ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

MAP-MATCHING SU PIATTAFORMA BIGDATA

Tesi in

DATA MINING

Relatore

Prof. MATTEO GOLFARELLI

Co-relatore

Dott. MATTEO FRANCIA

Presentata da

FEDERICO VITALI

Terza Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

Map Matching

Trajectory Mining

Big Data

Hidden Markov Model

Spark

Alla mia famiglia

Indice

Sommario

Nell'ambito dell'analisi dei dati di movimento atto all'estrazione di informazioni utili, il map matching ha l'obiettivo di proiettare i punti GPS generati dagli oggetti in movimento sopra i segmenti stradali in modo da rappresentare l'attuale posizione degli oggetti. Fino ad ora, il map matching è stato sfruttato in ambiti come l'analisi del traffico, l'estrazione dei percorsi frequenti e la predizione della posizione degli oggetti, oltre a rappresentare un'importante fase di pre-processing nell'intero procedimento di trajectory mining. Sfortunatamente, le implementazioni allo stato dell'arte degli algoritmi di map matching sono tutte sequenziali o inefficienti. In questa tesi viene quindi proposto un algoritmo il quale si basa su di un algoritmo sequenziale conosciuto per la sua accuratezza ed efficienza il quale viene completamente riformulato in maniera distribuita in modo tale da raggiungere anche un elevata scalabilità nel caso di utilizzo con i big data. Inoltre, viene migliorata la robustezza dell'algoritmo, il quale è basato sull'Hidden Markov Model di primo ordine, introducendo una strategia per gestire i possibili buchi di informazione che si possono venire a creare tra i segmenti stradali assegnati. Infatti, il problema può accadere in caso di campionamento variabile dei punti GPS in aree urbane con un elevata frammentazione dei segmenti stradali. L'implementazione è basata su Apache Spark e testata su un dataset di oltre 7.8 milioni di punti GPS nella città di Milano.

Introduzione

L'utilizzo di massa negli ultimi decenni di dispositivi portatili, come per esempio gli smartphone, considerati oramai indispensabili per lo stile di vita moderno, ha fatto aumentare in maniera esponenziale il quantitativo di studi effettuati dalle aziende e dagli enti sul comportamento delle persone stesse, cercando di sfruttare i dati che tali dispositivi emanano in continuazione. Una delle metodologie maggiormente sfruttate negli ultimi anni atte allo studio dei comportamenti delle persone per trarre da tali studi possibili vantaggi strategici in ambito aziendale è basarsi sulle informazioni che si possono estrapolare dagli spostamenti effettuati. Se si pensa oltretutto che ogni dispositivo tecnologico considerato "smart" possiede un rilevatore di movimento è facile intuire l'enorme quantitativo di dati di spostamento che è possibile raccogliere in modo da ricostruire i movimenti. Durante lo studio di tali spostamenti è molto spesso fondamentale (in molti casi un vero e proprio requisito tecnico) sapere il percorso che è stato fatto e, di norma, data la sola rilevazione di posizione dell'oggetto, non è presente tale informazione.

Il Map Matching, che ha il compito di associare gli spostamenti con il reale percorso che lo spostamento fa, interviene per risolvere questo tipo di problematica. In questo modo avendo a disposizione anche tale informazione, lo studio può essere più accurato e maggiormente approfondito ponendosi come base per tutte le successive tecniche applicabili in ambito di estrazione di informazioni utili dagli spostamenti. Se si aggiunge a ciò la problematica che il quantitativo di dati da analizzare aumenta in maniera esponenziale ogni giorno diventa fondamentale riuscire a gestirli efficacemente e nel minor tempo possibile.

Questa tesi presenta quindi una metodologia di map matching applicata nell'ambito dei Big Data, di conseguenza con lo scopo di gestire una grande mole di dati in maniera efficiente. Per riuscire a raggiungere tale scopo si è sfruttato il famoso framework Apache Hadoop. Perché la procedura di map matching risulti invece efficace ed accurata è stato effettuato uno studio approfondito dello stato dell'arte per capire le varie metodologie sviluppate e se fossero presenti anche tecniche facenti uso di metodologie distribuite valide. Si è notato una carenza di approcci robusti in ambito distribuito e quindi si è deciso di utilizzare un approccio sequenziale e distribuirlo completamente. L'approccio è basato sui modelli Markoviani nascosti (HMM) che garantiscono un'ottima accuratezza e una buona robustezza a casi particolarmente complessi da associare alla strada corretta. Tale approccio viene ulteriormente esteso includendo una metodologia di ricostruzione del percorso in modo da completare il map matching in caso di segmenti stradali risultanti dall'algoritmo non connessi.

La tesi è composta dal Capitolo 1 dove viene introdotto l'ambito di studio, cioè tutto ciò che riguarda le traiettorie e il processo di analisi applicabile ad esse, seguito dal caso di studio specifico di questa tesi cioè il map matching. Il Capitolo 2 introduce invece il secondo elemento principale dell'ambito di tesi cioè i Big Data e le varie tecnologie usate per gestirli al meglio e nel modo più efficace ed efficiente. A seguire, il Capitolo 3 introduce il modello Markoviano nei suoi dettagli e come è possibile risolverlo e applicarlo al nostro caso di studio del map matching. Il Capitolo 4 è dedicato allo studio che è stato fatto sull'intero panorama del map matching per capire lo stato dell'arte delle varie metodologie applicabili ed estrarre la miglior tecnica possibile da utilizzare poi nell'ambito distribuito. Nel Capitolo 5 viene definito l'algoritmo che è stato implementato con la relativa implementazione distribuita ed infine l'ultimo capitolo, cioè il Capitolo 6 è incentrato su tutti i test effettuati per capire l'efficacia e l'efficienza della metodologia proposta, punto fondamentale per un qualsiasi elaborato in ambito Big Data.

Capitolo 1

Il Map Matching

In questo primo capitolo viene introdotto nel dettaglio l'elemento principale della tesi ossia la traiettoria spiegando che cosa è e come viene sfruttata per estrarre da essa informazioni utili. Successivamente viene descritto il caso di studio su cui si concentra il lavoro di tesi, cioè il map matching, introducendolo e analizzandolo nel dettaglio.

1.1 Le Traiettorie

Una traiettoria di un oggetto in movimento è una traccia, un percorso che l'oggetto percorre nello spazio geografico. Generalmente viene identificata come una sequenza di posizioni geografiche a cui è corrisposto uno specifico *timestamp* (cioè una marca temporale standardizzata dall'ISO, International Organization for Standardization, con l' ISO 8601 nel piano spazio temporale. Può quindi essere rappresentata come una sequenza di insiemi di attributi all'interno di un unico elemento dove ogni elemento della sequenza è formato dalla posizione geografica composta a sua volta da latitudine e longitudine e dalla caratteristica temporale, il timestamp. Questo insieme di elementi spaziali è infatti ordinato nel tempo. È importante notare che le caratteristiche delle traiettorie variano a seconda di molteplici fattori tra cui:

- La tipologia di oggetto su cui è definita la traiettoria, perché ne cambia

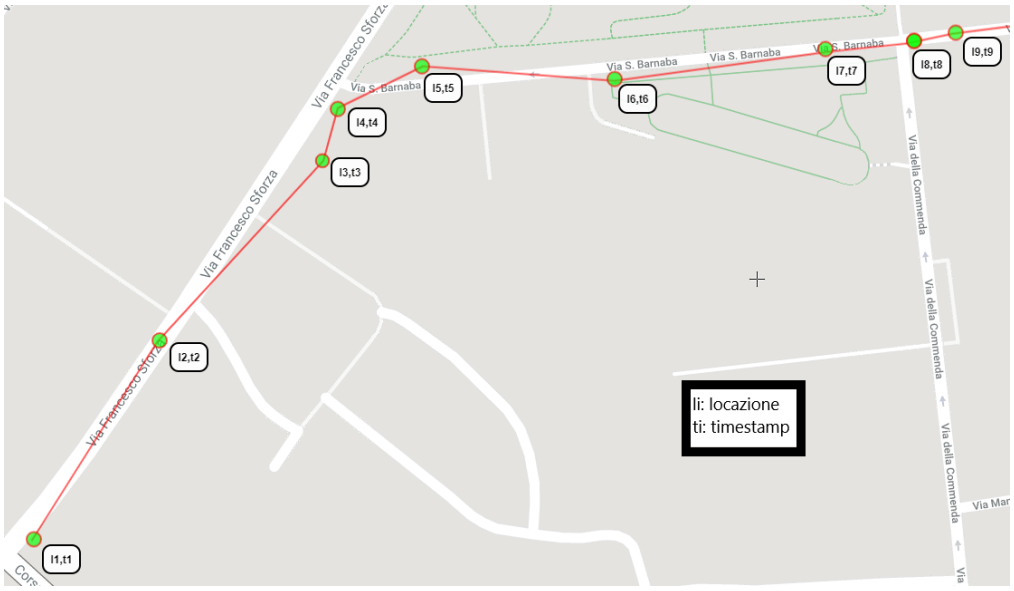


Figura 1.1: Esempio di una traiettoria GPS

il modello e il livello di dettaglio; basti infatti pensare alla differenza tra una traiettoria di una persona, che ne modella il comportamento di giorno e di notte, comprendendo tutti i suoi spostamenti e una traiettoria di un veicolo, la quale ne modella la tratta percorsa in un certo periodo di tempo. O addirittura la differenza di dettaglio di spostamento tra una persona e un tifone, e così via.

- Come tale traiettoria è ottenuta, dipendente sopratutto dal dispositivo di rilevazione della posizione e dalla tecnica di campionamento che determina quanto la traiettoria è definita nel tempo. Si possono infatti suddividere i vari metodi di creazione di traiettorie in:

  – **Global Positioning System** GPS, cioè quel componente del dispositivo sempre connesso ad una rete satellitare dedicata che comunica con esso fornendo la posizione in coordinate geografiche dell'oggetto.

  – **Global System for Mobile Communications** GSM, cioè la rete di comunicazione cellulare, dove da essa possono essere estratte

traiettorie definite da sequenze di posizioni delle celle telefoniche attraversate.

- **Geo-social network** Inteso come il contenuto estratto dai vari social media il quale può essere codificato geograficamente.
- **Radio Frequency Identification** RFID, la quale rilevazione dei punti della traiettoria è data dai lettori RFID attraversati.
- **Wi-Fi** Dove lo spostamento è codificato in una sequenza di reti wi-fi dove il dispositivo si è collegato.

1.2 Mining di Traiettorie - Trajectory Data Mining

Dato l'aumentare negli ultimi anni del quantitativo di dispositivi che rilevano il posizionamento delle persone e il susseguente aumento dei dati che vengono raccolti da essi sono aumentati gli studi su come estrarre informazioni da tali dati. L'estrazione di conoscenza dai dati è chiamata Data Mining, le quali tecniche richiedono un quantitativo notevole di dati per estrarre informazioni utili da essi. Per quanto riguarda le traiettorie, il Data Mining su di esse (chiamato anche Trajectory Data Mining) sfocia in diversi ambiti di studio come:

- Utilizzo degli spostamenti di specie animali per capirne il comportamento e studiarli a fondo. Da essi vengono anche dedotti possibili cambiamenti del clima e dell'ambiente che portano gli animali a spostamenti non standard [35].

- Predizione di eventi catastrofici come tornado o tifoni basandosi sullo studio di traiettorie dei fenomeni atmosferici [2].

- Studiare il comportamento degli esseri umani dai suoi spostamenti, profilando gruppi di persone cercando di capirne i maggiori interessi per

esempio i posti che vengono visitati maggiormente o i ristoranti più apprezzati. Questo tipo di studio può essere molto utile a diverse aziende, che si possono basare sui risultati per cercare di ottenere dei vantaggi in ambito strategico e concorrenziale [34].

- Studi sui flussi di persone a diversi orari del giorno i quali aiutano le agenzie dei trasporti (come taxi o bus) o gli stessi comuni nel miglioramento della gestione del flusso di traffico nelle proprie strade [33].

Tecnicamente parlando invece, il Trajectory Data Mining può essere considerato come una collezione di varie tecniche le quali possono essere raggruppate in procedure standard appartenenti al mining di dati. Tutto ciò può essere infatti visto come un vero e proprio framework i quali strati, o *layer*, comprendono una fase del mining di dati. Come si può notare in Figura 1.2, schema

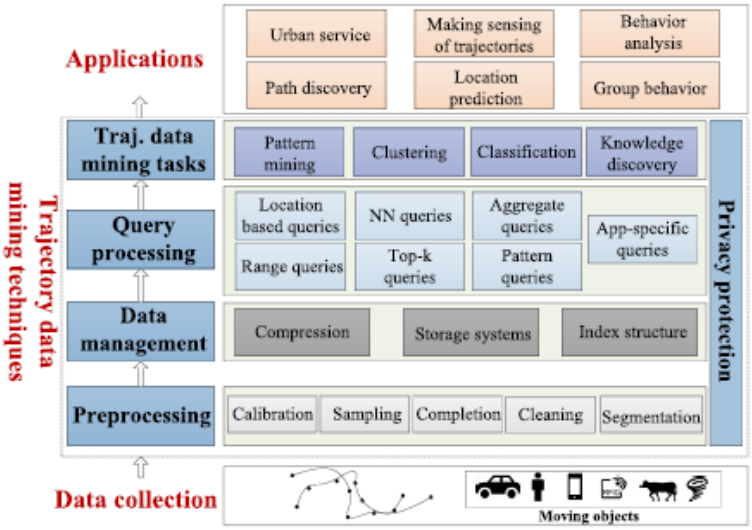


Figura 1.2: Framework del Trajectory Mining [1]

redatto e descritto in [1], si possono distinguere tre layer più generici tra essi strettamente collegati definiti come:

- **Raccolta dei dati** - include come detto in precedenza tutte le tecniche di raccolta delle traiettorie dai più svariati oggetti di rilevazione della posizione.

- **Tecniche di mining delle traiettorie** - include strategie e metodi per estrarre conoscenza dai dati, richiede come ogni tecnica di data mining un insieme consistente di dati a disposizione per lavorare al meglio.

- **Applicazioni** - parte finale del processo che sfrutta le tecniche di mining dei dati del livello precedente per i più svariati scopi. Si possono suddividere in:

  - Applicazioni di *Path Discovery* dove l'obiettivo è quello di scoprire il percorso maggiormente consigliato, a seconda del caso d'uso e dai vincoli dati dalla rete stradale.
  - Applicazioni di predizione della destinazione utilizzate per inviare pubblicità mirata a seconda della destinazione che viene predetta.
  - Applicazioni utilizzate per il *profiling* degli utenti o di gruppi di utenti dove si cerca di capire i motivi di certi spostamenti a diversi orari del giorno.
  - Applicazioni utilizzate per il servizio urbano, per capire come gestire le strade e tutto ciò che gira intorno ad esse (come per esempio dove inserire le piazzole di sosta per la ricarica di macchine elettriche dati studi sulle traiettorie prodotte da esse).

Concentrandosi sulla parte focale del mining di traiettorie, cioè la parte inclusa nelle tecniche di mining in Figura 1.2, essa viene suddivisa a sua volta in cinque passi i quali sono strettamente connessi tra loro e definiscono tutte le procedure che si possono attuare sulle collezioni di traiettorie alcune delle quali sono comprese come parti fondamentali di questa stessa tesi:

- **Preprocessing** - Il pre-processamento delle traiettorie è un fondamentale passo base da effettuare all'inizio dell'intero processo perché punta a migliorare la qualità dei dati della traiettoria e al generare le traiettorie migliori possibili. Infatti, nei casi reali, i dati arrivano semplicemente identificati come: chi ha effettuato il movimento ed insieme delle rilevazioni spaziali e temporali dello spostamento. Tale traiettoria deve essere

successivamente definita a seconda dello scopo prossimo degli step successivi, per esempio, possono essere collezionati spostamenti umani che comprendono intere settimane, tale spostamento va suddiviso secondo uno specifico criterio per estrarre da esso tutti gli spostamenti singoli che possono comprendere traiettorie in macchina, in bicicletta, di corsa e così via, definendoli a seconda degli orari di campionamento degli spostamenti (ciò in gergo è chiamata **Segmentazione**). Un'altra definizione di Segmentazione la quale è lo scopo di questa tesi, è il **Map-Matching**, che verrà dettagliato accuratamente nel Paragrafo 1.3. Un altro punto fondamentale di questa branchia è la pulizia del dato, è infatti normale che una traiettoria contenga al proprio interno molteplici punti fuori posto, causati da errori di rilevazione dei sensori, i quali possono inficiare negativamente nei successivi step di mining. Questa tecnica è infatti chiamata in gergo **Ricerca dell'Outlier**. D'altro canto, per alcune tecniche, le traiettorie sono fin troppo definite e i troppi dati vanno a inficiare inutilmente sui tempi di esecuzione degli algoritmi senza migliorarne il risultato. Su di esse è da attuare quello che viene chiamato in gergo **Sampling**. Per altre traiettorie invece, si ha il caso contrario, ed essendo esse troppo poco definite a causa del basso periodo di campionamento si cerca di completarle sulla base di criteri specifici come per esempio l'insieme degli elementi descrittivi delle strade sottostanti alla traiettoria.

- **Data Management** - In quasi tutti i casi di utilizzo, le traiettorie vengono sfruttate in maniera massiva e di conseguenza la gestione di tale mole di dati è fondamentale. Qui entrano in gioco una serie di metodologie e di tecniche per salvare il dato in maniera distribuita ed efficiente, effettuando su esso anche tecniche di compressione, semplificazione e indicizzazione, per velocizzarne il caricamento e il salvataggio.

- **Query Processing** - La base di un applicazione di data mining fa ovviamente largo uso delle query, cioè delle interrogazioni sulla mole di dati. Per quanto riguarda le traiettorie vengono utilizzate varie tipologie di

query per raccogliere i dati, molte delle quali fanno largo uso del calcolo della distanza. Fanno parte di questo gruppo le *range qeury*, dove viene specificato un qualche costrutto spaziale come l'intersezione o l'inclusione delle traiettoria tra di esse, o le *distance query* dove invece viene calcolata la distanza. Naturalmente sulle traiettorie vengono anche applicate interrogazioni considerate più "classiche", come filtri su specifici attributi, aggregazioni o selezioni per estrarre specifiche informazioni. La cosa importante per tutte queste tipologie di interrogazioni è che necessitano di una tecnica per l'indicizzazione la quale riesca a cooperare in maniera efficiente con la grande mole di dati a disposizione.

- **Trajectory Data Mining Task** - Sezione che comprende i veri e propri algoritmi utilizzati in Data Mining. Dal Pattern Mining, che è utilizzato per analizzare i pattern di movimento, alla classificazione, dove le traiettorie vengono raggruppate perché accomunate da specifiche fondamenta e determinate regole, alla clusterizzazione, dove vengono identificati dei gruppi omogenei ma a differenza della classificazione sono accomunati da regole occulte fino al momento della loro scoperta.

- **Privacy Protection** - Il tentativo di preservare la privacy è un problema cruciale in qualsiasi procedura di mining di traiettorie. Negli ultimi anni si sono moltiplicati gli esempi di algoritmi che illustrano come processare le traiettorie e allo stesso tempo proteggere le informazioni sensibili degli utenti.

1.3 Formalizzazione del Problema di Map Matching

Entrando più nello specifico della tesi, lo studio effettuato si è concentrato in una problematica che affligge il mining di traiettorie e nello specifico si posiziona nella parte di pre-processamento. È infatti importante per molte applicazioni utilizzare traiettorie molto accurate nell'ordine delle strade attraver-

sate e percorse. Anche le stesse query utilizzate per l'estrazione di conoscenza dai dati possono richiedere come livello di dettaglio della traiettoria la strada che ha realmente percorso, sia per aumentare l'accuratezza dell'analisi, sia per aggiungere informazioni utili da cui si possono estrarre deduzioni inerenti ai percorsi stradali. Per trasformare quindi le traiettorie, le quali non includono l'informazione sulla strada percorsa, nel percorso realmente attraversato interviene il così definito *Map Matching*. Si definisce infatti come il problema di come accoppiare una lista di posizioni composte da coordinate geografiche ad un modello logico del mondo reale. Gli elementi necessari atti a descrivere un qualsiasi tipo di problema di map matching comprendono questi elementi così definiti:

- **Collezione di punti GPS** - chiamato anche *GPS-Log*, è un insieme di punti GPS L=$\{p_1, p_2, p_3 ... p_n\}$ dove ogni punto p contiene al suo interno le coordinate spaziali composte da latitudine $p.lat$ e longitudine $p.lon$ e il valore temporale dato dal timestamp $p.t$. Può essere confuso come il termine traiettoria perché in molti lavori è il livello di dettaglio adatto del dato su cui poi andare a creare algoritmi di Trajectory Mining.

- **Traiettoria** - come già spiegato nel Paragrafo 1.1 e visibile nella Figura 1.1 la traiettoria è una sequenza di punti GPS. Ma più specificatamente, nel map matching, una traiettoria rispetta per ogni coppia di punti una soglia temporale δt tale che $T : p_1 \rightarrow p_2 \rightarrow ... \rightarrow p_n$, dove $p_i \in L$, e $0 < p_{i+1}.t - p_i.t < \delta t$ con $(1 <= i < n)$. È un vincolo che va rispettato perché é immaginabile fare un map matching accurato con traiettorie dove i punti sono distanti temporalmente ore o giorni interi.

- **Segmento Stradale** - un segmento stradale e viene definito come un arco direzionato a cui è associato un identificativo univoco $e.id$ il quale ha una certa lunghezza $e.l$ ed è generalmente delimitato da un punto iniziale $e.inizio$ ed da un punto finale $e.fine$ ed una serie di punti intermedi che ne mappano la struttura. Importante notare che una strada può essere composta a sua volta da molteplici segmenti stradali.

- **Rete Stradale** - è un grafo direzionato $G(V, E)$ dove V è un set di vertici i quali rappresentano le intersezioni e i punti terminali dei segmenti stradali, ed E che è un set di archi i quali rappresentano i segmenti stradali.

- **Percorso** - dati due vertici V_i, V_j in una rete stradale G, un percorso P è un set di segmenti stradali tali che inizino in V_i e finiscano in V_j, tali che P: $e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_n$, dove $e1.inizio = V_i$, $e_n.fine = V_j$, $e_k.fine = e_{k+1}.inizio$ con $i <= k < n$.

Dove quindi fare map matching **significa**: **data una collezione di punti GPS dalla quale vengono estratte le traiettorie T e data una rete stradale $G(V, E)$ composta da segmenti stradali E connessi da nodi in comune V, lo scopo è trovare il miglior percorso P in G accoppiato alla traiettoria T tale che esso sia identico con il percorso effettuato realmente.**
Formalizzato l'obiettivo, tecnicamente parlando la tipica struttura di un algoritmo che fa map matching si può riassumere in due macro parti:

- **Selezione dei candidati** - secondo una determinata tecnica vengono selezionati per ogni punto da analizzare un certo quantitativo di segmenti di strada considerati come possibili segmenti realmente percorsi dalla traiettoria.

- **Costruzione della strada percorsa** - dati i candidati per ogni punto, si costruisce una qualche struttura che possa connetterli tra di loro e si calcolano determinati punteggi chiamati *score* per ogni possibilità di connessione a seconda di vari fattori. In qualsiasi caso, a fine algoritmo viene selezionata la sequenza di strade tale che la sommatoria degli score dati dalle connessioni tra esse è massima (se è un problema dove si cerca di massimizzare la probabilità che un segmento sia associato al punto GPS) o minima (se invece è il problema inverso).

E' bene specificare che entrambi gli step sono considerati di pari importanza, dove la selezione dei candidati è cruciale per l'accuratezza finale del risultato,

ponendo la base su cui poi applicare le tecniche della seconda fase, la quale varia maggiormente a seconda della tipologia di algoritmo, le cui differenze verranno specificate nei capitoli seguenti.

1.4 Applicazioni e Casi d'Uso

Essendo il map matching una metodologia considerata "fondamenta" di tantissime applicazioni che estraggono informazioni dalle traiettorie, i casi d'uso reali in letteratura sono molteplici. Si può passare ad applicazioni che puntano sulla ricerca dinamica del percorso da consigliare a seconda di precedenti studi sui dati [3] alle applicazioni di aiuto ai tassisti o alle società di trasporto automobilistico che creano dei veri e propri sistemi intelligenti per consigliare in che zone spostarsi per ottimizzare il guadagno delle corse sui costi di trasporto [4]. Esistono studi sulla morfologia delle strade e sui percorsi effettuati su di esse dai quali vengono estratte informazioni socio-culturali, chiamate "zone funzionali" [5], ai più classici casi di mining delle traiettorie come l'utilizzo di tecniche di Clustering per estrarre informazioni su gruppi di persone e profilarle per scopi di marketing specifico come già spiegato nel Paragrafo 1.2.

Capitolo 2

I Big Data e le Tecnologie Utilizzate

In questo capito sono descritte tutte le tecnologie impiegate nell'ambito della tesi. Considerando che l'obiettivo principale è stato quello di applicare una tecnica di segmentazione di traiettorie, cioè fare map matching, in un ambito dove il quantitativo di dati è spropositato, diventa fondamentale selezionare le giuste tecnologie per riuscire ad ottenere buoni risultati in termini di tempi ed efficienza di calcolo. È presente una introduzione all'ambito dei Big Data, che in questo caso di studio equivalgono ad una grande mole di traiettorie GPS, passando poi alla scelta del sistema per riuscire a gestirli al meglio, cioè Apache Hadoop, compreso delle sue sottoparti più importanti che sono state sfruttate. È oltretutto dettagliata, con un paragrafo a parte, una estensione non ufficiale dello stesso Hadoop che è stata di grande aiuto per alcuni passaggi del procedimento di map matching applicato nel contesto distribuito.

2.1 Big Data

Il termine *Big Data* può essere fuorviante, la sua traduzione italiana in "grandi dati" o "grossi dati" fa pensare all'enorme quantità di dati oggi disponibili nei più svariati settori aziendali e, in automatico, porta a concludere che

per "rivoluzione dei big data" si intenda l'opportunità oggi disponibile nell'avere così tante informazioni a disposizione delle aziende. Questa conclusione è vera solamente in minima parte perché la vera rivoluzione a cui ci si riferisce parlando di big data non è questa, quanto la capacità di usare tutte queste informazioni per elaborare, analizzare e trovare riscontri oggettivi su diverse tematiche.

2.1.1 Definizione e Caratteristiche

Si parla di big data quando si ha un insieme talmente grande e complesso di dati che richiede la definizione di nuovi strumenti e metodologie per estrapolare, gestire e processare informazioni entro un tempo ragionevole.[18] Si possono considerare i big data quell'insieme di elementi il quale possiede tre caratteristiche specifiche fondamentali, diventate poi cinque con il passare degli anni [19]:

- **Volume** - si riferisce ovviamente alla mole di dati generati ogni secondo. Basti pensare che le sorgenti da dove vengono presi i dati sono **eterogenee** comprendendo i database, tutti i dati estrapolati dai vari *social media*, i valori rilevati dai più svariati tipi di sensori e così via. Si è stimato infatti che nel 2014 il 90% dei dati generati in tutto il mondo era da ricondurre solamente ai due anni precedenti [1] e tale valore è addirittura aumentato negli ultimi anni esponenzialmente.

- **Varietà** - per ottenere analisi più accurate e approfondite bisogna prendere in considerazione i dati indipendentemente dal fatto che essi siano strutturati (cioè descritti da uno schema, dove un'esempio sono i dati gestiti dai classici database), semi-strutturati (che comprendono tutti quei dati strutturati non conformi alla struttura formale dei modelli di dati associati ai database relazioni) e non strutturati.

- **Velocità** - si riferisce alla velocità con cui i nuovi dati vengono generati. Non solo la celerità nella generazione dei dati, ma anche la necessità

[1] https://www.domo.com/learn/data-never-sleeps-6

che questi dati/informazioni arrivino in tempo reale al fine di effettuare analisi su di essi prima che diventino "obsoleti" al fine dell'analisi stessa.

- **Veridicità** - considerando le caratteristiche di Varietà e Velocità è molto difficile garantire che il dato da analizzare sia sempre di buona qualità. Se si possiede solamente dati imprecisi, anche l'analisi su di essi risulterà sbagliata e dovendo basare di solito delle decisioni strategiche aziendali sui risultati dell'analisi, bisogna sempre provare ad assegnare un livello di veridicità ai set di dati prima di analizzarli.

- **Valore** - si riferisce alla capacità di trasformare i dati in valore. È importante analizzare bene che vantaggi di business può portare l'analisi di tali dati perché l'intero processo, compreso sia dalla raccolta che dall'analisi, richiede un vero e proprio investimento economico e di tempo.

2.1.2 Tecnologie per i Big Data

Il crescere incessante della mole di dati generati da sorgenti di dati diverse tra loro ha posto l'attenzione per le aziende sul come archiviare, gestire ed elaborare tali dati in modo da ricavarne un profitto. Il problema principale è stata l'inadeguatezze dei classici database relazionali nel gestire tali dati, sia in termini di volumi che in termini di performance. Si sono studiati quindi nuovi modelli di elaborazione per riuscire a gestire i big data in maniera efficiente ed agile. Tali architetture sono accomunate da tre caratteristiche principali:

1. **Architettura distribuita** - utilizzo di *cluster* di computer connessi tra loro al fine di cooperare al raggiungimento di un obiettivo comune realizzando la così definita **scalabilità orizzontale**, cioè la possibilità di aggiungere alla struttura un numero teoricamente infinito di macchine aumentando il quantitativo di potenza di calcolo e di memoria. È una struttura che quindi riesce a scalare al numero di macchine dando la possibilità di utilizzare come componenti del cluster dei computer equipaggiati da hardware comune a basso costo, in totale confronto al caso dei "super computer" nei quali si punta alla **scalabilità verticale**

dove la macchina è una sola la quale sfrutta hardware specializzato in computazione massiva il quale possiede un costo molto elevato.

2. **Tolleranza ai guasti** - le architetture devono essere progettate per riuscire a tollerare i guasti ed i fallimenti delle varie macchine che le compongono, perciò vengono sfruttate diverse tecniche di replicazione delle risorse sulle differenti macchine che compongono il cluster.

3. **Calcolo distribuito** - il modello utilizzato per elaborare i dati è anch'esso distribuito in modo tale da sfruttare al meglio le risorse messe a disposizione dall'intero insieme di macchine.

Uno dei framework più diffusi ed utilizzati per l'archiviazione, la gestione e il calcolo dei big data, il quale è stato sfruttato per l'implementazione di questa tesi, è Apache Hadoop.

2.2 Framework Utilizzato - Apache Hadoop

Hadoop è un framework software sotto licenza libera che supporta il processamento distribuito di un grande quantitativo di dati tramite un cluster di computer dando la possibilità di scrivere facilmente applicazioni distribuite utilizzando semplici modelli di programmazione. È stato ispirato dalla metodologia MapReduce di Google e dal suo file system distribuito [20]. È stato inizialmente proposto come stessa implementazione della tecnica di Google ma in codice Java per poi diventare un progetto Apache indipendente dove il numero di contributori è andato in aumentando ogni anno. Il framework garantisce un'elevata affidabilità: le anomalie e tutti gli eventuali problemi del sistema sono gestiti a livello applicativo anziché utilizzare sistemi hardware specifici per garantire un'alta disponibilità. Un'altra caratteristica di Hadoop è la scalabilità che è realizzabile semplicemente aggiungendo nodi (cioè macchine composte da hardware "di comodità") al cluster in esercizio. È bene precisare infatti che Hadoop è un progetto di alto livello, nel senso che mette a disposizione librerie per programmare con tecniche distribuite nascondendo al

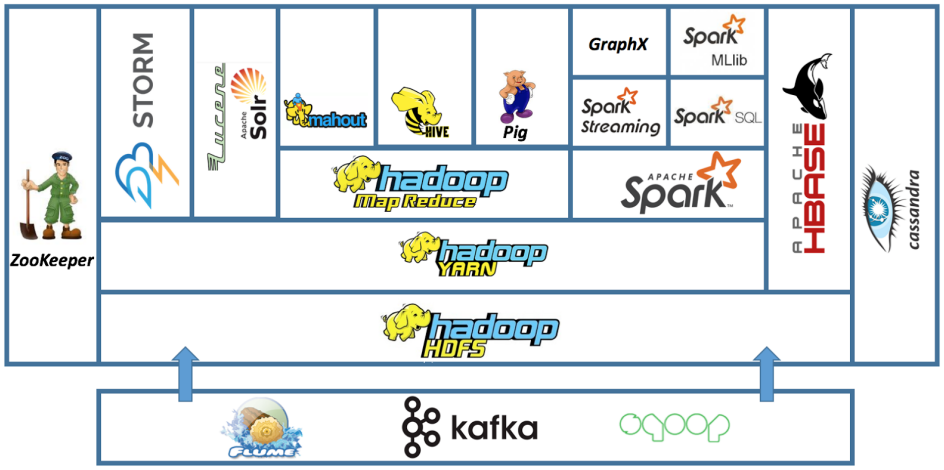


Figura 2.1: Ecosistema Hadoop [36]

suo interno tutti i dettagli di sistema separando quindi il "cosa" dal "come", semplificando enormemente lo sviluppo architetturale e l'implementazione dei programmi liberando l'utente dal come riuscire ad implementare in maniera distribuita e parallela fronteggiando tutte le problematiche del caso.

2.2.1 Architettura

Come si nota dalla Figura 2.1 Hadoop è composto da un insieme differente di componenti (o moduli) che cooperano tra di loro formando quello che può essere definito come un ecosistema. Sebbene sia possibile inserire manualmente solo i moduli software che si vogliono sfruttare nel cluster Hadoop, in molti casi è preferibile usare una distribuzione nella quale sono già presenti ed installati tutti questi software. Nel caso del cluster su cui si è svolta la tesi è stata installata una distribuzione **Cloudera**[2], la quale è anch'essa sotto licenza libera e include i principali pacchetti dell'ecosistema.

[2]https://www.cloudera.com/products/open-source/apache-hadoop/key-cdh-components.html

2.2.2 HDFS

L'Hadoop Distribuited File System è un file system distribuito, portabile e scalabile, scritto e progettato per l'ecosistema Hadoop ed ideato per contenere file di dimensioni elevate con pattern di accesso ai dati ad altissimo *throughput* (cioè mole di trasferimento dati). Garantisce che i dati siano ridondanti nel cluster rendendo le operazioni sui dati stessi immuni dall'eventuale guasto di un nodo. HDFS accetta dati in qualsiasi formato, strutturati e non strutturati organizzando i file in una struttura gerarchica di cartelle. Dal punto di vista dell'architettura, un cluster è costituito dai seguenti tipi di nodi operando con un pattern master-slave:

- **NameNode** - il *master* - è l'applicazione che gira sul server principale, gestisce il file system ed in particolare il namespace, cioè l'elenco dei nomi dei file (con relativi percorsi) e della posizione della sua suddivisione. Infatti, i file al suo interno sono organizzati in **blocchi** da 64 o 128Mb, creati con lo scopo di riuscire a contenere file di qualsiasi dimensione, anche quando i file stessi sono più grandi del disco fisso di una delle macchine del cluster. Ciò rende anche più semplice salvare i file e replicarli. Oltretutto controlla l'accesso ai vari file e determina come i blocchi devono essere distribuiti sui nodi del cluster e la strategia di replica che garantisce l'affidabilità del sistema. L'idea alla base è che quando si deve computare una grande quantità di dati è molto meno costoso portare il codice verso i dati che il contrario (proprietà della **Data Locality**).

- **DataNode** - lo *slave* - applicazioni che girano su altri nodi del cluster (generalmente una per nodo) e gestiscono fisicamente lo spazio di archiviazione di ciascun nodo. Queste applicazioni eseguono le operazioni di lettura e scrittura richieste dai client e gestiscono fisicamente la creazione, la cancellazione e la replica dei blocchi dati.

Oltre a questi due tipi principali sono presenti anche un altro paio di nodi importanti. Il **CheckPointNode**, il quale offre un servizio che aiuta il NameNode ad essere più efficiente scaricando periodicamente i "log" dei cam-

biamenti dal NameNode e unendoli in *snapshot* (definite letteralmente come delle istantanee che catturano lo stato corrente del sistema) i quali serviranno per recuperare lo stato in caso di fallimento del NameNode. Il **BackupNode**, che funziona come replica del CheckPointNode generato per mantenere le proprietà di elevata disponibilità per ogni elemento di HDFS.

2.2.3 YARN

Yet Another Resource Negotiator è un elemento fondamentale dell'ecosistema Hadoop, integrato ad esso solamente dalla sua seconda versione per migliorarne le prestazioni ed eliminare i vari problemi che si venivano a creare con la prima versione dati dall'assenza di scalabilità. Si occupa infatti di gestire le risorse del cluster mettendo a disposizioni un insieme di procedure le quali vengono sfruttate poi dai framework distribuiti come MapReduce e Spark (quindi non direttamente dagli utenti). Tali servizi sono resi tramite due programmi considerati "daemon" cioè eseguiti in background i quali sono sempre attivi:

- **Resource Manager** - presente uno per cluster, è composto dallo schedulatore che alloca le risorse per le varie applicazioni a seconda di requisiti esposti e dal gestore delle applicazioni il quale è responsabile di accettare inizialmente le richieste di avvio delle applicazioni e negoziare il primo contenitore per eseguire l'**Application Master Process** il quale verrà poi controllato con la possibilità di riavviarlo in caso di fallimenti.
- **Node Manager** - presente uno per nodo, è responsabile dei "container" cioè entità astratte utilizzate per eseguire i processi delle applicazioni con un insieme contenuto di risorse. Oltre a ciò si occupa di monitorarne il consumo, riportando tutto al Resource Manager.

Il punto forte di YARN sta nel fatto che per ogni applicazione lanciata sul cluster viene creato in un container un **Application Master Process** il quale sarà poi lui ad avere il compito di gestire l'applicazione decidendo se stanziare altri container per gestisce il calcolo distribuito, andando a sopperire ai

problemi della prima versione di Hadoop dove tutte queste funzionalità erano incentrate nel Resource Manager formando quello che viene definito "collo di bottiglia", cioè quel fenomeno che si verifica quando le prestazioni di un intero sistema o le sue capacità sono fortemente vincolate da un singolo componente.

2.2.4 MapReduce

Hadoop MapReduce è un framework per la creazione di applicazioni in grado di elaborare grandi quantità di dati in parallelo basandosi sul concetto di *functional programming*. A differenza della programmazione *multithreading*, in cui i thread condividono i dati oggetto delle elaborazioni presentando così una certa complessità proprio nel coordinare l'accesso alle risorse condivise, nel functional programming la condivisione dei dati è eliminata, e questi sono passati tra le funzioni come parametri o valori di ritorno. È un elemento fondamentale dell'ecosistema Hadoop perché implementa appunto il modello di programmazione MapReduce che, come si può notare dalla Figura 2.1, sta alla base di tanti altri progetti e framework che negli anni si sono sviluppati per ottimizzare il processo di analisi dei big data. MapReduce lavora secondo il principio del divide et impera, suddividendo l'operazione di calcolo in diverse parti processate in modo autonomo. Una volta che ciascuna parte del problema è stata calcolata, i vari risultati parziali sono "ridotti" (cioè ricomposti) a un unico risultato finale. È MapReduce stesso che si occupa dell'esecuzione dei vari task di calcolo, del loro monitoraggio e della ripetizione dell'esecuzione in caso si verifichino problemi. Un programma MapReduce è definito **Job** e consiste principalmente in due fasi:

- **Map** - applica una funzione di trasformazione definita dall'utente su ogni elemento di ogni blocco del file in ingresso ritornando una coppia chiave-valore il quale viene temporaneamente salvato nel disco.

- **Reduce** - i file intermedi vengono trasferiti nei nodi dove viene eseguita la funzione di reduce la quale combina tutti i valori aventi la stessa chiave a seconda del codice definito dall'utente. Importante notare che l'input

viene anche ordinato per chiave prima di essere passato alla funzione di Reduce.

Tutte le coppie chiave-valore che escono dalla procedura di Reduce vengono infine scritte in maniera definitiva su HDFS.

2.2.5 Hive

Apache Hive è un progetto software di *data warehouse* che come si può notare da Figura 2.1 si posiziona in cima allo stack e viene utilizzato per l'analisi e l'interrogazione dei dati strutturati e non strutturati presenti su HDFS. Mette a disposizione una interfaccia simile a quella SQL, utilizzata per interrogare i classici database relazionali, dove però ogni operazione è implementata in maniera distribuita tramite MapReduce. Tale linguaggio è chiamato HiveQL che in maniera trasparente converte tutte le tipologie di interrogazioni in Job MapReduce, senza quindi il bisogno di implementarle tramite il classico codice Java. Nella tesi è stato largamente utilizzato sia come metodologia di salvataggio in formato tabellare strutturato di tutti i tipi di dati utilizzati ma anche come strumento per analizzarli e poi prendere decisioni di implementazione.

2.2.6 Spark

Spark è un framework della famiglia Apache per il calcolo distribuito, nato per sopperire alle limitazioni che affliggono MapReduce. Infatti, MapReduce, è stato sviluppato più di un decennio fà dove scarseggiavano le aziende che lavoravano con i big data e si puntava di più al volume del dato che alla velocità e le performance delle analisi. Al giorno d'oggi invece, dove oramai tutte le aziende hanno a che fare con i big data, grazie anche alla facilità economica con cui è possibile creare e gestire un cluster, si punta di più alla reattività e velocità di analisi e viene quindi richiesta la possibilità di lavorare a 360 gradi sulla mole di dati applicando tecniche di analisi reattive e preformanti. Con MapReduce oltretutto si è obbligati a mantenere lo stile di programmazione dato dal suo paradigma composto da funzione di Map e funzione di Reduce,

rendendo molto complesso riuscire a costruire algoritmi senza concatenare tanti Job insieme, cosa che implica un numero spropositato letture e scrittura sul disco tramite HDFS rendendo molto lento il procedimento. Spark per risolvere ciò sfrutta delle primitive "*in memory*" multi livello fornendo prestazioni fino a 100 volte migliori rispetto ad applicazioni di MapReduce dando oltretutto la possibilità di specificare intere procedure, anche iterative, in un solo Job, riducendo quindi la complessità del codice dell'applicazione. I suoi punti di forza principali sono:

- **Caching dei dati in memoria** - mantiene i dati nella RAM fino al termine dell'intera sequenza di elaborazione, evitando di leggere e scrivere su disco.

- **Computazione "Lazy"** - ottimizza l'intera procedura di operazioni prima di eseguirla.

- **Pipelining efficiente** - in maniera automatica a seconda del tipo di operazioni da eseguire ed al quantitativo di dati da analizzare crea i task necessari concatenandoli e parallelizzandoli cercando sempre di rimanere in memoria e scrivere sul disco il minor numero di volte.

L'elemento fondamentale su cui si basa Spark è il **Resilient Distributed Dataset**, o RDD, che in sostanza è un set di dati distribuito in partizioni il quale possiede alcune proprietà chiave per il suo funzionamento efficiente e distribuito:

- Ogni RDD è immutabile, nel senso che una volta che è stato creato non lo si può cambiare, se non creandone un altro. Vincolo imposto per rendere più semplice la parallelizzazione, per riuscire a sfruttare la *laziness* e per mantenerli salvati in memoria.

- Restando in memoria è adatto a utilizzi ripetuti e iterati. Può essere facilmente ricreato in caso di fallimenti, migliorando drasticamente le performance.

- Ogni RDD è descritto da un set completo di metadati che consentono la ricostruzione di una delle sue partizioni in caso di problemi o fallimenti: dove si trovano le partizioni, quali sono gli RDD padre, quale è la sequenza di trasformazioni, detta **Lineage**, che lo ha generato.

- È valutato in maniera "lazy" nel senso che non viene effettivamente eseguito l'insieme di operazioni fino a quando non viene specificata quella che viene definita azione. E' bene precisare che ci sono solo due tipologie di operazioni per gli RDD: le **trasformazioni** e le **azioni**. Con le prime si va costruire un nuovo RDD dal precedente applicandoci una qualche modifica o filtraggio senza realmente computare il risultato ma modificando solamente i metadati interni all'RDD, con la seconda il risultato viene effettivamente computato. Per calcolare il risultato il programma viene passato allo *scheduler* di Spark che sulla base delle operazioni e dai metadati dei vari RDD interessati nel calcolo costruisce un grafo rappresentante la sequenza delle computazioni da eseguire sui dati. In base al grafo ottenuto (che è definito come **Directed Acyclic Graph** o DAG, cioè un grafo senza cicli in cui ogni nodo è una partizione di un RDD e ogni vertice è una trasformazione), lo scheduler determina il miglior modo possibile per distribuire le varie computazioni di trasformazione sui nodi. Il DAG viene infatti convertito in un piano di esecuzione applicando una divisione in **Stage** (dove i confini sono dati dalle operazioni che richiedono il mescolamento dei dati o l'utilizzo di partizioni che sono nella cache) e poi in **Task**, il quale rappresenta l'unità di esecuzione. Ogni task viene successivamente assegnato ai vari nodi del cluster basandosi sempre sul principio della *data locality*.

Ciò che è stato spiegato viene anche definito **Spark Core** cioè la base su cui si appoggiano i suoi moduli, o librerie, come si può notare da Figura 2.2, le quali si possono combinare in maniera semplice ed efficace. Ogni modulo ha dei compiti ben specifici di seguito descritti:

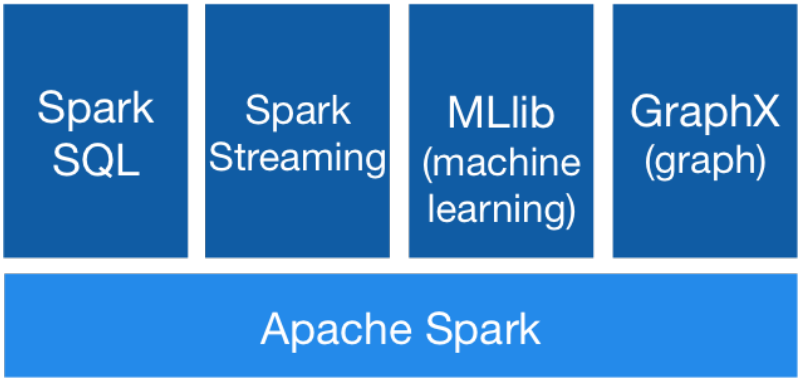


Figura 2.2: Stack di Spark [38]

SparkSQL

È il modulo di Spark atto a processare dati strutturati. Nasce con l'intento di integrare le tecniche del modello relazione, conosciuto e utilizzato da tantissime persone perché ampiamente usato nei sistemi relazionali, con la potenza di Spark. Dà la possibilità di mixare quelle che sono le interrogazioni classiche SQL con i programmi Spark e gli altri suoi moduli. Oltretutto, mette a disposizione un accesso ai dati univoco con la possibilità di caricarli da diverse sorgenti tra cui le tabelle Hive. Ciò è importantissimo perché la compatibilità con Hive da la possibilità di lanciare interrogazioni sui dati tramite codice in HiveQL nello stesso identico modo, con la fondamentale differenza che in Hive l'interrogazione viene tradotta in Job MapReduce, in SparkSQL tutto il procedimento (a cui può essere incluso anche altro codice) viene ottimizzato e convertito in procedure Spark, sfruttandone tutti i vantaggi. L'elemento fondamentale su cui si basa SparkSQL è il **DataFrame** il quale può essere visto come un'estensione dell'RDD (o come un RDD composto da oggetti riga), dal quale eredita tutte le caratteristiche principali, ma organizzato come una tabella relazionale, cioè descritta da colonne. L'insieme di operazioni fatte sul DataFrame vengono convertite in codice di esecuzione Spark tramite l'ottimizzatore relazionale **Catalyst** il quale traduce le varie interrogazioni SQL tramite una procedura logico-fisica simile a quella descritta per gli RDD al cui interno applica tecniche di ottimizzazione come per esempio il push-down dei

predicati, dove le operazione di filtraggio vengono eseguite come prima cosa in modo tale da scartare i dati irrilevanti per i calcoli successivi ottimizzando la procedura.

Spark Streaming

Nato dalla necessità delle aziende di attuare analisi sui dati in tempo reale Spark Streaming si pone come modulo di Spark per analizzare flussi di dati i quali possono essere presi sia da fonti statiche (come database) che da fonti le quali trasmettono dati in continuazione, anche ad elevata velocità. Spark Streaming rappresenta un flusso continuo di dati in ingresso con un flusso discretizzato chiamato **DStream** il quale è diviso in *batch* di grandezza fissa ognuno dei quali viene considerato come un RDD su cui possono venire applicate le operazioni. Generalmente le applicazioni Spark Streaming devono attendere una frazione di secondo per raccogliere ogni micro batch di eventi prima di inviare ogni batch per l'elaborazione. Al contrario, un'applicazione guidata dagli eventi elabora ogni evento immediatamente. La latenza di Spark Streaming è in genere inferiore a pochi secondi rendendole applicazioni "near real-time". D'altro canto, i vantaggi dell'approccio basato su micro batch consistono in un'elaborazione dati più efficiente mantenendo le proprietà di interoperabilità con gli altri moduli.

MLib - Machine Learning

Componente di Spark che implementa e mette a disposizione primitive per il **Machine Learning**, unendo l'efficacia del processare i dati in maniera distribuita su di una mole enorme di dati ai più importanti algoritmi di machine learning come classificazione, clustering, regressione ecc.. con la possibilità quindi di combinarli ai comandi di Spark Core ed agli altri moduli.

GraphX

GraphX è il modulo più recente di Spark e viene utilizzato per i grafi abilitando la computazione parallela e distribuita su di essi. Ad alto livello, estende

i classici RDD introducendo il **Resilient Distributed Property Graph** cioè un grafo direzionato che può possedere proprietà sia negli archi che nei nodi. Mette a disposizione un insieme di operatori per lavorare con la struttura grafo cosi come una variante ottimizzata dell'API di Pregel [37] con la quale si possono definire complesse operazioni iterative in maniera rapida ed efficiente sull'interno grafo. Con il passare dei mesi ha iniziato ad integrare anche diversi algoritmi come per esempio il PageRank (utilizzato per quantificare l'importanza relativa di un sito web a seconda dei vari collegamenti che esso ha con gli altri siti) ed il calcolo del cammino minimo (utilizzato per trovare i percorsi di minor durata all'interno del grafo) in modo tale da semplificare molti task di analisi.

2.3 GEOSpark

Per quanto riguarda Hadoop e la sua infrastruttura manca ancora un elemento che dia la possibilità di lavorare con i tipi di dati spaziali ed effettuare analisi su di essi tramite operatori geometrici anche complessi. Essendo questo un requisito fondamentale per il mining di traiettoria e per il map matching in generale, la ricerca di un possibile framework che abilitasse il sistema distribuito a fare ciò è stato un elemento importante della parte di ricerca della tesi. Sono stati analizzati diversi sistemi che potessero estendere Hadoop come SpatialHadoop[3] e HadoopGIS[4] ma entrambi estendono direttamente il framework MapReduce, ereditando quindi tutte le problematiche che il framework possiede, come spiegato ad inizio Paragrafo 2.2.6. Per risolvere questi problemi è stato sviluppato negli ultimi anni un framework open-source per caricare, processare e analizzare dati spaziali in larga scala su Apache Spark chiamato GEOSpark [27].

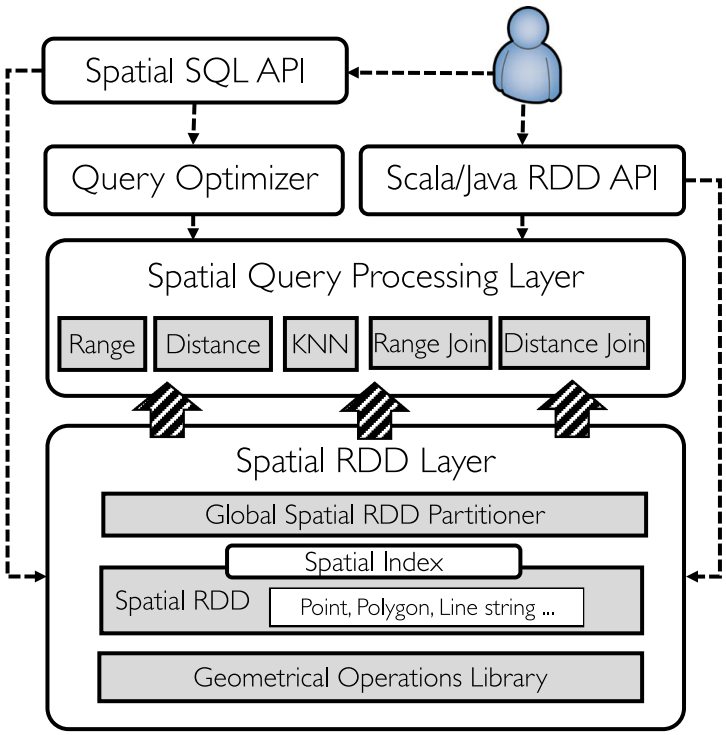


Figura 2.3: Struttura di GEOSpark [27]

2.3.1 Descrizione del sistema

La Figura 2.3 mostra una panoramica di come è strutturato GEOSpark. L'utente può interagire con il sistema utilizzando due differenti moduli: l'API *Spatial SQL* e l'API *Scala RDD*. Con l'API Scala RDD è possibile utilizzare specifiche funzioni create per l'analisi spaziale. L'utente può creare uno **SpatialRDD**, richiamare le varie librerie geometriche messe a disposizione e far partire analisi sull'RDD. Con l'API di Spatial SQL, invece, viene seguito lo standard "SQL/MM Part 3"[28] il quale specifica come standard i codici per tutte le operazioni spaziali sui dati applicabili in SQL. In particolare vengono supportate tre tipi di interfacce SQL:

1. **Costruttori** - danno la possibilità di costruire un RDD con caratteri-

[3]http://spatialhadoop.cs.umn.edu/
[4]https://esri.github.io/gis-tools-for-hadoop/

stiche spaziali dai più comuni tipi di file come i file di testo e i comma-separated values (csv) agli specifici file geometrici come i GEOJson (file JSON con indentazione aggiuntiva per le caratteristiche spaziali), i Well-Known Text(WKT). Viene data la possibilità anche di definire geometrie direttamente da codice specificando le forme geometriche (punto, linea, cerchio, poligono ecc..) e le caratteristiche.

2. **Funzioni Geometriche** - rappresentano le operazioni che si possono applicare sugli Spatial RDD. Sono compresi in questa sezione la distanza, il calcolo del raggio, la lunghezza della forma geometrica, l'area di superficie e così via.

3. **Predicati** - comprendono tutte quelle interrogazioni con vincoli spaziali che si possono effettuare sui dati i quali ritornano solo i dati che riescono a soddisfare tali predicati come per esempio il riuscire a contenere completamente o meno una geometria o l'intersezione tra figure geometriche.

Oltretutto tale API SQL utilizza un ottimizzatore per le interrogazioni il quale produce dei piani di esecuzioni perfezionati a seconda delle query spaziali basandosi sull'ottimizzatore Catalyst già presente in SparkSQL aggiungendovi tecniche come la scelta delle query a seconda del costo computazionale o il predicate pushdown spaziale dove trova i predicati che filtrano i dati e il porta all'inizio della computazione del piano per ridurre la quantità di dati da analizzare. Entrambi i moduli di interfacciamento con l'utente passano poi dallo **spatial query processing layer** il quale provvede ad implementazioni efficienti degli operatori spaziali più utilizzati per interrogare i dati come i filtri per range di distanza, la ricerca dei vicini in termini di posizione e i join tra elementi geometrici a seconda di diversi vincoli spaziali. Il layer che sta alla base della struttura è ovviamente il **Spatial Resilient Distributed Dataset layer** che estende Spark aggiungendo gli SpatialRDD, o SRDD, i quali sono programmati per partizionare in maniera efficiente i dati spaziali all'interno del cluster. La tecnica alla base di queste migliorie è il partizionamento automatico dei dati dell'SRDD a seconda della distribuzione spaziale

interna ad essi tramite una tecnica di suddivisione ad indice globale che può variare dalla divisione in griglie spaziali uniformi alle tecniche ad indicizzazione ad albero come per esempio il Quad-Tree e il KDB-Tree. È bene precisare che l'utilizzo di indici spaziali è troppo oneroso in termine di spazio (circa il 15% dello spazio occupato dai dati[28]) quando si hanno a che fare con i big data, di conseguenza GEOSpark utilizza una tecnica di campionamento dove costruisce l'indicizzazione su un piccolissimo sottoinsieme di dati andando poi ad applicare tale indice a tutti i dati e ripartizionando l'SRDD di conseguenza. L'indicizzazione completa è comunque attuabile localmente: se infatti c'è il bisogno di interrogare un SRDD molteplici volte un indicizzazione completa dei dati velocizza enormemente l'esecuzione delle interrogazioni spaziali andando a sopperire all'aumento della dimensione del dataset data dallo spazio occupato dall'indice. Per fare ciò GEOSpark genera una indicizzazione per ogni partizione dell'SRDD andando a ripartizionare successivamente gli elementi internamente ad ogni partizione risparmiando la gestione dell'indice globalmente attraverso le varie partizioni.

Capitolo 3

Hidden Markov Model

In questo capitolo viene analizzata nel dettaglio la tecnica sfruttata per effettuare il map-matching delle traiettorie nell'ambito di questa tesi cioè l'utilizzo del modello di Markov nascosto (Hidden Markov Model). Essendo un approccio considerato applicabile in svariati campi, verrà inizialmente descritto in modo generale per poi passare all'applicazione di tale modello al map-matching delle traiettorie.

3.1 Modelli Markoviani e Catene di Markov

Prima di introdurre il modello nascosto di Markov bisogna capire su cosa si basa, cioè i processi Markoviani e le catene di Markov. Innanzitutto un **Processo Markoviano** è processo di tipo stocastico cioè definito da un insieme di variabili aleatorie le quali cambiano nel tempo in maniera incerta. L'insieme di valori che possono assumere le variabili formano il cosiddetto **spazio degli stati** nel quale il processo si modifica ed evolve. Nel processo Markoviano la probabilità di transizione la quale determina il passaggio da uno stato di sistema ad un altro dipende solo dallo stato di sistema immediatamente precedente e non da come si è giunti in questo stato. Questa proprietà è fondamentale per questi tipi di processi ed è infatti definita **proprietà di Markov**. Una **Catena di Markov** invece, non è altro che un processo Markoviano il quale

spazio degli stati è discreto, quindi è un processo stocastico che assume valori in uno spazio discreto e che gode della proprietà di Markov.

3.2 Modello di Markov Nascosto

Un modello di Markov nascosto non è altro che una catena di Markov in cui gli stati sono sconosciuti e quindi non osservabili direttamente. A confronto dei processi Markoviani classici infatti ogni stato della sua catena genera un evento con una certa distribuzione di probabilità il quale dipende solamente dallo stato. Tale processo è anch'esso stocastico andando ad aggiungersi al modello classico del processo Markoviano, il quale è modellato solamente dalla probabilità di transitare da uno stato ad un altro.

3.2.1 Formalizzazione

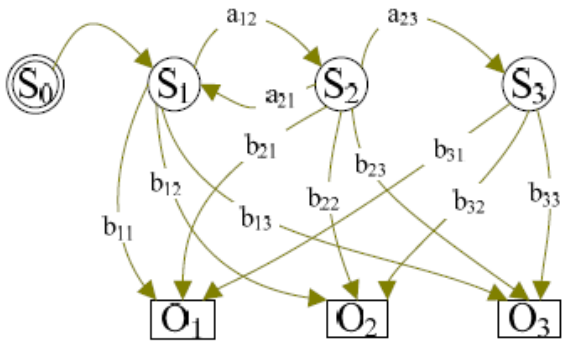


Figura 3.1: Struttura di HMM con i suoi parametri

Un qualsiasi modello di Markov nascosto è rappresentabile come si può notare dalla Figura 3.1 e comprende:

- Un insieme $S = \{S_1, S_2, S_3, ...S_K\}$ di stati (nascosti).
- Un insieme di osservazioni $O = \{O_1, O_2, O_3, ...O_N\}$ sull'insieme degli stati. Tali osservazioni sono definite come l'output fisico osservabile del sistema. Se le osservazioni sono in un quantitativo finito si parla di

HMM Discreto, in caso invece le osservazioni siano modellate come un valore continuo si parla di **HMM Continuo**.

- Una probabilità degli stati iniziali con la quale il modello viene inizializzato $\pi = P(0|S_i)$ con i = stati iniziali e con 0 che rappresenta lo stato iniziale del modello.

- La **probabilità di transizione tra stati** $A = \{a_{ij}\} \rightarrow a_{ij} = P(t = S_j|t-1 = S_i)$ che quantifica la probabilità di passare da un certo stato ad un altro dell'insieme S al passare del tempo.

- La **probabilità di emissione** delle osservazioni O per gli stati di S tale che $B = \{b_{ij}\} \rightarrow b_{ij} = P(O_i$ emesso al tempo t $|t = S_j)$ che invece quantifica la probabilità che tale osservazione si sia verificata per tale stato in quel preciso momento.

3.3 Tipologia di Problemi Applicabili al Modello

A seconda dell'obiettivo si possono definire tre problemi di base su cui lavorare con i modelli Markoviani a stati nascosti:

- **Valutazione** - data una sequenza di Osservazioni O e un modello M si calcola l'effetto di tali osservazioni sul modello con la probabilità $P(O|M)$. Utilizzano questa tecnica tutte le applicazioni che richiedono di determinare la probabilità di essere in uno specifico stato quando si sa la sequenza delle osservazioni. Questo tipo di problema può essere gestito in maniera efficace con il **Foward algorithm**[42] il quale calcola le combinazioni tra probabilità di emissione e transizione tra tutti i possibili set di stati del modello fino ad arrivare allo stato di cui si è interessati a sapere la probabilità per un certo numero di osservazioni specificato. Sfruttando l'indipendenza condizionale del modello di Markov il calcolo può essere ricorsivo e quindi risparmiare in termini di tempi di calcolo.

- **Decodifica** - data una sequenza di Osservazioni O e un modello M si calcola la sequenza ottimale di stati $S = \{S_1, ..., S_K\}$ che ha generato tale sequenza O. A confronto della Valutazione, qui si richiede il calcolo della probabilità congiunta dell'intera sequenza di stati nascosti che hanno generato una particolare sequenza di osservazioni. Per risolvere questo tipo di problema si utilizza l'algoritmo di Viterbi, utilizzato anche in ambito di tesi, il quale verrà spiegato nel dettaglio nel Paragrafo 3.4 a lui dedicato.

- **Addestramento** - dato un insieme di sequenze di Osservazioni $\{O_i\}$ si determina il miglior modello M tale che il modello per cui $P(\{O_i\}|M)$ è massimizzata. L'obiettivo è quindi quello di addestrare un modello in modo tale che restituisca i migliori insiemi di probabilità di trasmissione ed emissione dati un set specifico di osservazioni. L'algoritmo più utilizzato per risolvere l'addestramento del modello è il **Baum-Welch**[43] il quale, per ogni sequenza di osservazioni, calcola il modello e successivamente estrae delle variabili che rappresentano quanto la sequenza sotto esame contribuisce alle transizioni e alle emissioni nel modello. Con tali variabili va poi a ricalcolare i parametri del modello e continua cosi fino a quando non si raggiunge un desiderato livello di convergenza o un numero massimo di iterazioni.

3.3.1 Applicazioni e Ambiti di Studio

Tutte e tre le tipologie di problemi sono ampiamente sfruttate in svariati ambiti. In quello **finanziario** vengono utilizzati i problemi di valutazione per decidere se vendere o comprare determinati *beni* e i problemi di addestramento per creare sistemi atti al valutare il *rating* (cioè la capacità di una società di pagare o meno i propri debiti) delle aziende. Uno dei primi ambiti al quale sono stati applicati in maniera massiva vari algoritmi basati sull'HMM è quello del **riconoscimento vocale** nel quale viene utilizzato sia tramite problemi di decodifica per attuare l'analisi grammaticale automatica del testo sia tramite problemi di addestramento per effettuare sintesi vocale che per riconoscimento

della voce (ad esempio, Siri di Apple utilizza HMM per il suo riconoscimento vocale [1]). Negli ultimi anni sta aumentando l'utilizzo di HMM per quanto riguarda l'ambito **biomedico** dove viene sfruttato per la predizione dei geni e per studi sul DNA[44]. Come ultimo ma non meno importante sono gli svariati utilizzi in ambito di **sicurezza dei sistemi** dove per esempio viene utilizzato per addestrare modelli di decriptazione[45].

3.4 Algoritmo di Viterbi

Merita un paragrafo a parte questo algoritmo utilizzato per risolvere i problemi di Decodifica perché è l'algoritmo che è stato poi selezionato ed implementato per risolvere i modelli Markoviani nell'elaborato di tesi. L'algoritmo di Viterbi[46] è un algoritmo inventato dall'ingegnere Andrew Viterbi e viene generalmente utilizzato per trovare la migliore sequenza di stati (chiamata Viterbi Path) data una sequenza di eventi osservati in un processo Markoviano. E' un algoritmo molto generale e dunque è possibile adeguarlo alla descrizione di diversi fenomeni, ma la sua applicazione di maggiore successo è con i modelli Markoviani e gli HMM. Dati in input:

- Spazio delle osservazioni $\mathbf{O} = \{o_1, o_2, ..., o_N\}$.
- Spazio degli stati $\mathbf{S} = \{s_1, s_2, ..., s_K\}$.
- Array di probabilità iniziali $\mathbf{I} = \{\pi_1, \pi_2, ..., \pi_K\}$ tale che π_i sia la probabilità che $x_i = s_i$.
- Sequenza di osservazioni $\mathbf{Y} = (y_1, y_2, ..., y_T)$ tale che $y_t = i$ se l'osservazione al tempo t è o_i.
- Matrice di transizione $\mathbf{A}$ di grandezza $N \times N$ in modo tale che A_{ij} sia la probabilità di transizione da uno stato s_i allo stato s_j.
- Matrice di emissione $\mathbf{B}$ di grandezza $N \times M$ in modo tale che B_{ij} sia la probabilità di osservare o_j dallo stato s_i.

[1] https://www.makeuseof.com/tag/alexa-siri-work-voice-control-explained/

Algoritmo 1 Algoritmo di Viterbi

function VITERBI(S,Y,I,A,B)
FASE DI INIZIALIZZAZIONE
for each stato $i \in \{1, 2, ..., K\}$ **do**
$T_1[i, 1] \leftarrow \pi_i * B_{iy_1}$
$T_2[i, 1] \leftarrow 0$
end for
FASE DI CALCOLO DELLE PROBABILITÀ CONGIUNTE
for each osservazione $i = 2, 3, ..., T$ **do**
for each stato $j \in \{1, 2, ..., K\}$ **do**
$T_1[j, i] \leftarrow \max_k(T_1[k, i-1] * A_{kj} * B_{jy_i})$
$T_2[j, i] \leftarrow \arg\max_k(T_1[k, i-1] * A_{kj} * B_{jy_i})$
end for
end for
FASE DI ESTRAZIONE DEL PERCORSO A RITROSO
$z_T \leftarrow \arg\max_k(T_1[k, T])$
$x_T \leftarrow s_{z_T}$
for $i \leftarrow T, T-1, ..., 2$ **do**
$z_{i-1} \leftarrow T_2[z_i, i]$
$x_{i-1} \leftarrow s_{z_{i-1}}$
end for
return X
end function

L'algoritmo ritorna quindi la sequenza ottimale di stati nascosti $\mathbf{X} = (x_1, x_2, ..., x_N)$. Come si può evincere dall'algoritmo 1 vengono utilizzate due tabelle bidimensionali dove T_1 e T_2 come supporto all'algoritmo. Ogni elemento di $T_1[j, i]$ di T_1 mantiene la probabilità dell'insieme degli stati ottimale fino allo stato j all'osservazione i. $T_2[j, i]$ di T_2 invece serve a mantenere lo stato precedente a j che ha portato al percorso ottimale. Definite queste strutture di appoggio l'algoritmo funziona principalmente in tre passi:

1. **Passo iniziale** - viene sfruttata la probabilità iniziale π e l'osservazione

di ogni stato per inizializzare la matrice T_1 e T_2.

2. **Passo in avanti** - chiamato così perché viene cercata la sequenza di stati migliore per ogni possibile combinazione di stati che producono le osservazioni passate in input all'algoritmo mantenendo la probabilità che tale sequenza ha in T_1 e il puntatore all'ultimo stato della sequenza in T_2. Questa procedura viene effettuata al passare del tempo, ergo allo scorrere delle osservazioni.

3. **Backtracking** - in questa ultima fase viene ricercata la sequenza di stati che ha portato alla probabilità congiunta più alta partendo dalla fine della catena di Markov e selezionando la probabilità maggiore ed estraendo la sequenza di stati andando a ritroso a seconda dei puntatori agli stati salvati in T_2.

3.5 HMM Applicato al Map-Matching

Entrando nello specifico della segmentazione di traiettorie, un modello nascosto di Markov riesce a modellare perfettamente la situazione di associare una traiettoria ad un'insieme di segmenti stradali. Ogni elemento del Map-Matching è cosi trasposto in modello Markoviano:

- Lo stato del sistema consiste in un insieme di strade cioè il dataset dei segmenti stradali.

- Il set delle osservazioni è dato dalle misurazioni degli stati del sistema i quali non sono altro che i punti che compongono la traiettoria GPS.

- La probabilità di transizione tra stati dipende dalla connettività tra segmenti della strada all'interno della rete stradale.

- La probabilità di emissione invece indica la probabilità di quanto un punto GPS della traiettoria (osservazione) corrisponda ad una strada (stato).

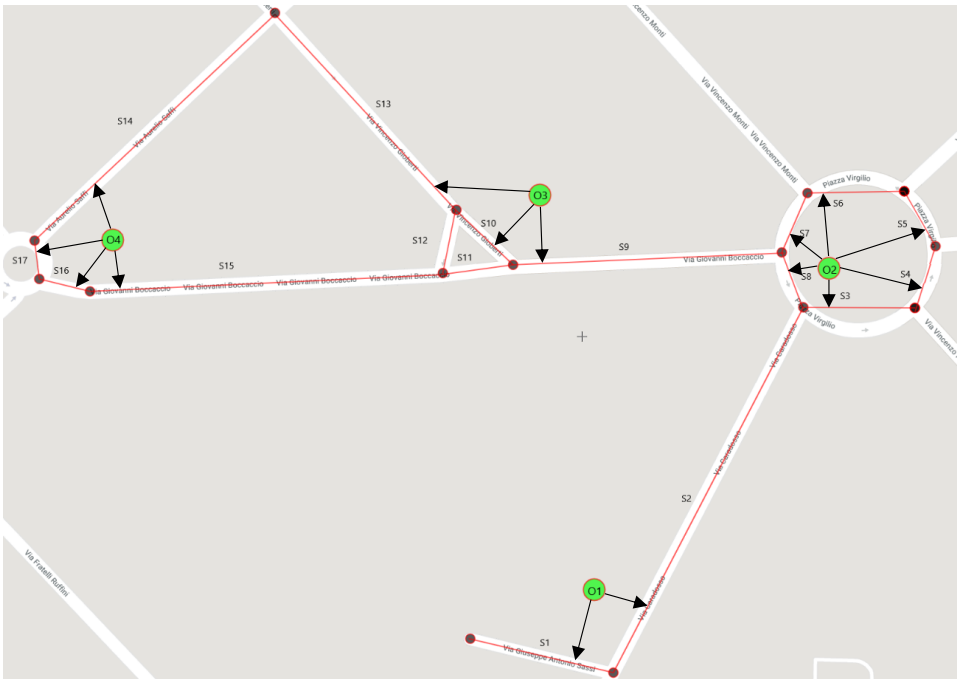


Figura 3.2: Esempio di traiettoria con quattro punti GPS (in verde) e relativi segmenti stradali

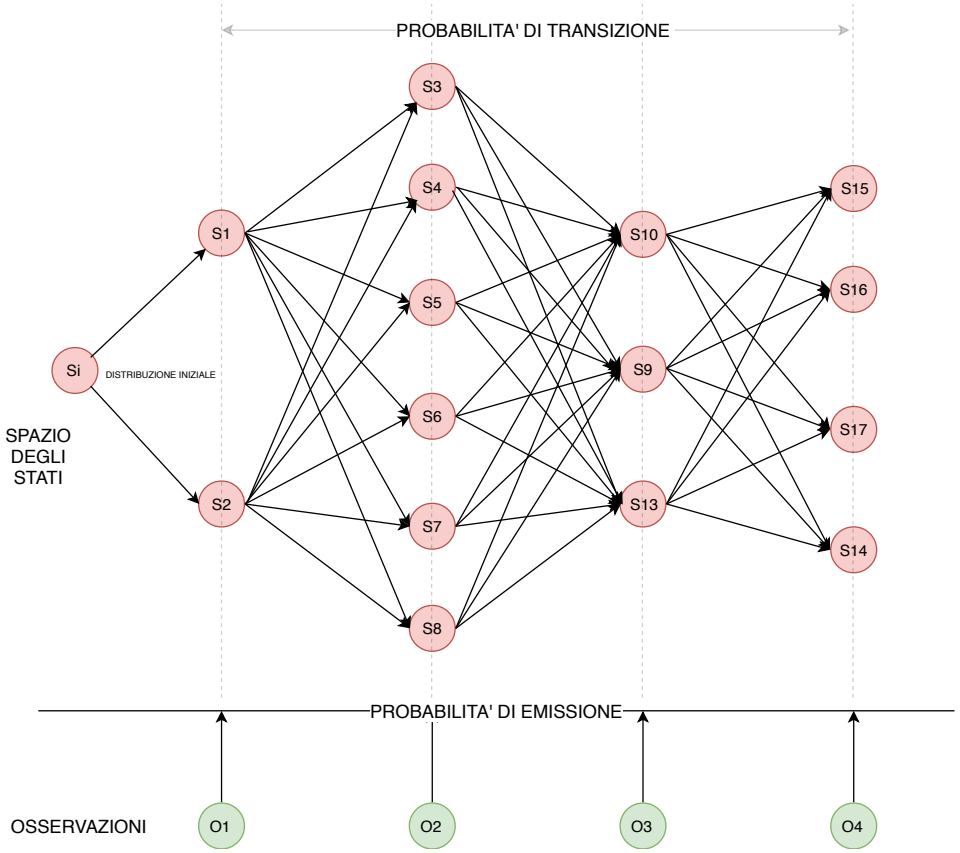


Figura 3.3: Conversione in catena di Markov dell'esempio in Figura 3.2

In parole più semplici si può considerare il modello come la rete stradale dove all'interno un oggetto si sposta di segmento in segmento all'avanzare del tempo e ogni segmento "emette" un valore a seconda di dove l'oggetto si sta spostando. Considerando quindi i segmenti realmente percorsi dalla traiettoria come gli stati "nascosti" del sistema il problema diventa decodificare il modello dati i punti GPS (cioè i valori emessi dai segmenti a un tempo specifico)), i segmenti stradali (cioè gli stati) e le varie probabilità di connessione tra segmenti e di vicinanza tra i punti GPS e i segmenti stessi. Per decodificare il modello può essere ovviamente utilizzato l'algoritmo di Viterbi il quale può venire abbondantemente velocizzato perché nel passo in avanti itera solamente tra gli stati che sono direttamente connessi con lo stato corrente, ergo, per ogni segmento che emette il punto GPS (osservazione) o_t considera solo i segmenti stradali che "emettono" il punto GPS precedente o_{t-1}.

Capitolo 4

Stato dell'Arte

In questo capitolo viene descritto lo stato dell'arte della metodologia di map matching di traiettorie. Inizialmente viene dettagliata una suddivisione degli algoritmi a seconda di specifiche caratteristiche che possono possedere, successivamente vengono dettagliate le metodologie presenti in letteratura ed analizzate mostrandone i vari pro e contro.

4.1 Tipologie di Algoritmi

Gli algoritmi esistenti in letteratura possono essere categorizzati a seconda di diverse prospettive che svariano dal punto di vista della tecnica utilizzata per calcolare lo score di similarità alla struttura dell'algoritmo. Tale score di similarità può essere chiamato anche *match*. Di seguito verranno analizzate le varie categorie sintetizzate in Tabella 4.1. È bene far notare che la suddivisione per metrica non è esclusiva infatti gli algoritmi più recenti combinano diverse metriche tra di loro.

4.1.1 Suddivisione per Metrica

Per metrica si intende il modo in cui viene calcolata la qualità del match che c'è tra la traiettoria e la rete stradale. Si suddividono gli algoritmi in questo

Categorizzazione	**Suddivisione**
Metriche di matching	• Geometriche • Topologiche • Probabilistiche • Statistiche
Livello di località dell'approccio	• Incrementale • Globale
Contesto di applicazione	• Offline • Online
Distribuzione del processo	• Centralizzato • Distribuito

Tabella 4.1: Tipologie di suddivisioni per algoritmi di map matching

modo a seconda che vengano utilizzate informazioni geometriche, topologiche, probabilistiche o statistiche.

- **Metriche basate sulla geometria** - quantificano la qualità del match basandosi sulla similarità di caratteristiche geometriche tra traiettoria e percorso, come la distanza, l'angolo tra le due curve formato tra la traiettoria e la mappa sottostante e così via. Sono tra le metriche maggiormente utilizzate. Sono adatte a traiettorie campionate ad alta frequenza, infatti, maggiore è la distanza tra i punti, minore è il significato del risultato di queste metriche [39]. Gli algoritmi che utilizzano metriche geometriche sono anche i primi nati, come descritto in [6] e [7]. Sono algoritmi molto veloci e "real-time". D'altro canto però non riescono a raggiungere un elevata accuratezza per le stesse ragioni. Fanno parte di questa categoria le tecniche "point-to-point","point-to-curve matching" e "curve-to-curve matching". I primi due approcci identificano rispettivamente il nodo e l'arco più vicino al punto della traiettoria sfruttando ricerche basate sulla distanza nella rete stradale. Il terzo approccio in-

vece è un evoluzione diretta dei due precedenti perché come primo passo va ad identificare i nodi associati ai punti della traiettoria come nel caso del point-to-point ma successivamente costruisce il percorso dato dagli archi tra i nodi e lo confronta direttamente con il percorso dato dalla connessione dei corrispondenti punti della traiettoria. Tale confronto si attua ad esempio sfruttando la distanza di Fréchet e tutte le sue varianti. La distanza di Fréchet infatti è una misura di similarità tra curve la quale sfrutta la posizione e l'ordinamento dei punti nelle curve. Anche tale algoritmo ha delle limitazioni tra cui il fatto che sfrutta al suo interno solamente un insieme di distanze con i punti della traiettoria, di conseguenza il risultato è molto influenzato dalle rilevazioni GPS "anomale". È comunque un metodo molto utilizzato anche oggi, sopratutto in caso di implementazione dell'algoritmo in ambiti ancora poco approfonditi, come appunto l'ambito distribuito [21, 22].

- **Metriche basate sulla topologia** - sfruttano le informazioni sulla topologia della strada includendo connettività, adiacenza, relazione di confini in comune ecc.. tra segmenti, curve o poligoni. Tali metriche considerano non solo la distanza tra i punti della traiettoria e le potenziali strade di match ma anche la connettività topologica della strada stressa, diventando quindi metriche maggiormente consigliate in caso di traiettorie a bassa frequenza che possono anche includere errori di campionamento a confronto di quelle geometriche [39]. Tutti gli algoritmi che sfruttano informazioni topologiche, come per esempio [8, 9, 10, 11] riescono ad eguagliare le performance in termini di tempi di calcolo degli algoritmi geometrici sopra descritti, ma possono funzionare in modo migliore in termini di robustezza e capacità di funzionamento in tempo reale. L'approccio in comune di tali algoritmi è quello di utilizzare le informazioni topologiche per ridurre drasticamente il numero di segmenti candidati per ogni punto della traiettoria, ed utilizzare un sistema a pesi per misurare la similarità tra la geometria di una parte di traiettoria e i segmenti candidati per cercare l'arco più adatto. Il sistema di calcolo

del peso può cambiare a seconda del quantitativo di elementi topologici che vengono presi in considerazione come velocità, distanza, direzione, numero di svolte e così via. Bisogna specificare che, andando ad aggiungere più vincoli diventa sempre più difficile per questi algoritmi riuscire ad aggiustare tali fattori nelle varie formule per mantenersi robusti verso tutti i casi possibili, sopratutto in presenta di incroci tra molteplici segmenti [39].

- **Metriche basate sulla probabilità** - utilizzano la probabilità che una traiettoria possa realmente passare attraverso un certo segmento di strada per misurare la qualità di match finale. A seconda della precisione di misurazione, l'attuale posizione di ogni punto GPS della traiettoria è ristretto a una area ellittica di "confidenza", e la probabilità che tale punto passi da una certa strada può essere calcolata basandosi sulla relazione tra il punto e le parti di strada all'interno dell'area di "confidenza". Sono metriche più complesse delle precedenti perché calcolano il match anche unendo in combinazione metriche geometriche e topologiche combinandole a tecniche probabilistiche [29, 30, 31, 32, 48].

- **Metriche basate sull'accuratezza** - tali metriche utilizzano la statistica per misurare quanto accurata una strada è associata a una traiettoria. Per esempio, data una strada, la metrica di accuratezza utilizza il ratio dei segmenti correttamente associati sul numero totale di segmenti nella traiettoria per valutare l'accuratezza e quindi la qualità del percorso trovato. Il problema principale con queste metriche è l'obbligatorietà della cosiddetta *Ground Truth*, cioè il percorso reale ed effettivo che andrebbe associato alla traiettoria. Essendo però nella maggior parte dei casi ignoto il reale percorso, questa tipologia di metrica viene poco utilizzata.

4.1.2 Suddivisione Metodo Incrementale/Globale

Un'altra metodologia con cui si possono descrivere gli algoritmi di map matching è la suddivisione in metodi Incrementali (o Locali) e Globali. Nel meto-

do **Incrementale** l'algoritmo procede ad eseguire l'accoppiamento segmento-strada per ogni punto della traiettoria presi singolarmente ed andando ad effettuare la scelta nel suo intorno, considerando come precedente il segmento associato al punto prima. È una metodologia molto utilizzata nelle applicazioni di navigazione stradale per la capacità di calcolo quasi istantanea, che però ha un grave contro, il quale consiste nel fatto di soffrire tremendamente agli *outlier* (cioè punti anomali causati da problemi di rilevazione spaziale). Se viene assegnata una strada sbagliata ad un punto tale errore avrà un peso anche per il punto successivo inficiando sulla selezione del punto ed espandendosi così anche per i punti successivi. Nel metodo **Globale**, invece, il percorso viene costruito considerando l'intera traiettoria. Tale proprietà li rende molto più adatti per processare dati GPS a larga scala. Nella maggior parte degli algoritmi globali, viene estratto un set di segmenti candidati e mantenuto per ogni punto dell'intera traiettoria. Il miglior percorso viene selezionato da tale set come combinazione dei segmenti al suo interno a seconda di una metrica o come combinazioni di alcune esse. È un metodo sicuramente più accurato di quello Incrementale ma d'altro canto la gestione dell'insieme di candidati è tutt'altro che triviale e considerato un vero e proprio "trade-off". Bisogna infatti tenere in considerazione la grandezza che può raggiungere tale insieme, il quale aumenta a seconda del numero di punti della traiettoria e dal quantitativo di candidati per punto. Formalmente si può rappresentare per la traiettoria T il set S dei segmenti candidati di grandezza $\sum_{n=1}^{s} T_n * numSegCandidati$. Per qualsiasi algoritmo Globale, più è elevata tale grandezza maggiori sono il numero dei calcoli attuati dall'algoritmo per capire l'appartenenza o meno di ogni segmento candidato al punto. Per ottimizzare l'algoritmo è buona norma riuscire a stimare il miglior numero di segmenti candidati per traiettoria, ma è un compito molto complesso, perché ogni traiettoria ha caratteristiche specifiche le quali dipendendo dalle sezioni di rete stradale che attraversa dove possono possedere densità di segmenti differenti al loro interno e quindi richiedono un quantitativo di candidati adatto in modo tale da non prenderne un numero eccessivo o, al contrario, un quantitativo insufficiente tale che il segmento reale attraversato non sia presente nell'insieme dei candidati.

4.1.3 Suddivisione Online/Offline

Questa tipologia di suddivisione degli algoritmi di map matching è differente perché riguarda il contesto in cui l'algoritmo viene applicato. Negli anni si sono moltiplicate le implementazioni di diversi algoritmi sia per la modalità Online, sia per la modalità Offline. In quelli **Online** i punti della traiettoria arrivano uno alla volta, o in gruppi di punti, di conseguenza il calcolo è effettuato a metodo Incrementale o a metodo Globale considerando solamente una finestra mobile di punti. L'obiettivo è quello di associare il punto appena arrivato in analisi alla strada sottostante basandosi solamente sul percorso precedente. Infatti questi tipi di algoritmi differiscono da quegli Offline anche perché non sono a conoscenza di dove il prossimo punto sarà posizionato. Questo insieme di caratteristiche porta ad un compromesso perché si ha una maggiore efficienza in termini di performance essendo il calcolo basato su un solo punto alla volta in favore dell'accuratezza la quale può diminuire per i problemi spiegati sopra. Sono però le uniche soluzioni applicabili a contesti in tempo reale come per esempio tutte quelle applicazioni che utilizzano notizie sul traffico sempre aggiornato. D'altro canto, quelli **Offline** non possiedono questi vincoli e possono considerare tutti i punti delle intere traiettorie sfruttando quasi esclusivamente i metodi Globali. Di conseguenza, a confronto degli algoritmi Online, tollerano peggiori performance a favore di una maggiore accuratezza nel risultato [39].

4.1.4 Suddivisione Centralizzato/Distribuito

Ultima prospettiva definita in ordine di tempo del modo di categorizzare gli algoritmi di map matching è stata definita da Douglas Alves Peixoto et al. in [42] e vede gli algoritmi suddivisi a seconda del modo con cui essi sono pensati e implementati. Possono essere infatti considerati **Centralizzati** se sono pensati per funzionare su di una singola macchina. Sono quindi presenti per la maggior parte in letteratura perché comprendono tutti quegli algoritmi pensati quando ancora il fenomeno dei big data non era esploso e sfruttato. Puntano quindi sopratutto sull'accuratezza del risultato a discapito delle per-

formance. Al contrario, gli algoritmi di tipo **Distribuito** sono pensati per funzionare su un insieme di macchine dove tali macchine comunicano tra di loro per cooperare e raggiungere un obiettivo comune. Di conseguenza, cambia completamente la metodologia con cui sviluppare l'algoritmo il quale deve sfruttare anche la suddivisione del lavoro tra le macchine in modo da ottenere migliori performance.

4.2 Algoritmi Analizzati

Dopo aver analizzato le varie caratteristiche e tipologie che possono avere i vari algoritmi di map matching, si sono ricercate in letteratura i vari studi e le varie metodologie applicate. La Tabella 4.2 mostra tutte le pubblicazioni analizzate nel dettaglio, il quale studio ha portato poi alla selezione dell'algoritmo sviluppato durante il periodo di svolgimento della tesi. Come si può notare dalla tabella gli algoritmi sono stati ricercati inizialmente considerando solamente l'argomento del map matching, quindi analizzando classici algoritmi **Centralizzati**, passando successivamente all'ambito dell'applicare un algoritmo di map matching sui big data, passando quindi ad algoritmi **Distribuiti**, ed infine concludendo con gli algoritmi facenti uso del Hidden Markov Model.

4.2.1 Algoritmi Centralizzati

ST-Matching

Uno degli algoritmi più famosi nato negli ultimi anni è sicuramente ST-Matching, il quale è si un algoritmo **topologico** e **globale**, ma sfrutta anche elementi **geometrici** e analitici delle traiettorie e li combina assieme [12]. Si basa su una analisi spazio-temporale e punta a creare una matrice, o grafo, di probabili percorsi da dove estrarre infine quello con il più alto score. È infatti anche chiamato **Graph-based Map-Matching**. L'ST-Matching può essere diviso in due parti ben distinte:

Nome	**Metriche**	**Localizzazione**	**Online**	**Distribuito**
ST-Matching[12]	Geometriche Topologiche	Globale	-	-
Segment-based Probabilistic [17]	Probabilistiche	Globale	-	-
Recursive LCS-based [40]	Geometriche	Globale	-	-
HOM [21]	Geometriche	Globale	-	✓
tMM+Leapfrog [23]	Geometriche	Globale	-	✓
HBase+Geohash [24]	Geometriche	Iterativo	✓	✓
DMM [25]	Geometriche Topologiche	Globale	-	✓
Serial Real-time [26]	Topologiche	Globale	-	✓
Orig. HMM[29]	Probabilistiche	Globale	-	-
Simp. HMM[30]	Probabilistiche	Globale	-	-
Online HMM + RCM [31]	Probabilistiche	Iterativo	✓	-
"Fast" HMM [32]	Probabilistiche	Globale	-	-
Green Self-Adaptive[41]	Probabilistiche	Iterativo	✓	-

Tabella 4.2: Metodologie analizzate con le relative proprietà di map matching

1. **Filtraggio dei candidati** - inizialmente si ottiene il set di segmenti stradali candidati interni ad un certo raggio di lunghezza parametrica r per ogni punto della traiettoria. Estratti tutti i segmenti candidati il problema diventa sceglie la miglior combinazioni di essi in modo che tale combinazione sia l'associazione corretta della traiettoria.

2. **Analisi Spazio-Temporale** - la funzione di analisi spaziale misura la

similarità del segmento tra due punti della traiettoria con il percorso più corto tra i due corrispondenti segmenti candidati. Tale similarità è combinazione di una probabilità punto della traiettoria-segmento la quale è modellata con una gaussiana che include la distanza, ed una probabilità tra due segmenti consecutivi che include in essa le informazioni sulla connessione dei segmenti stessi. Tale analisi può bastare nella maggior parte dei casi comuni, ma nel caso per esempio una traiettoria si trovi in mezzo a due strade dove la prima è una autostrada e la seconda un strada di servizio vicino ad essa la sola analisi spaziale non può bastare. Viene quindi aggiunta un analisi temporale la quale considera la velocità della traiettoria e la compara con il limite di velocità dei segmenti candidati tramite la distanza coseno, che rappresenta una tecnica euristica per la misurazione della similitudine tra due vettori effettuata calcolando il coseno tra di loro. Il risultato della distanza coseno è un valore di similitudine compreso tra -1 e 1, dove -1 indica una corrispondenza esatta ma opposta, mentre 1 indica che i due vettori sono uguali. Combinando insieme le due analisi si ottiene la funzione per il calcolo del punteggio dato il percorso candidato.

Il punto a maggior sfavore di questo algoritmo è sicuramente il tempo di calcolo, perché richiede di sapere il punteggio di similarità di tutti i percorsi possibili dal punto di inizio al punto di fine e tale tempo, oltretutto, aumenta esponenzialmente a seconda del raggio nel quale si cercano i candidati che moltiplicano le possibilità dei percorsi. Sono state sviluppate infatti alcune migliorie a tale algoritmo tra cui GridST [13] che punta ad ottimizzare la parte del filtraggio dei candidati andando a modellare dinamicamente il raggio di ricerca a seconda dell'intorno del punto della traiettoria, diminuendo quindi il numero totale di combinazioni da calcolare. Per fare ciò suddivide l'intera mappa stradale in griglie e calcola anticipatamente la densità di ognuna di esse in modo da velocizzare il procedimento di settaggio del raggio. Questa tecnica a griglie verrà ampiamente utilizzata anche dagli algoritmi distribuiti. Un altra estensione studiata è proposta da Yuan in [14] la quale invece va ad ottimizzare

la seconda fase, inserendo un processo di votazione tra i segmenti candidati. Dati i valori delle analisi spazio-temporali del ST-Matching viene costruita una matrice dove ogni termine corrisponde alla possibilità di un segmento candidato di essere il corretto match. Viene moltiplicata ad essa per ogni valore al suo interno con la diagonale di un'altra matrice calcolata per ogni segmento candidato che rappresenta il "peso" o influenza che tale segmento ha. Data la matrice in output l'algoritmo calcola per ogni segmento candidato quante volte esso compare nel percorso ottimale (è qui che avviene la votazione) e fa ciò per tutti i segmenti candidati andando ad estrapolare il percorso dato dai segmenti più votati. L'ultimo miglioramento in ordine cronologico proposto in letteratura è l'IF-Matching[16] il quale sfrutta una maggiore quantità di dettagli per quanto riguarda il calcolo dell'analisi temporale sfruttando due componenti aggiuntivi. Il primo è chiamato **History Speed Mining** che consiste nell'utilizzo di dati storici per costruire un modello per la velocità a seconda dell'orario del giorno. Il secondo è chiamato **Surrounding Speed Estimation** dove viene preso in considerazione anche il verso della traiettoria e le condizioni reali dei segmenti della strada in esame. Di conseguenza con queste aggiunte il metodo risulta più robusto di ST-Matching, andando però a inficiare sulle performance.

Segment-based Probabilistic Map-Matching

Questa metodologia **globale**, introdotta da Yin et al. in [17] sfrutta metriche **probabilistiche** unendo ai vincoli spazio temporali più comuni come distanze tra segmenti e velocità della traiettoria anche fattori specifici della rete stradale come ad esempio la tipologia di svolta tra segmenti in modo da modellare al meglio il valore della scelta di una strada dal punto di vista dell'utente. È considerato un modello a suddivisione di segmenti perché inizialmente divide una traiettoria in punti GPS chiave presenti in essa utilizzando l'algoritmo di Douglas-Peucker il quale, data una curva composta da segmenti di linee trova la curva maggiormente simile ad essa ma con un numero inferiori di punti, mantenendo un grado di similarità tra curve elevato. La curva sem-

plificata consisterà quindi in un sottoinsieme dei punti che definivano la curva originaria. Data la traiettoria semplificata, vengono considerate solo le possibili coppie di strade candidate (e i relativi percorsi tra di esse) di questi punti chiave diminuendo cosi il costo computazionale totale. Un altra caratteristica di questo approccio è la generazione per ogni coppia di segmenti candidati dei punti chiave di un grafo, chiamato *Action Graph*, il quale possiede come nodi i segmenti stradali compresi tra la coppia di segmenti candidati e come archi le "azioni" tra ognuno di essi. Tale azione è data dall'estrazione delle caratteristiche spaziali date dalle distanze tra punti e segmenti unite alle caratteristiche della strada viste dal punto di vista dell'utente come la formazione del percorso, modellato come lunghezza, numero di transizioni tra segmenti e tipologia di svolte presenti. Da tale grafo si estrae il percorso che porta ad un costo totale minore. L'algoritmo sfrutta il costo per migliorare il calcolo della probabilità di transitare dai due segmenti candidati. L'algoritmo infatti semplifica inizialmente la traiettoria tramite Douglas-Peucker, calcola i segmenti stradali candidati in un intorno di essi e le relative probabilità di appartenenza tra il segmento e il punto, modellato tramite una Gaussiana in modo tale da gestire al meglio il possibile rumore del GPS. Infine seleziona la combinazione di segmenti candidati per ogni punto chiave GPS tale che sia massimizzata la probabilità congiunta data dalle probabilità punto-segmento e le probabilità segmento-segmento, le quali utilizzano il valore del costo minimo ottenuto dal grafo delle azioni tramite la formula $\exp^{-costo}$ che quindi è maggiore al diminuire del costo.

Recursive Longest Common Subsequence

Algoritmo proposto da Zhu et al. [40] il quale sfrutta una metodologia **globale** con metriche puramente **geometriche** che punta a separare la traiettoria in segmenti in modo tale da cercare il migliore percorso da associarsi. Tale assunzione, chiamata "small segment" mira a suddividere in maniera ricorsiva una traiettoria fino a quando tutte le sue sotto parti possiedono una similarità elevata con la rete stradale. Il concetto di similarità tra traiettoria e strada

è definito come **Longest Common Subsequence**, o LCS. Tale funzione è ricorsiva e viene quindi applicata su tutti i segmenti stradali che comprendono il path in esame con tutti i punti della traiettoria. La base di questa funzione ricorsiva è un calcolo di similarità che si basa sulla distanza tra il punto e la strada: se tale valore supera una certa soglia, allora la similarità è zero, in caso contrario è un valore tra zero e uno basato su tale distanza. Il procedimento dell'algoritmo è il seguente:

- **Inizializzazione** - data la traiettoria e la rete stradale si calcola il percorso più corto basandosi sui punti di inizio e di fine della traiettoria e si calcola lo score di similarità. Se tale valore è più alto della soglia (viene utilizzato 0.95) allora il percorso è il migliore e l'algoritmo si ferma. Altrimenti il segmento viene suddiviso e si passa allo step successivo.

- **Processo ricorsivo** - Il processo ricorsivo continua a separare le traiettoria in segmenti fino a quando il calcolo della similarità non raggiunge la soglia prefissata. Per ogni segmento viene applicato lo stesso procedimento presente in Inizializzazione. Per quanto riguarda la segmentazione invece, si segmenta la traiettoria solamente nei punti considerati di *cutting* cioè quei punti della traiettoria che hanno maggiore distanza dal percorso a cui erano stati associati.

- **Criterio di Stop** - Se per due ricorsioni consecutive lo schema dei segmenti traiettoria non cambia, quindi non è più avvenuta segmentazione a causa della similarità sopra soglia, il matching è avvenuto e l'algoritmo finisce.

Come viene dimostrato dallo studio associato all'implementazione è un metodo il quale tempo di computazione dipende dalla soglia per il calcolo di similarità: aumentandola migliora la precisione ma diminuiscono le performance.

4.2.2 Algoritmi Distribuiti

Oltre agli algoritmi appena descritti si sono ricercate anche possibili metodologie per provare a distribuire tali algoritmi, punto focale della tesi. È bene

puntualizzare che qualsiasi tecnica analizzata applicata nel distribuito è implementata sulla base del framework Hadoop, spiegato nel dettaglio nel Paragrafo 2.2. Come si potrà notare, in letteratura non sono ancora presenti algoritmi implementati appositamente con l'obiettivo di fare map matching in ambito distribuito e tutte le tecniche analizzate o sfruttano algoritmi poco robusti, come quelli specificati nel Paragrafo 4.1.1, i quali vengono sfruttati tramite paradigmi come MapReduce e quindi non completamente adatti ai cluster recenti, o poco distribuiti e non applicabili al mio caso di studio. Questi fattori hanno infatti portato alla decisione di provare a distribuire il procedimento di un algoritmo che in letteratura non è presente cioè l'Hidden Markov Model.

Algoritmi Utilizzanti MapReduce

Presenti per la maggioranza in letteratura, essendo il primo metodo di programmazione distribuita messa a disposizione per piattaforme Hadoop, sfruttano tutti algoritmi **geometrici** e **topologici** distribuendoli con tale paradigma. In letteratura, Jian Huang et al. in [21] e Jinhui Qie et al. in [22] utilizzano due Job MapReduce per implementare un'algoritmo geometrico basato sulla calcolo della distanza di Fréchet. La parallelizzazione è ottenuta tramite due Job concatenati ed ottimizzata suddividendo la rete stradale in griglie geometriche di ugual dimensione. Grazie a questa suddivisione durante il calcolo della distanza punto-segmento si risparmiano calcoli considerando solamente i segmenti all'interno della griglia su cui è il punto. Il primo Job consiste quindi dato ogni punto delle traiettorie nell'associarli con una griglia dalla quale si estrae il vicinato e le relative distanze con il punto. Nel secondo Job si raggruppano infine tutti i punti per ogni traiettoria e le relative strade candidate andando a selezionare l'insieme di segmenti combinati tali che la sommatoria delle distanze sia la minore. Da tale metodologia è stata proposta anche una versione più robusta ed efficiente, introdotta dallo stesso Jian Huang et al. in [23]. Consiste in due processi separati: il primo è un filtraggio iterativo basato su tutti quei fattori (distanza, connettività stradale, informazioni di svolta ecc..) che in [21, 22] venivano applicati all'unisono generando il valore del peso

di ogni segmento candidato. In questo algoritmo, tali fattori vengono precedentemente ordinati per efficacia la quale è basata sul rapporto tra il numero di candidati che tale fattore può escludere e il costo in termini di tempo per calcolare tale fattore. Successivamente, i segmenti candidati vengono filtrati per tali fattori basandosi sull'ordine e fermando il filtraggio solamente quando è rimasto un segmento candidato. Il secondo miglioramento è l'aggiunta di un metodo "Leapfrog" il quale sfrutta una tecnica con la quale vengono presi in considerazione solamente i punti posizionati sulle intersezioni della mappa, considerati quindi realmente discriminante per il calcolo del percorso. Grazie a ciò viene nettamente ridotto il numero di punti per traiettoria, mantenendo la stessa accuratezza di [21, 22], ma raddoppiando le performance in caso di molte traiettorie campionate ad alta frequenza, con quindi tanti punti vicini in pochi metri.

Algoritmi Utilizzanti HBase

Ideato da Wonhee Cho et al. in [26] sfrutta sempre come base il paradigma MapReduce ma lo utilizza solo per raggruppare gli elementi utili ad ogni punto della traiettoria (che trova in fase di Map) in modo tale da applicare l'algoritmo interamente nella fase di Reduce. Si differenzia dal fatto che sfrutta HBase, il database NoSQL distribuito della piattaforma Hadoop, al quale unisce come indicizzazione della mappa stradale *Geohash* (un codificatore che trasforma coordinate geografiche in stringhe di lettere e numeri), con il quale suddivide in maniera iterativa la mappa in rettangoli di dimensione variabile a seconda della densità al loro interno. Con Geohash oltretutto si riesce a mantenere informazioni per quanto riguarda la vicinanza tra griglie e di conseguenza, unendolo con la ricerca indicizzata di HBase risulta velocissimo trovare la cella più adatta ad ogni punto della traiettoria per l'estrazione dei segmenti candidati. Essendo questa solamente la base del map matching, cioè la ricerca dei segmenti candidati, per la fase di reduce implementa una tecnica "point-to-curve" unita alla concatenazione topologica data dalla similitudine tra codici Geohash dei segmenti. Per migliorare la robustezza dell'approccio aggiunge

un doppio filtraggio per i segmenti candidati: il primo è basato sulla velocità e si basa sul semplice confronto tra la velocità di movimento della traiettoria e la velocità massima consentita nel segmento candidato, il secondo è invece basato sulla direzione azimuth delle coordinate GPS sui segmenti candidati selezionando così in caso di diverse corsie adiacenti solamente quelle corrette seguendo il senso di marcia della traiettoria.

Algoritmi Utilizzanti Spark

Come dettagliato nel Paragrafo 2.2.6 Spark può essere visto come un'evoluzione in tutto e per tutto di MapReduce, il quale si basa su esso e sfrutta al meglio le risorse dei cluster moderni. Hongyu Wang et al. in [24] è l'unico che parallelizza anche l'algoritmo e quindi l'intera procedura. Sfrutta una metrica **geometrica**, poco robusta anche se molto rapida nell'esecuzione utilizzando una tecnica in tempo reale sfruttando **Spark Streaming**. Non essendo Spark realizzato per lavorare realmente in tempo reale, sfrutta l'utilizzo di una finestra temporale dove va ad applicare delle trasformazioni sui dati ogni volta che arrivano all'interno della finestra, aggiornando quelli precedenti di conseguenza. Il procedimento per ogni dato incluso nella finestra è sempre simile a quelli precedenti: calcola il vicinato basandosi su una indicizzazione a griglia della mappa stradale, raggruppa quindi le strade candidate calcolandone il "peso" che ognuna di esse ha per il punto basandosi sulla vicinanza punto-segmento e sempre internamente ai dati della finestra raggruppa i risultati per traiettoria, tale risultato viene poi collegato con i segmenti uscenti dei dati della stessa traiettoria per la finestra precedente aggiungendo ai pesi anche le informazioni topologiche riguardanti il collegamento tra di essi, mantenendo le sommatorie dei pesi cumulativi per i diversi percorsi possibili in memoria. Il risultato è quindi aggiornato e restituito in output ad ogni elaborazione di una finestra. Tale algoritmo ha però diverse problematiche come la bassa robustezza al rumore del segnale GPS dato dall'utilizzo di metriche solamente geometriche ed al vincolo di utilizzarlo come algoritmo real-time. Oltretutto la troppa ripetizione di calcoli considerati pesanti come la ricerca del vicinato, lo rende

molto oneroso in termini di performance. D'altro canto, Almeida et al. in [25] sfrutta Spark per implementare un algoritmo molto simile ad ST-Matching spiegato in 4.2.1. Tale algoritmo però non viene realmente distribuito come in quello precedente di [24] sull'intero insieme di traiettorie ma viene sfruttato per calcolare in precedenza tutti i percorsi minimi tra ogni coppia di vertici della mappa e successivamente durante il calcolo dell'algoritmo per trovare il percorso migliore dato il grafo dei candidati sfruttando **Spark GraphX**, una libreria che abilita il calcolo distribuito di grafi. Può essere visto come una estensione distribuita di ST-Matching perché ne sfrutta le metriche (escludendo quelle temporali) e la tecnica di utilizzo del grafo dei segmenti candidati. Il problema principale sta nel fatto che viene sfruttata una libreria distribuita per il calcolo di percorsi in grafi che presi per singola traiettoria sono di dimensione contenuta. Oltretutto questa procedura viene ripetuta per tutte le traiettorie dove in un caso reale di utilizzo la creazione e l'esecuzione di cosi tanti piccoli grafi non farebbe altro che intaccare pesantemente le performance del programma.

4.2.3 Algoritmi HMM per il Map Matching

Partendo dal presupposto che non esistono implementazioni distribuite di algoritmi di map matching facenti uso di modelli Markoviani a stati nascosti, è stata comunque effettuata una ricerca approfondita in letteratura per cercare implementazioni non distribuite, per capire se fosse possibile distribuirne il processo. In tutte le implementazioni [29] [30] [31] [32] viene effettuata una ricerca dei segmenti candidati per ogni punto della traiettoria basandosi su un attorno del punto in metri. Questo è già un punto a sfavore delle implementazioni perché non considerano la densità di segmenti nelle città che può far esplodere il numero di candidati e far aumentare inutilmente il tempo di calcolo del modello Markoviano in maniera esponenziale. Entrando nel dettaglio delle implementazioni sia Newson et al. in [29] che Jagadeesh et al. in [31] gestiscono la probabilità di emissione partendo dall'intuizione che più un segmento è lontano dal punto meno probabilità ha che sia stato lui ad "emettere"

l'osservazione unendola al fatto che il modello del rumore del GPS può essere rappresentato con una Gaussiana a media nulla. In questo modo si riescono ad attutire errori di misurazione del GPS, a sfavore però di una normalizzazione della probabilità entro un certo quantitativo di metri. Per quanto riguarda invece la probabilità di transizione tra segmenti, Newson et al. in [29] una probabilità esponenziale parametrizzata data dall'osservazione che la differenza tra la distanza di due osservazioni susseguenti e le rispettive distanze dei loro segmenti candidati è molto simile. Jagadeesh et al. in [31] invece modifica tale osservazione aggiungendovi anche il fattore dell'implausibilità temporale con il tempo di percorrenza del tratto in secondi il quale assegna bassa probabilità ai percorsi i quali non possono essere attraversati nel tempo trascorso tra le due osservazioni, se non ad una velocità fuori dal comune. Oltretutto questo algoritmo attua un map matching **online** a confronto degli altri analizzati, ed utilizzando l'algoritmo di Viterbi che richiede tutta la serie di osservazioni per calcolare il percorso corretto, cosa che in questo caso non si possiede. Per utilizzare Viterbi in casi Online la teoria che sta alla sua base è la seguente: dato in input una nuova osservazione se tutti i suoi stati candidati nella fase di *backtracking* convergono in un'unico stato, quello stato è considerato **convergente** e il percorso da esso all'inizio della catena markoviana può essere ritornato come output perché non verrà mai più alterato da osservazioni successive. Di conseguenza i path ottimali trovati da Viterbi Online possono essere incompleti. Oltretutto, ci possono essere casi in cui il punto convergente non venga trovato, in questi casi si aggiunge una nuova osservazione e si ripete il procedimento, andando però a ricalcolare le varie probabilità congiunte del modello Markoviano, cosa che aggrava non poco sulle performance dell'algoritmo. Oltre a ciò, aggiunge un passaggio dopo aver ricevuto il percorso ottimo da Viterbi Online il quale consiste nel generare una serie di percorsi alternativi basati sul tempo di percorrenza minimo e per ognuno di essi calcola una probabilità di percorrenza data da una combinazione di fattori stradali come il numero di semafori, le classi delle strade che compongono il percorso, il quantitativo di cambi di tali classi e il tempo di percorrenza. Come si può notare è un metodo molto complesso che richiede una mole di calcolo aggiuntivo

importante, aumentando la robustezza del map matching in molti casi particolari ma allo stesso tempo aumentando esponenzialmente il tempo di calcolo. Oltretutto per la probabilità di percorrenza richiede molte informazioni che non sono sempre presenti in tutti i segmenti dei dataset che si possono trovare in rete. Un'altro approccio **onlne** è proposto da An et al. in [41] il quale comprende in maniera identica l'implementazione di HMM in ambito online di [31] aggiungendo all'intero processo due elementi: una finestra di controllo di punti a scorrimento la quale continua a crescere di grandezza fino a quando non viene trovato un punto convergente e un metodo di campionamento dei punti adattivo il quale va a campionare la posizione basandosi sulla velocità del veicolo, in modo tale da escludere dalla computazione punti non significativi in merito al risultato di matching finale. Infine, An Luo, in [32] implementa una versione velocizzata rispetto alle altre implementazioni, ma allo stesso tempo molto semplificata e poco generalizzabile. Per velocizzare il procedimento di matching calcola in precedenza all'algoritmo di Viterbi interamente la matrice di probabilità di transizione tra segmenti. Per fare ciò stima la media globale delle lunghezze dei segmenti della intera rete stradale e la media globale delle distanze tra punti GPS per stimare la probabilità di transizione sullo stesso segmento, su segmenti adiacenti e su segmenti separati da un altro segmento. Nel caso il numero di segmenti di separazione tra i due segmenti supera due, allora la probabilità è direttamente zero. Sfortunatamente, questa distribuzione di probabilità non funziona bene in caso di segmenti ad elevata varianza o di traiettorie ad elevata varianza in termini di periodi di campionamento dei punti. Oltretutto più è definita la rete stradale, meno adatto è tale approccio, perché assegna zero a tutte le probabilità di transizione ai punti che possono anche essere molto vicini, ma a distanza maggiore di tre segmenti.

Capitolo 5

Map Matching Distribuito

Dopo aver descritto nel dettaglio i vari elementi aventi a che fare con l'obiettivo della tesi, in questo capitolo è presente la spiegazione dettagliata della metodologia con la quale si sono combinati assieme tali elementi nell'ambito teorico e pratico in modo tale da raggiungere lo scopo di applicare una metodologia di map matching in maniera efficiente ed efficace su un grande quantitativo di dati. Viene inizialmente definito l'algoritmo della procedura e successivamente dettagliato. Il capitolo si chiude con la panoramica delle strutture dati utilizzate e i vari filtraggi applicati ad esse.

5.1 Formalizzazione dell'Algoritmo

Come già accennato nei capitoli precedenti è stato utilizzato una tecnica facente uso dell'Hidden Markov Model per riuscire ad effettuare un map matching preciso e sopratutto robusto ai possibili rumori GPS ed ai differenti periodi di campionamento all'interno della traiettoria. Per formalizzare l'algoritmo utilizzato è necessario definire nel dettaglio anche il concetto di traiettoria GPS e rete stradale. Partendo dal presupposto di avere una **traiettoria GPS** T, cioè una sequenza temporale di punti $\{p_1, p_2, ..., p_t\}$ generata da un oggetto in movimento dove ogni suo punto p_t è una posizione GPS composta da $(latitudine, longitudine)$ al tempo t. L'insieme delle traiettorie GPS viene denotato da $\mathcal{T}$. Si vuole mappare tale insieme su di una **rete stradale**

$G(V, S)$ dove S è il set dei segmenti stradali i quali sono connessi tra di loro dal set di punti terminali V. Una strada è una sequenza di segmenti stradali consecutivi. Per sfruttare il modello Markoviano a stati nascosti è necessario innanzitutto definire la metodologia per calcolare le due probabilità che definiscono il comportamento del modello, come specificato nel Paragrafo 3.2.1 inerente ai modelli di Markov:

- **Probabilità di Emissione** - Dato il set di segmenti stradali S e la posizione dell'oggetto p_t al tempo t, la probabilità di emissione $P(p_t|X_t = i)$, dove con X_t si denota l'indice del segmento stradale dove l'oggetto può essere al tempo t, è la probabilità di uno stato i di generare l'osservazione p_i:

$$P(p_t|X_t = i) = \frac{d_H(p_t, s_i)^{-1}}{\sum_{s_j \in N(p_t,S,\alpha,\tau)} d_H(p_t, prj(p_t, s_j))^{-1}} \tag{5.1}$$

  $N(p_t, S, \alpha, \tau)$ è l'operazione che ritorna il *vicinato* di un punto p_t cioè l'insieme α di segmenti più vicini in S la quale distanza a p_t è minore della soglia spaziale τ. d_h è la "great-circle distance" (cioè la distanza minima che c'è tra due punti sulla superficie di una sfera) tra il punto p_t e la sua proiezione prj al segmento stradale s_t. Tale formulazione presa da [32] è stata resa più robusta aggiungendo un vincolo sulla distanza d_H:

$$d_H = \begin{cases} 5 & \text{se } d_H(p_t, s_i) \leq 5 \\ d_H & \text{se } d_H(p_t, s_i) > 5 \end{cases}$$

  Ciò è stato fatto per aiutare il processo di matching in caso di punti GPS compresi tra molti segmenti ravvicinati. In quel caso, infatti, senza tale vincolo verrebbe avvantaggiato il segmento stradale più vicino al punto dando troppa poca considerazione agli altri segmenti anche se anch'essi molto vicini al punto GPS.

- **Probabilità di Transizione** - Essendo i segmenti stradali molto densi in aree urbane, la topologia della rete stradale fornisce vincoli al processo di map matching (per esempio, un oggetto in movimento non può passare

da un segmento ad un altro se non sono connessi in qualche modo), quindi devono essere presi in considerazione per la definizione di probabilità di transizione. Come già spiegato nel Paragrafo 4.2.3, [30] non è adatto a reti stradali con elevata varianza nella lunghezza dei segmenti e nella distanza di campionamento tra i punti delle traiettorie. Avendo come compito quello di utilizzare come area urbana Milano, molto densa di strade, è stata selezionata una probabilità di trasmissione più robusta [29]. Dato un set di segmenti stradali S, la transizione stato-a-stato $P(X_t|X_{t-1})$ è una matrice SxS, dove $P(X_t = j|X_{t-1} = i)$ è la probabilità di transizione da un segmento i (cioè lo stato al tempo $t-1$) al segmento j (cioè lo stato al tempo t):

$$P(X_t = j|X_{t-1} = i) = \frac{1}{\beta} e^{\frac{-d}{\beta}} \tag{5.2}$$

dove $d = |d_h(p_t, p_{t-1}) - d_R(prj(p_t, s_j), prj(p_{t-1}, s_i))|$
prj proietta ortogonalmente p_t e p_{t-1} ai segmenti s_j e s_t, d_R è la lunghezza del percorso più corto in metri tra i due punti e β è il parametro di *smoothing*[29]. In altre parole, dati due punti GPS p_t e p_{t-1} la probabilità di transizione è definita come la differenza tra la great circle distance tra i due punti $d_h(p_t, p_{t-1})$ e la distanza basata sulla topologia stradale data dalla somma delle lunghezze dei segmenti facenti parte del percorso $d_R(prj(p_t, s_j), prj(p_{t-1}, s_i))$. Infatti, secondo [29] nella reale strada percorsa dalla traiettoria c'è tipicamente una piccolissima differenza tra d_H e d_R e il valore assoluto del loro confronto nel caso di match corretto segue una distribuzione esponenziale della probabilità. Per quanto riguarda il parametro β, descrive la differenze tra le great circle distance e le lunghezze del percorso corrispondente al matching corretto (in gergo tecnico si definisce utilizzare una *Groud Truth* dei dati) ed è stimato come:

$$\beta = \frac{1}{\ln(2)} mediana_t(d_t) \tag{5.3}$$

Algoritmo 2 Algoritmo di Map-Matching

Input:

T: traiettoria, $G(V, S)$: rete stradale, β: parametro di *smoothing*, α: numero di vicini, τ: soglia spaziale, γ: massima distanza del percorso, θ: massima profondità del percorso

Output:

M: traiettoria mappata sulla rete stradale

1: $percorsi \leftarrow$ percorsi con distanza massima γ e profondità massima θ in $G(V, S)$
2: $I \leftarrow 1/|S|, \forall_{s_i} \in S$ ▷ Tutti i segmenti sono equamente probabili a p_0
3: $O \leftarrow P(p_t|X_t = i), \forall_{p_t} \in T, \forall_{s_i} \in N(p_t, S, \alpha, \tau)$ ▷ Creazione vettore probabilità di emissione
4: $R \leftarrow P(X_t = j|X_{t-1} = i), \forall_{p_t, p_{t-1}} \in T, \forall_{s_j} \in N(p_t, S, \alpha, \tau), \forall_{s_i} \in N(p_{t-1}, S, \alpha, \tau)$ ▷ Creazione matrice probabilità di transizione
5: $M \leftarrow Viterbi(S, T, I, R, O)$ ▷ Viterbi(spazio degli stati, traiettorie,probabilità iniziale, di emissione e di transizione)
6: **return** M

Più β aumenta, maggiormente vengono tollerati percorsi articolati tra segmenti, funzionando quindi come un vero e proprio "smussatore" del risultato finale.

La definizione formale del processo di map matching effettuato è rappresentata nell'Algoritmo 2. È importante notare come l'insieme delle osservazioni Y equivalga ad O, la matrice delle probabilità di transizione A ad R e la matrice delle probabilità di emissione B al vettore (per ogni punto della traiettoria) O.

5.2 Implementazione

L'algoritmo 2 è la formalizzazione del map matching utilizzato in contesto generale e quindi non è strettamente legato ad un ambito (centralizzato o distribuito). Può quindi trarre in inganno perché non presenta al suo interno metodologie distribuite, scopo del lavoro di tesi, ma in realtà tale algoritmo è stato completamente distribuito, semplicemente si è deciso di non inserire terminologie le quali sarebbero state così specifiche da tradursi in codice imple-

mentativo. Di conseguenza, l'implementazione distribuita, la quale può essere riassunta nella Figura 5.1, verrà spiegata con un maggiore livello di dettaglio in seguito. È stata volontariamente suddivisa in tre macro "step" perché ognuno dei quali rappresenta concettualmente una specifica fase del map matching con Hidden Markov Model e anche perché computazionalmente parlando ognuno di essi ha un peso specifico sulle performance globali del processo completo. L'algoritmo segue i seguenti step per ogni traiettoria all'intero del dataset:

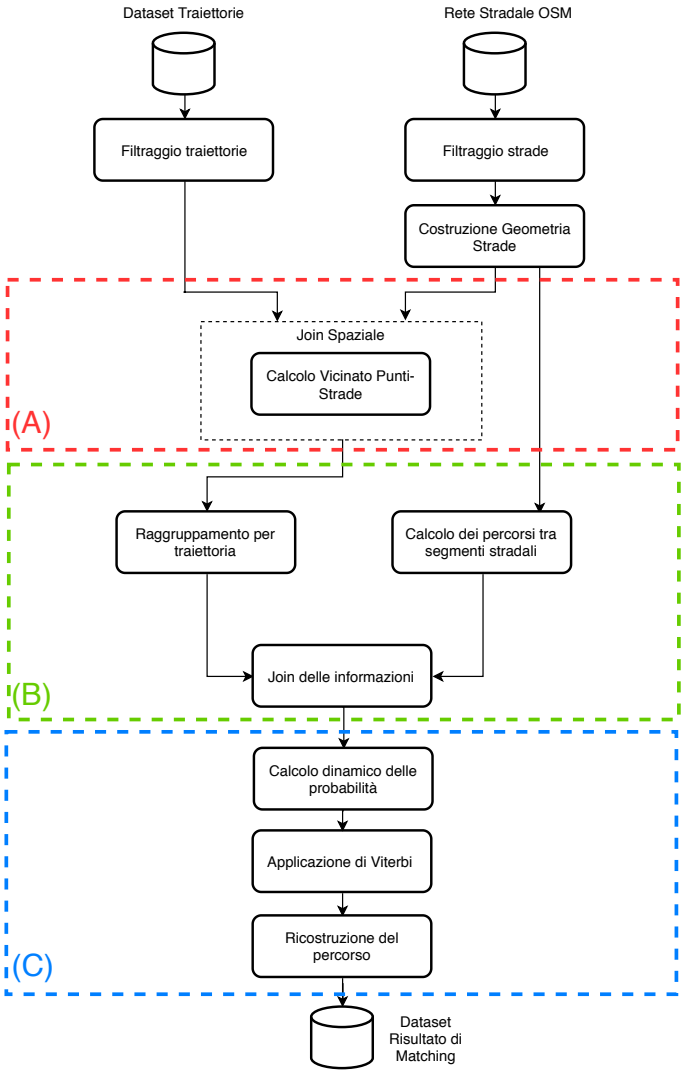


Figura 5.1: Implementazione del map matching distribuito con suddivisione in tre macrostep

5.2.1 Step A

Il primo step si concentra sul calcolo della probabilità di emissione e viene eseguita in maniera massivamente distribuita. Dato un punto GPS p_t (cioè un simbolo osservabile nel modello Markoviano a stati nascosti), la stima della sua probabilità di emissione richiede la computazione del vicinato $N(p_t, S, \alpha, \tau)$ per ogni punto in nel dataset delle traiettorie $\mathcal{T}$. Ciò richiede un *range-join* spaziale, cioè una operazione spaziale che combina due dataset nel rispetto della distanza delle loro caratteristiche geometriche. In altre parole, cerca per ogni punto GPS osservato, tutti i segmenti stradali vicini. È importante notare che tale trasformazione viene applicata a tutti i punti GPS di $\mathcal{T}$ indipendentemente dalla traiettoria a cui appartengono, in modo tale da raggiungere il grado massimo di parallelismo. Per ottenere ciò si è sfruttata la libreria GEOSpark, già dettagliata nel Paragrafo 2.3, con la quale basandosi sull'estensione dell'RDD di Spark (SpatialRDD) rende possibile la caratterizzazione e l'utilizzo delle geometrie degli elementi al suo interno, distribuendo ed ottimizzando le operazioni spaziali. Sfruttando quindi un range-join spaziale sul set dei punti GPS e sull'intera rete stradale, il risultato è un set di α segmenti stradali vicini a p_t entro una determinata soglia spaziale τ. L'efficienza dell'operazione spaziale è fortemente correlata a τ perché determina lo spazio di ricerca dei segmenti candidati, la quale traduzione in operazione di GEOSpark determina un maggiore *shuffle* (cioè quando i dati vengono spostati da diverse partizioni) tra gli elementi del dataset. Al contrario, α determina semplicemente la grandezza dell'output. Il settaggio di questi parametri dipende fortemente dalle specifiche del cluster: maggiori saranno τ e α, maggiore sarà il numero di candidati che verranno considerati (e infine ritornati) per ogni punto. Tutte le considerazioni in merito saranno spiegate nel dettaglio nel Capitolo 6. Dato il risultato uscente da ciò, si sfrutta la potenza di SparkSQL per combinare e generare tramite l'utilizzo di una finestra mobile sui risultati delle distanze del punto p_t dagli α segmenti stradali la probabilità di emissione come nella Definizione 5.1.

5.2.2 Step B

Lo step B può essere considerato come una fase di preparazione a ciò che avverrà poi nello step C. In S sono presenti molti segmenti la cui probabilità di transizione tra essi è quasi nulla (come ad esempio segmenti troppo distanti dal match corretto o connessi da più percorsi con lunghezze differenti). Per alleggerire il carico di lavoro e migliorare le performance, si impedisce il calcolo delle probabilità di transizione che sono vicine allo zero (seguendo la Definizione 5.2). Data infatti ogni coppia di punti p_t e p_{t-1} si considerano come insieme di stati candidati dei rispettivi punti solamente i segmenti stradali che sono all'interno del vicinato formato da α segmenti distanti massimo τ metri dal punto in questione. Oltre a ciò viene aggiunto un ulteriore filtraggio sulla lista dei segmenti stradali non considerando tutte le coppie di segmenti le quali sono connesse da un'insieme di segmenti maggiore di θ o che sono ad una distanza data dalla somma dei segmenti componenti il percorso (quindi d_R, come definita in 5.2) maggiore di γ. Mentre θ limita i percorsi i quali potenzialmente produrranno probabilità di transizione vicina allo zero, γ serve ad alleggerire il processo di creazione dei percorsi tra segmenti prevenendo la computazione di percorsi inutilmente lunghi i quali verrebbero sicuramente scartati nella fase C di calcolo della probabilità di transizione dinamica. Bisogna però far notare che la computazione di percorsi "profondi" (cioè ad elevato θ) è necessario in caso di aree urbane ad elevata frammentazione in segmenti stradali. Visto che il percorso minimo tra i due segmenti con cui si calcola d_R sarebbe servito sia nel calcolo della probabilità di transizione, sia per quanto riguarda la tecnica di costruzione del risultato finale e tale percorso dipende solo e solamente dal grafo $G(V, S)$, può essere computato in precedenza e fatto persistere nel sistema distribuito, per poi essere utilizzato a tempo debito. Per riuscire a distribuire tale calcolo, il quale è computazionalmente molto oneroso se si considerano reti stradali molto grandi e dense di segmenti, si è sfruttata una procedura automatizzata in SparkSQL la quale concatena i segmenti stradali adiacenti fino ad un massimo di θ o al raggiungimento di γ metri, selezionando tra i possibili percorsi tra le coppie di segmenti solamente il più

breve. Per riuscire ad avere disponibile nello step C tutto ciò per calcolare dinamicamente la probabilità di transizione come in 5.2 viene preso il risultato dallo step A, vengono raggruppati i punti GPS per traiettorie e viene creata una finestra mobile di grandezza 2 per ogni traiettoria dove per ogni coppia di punti p_t e p_{t-1} vengono inserite le informazioni inerenti a tutti i percorsi minimi dei segmenti candidati da ogni s_{p_t} in $N(p_t, S, \alpha, \tau)$ ad ogni $s_{p_{t-1}}$ in $N(p_{t-1}, S, \alpha, \tau)$ per poi passare all'ultimo step. Grazie a questa metodologia si arriva alla computazione del modello Markoviano a stati nascosti in modo totalmente distribuito e scalabile a seconda dei parametri selezionati. È bene precisare che sono state provate anche altre tecniche per riusce ad ottenere i percorsi minimi tra segmenti a disposizione nello step C come a collezionare l'intera mappa dei percorsi sul *driver* (cioè il nodo principale dell'applicazione Spark) e copiarla in ogni altro esecutore nel cluster (cioè fare *broadcast* della mappa dei percorsi minimi), in modo tale da avere accesso istantaneo ai percorsi indipendentemente dalla coppia di segmenti durante il processo distribuito in C. Sfortunatamente è una procedura che dipende dalla memoria disponibile nel cluster e non riesce a scalare all'aumentare di θ e γ. Un'altra metodologia invece era l'utilizzare GraphFrames[1] il quale è l'estensione per SparkSQL di GraphX con cui è possibile generare un grafo basandosi sull'intera rete stradale $G(V, E)$ ed interrogalo in maniera intuitiva e distribuita. Anche in questo caso però, venivano raggiunte prestazioni peggiori del caso con SparkSQL a causa del filtraggio per γ effettuato solo dopo aver calcolato tutte le possibilità di profondità θ generando un quantitativo di dati spropositato.

5.2.3 Step C

Questo ultimo step è atto alla computazione parallela e distribuita del map matching delle traiettorie. Per una traiettoria T con $|T|$ punti GPS si può pensare di provare a computare $S^{|T|}$ sequenze per cercare il miglior percorso che genera i punti GPS osservati. Questa soluzione però ha una complessità esponenziale. Come si nota dall'Algoritmo 2 è stato infatti utilizzato come

[1] https://databricks.com/blog/2016/03/03/introducing-graphframes.html

ultimo passo l'algoritmo di Viterbi 1 che estrae la sequenza degli stati ottimale da un modello Markoviano a stati nascosti con una complessità polinomiale $O(|S|^2|T|)$. Pensare interamente l'algoritmo di Viterbi in un contesto distribuito non è per niente banale a causa della sua struttura e della sua natura ricorsiva. Di conseguenza ci si è limitati a parallelizzare massivamente il calcolo del map matching distribuendo le varie traiettorie su tutti gli esecutori del cluster e applicando su ognuna di esse l'algoritmo. Innanzitutto, prima di applicare l'algoritmo viene calcolata la matrice delle probabilità di transizione R secondo Definizione 5.2 dove viene creata una finestra a scorrimento di grandezza 2 nella traiettoria (o in altre parola vengono create delle coppie di p_t e p_{t-1}) e si calcola la probabilità di transizione $P(X_t = j|X_{t-1} = i)$ per ogni coppia di segmenti candidati s_j e s_i, dove $s_j \in N(p_t, S, \alpha, \tau)$ ed $s_i \in N(p_{t-1}, S, \alpha, \tau)$. Naturalmente tutte le informazioni che servono al calcolo, come il vicinato N e i percorsi tra segmenti per d_R, sono già presenti perché generate negli step precedenti rendendo così molto più rapida la computazione. Oltre a tale matrice delle probabilità di transizione, per l'algoritmo di Viterbi è necessario: la traiettoria T, lo spazio dei segmenti (cioè gli stati del modello) S, una vettore I contenente la probabilità iniziale, dove $|I| = |S|$ e $I_i = 1/|S|$ per ogni segmento in S (nel senso che vengono considerati tutti i segmenti equamente probabili nel rispetto del punto GPS iniziale) ed infine la matrice di probabilità di emissione O (computata per ogni punto nello step A come per Definizione 5.1). È bene precisare che l'intero algoritmo di Viterbi è stato passato allo spazio logaritmico (modificando così le moltiplicazioni in somme) per riuscire a gestire traiettorie con $|T|$ molto elevato che avrebbero portato a troppe moltiplicazioni tra numeri molto piccoli (con possibilità di problemi tecnici come l'*underflow*, con il quale l'elaboratore confonde numeri molto piccoli con 0). La complessità computazionale dell'algoritmo può essere abbassata sia diminuendo $|T|$ che $|S|$. Per quanto riguarda $|T|$, vuol dire diminuire la lunghezza della traiettoria, cosa che richiede un processo di semplificazione da applicare prima del processo di map matching. Per quanto concerne invece $|S|$, lo spazio dei segmenti può essere sensibilmente abbattuto assumendo che, tra tutti i segmenti in S, solamente i segmenti vicini al punto osservato contribuiscono al

processo di matching con delle probabilità non nulle. Per ogni punto p_t, vengono considerati solo i segmenti candidati all'interno del vicinato $N(p_t, S, \alpha, \tau)$, dove α (che è il numero dei vicini estratti) determina lo spazio di ricerca di Viterbi. Intuitivamente, applicare l'algoritmo di Viterbi a T, ritorna il percorso con cui T è accoppiato associando ad ogni punto GPS il segmento del percorso ottimale. Sfortunatamente, non c'è la certezza che punti GPS conseguenti vengano mappati a segmenti stradali adiacenti, formando così percorsi ottimali frammentati. È stato quindi aggiunto a fine dell'algoritmo di Viterbi un processo di ricostruzione del percorso ottimale per riuscire a completare i percorsi frammentati. In ogni iterazione tra due punti conseguenti p_t e p_{t-1}, se il percorso più corto tra i segmenti mappati s e s' è più corto di γ ed ha una profondità più piccola di θ, vengono mappati a p_{t-1} anche tutti i segmenti inclusi nel percorso più corto tra s e s'. Per fare ciò, viene sfruttato il percorso computato in precedenza nello step B. Tale passo aggiuntivo, aggiunto in questo punto della computazione (invece che successivo allo step C) previene una iterazione extra su tutte le traiettorie che erano state analizzate, migliorando le performance.

5.3 Strutture Dati

Un qualsiasi progetto in ambito di data mining richiede sempre una ottima conoscenza del dominio applicativo e della relativa base di dati su cui ci si appoggia. Il primo passo effettuato in questo caso di studio è stato infatti quello di ricercare ed analizzare le strutture dati alla base del map matching cioè una collezione dati per le traiettorie ed una per la rappresentazione della rete stradale.

5.3.1 Dataset Traiettorie

La collezione di dati delle traiettorie di movimento è stata messa a disposizione dal laboratorio di Business Intelligence dell'università di Bologna e include spostamenti di utenti per tutto il territorio nazionale. È stato espressa-

mente richiesto però di concentrarsi sull'area amministrativa milanese dovendo quindi effettuare diversi filtraggi su di esso.

Struttura

La collezione di dati è stata memorizzata sul cluster Hadoop e gestita in modo relazionale tramite una tabella Hive con la seguente struttura che ne descrive il comportamento:

- **timestamp** - Il valore numerico della data e dell'ora specifica della rilevazione di posizione.
- **customID** - Identificatore **univoco** dell'oggetto a cui è stata rilevata la posizione.
- **latitude** - Coordinata geografica che rappresenta la distanza angolare misurata in gradi lungo l'arco di meridiano compreso tra l'Equatore e il parallelo passante per il punto considerato.
- **longitude** - Coordinata geografica che rappresenta distanza angolare misurata in gradi, lungo l'arco di parallelo compreso tra il Meridiano fondamentale, Meridiano di Greenwich, e il meridiano passante per il punto considerato.
- **accuracy** - Valore in metri che rappresenta la precisione nella rilevazione della posizione da parte del rilevatore GPS. È bene precisare che più piccolo è tale valore maggiore è la sicurezza che la rilevazione sia accurata.
- **timezoneOffset** - Misura in secondi del fuso orario al tempo della misurazione del timestamp.

Da questa descrizione è facile notare come il concetto di traiettoria non sia mappato e rappresentato correttamente. Viene si modellato il movimento di un *customID* ma esso comprende l'intero insieme di spostamenti che vengono catturati in un grande lasso di tempo, il quale può arrivare a intere settimane. È stato quindi necessario applicare una tecnica di **segmentazione** in modo

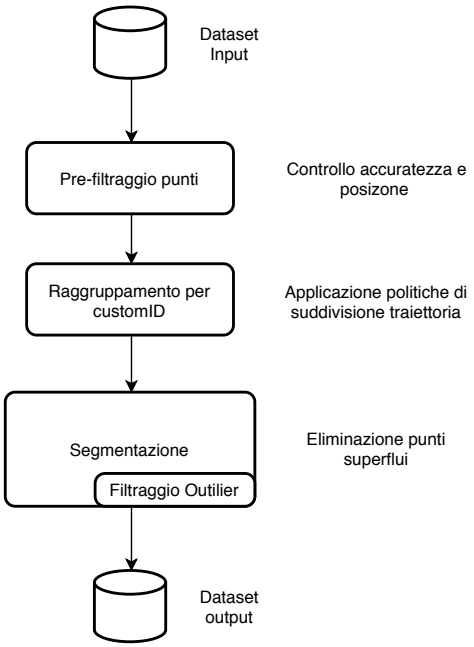


Figura 5.2: Processo di suddivisione del dataset in traiettorie di movimento

tale riconoscere e suddividere per ogni *customID* i movimenti singoli effettuati durante l'intero periodo di campionamento della posizione.

Pre Processamento sul Dataset

Per riuscire a segmentare l'intero dataset delle traiettoria si è sfruttato sempre Apache Spark tramite il quale è stato possibile effettuare anche la segmentazione in maniera rapida ed efficace. Durante il processo si è notato che molti punti all'interno delle traiettorie presentavano spostamenti sostanziali in lassi di tempi molto brevi per appartenere alla traiettoria in analisi, possedendo comunque un valore di accuracy basso e quindi rappresentando devi veri e proprio errori di rilevazione. Si è quindi unito al processo di segmentazione anche una tecnica di eliminazione degli *Outlier*. Come si nota dalla Figura 5.2 la quale schematizza e riassume il processo, si può suddividere il procedimento in quattro passi fondamentali:

- **Filtraggio punti** - Per recuperare il dataset da Hive viene effettuato un filtraggio iniziale dei punti tramite una interrogazione HiveQL escluden-

do tutti quei punti che non sono all'interno dei confini amministrativi milanesi o che hanno una accuratezza così elevata (si ricorda che maggiore è tale valore, minore è la sicurezza che la posizione rilevata sia corretta) da risultare poi problematici al fine del matching di traiettorie perdendo l'informazione della posizione reale dell'oggetto.

- **Raggruppamento per customID** - Viene sfruttata la potenza di Spark generando un RDD dal dataset estratto e raggruppando tutti i punti a seconda del customID.

- **Segmentazione** - Passo fondamentale del procedimento. Per ogni insieme di punti raggruppato si attua la segmentazione basandosi fondamentalmente sulla seguente assunzione: una traiettoria è considerata tale solamente quando la distanza tra ognuno dei suoi punti non supera una soglia temporale di un minuto. Per attuare ciò viene applicato un filtro su ogni punto della traiettoria dove si mantiene in memoria sempre il punto precedentemente analizzato in modo tale da riuscire a calcolare l'intervallo temporale tra i due punti. Sfruttando tale punto precedente viene anche calcolata la velocità utilizzando la differenza di tempo e la distanza tra i due punti. Per calcolare la distanza viene utilizzata la great circle distance la quale determina la distanza tra due punti su una sfera date le coordinate.

- **Eliminazione degli Outilier** - All'interno del processo di scorrimento dei punti, viene mantenuta un'ulteriore finestra di punti p precedenti $F = \{p_1, p_2, ..., p_N\}$ di grandezza N, con N dispari, dove il punto analizzato nella procedura è $p_{(N/2)+1}$ quindi in posizione centrale alla finestra, come si può notare dalla rappresentazione in Figura 5.3. Viene applicato un filtraggio basato sull'utilizzo di una **media mobile** centrata basata sulla velocità del punto. Vengono infatti calcolate le medie per i punti precedenti e per i punti antecedenti al punto in esame e viene confrontata che la differenza tra la velocità del punto in esame e le corrispettive medie non sia superiore ad una soglia, in caso contrario il punto è etichettato

come outilier. La soglia viene anch'essa calcolata dinamicamente ed è data da:

$$soglia = \frac{p_{(N/2)+1}.v}{3} + thsP \tag{5.4}$$

dove $p_{(N/2)+1}.v$ indica la velocità del punto in esame e $thsP$ rappresenta la velocità minima di una persona a piedi necessaria nella formulazione della soglia perché la irrobustisce (evitando di eliminare punti la quale velocità è in realtà tollerabile ma secondo formulazione dovrebbe essere eliminata) nel caso di ricerca di outlier con velocità molto basse. Con tale soglia tutti gli outlier vengono di fatto eliminati tramite filtraggio alla fine del procedimento di segmentazione e la tabella risultate viene salvata su Hive. È bene aggiungere che utilizzando una media mobile, vengono persi durante la segmentazione i punti successivi all'interno della finestra dell'**ultimo** punto preso in esame per il controllo degli outlier all'interno della intera sequenza di punti che vengono controllati. Questo perché i punti successivi non hanno più punti a disposizione per costruire la finestra necessaria al calcolo delle medie mobili. È quindi importante scegliere correttamente la grandezza della finestra la quale più grande è, maggiore sarà la robustezza del calcolo delle medie e quindi della precisione nel trovare gli outlier ma porterà a una perdita di $(N/2) + 1$ punti totali alla fine del processo di segmentazione.

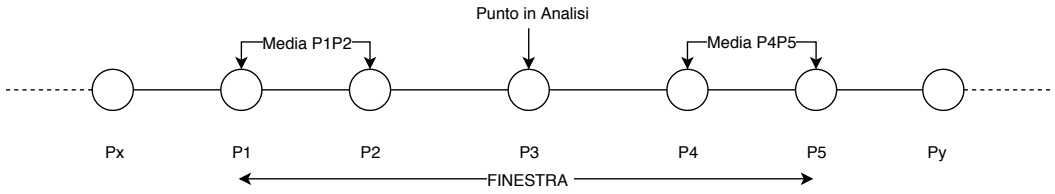


Figura 5.3: Calcolo della media mobile data una finestra di grandezza 5

5.3.2 Dataset OpenStreetMap

Per quanto riguarda la rete stradale è fondamentale riuscire ad ottenerla nella forma richiesta dal map matching, cioè rappresentata da un grafo com-

posta da segmenti e nodi. In letteratura e sul web sono presenti tanti tipi di dataset sulle reti stradali, ma sono tutti circoscritti alle più famose città cinesi [12] [29] e americane [32]. Essendo l'obiettivo quello di mappare un insieme di traiettorie nella città di Milano, si è cercata una metodologia per poter generalizzare l'estrazione e la creazione di una rete stradale data la città o addirittura la nazione intera su cui poi applicare il map matching. La soluzione è stata possibile grazie ad *OpenStreetMap*[2], un progetto collaborativo finalizzato a creare mappe del mondo intero a contenuto libero, dove qualsiasi utente può modificarle, integrarle con nuovi elementi ed appunto, scaricarle e utilizzarle per gli scopi più disparati.

Struttura della Mappa

OpenStreetMap è stato concepito con lo scopo di rappresentare il mondo in modo più semplice e chiaro possibile. È da considerarsi un database dove tutti gli elementi che possono essere inseriti (strade, negozi, ecc..) sono inclusi in quattro tipologie:

- **Nodo** - Elemento chiave, definito come un punto nello spazio geografico il quale possiede latitudine, longitudine e un identificatore univoco. Può aiutare a definire la forma di una strada o di un poligono o essere un punto singolo per rappresentare oggetti come per esempio una panchina in un parco.

- **Strada** - Lista ordinata di nodi. Può rappresentare forme geometriche come linee (strade, fiumi ecc..) o poligoni con i quali vengono rappresentate tutte le tipologie di aree e confini.

- **Relazione** - Struttura atta a documentare diverse tipologie di relazioni tra due o più elementi (nodi,strade o relazioni). Un membro di una relazione può opzionalmente avere un ruolo che ne descrive la parte. Esempi di relazioni sono per esempio la modellazione di "multipoligoni"

[2]https://www.openstreetmap.org

per rappresentare aree complesse dove un poligono classico non basta o per rappresentare percorsi di linee del bus o dei treni.

- **Tag** - Tutti i tipi di elementi (nodi,strade o relazioni) per essere descritti possono avere svariati *tag* i quali ne modellano il contesto e le varie caratteristiche.

Passaggio Mappa OSM su Apache Hive

Estratto quindi il file corrispondente alla mappa, si presenta in formato **OSM XML**. Per riuscire ad inserire correttamente tale file in tabelle Hive ci si è avvalsi di **OSM2Hive**[3]. Tale progetto è una collezione di funzioni aggiuntive per Hive le quali riescono ad analizzare ogni elemento del file OSM XML estraendone la tipologia e tutti gli attributi. Con esso vengono infatti create tre tabelle su Hive, una per ogni elemento (nodo, strada e relazione) le quali condividono alcune colonne come l'identificatore, la versione, il timestamp di modifica e i tag i quali vengono rappresentati con una colonna ad attributo mappa di stringhe. In questo modo tutti gli attributi specifici vengono collezionati come coppie chiave-valore al suo interno e saranno facilmente estraibili e utilizzabili. Per quanto riguarda le colonne specifiche per tipologia, i nodi possiedono latitudine e longitudine, le strade vengono identificate tramite colonna contenente una sequenza di identificatori dei nodi e le relazioni aggiungono una colonna la quale mappa i componenti della relazione.

Filtraggi

Data la descrizione di *Strada* nella struttura di OpenStreetMap si nota che essa può rappresentare un'enorme quantitativo di figure geometriche, non solamente i segmenti stradali. È stato quindi necessario operare un filtraggio al dataset delle strade in modo tale da ottenere solamente i segmenti stradali necessari al processo di matching. Ci si è avvalsi del tag *highway* il quale è fondamentale perché viene utilizzato per identificare le varie tipologie di

[3]https://github.com/PanierAvide/OSM2Hive

strade e sopratutto è un tag obbligatorio, di conseguenza sfruttandolo non si rischia di scartare delle strade necessarie. Naturalmente OpenStreetMap punta a mappare qualsiasi elemento presente su una mappa di conseguenza il tag highway include anche marciapiedi, strade di servizio, linee dei tram che potrebbero sovrapporsi con le strade realmente percorse perché attigue ad esse aumentando così la possibilità di errore da parte del procedimento di matching. Il dataset della rete stradale viene quindi ulteriormente filtrato andando a selezionare solamente le righe della tabella che hanno il tag $highway \subset (motorway, trunk, primary, secondary, tertiary, unclassified, residential)$ che rappresentano la scala di importanza delle strade.

Capitolo 6

Testing

In questo capitolo sono analizzati e dettagliati tutti i test effettuati con l'implementazione distribuita dell'algoritmo: è infatti fondamentale in un applicazione con l'obiettivo di lavorare con i big data gestire al meglio le risorse che il cluster ha a disposizione e raggiungere il miglior compromesso possibile tra l'accuratezza del risultato e le performance di esecuzione dell'algoritmo.

6.1 Dataset

Il dataset utilizzato per i vari test effettuati presenta le caratteristiche evidenziate in Tabella 6.1. Si nota che tale dataset è stato ottenuto secondo segmentazione come dettagliato nel paragrafo 5.3.1. Oltre a ciò sono stati aggiunti due filtraggi aggiuntivi all'interno della procedura:

- Sono mantenute le traiettorie che possiedono almeno una velocità media di 20 km/h. Ciò è stato fatto per assicurarsi che le traiettorie siano realmente significative in termini di movimento dell'oggetto (escludendo le traiettorie delle persone in casa o comunque che si spostano in luoghi al chiuso come centri commerciali, che porterebbero il map matching ad eseguire l'associazione mappando tutti i punti della traiettoria nello stesso singolo segmento stradale affianco all'edificio).

- Vengono considerate solo le traiettorie composte da almeno 20 punti in modo tale da possedere una lunghezza significativa per modellare lo spostamento dell'oggetto.

Il dataset include così 120000 traiettorie (T) le quali contengono un numero medio di 65 punti GPS (risultando così in un totale di 7.8 milioni di punti GPS). La distanza tra due punti GPS consecutivi è intorno ai 99 metri e la media del campionamento di tali punti è di circa 7 secondi (cioè a circa 0.14Hz). La media del campionamento può trarre in inganno perché rimangono comunque presenti casi spuri in cui due punti sono molto più distanti. Ciò ha portato a diverse considerazioni le quali vengono esplicitate nei paragrafi successivi.

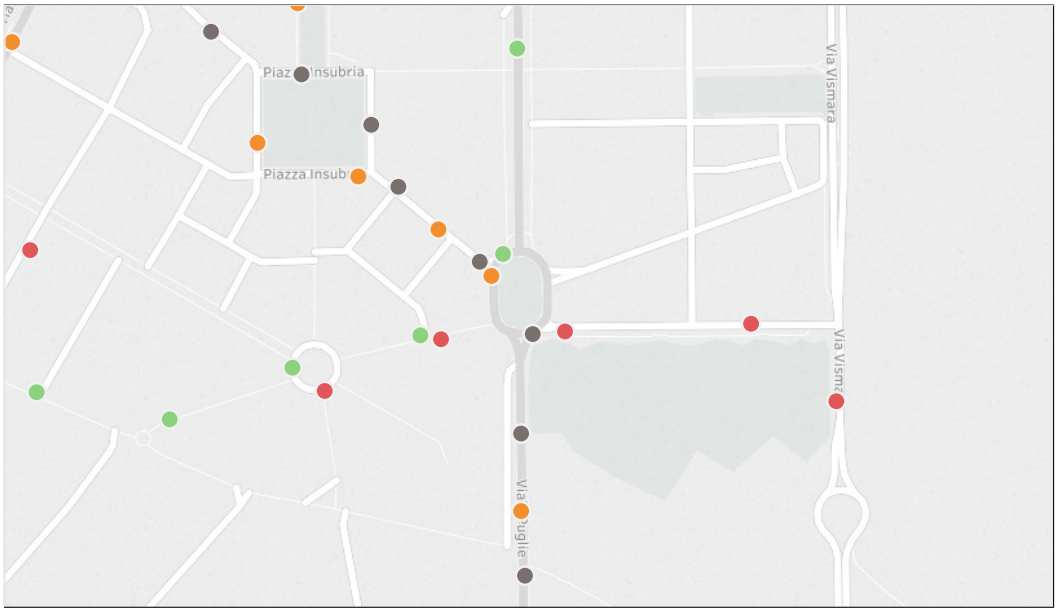
Caratteristica	**Valore**
Punti GPS	$7.8 \cdot 10^6$
Traiettorie	$12 \cdot 10^4$
Media punti in traiettoria	65 ± 42
Media distanza tra punti (metri)	99 ± 55
Media intervallo di campionamento posizione (secondi)	7 ± 6

Tabella 6.1: Caratteristiche del dataset delle traiettorie in analisi

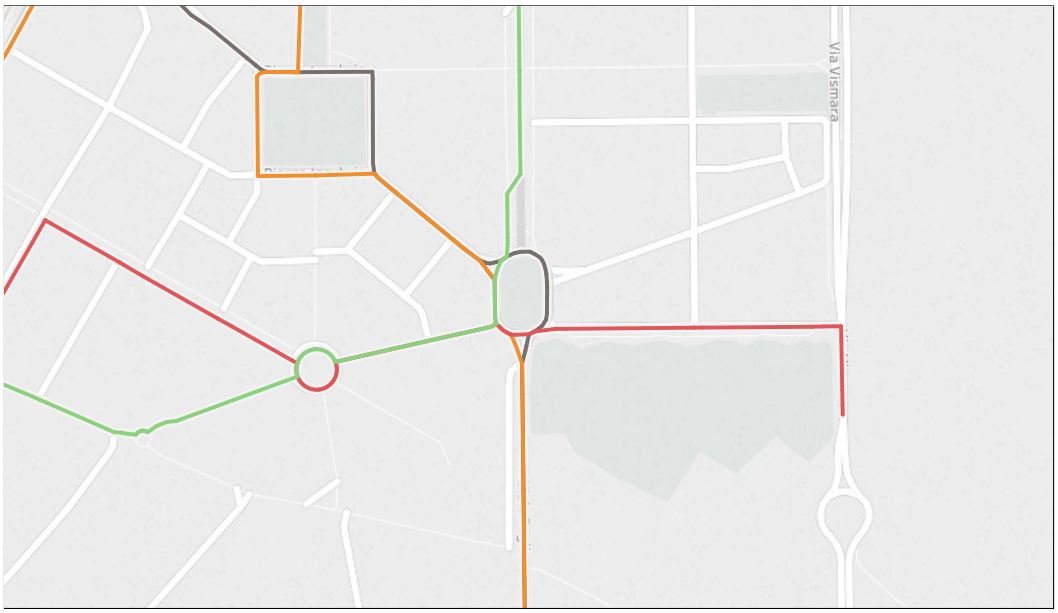
6.2 Accuratezza del risultato

Un'esempio di risultato del procedimento di map matching è mostrato nella Figura 6.1. L'immagine superiore mostra 4 traiettorie ben definite da colori differenti tra di loro e in quella inferiore il conseguente risultato di matching con i segmenti che possiedono lo stesso colore dei punti della traiettoria associata.

Dal dataset descritto in 6.1 sono state estratte 50 traiettorie molto dettagliate in termini di punti in modo tale che possa essere evidente il percorso effettuato. Tale insieme di traiettorie ha infatti formato la *Ground Truth*. Con Ground Truth si intende quell'insieme di dati empirici "veri", raccolti tramite osservazione diretta e quindi utilizzati per testare la capacità predittiva di un algoritmo.



(a) Traiettorie GPS



(b) Risultato di map matching

Figura 6.1: Funzionamento della procedura di map matching su 4 traiettorie GPS. Punti della traiettoria e risultato corrispondente presentano lo stesso colore.

Parametro	Valore	Significato
β	10	Smoothing di probabilità
α	$[4, 10]$	Segmenti candidati
τ	$[100, 200]$	Soglia di ricerca del vicinato (metri)
γ	$[4, 10]$	Numero di segmenti del percorso
θ	$[300, 800]$	Lunghezza massima percorso (metri)

Tabella 6.2: Range di valori analizzato per ogni parametro dell'algoritmo

Tali dati si sono infatti sfruttati per testare l'accuratezza dell'algoritmo di map matching implementato, nel rispetto dei parametri α, τ, θ, γ. Oltretutto, queste 50 traiettorie sono state utilizzate per stimare α, come da Definizione 5.3. La Ground Truth è infatti definita rispetto ai valori massimi indicati nella Tabella 6.2 e i test di accuratezza sono seguiti variando ogni volta uno dei parametri. Per calcolare l'accuratezza del risultato al variare dei parametri si è fatto uso di un *Key Performance Indicator*, o KPI, chiamato *Conformità*, il quale è in grado di misurare la bontà del risultato nel rispetto della Ground Truth descrivendo quanto il risultato di map matching è conforme a quello associato alla Ground Truth. Infatti tale KPI è calcolato come:

$$Conformità = \frac{Corretti}{(Corretti + Non\ Assegnati + Non\ Corretti)} \tag{6.1}$$

Dove *Corretti* indica il quantitativo di segmenti associati correttamente, *Non Assegnati* indica il quantitativo di segmenti corretti che non sono stati associati ed infine *Non Corretti* indica il quantitativo di segmenti che sono stati associati ma non sono presenti nella Ground Truth. Il risultato di questo KPI è un valore da 0 ad 1 ed è stato utilizzato perché riesce a dare un peso anche al quantitativo di segmenti associati che in realtà sono sbagliati. Oltre alla Conformità è stato calcolato anche un valore che riesca a rappresentare in che modo il la Conformità differisce dalla Ground Truth. Tale valore viene definito come *Deviazione* che equivale alla differenza tra $NonCorretti$ e $NonAssegnati$. Si può notare infatti dai grafici in Figura 6.3 la colonna di destra la quale mostra in che percentuale il risultato di map matching devia dalla Ground Truth:

se è negativo significa una maggioranza di segmenti corretti non associati, se positivo una maggioranza di segmenti non necessari assegnati in più del dovuto. I punti fondamentali che si possono notare dai grafici in Figura 6.3 sono i seguenti:

- Diminuire α porta ad un peggioramento del risultato. Diminuendo il parametro aumenta la probabilità di non prendere in considerazione il segmento realmente percorso dall'oggetto nel caso il campionamento del punto non sia molto preciso, o anche nel caso di punti GPS in parti di mappa molto dense di segmenti come rotonde ed incroci stradali. Nella maggior parte dei casi in cui non venga preso in considerazione il segmento corretto, Viterbi assegna il punto ad un segmento vicino sbagliato. È importante notare che però non porta ad errare tutto il map matching ma solamente la parte inerente a quel segmento (il quale può comprendere anche un diverso percorso ricostruito andando ad aumentare il valore dei *Non Corretti*). Si nota infatti che più si abbassa α maggiormente la *Deviazione* tende ad aumentare positivamente a causa dell'aumentare del numero di segmenti $NonCorretti$ assegnati da Viterbi. È infatti l'unico grafico della Figura 6.3 che ha un netto cambiamento della *Deviazione* passando da una percentuale negativa in caso di α 8 a una percentuale positiva abbassando il valore del parametro. Questo è dato dal fatto che il numero dei segmenti $NonCorretti$ aumenta in modo così repentino da superare il quantitativo dei segmenti $NonAssegnati$ a causa delle assegnazioni sbagliate e dalle rispettive ricostruzioni del percorso che possono comprendere un numero elevato di segmenti $NonCorretti$ se tra il segmento assegnato in maniera errata e il successivo (il quale può essere a sua volta non corretto) sono presenti un elevato numero di segmenti (ciò dipende dalla distanza tra i punti della traiettoria GPS).

- γ e τ portano ad un netto peggioramento della soluzione dato dal fatto che si diminuisce la profondità del calcolo dei *percorsi* e di conseguenza ci sono più probabilità che in un punto della catena Markoviana i segmenti stradali del punto p_t non siano connessi ad i segmenti stradali del punto

p_{t-1}. Se ciò accade la computazione si interrompe e vengono mantenuti solo i segmenti associati fino a quel punto. Per questo motivo si nota un netto aumento negativo della *Deviazione* dato dai tanti segmenti persi a causa di tale problema. Essendo una fase che si computa in maniera massivamente distribuita e può essere calcolata in precedenza e salvata in modo anch'esso distribuito è altamente consigliato calcolare i *percorsi* con un γ e τ elevato, sempre tenendo in considerazione della densità che ha la mappa stradale che si utilizza. Nel caso di studio della mappa presa da OSM, il quantitativo di segmenti presenti è molto elevato, come mostrato dalla Figura 6.2, per questo motivo γ e τ portano ad un abbassamento così notevole del KPI.

- Per quanto riguarda invece τ il risultato resta stabile, cambiando di un valore molto contenuto. Questo perché il parametro τ porta solamente ad un ingrandimento del raggio di ricerca dei segmenti durante il calcolo del *vicinato* dei punti. In questo caso di studio, dato il filtraggio e la segmentazione spiegata nel paragrafo 5.3.1 il rumore nella rilevazione della posizione non è presente e di conseguenza basta un raggio di ricerca di 100 metri per includere tutti i segmenti candidati necessari. Nel caso invece si utilizzasse un dataset con all'interno molto rumore nella rilevazione è consigliato un τ più elevato per riuscire ad includere anche il segmento corretto.

Figura 6.2: Caso di studio che dimostra la definizione della rete stradale presa in esame, ogni segmento stradale è identificato da un colore diverso.

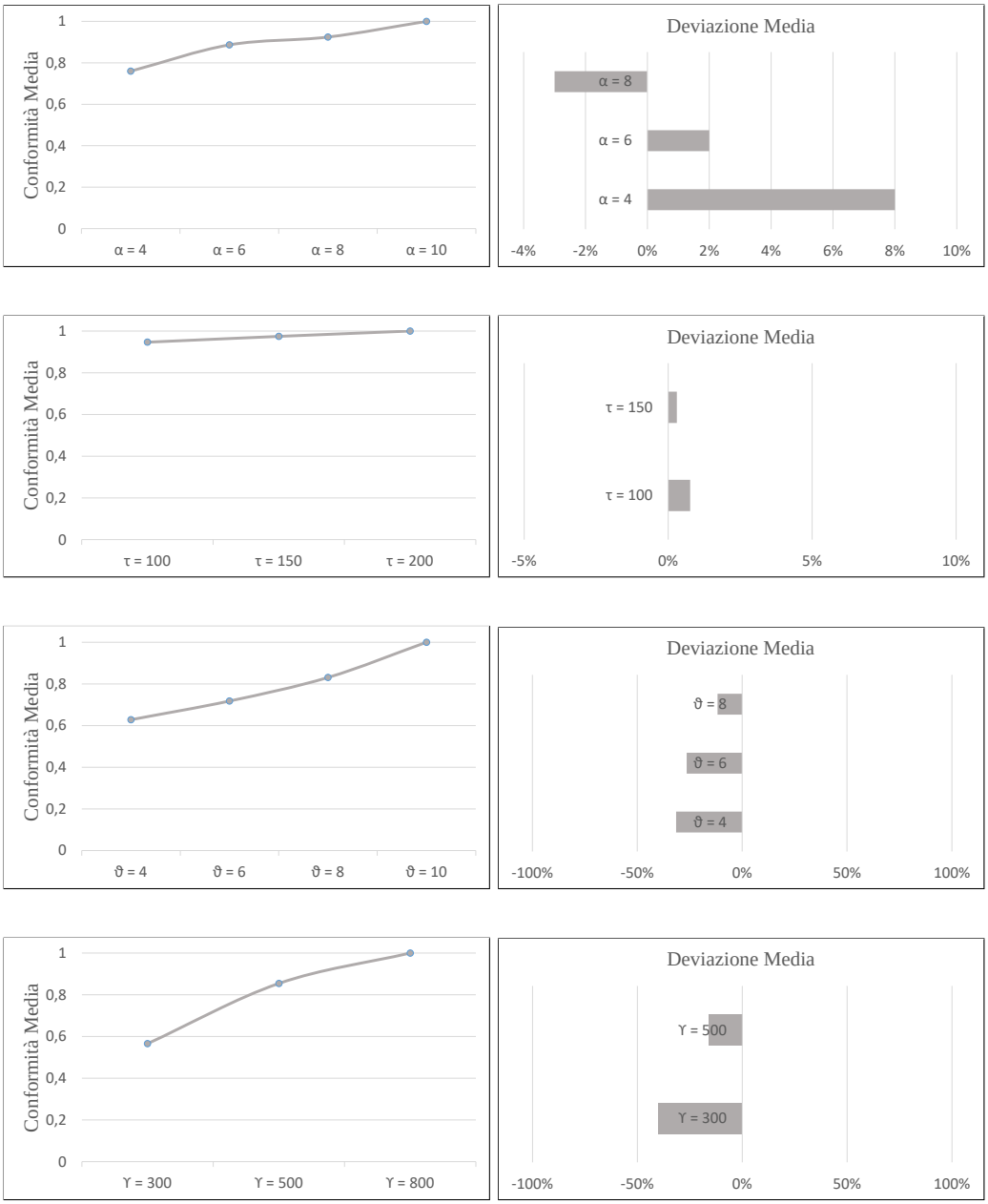


Figura 6.3: Calcolo dei KPI inerenti all'accuratezza del map matching al variare dei parametri α, τ, θ, γ. Nella colonna di sinistra il risultato di conformità, a destra il valore della deviazione, entrambi risultanti confrontando il risultato modificando un parametro con la Ground Truth.

6.3 Performance di esecuzione

Per quanto riguarda i test sulle performance, sono tutti stati effettuati nel caso di studio big data comprendente tutte le traiettorie in Milano come descritto in 6.1. I test sono stati eseguiti su un cluster composto da 11 nodi, ognuno dei quali equipaggiato con un 8-core i7 CPU 3.60Hz e 3GB di RAM e interconnessi da una ethernet Gigabit. Tutti i parametri indicati nel Capitolo 5 sono stati testati seguendo un range di valori come indicati dalla tabella riassuntiva 6.2. Per valutare l'implementazione in termini di efficienza, è stato misurato l'impatto di ogni parametro sul tempo di esecuzione di ogni step dell'algoritmo. Sono state sfruttate le metriche della GUI di Spark web per ottenere i tempi di esecuzione della computazione dei *percorsi* e la computazione dei vari step A, B e C. Si è definita una configurazione di default dove tutti i parametri sono stati valutati al loro minimo, nel rispetto delle considerazioni fatte nel paragrafo sull'accuratezza 6.2 e successivamente sono stati progressivamente scalati fino al loro massimo. Come si può notare dai grafici in Figura 6.4 ogni parametro impatta solamente su uno degli step dell'algoritmo:

- α - È sicuramente il parametro che impatta maggiormente sulle performance dell'intera procedura di map matching. Aumentare infatti α significa generare un maggior numero di segmenti candidati per ogni punto della traiettoria e questo si traduce in una maggiore complessità nel calcolo dell'algoritmo di Viterbi (step C) il quale, come spiegato nel paragrafo 5.2.3 è l'unico elemento non distribuito data la sua natura ricorsiva. È molto importante tarare bene questo parametro per riuscire a raggiungere una giusto compromesso tra accuratezza e performance basandosi sulla densità della mappa che si utilizza e sul periodo di campionamento dei punto che compongono le traiettorie.

- τ - Per quanto riguarda τ aumentarlo significa aumentare lo spazio di ricerca del join spaziale per calcolare il vicinato nello step A. Le performance vengono quindi intaccate solamente in questo step.

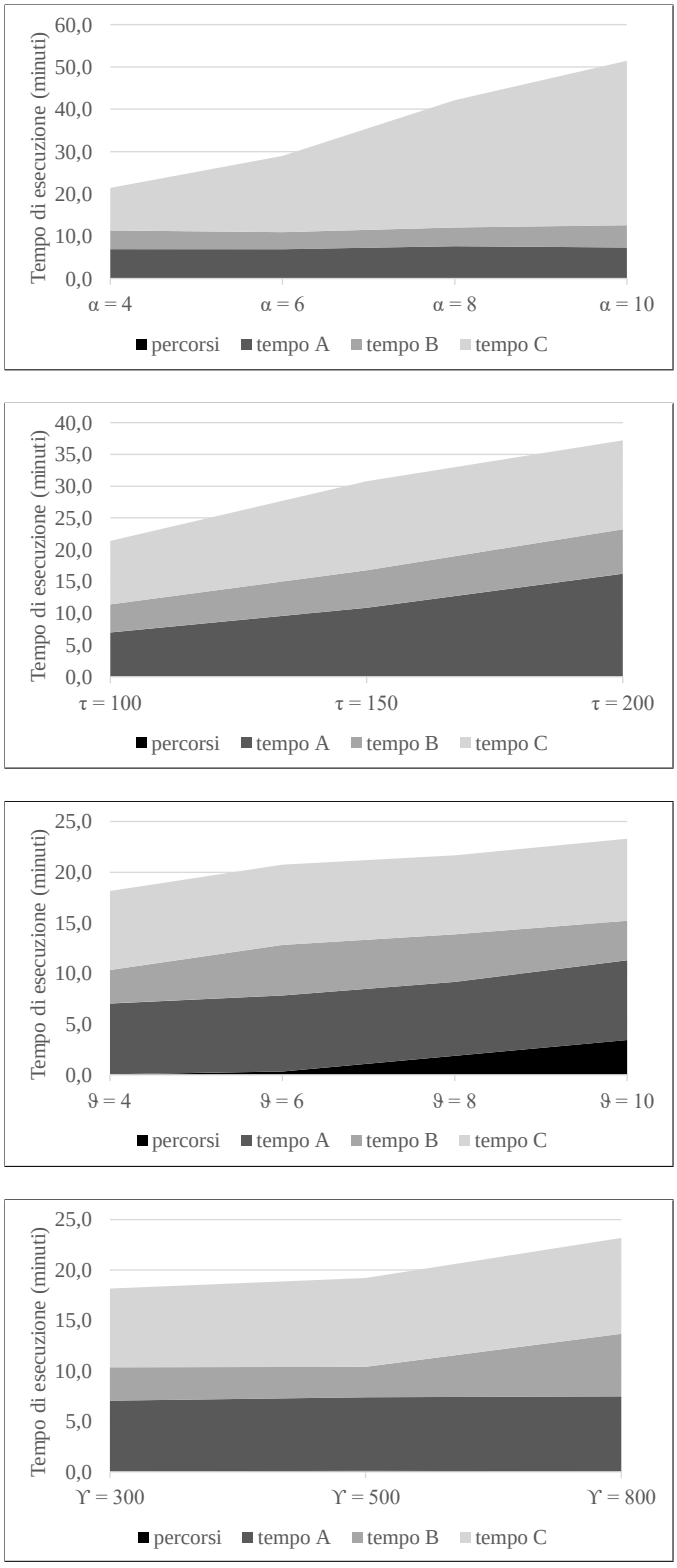


Figura 6.4: Tempi di esecuzione dei diversi step dell'algoritmo sotto a differenti valore dei parametri. Quando non è specificato sono: $\alpha = 4$, $\tau = 100$, $\theta = 4$, and $\gamma = 800$.

Ciò è dato dall'aumentare esponenziale di dati che GEOSpark deve gestire per trovare il vicinato di ogni punto. Basti pensare in aree urbane molto dense il quantitativo di segmenti presi in considerazione se il raggio di ricerca aumenta da 100 metri a 200 metri: per ogni segmento GEOSpark deve calcolare la distanza con il punto e poi recuperarla spostandola tra le varie partizioni per riuscire ad ordinarle ed estrarre le α distanze più piccole.

- θ e γ - Il parametro della profondità θ aumenta la complessità della computazione dei *percorsi* ed è strettamente legato alla grandezza di γ (in metri) della ricerca di tali percorsi. Questo perché come si può notare dai tempi espressi dai grafi nella Figura 6.4 finché γ è limitato ai 300 metri, la profondità dei percorsi non aumenta, di conseguenza non vengono prodotti effetti sullo step B associato. Al contrario, un valore maggiore di γ produce un maggior numero di percorsi, aumentando la computazione nello step B. È bene considerare sempre che questi due parametri impattano sulle performance di una computazione che può essere eseguita anticipatamente al processo di matching, di conseguenza si può pensare di avere un θ e un γ molto elevati.

Concludendo, analizzate anche le performance, per quanto riguarda il dataset utilizzato e la mappa stradale presa da OpenStreetMap, il settaggio dei parametri per quanto riguarda il miglior compromesso tra accuratezza e performance date le caratteristiche del cluster spiegate nel paragrafo sono stati:

- α a 7 per riuscire ad includere tutti i segmenti candidati necessari.
- τ a 100 metri è sufficiente per riuscire a raggiungere tutti gli α segmenti per ogni punto.
- θ a 10 per riuscire ad includere tutti i percorsi necessari al calcolo successivo delle probabilità di transizione data la mappa di OSM molto dettagliata sopratutto in caso di rotonde e incroci complicati. Esempio lampante è la Figura 6.2.

- γ a 500 è sufficiente per considerare tutti i percorsi tra segmenti stradali candidati di due punti GPS successivi dato il campionamento dei punti basso, come si può evincere dalle caratteristiche del dataset 6.1.

Conclusioni

Concludendo, il lavoro svolto rende possibile applicare con un elevato grado di accuratezza nei risultati, un algoritmo di map matching probabilistico facente uso di un modello complesso come il modello Markoviano a stati nascosti in maniera distribuita e scalabile, tramite una implementazione totalmente open-source. La metodologia proposta è stata applicata efficacemente in un contesto di studio reale su oltre 120000 traiettorie di movimento (per un totale di 7.8 milioni di punti GPS). Considerando che in letteratura gli approcci di map matching nell'ambito distribuito peccano di robustezza, questo algoritmo riesce a sfruttare i modelli Markoviani a stati nascosti i quali rendono il processo di map matching robusto e resistente alle possibili diversità di caratteristiche delle traiettorie. Infatti, per quanto riguarda l'accuratezza, l'algoritmo e l'implementazione distribuita proposta non inficia in alcun modo sui risultati di matching che mantengono la stessa accuratezza degli approcci sequenziali utilizzati nei vari paper scientifici che sfruttato i modelli Markoviani a stati nascosti per il map matching in maniera sequenziale. L'algoritmo proposto utilizzante HMM viene oltretutto reso maggiormente robusto in caso di elevata frammentazione delle traiettorie e della rete stradale sottostante aggiungendo una metodologia per la ricostruzione dei percorsi in modo tale da completarli in caso di soluzioni con segmenti stradali scollegati. Per quanto riguarda le performance l'algoritmo raggiunge un elevato grado di scalabilità grazie alla programmazione distribuita effettuata. Sono state identificate nel processo distribuito di map matching quattro variabili che ne modellano caratteristiche specifiche le quali possono essere tarate a seconda della potenza di calcolo che si ha a disposizione e dalle strutture dati (dataset delle traiettorie e della rete

stradale) sulle quali si vuole applicare la procedura di map matching in modo da raggiungere il miglior risultato possibile con le migliori performance possibili. Il procedimento è stato infatti ottimizzato in ogni sua parte cercando di sfruttare il più possibile la potenza e le tecniche di ottimizzazione delle performance messe a disposizione dagli strumenti Apache Hadoop utilizzati per parallelizzare il maggior numero possibile di elementi dell'algoritmo sviluppato.

Sviluppi Futuri

L'elaborato implementato riesce nel suo obiettivo di associare correttamente un grande numero di traiettorie con i percorsi associati nell'intera mappa di Milano. Sono comunque diverse le possibili estensioni che possono essere applicate per migliorarne ulteriormente i vari aspetti:

- Aggiungere al calcolo delle probabilità del modello Markoviano a stati vincoli legati al tempo e alla velocità della traiettoria in confronto alle proprietà della rete stradale sottostante. In questo caso di studio infatti, si è deciso di non inserire questi vincoli perché la rete stradale utilizzata non possedeva nella sua interezza le caratteristiche richieste, come il limite di velocità di ogni sua strada.

- Estendere il processo di map matching dal confine amministrativo milanese all'intera regione Lombardia. Essendo questa parte totalmente distribuita nell'implementazione è possibile estendere i confini della mappa scalando nel processo di calcolo dei percorsi e senza intaccare le performance il calcolo del vicinato per i punti delle traiettorie GPS.

- Applicare un ulteriore filtraggio iniziale alle traiettorie in modo tale da effettuare una riduzione dei punti nei casi di sovrabbondanza di punti in breve lasso di tempo o di spazio in modo tale da perfezionare ulteriormente i tempi di esecuzione eliminando calcoli aggiuntivi superflui.

- Migliorare la ricostruzione del percorso in caso di traiettorie a bassa frequenza di campionamento, aggiungendo tecniche per considerare i sensi di marcia e i vari limiti quando si vanno a calcolare in precedenza i percorsi stradali. Un'altra possibile estensione può essere quella di costruire un modello di scelta dei percorsi multipli a seconda dei possibili vincoli.

- Cercare una possibile approssimazione dell'algoritmo di Viterbi per riuscire a migliorarne ulteriormente le prestazioni.

Ringraziamenti

Ci tengo innanzitutto a ringraziare vivamente il mio relatore, il prof. Matteo Golfarelli per la possibilità a me data nell'affrontare il mio percorso di tesi sull'ambito che maggiormente preferivo. Lo ringrazio oltretutto per essere stato sempre disponibile per ogni tipo di chiarimento o dubbio in merito allo studio e al successivo sviluppo dell'elaborato. Ringrazio il mio co-relatore, il dott. Matteo Francia, per tutti gli aiuti a me dati durante l'intero periodo di sviluppo, facendosi sempre trovare disponibile per qualsiasi tipologia di domanda tecnica o teorica. Ringrazio oltretutto il dott. Enrico Gallinucci per tutti i consigli pratici che mi hanno aiutato nel raggiungere gli obiettivi prefissati e dal quale ho imparato molto. Ringrazio i miei colleghi tesisti del Business Intelligence Group per avermi fatto compagnia in questi mesi difficili, sopratutto Mattia, compagno di innumerevoli progetti ed esami universitari, con cui ho condiviso praticamente tutti i miei momenti passati all'università. Un ringraziamento speciale alla mia famiglia, in particolare ai miei genitori ed ai miei nonni: è grazie a loro sostegno e al loro incoraggiamento se oggi sono riuscito a raggiungere questo traguardo. Infine, una dedica speciale ai miei amici, che ogni giorno hanno condiviso con me gioie, sacrifici e successi, senza voltarmi mai le spalle. L'affetto e il sostegno che mi hanno dimostrato rendono questo traguardo ancora più prezioso.

Bibliografia

[1] Zhenni Feng , Yanmin Zhu, *A Survey on Trajectory Data Mining: Techniques and Applications*, IEEE Access (Volume: 4), 13 Aprile 2016.

[2] Yu Zheng, *Trajectory Data Mining: An Overview*, Microsoft - ACM Transaction on Intelligent Systems and Technology , Settembre 2015.

[3] Wuman Luo, Haoyu Tan, Lei Chen, Lionel M. N, *Finding Time Period-Based Most Frequent Path in Big Trajectory Data*, Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD, 2013.

[4] Qu, M., Zhu, H., Liu, J., Liu, G., Xiong, *A Cost-Effective Recommender System for Taxi Drivers*, The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD, 2014.

[5] Yuan, N.J., Zheng, Y., Xie, X., Wang, Y., Zheng, K., Xiong, H, *Discovering Urban Functional Zones Using Latent Activity Trajectories*, EEE Trans. Knowl. Data Eng.27, 2015.

[6] D. Bernstein and A. Kornhauser, *An introduction to map matching for personal navigation assistants*, New Jersey TIDE Center, 1996.

[7] C. White, D. Bernstein, and A. Kornhauser, *Some map matching algorithms for personal navigation assistants*, Transportation Research Part C: Emerging Technologies, 8(1-6):91–108, 2000.

[8] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, *On map-matching vehicle tracking data*, Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005.

[9] S. Chawathe, *Segment-Based map matching*, In Intelligent Vehicles Symposium, IEEE, 2007.

[10] H. Yin and O. Wolfson, *A weight-based map matching method in moving objects databases*, 2004.

[11] M. Quddus, W. Ochieng, L. Zhao, and R. Noland, *A general map matching algorithm for transport telematics applications*, GPS solutions, 7(3):157–167, 2003.

[12] Lou, Y., Zhang, C., Zheng, Y., Xie, X., Wang, W., Huang, Y, *Map-matching for low-sampling-rate GPS trajectories*, Proceedings of 17th ACM SIGSPATIAL International Symposium on Advances in Geographic Information Systems, ACMGIS, 2009.

[13] Chandio, A.A., Tziritas, N., Zhang, F., Xu, C.-Z., *An approach for map-matching strategy of GPS-trajectories based on the locality of road networks*, IOV 2015. LNCS, vol. 9502, pp. 234–246, Springer, 2015.

[14] Yuan, J., Zheng, Y., Zhang, C., Xie, X., Sun, G., *An interactive-voting based map matching algorithm*, Eleventh International Conference on Mobile Data Management, MDM, 2010.

[15] Haibin, S., Jiansheng, T., Chaozhen, H, *A integrated map matching algorithm based on fuzzy theory for vehicle navigation system* vol.1, pp. 916–919, 2006.

[16] G. Hu, J. Shao, Y. Wang, H. T. Shen, and F. Liu, *IF-matching: Towards accurate map-matching with information fusion*, IEEE Trans. Knowl. Data Eng., vol. 29, no. 1, pp. 114/127, 2016.

[17] Y. Yin, R. R. Shah, and R. Zimmermann, *A general feature-based map matching framework with trajectory simplification*, Proc. ACM SIGSPATIAL Int. Workshop Geostreaming, 2016.

[18] Snijders, C., Matzat, U., Reips, *Big Data': Big gaps of knowledge in the field of Internet*, International Journal of Internet Science, 7, 1-5, 2012.

[19] *https://www.ibm.com/blogs/watson-health/the-5-vs-of-big-data/*, 2016.

[20] Rob Pike, Sean Dorward, Robert Griesemer, Sean Quinlan, *Big Data': Big gaps of knowledge in the field of Internet*, Scientific Programming Journal, 277-298, 13, 2005.

[21] Jian Huang, Shaoqing Qiao, Haitao Yu, Jinhui Qie, Chunwei Liu, *Parallel Map Matching on Massive Vehicle GPS Data Using MapReduce* IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing, 2013.

[22] Jian Huang, Jinhui Qie, Chunwei Liu, Siyang Li, Jingnong Wenga, Weifeng Lv, *Cloud computing-based map-matching for transportation data center*, Electronic Commerce Research and Applications, 2015.

[23] Jian Huang, Chunwei Liu, Jinhui Qie, *Developing Map Matching Algorithm for Transportation Data Center*, Ninth International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 2014.

[24] Hongyu Wang, Jin Li, Zhenshan Hou, Ruochen Fang, Wenbo Mei, Jian Huang, *Research on parallelized real-time map matching algorithm for massive GPS data*, Cluster Comput, 2017.

[25] Antonio M. R. Almeida, Maria I. V. Lima, Jose A. F. Macedo, Javam C. Machado, *DMM: A Distributed Map-matching algorithm using the MapReduce Paradigm*, IEEE 19th International Conference on Intelligent Transportation Systems, 2016.

[26] Wonhee Cho, Eunmi Choi, *A basis of spatial big data analysis with map-matching system*, Cluster Comput, 2017.

[27] Jia Yu, Zongsi Zhang, Mohamed Sarwat, *Spatial data management in apache spark: the GeoSpark perspective and beyond*, Geoinformatica, 2018.

[28] Ashworth M, *Information technology – database languages – sql multimedia and application packages – part 3: Spatial*, Geneva, Switzerland, 2016.

[29] Paul Newson, John Krumm, *Hidden Markov Map Matching Through Noise and Sparseness*, 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2009.

[30] Hannes Koller, Peter Widhalm, Melitta Dragaschnig, Anita Graser, *Fast Hidden Markov Model Map-Matching for Sparse and Noisy Trajectories*, IEEE 18th International Conference on Intelligent Transportation Systems, 2015.

[31] George R. Jagadeesh, Thambipillai Srikanthan, *Online Map-Matching of Noisy and Sparse Location Data With Hidden Markov and Route Choice Models*, IEEE Transaction on Intelligent Transportation Systems, VOL. 18, NO. 9, 2017.

[32] An Luo, Shenghua Chen, Bin Xv, *Enhanced Map-Matching Algorithm with a Hidden Markov Model for Mobile Phone Positioning*, International Journal of Geo-Information, 2017.

[33] Castro P, Zhang D, Chen C, Li S, Pan G, *From taxi GPS traces to social and community dynamics: A survey*, ACM Computing Surveys, 2013.

[34] Gaito S., Rossi G., Zignami M., *From mobility data to social attitudes: A complex network approach*, International Workshop on Finding Patterns of Human Behaviors in Networks and Mobility Data, 2011.

[35] Carneiro C., Alp A., Macedo J., Spaccapietra S., *Advanced data mining method for discovering regions and trajectories of moving objects: "ciconia ciconia" scenario*, AGILE Conference, 2008.

[36] *http://blog.newtechways.com/2017/10/apache-hadoop-ecosystem.html*

[37] Grzegorz Malewicz, Matthew H. Austern, Aart J. C.Bik, James C. Dehnert, Ilan Horn,Naty Leiser, and Grzegorz Czajkowski, *Pregel: A System for Large-Scale Graph Processing*, SIGMOD, 2010.

[38] Spyros Sioutas, Phivos Mylonas, Alexandros Panaretos, Panagiotis Gerolymatos, Dimitrios Vogiatzis, Eleftherios Karavaras,Thomas Spitieris, Andreas Kanavos, *Survey of machine learning algorithms on Spark over DHT-based Structures*, International Workshop of Algorithmic Aspects of Cloud Computing, 2016.

[39] Na Ta, Jiuqi Wang, and Guoliang Li, *Map Matching Algorithms: An Experimental Evaluation*, Joint International Conference on Web and Big Data, 2018.

[40] Lei Zhu, Jacob Holden, Jeffrey Gonder, *A Trajectory Segmentation Map-Matching Approach for Large-Scale, High-Resolution GPS Data*, Transportation Research Board 96th Annual Meeting, 2017.

[41] Qi An, Zhiyong Feng, Shinzhan Chen, Keman Huang, *A Green Self-Adaptive Approach for Online Map Matching*, IEEE Access, 2018.

[42] Douglas Alves Peixoto, Hung Quoc Viet Nguyen, Bolong Zheng, Xiaofang Zhou, *A framework for parallel map-matching at scale using Spark*, Distributed and Parallel Databases, pages 1– 24, 2018.

[43] Paul M. Baggenstoss, *A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces*, IEEE International Conference, 2000.

[44] Byung-Jun Yoon, *Hidden Markov Models and their Applications in Biological Sequence Analysis*, Curr Genomics, 2009.

[45] Qiang Zhang, Dapeng Man, Wu Yang, *Using HMM for Intent Recognition in Cyber Security Situation Awareness*, Knowledge Acquisition and Modeling, International Symposium, 2009.

[46] G. David Forney, *The Viterbi Algorithm*, Proceedings of the IEEE, 1973.

[47] L.R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, IEEE Magazine, 1989.

[48] Michel Bierlaire, Jingmin Chen, Jeffrey Newman, *A Probabilistic Map Matching Method for Smartphone GPS data*, Transportation Research Part C Emerging Technologies, 2013.