ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

CLUSTERING DI TRAIETTORIE SU PIATTAFORMA BIG DATA

Tesi in
DATA MINING

Relatore
Prof. MATTEO GOLFARELLI

Co-relatore
Dott. MATTEO FRANCIA

Presentata da
MATTIA ORIANI

Terza Sessione di Laurea
Anno Accademico 2017 – 2018

PAROLE CHIAVE

trajectory clustering

trajectory mining

big data

spark

Alla mia famiglia

Indice

Sommario

Nell'ambito di analisi dei dati di traiettoria, le tecniche di clustering sono utilizzate con diversi obiettivi, dalla scoperta di strade ad alta percorrenza, predizione di destinazioni, fino allo studio del movimento. A seconda della tipologia, gli algoritmi di clustering si suddividono in: algoritmi basati sulla densità, algoritmi basati sul flusso, algoritmi basati sulla distanza.

Tra gli algoritmi di clustering basati su flussi di traiettorie, NEAT è tra i più recenti ed è sequenziale, tiene conto dei vincoli della rete stradale, della prossimità fra le strade e del flusso di movimento fra porzioni di strada consecutive, con l'obiettivo di individuare flussi di traffico ad alta densità, e significativi in termini di continuità di movimento, all'interno di una rete stradale.

Il contributo di questa tesi riguarda l'estensione in ambito big data di NEAT. L'estensione prodotta aggiunge la possibilità di eseguire clustering a diversi livelli di granularità, oltre ad eliminare il vincolo di dover conoscere la rete stradale a cui si riferiscono i dati e dover eseguire fasi di mapping delle traiettorie sulle strade. La struttura dell'algoritmo originario è stata modificata per ottenere un'implementazione dell'algoritmo e dell'estensione prodotta su una piattaforma big data, Apache Spark, al fine di eseguire clustering di traiettorie su un dataset formato da circa 295 milioni di ping gps.

Introduzione

L'esponenziale diffusione di dispositivi mobili, sensori di movimento e tecnologie di localizzazione, ha portato alla generazione di grandi quantità di dati di traiettorie, i quali rappresentano i percorsi e gli spostamenti di oggetti in movimento in una certa porzione di spazio. La generazione sempre più massiccia di questo tipo di dato, ha fatto emergere la necessità di estrarre informazioni e conoscenze utili oltre al bisogno di dover classificare i dati ed effettuare statistiche su di essi. In questa ottica sono stati sviluppati negli ultimi anni diversi algoritmi in grado di identificare percorsi comuni o movimenti tipici, effettuati da persone, veicoli o fenomeni naturali. Ci sono diverse tecniche e applicazioni nell'ambito di trajectory data mining le quali affrontano questa necessità, considerando che tali applicazioni possono portare significativi aiuti nel miglioramento della qualità della vita o nella scelta di decisioni in ambito industriale o governativo. Fra gli scenari reali in cui queste applicazioni vengono utilizzate troviamo ad esempio la scoperta di percorsi effettuati da oggetti in movimento, predizione di una destinazione o analisi del movimento comune.

In particolare, una delle tecniche utilizzate per l'analisi di questo tipo di dato, riguarda il clustering di dati di traiettoria, il cui obiettivo è creare gruppi di traiettorie simili che siano identificativi di un movimento frequente o di una zona stradale altamente percorsa. Le metodologie di clustering sono diverse e basate su concetti differenti nella formazione dei cluster, ad esempio metodi basati su densità, su densità e flusso, o sulla distanza fra le traiettorie. Le tipologie di clustering di traiettorie possono essere distinte inoltre in due categorie: clustering di intere traiettorie e clustering di sotto-traiettorie, nel secondo caso le traiettorie vengono segmentate al fine di trovare porzioni di strada comuni

o un movimento tipico fra diverse persone, mentre nel primo caso si ricercano oggetti in movimento che abbiano un percorso complessivo similare.

In questa tesi, il lavoro effettuato può essere distinto in due macro fasi, dopo uno studio del contesto riguardante il clustering di traiettorie e un'analisi dello stato dell'arte al fine di analizzare i possibili approcci presenti in letteratura, è stata effettuata la progettazione di un'estensione ad un algoritmo e l'implementazione della stessa in ottica big data. L'obiettivo del processo di clustering implementato è l'identificazione di flussi di traffico che siano significativi dei maggiori percorsi utilizzati dai veicoli, e partendo da un algoritmo di clustering basato su densità e flusso, è stata aggiunta anche la possibilità di effettuare clustering a diversi livelli di granularità, introducendo il concetto di griglia, formata da un insieme di celle, all'interno della quale mappare le traiettorie. L'implementazione è stata progettata per permettere l'esecuzione dell'algoritmo in ottica big data, al fine di ottimizzare il numero di operazioni distribuite e la quantità di dati elaborata.

Per quanto riguarda la struttura del documento, nel capitolo 1 viene descritto il problema affrontato e il tipo di dato utilizzato, unitamente agli scenari e ai casi reali in cui è possibile applicare le tecniche di analisi relative ai dati di traiettoria, mentre nel capitolo 2 è effettuata una descrizione delle tecnologie utilizzate in ambito big data. Nel capitolo 3 sono descritti gli algoritmi di clustering sia in ambito big data che non, presenti allo stato dell'arte attuale, unitamente ai confronti fra le diverse caratteristiche degli approcci analizzati. L'algoritmo utilizzato come base di partenza è descritto nel dettaglio nel capitolo 4, mentre nel capitolo 5 viene presentata l'estensione prodotta e la conseguente implementazione effettuata. Nel capitolo 6 è descritto il dataset su cui è stato testato l'algoritmo, e di conseguenza sono discussi i risultati ottenuti, confrontando diversi esempi e tempi d'esecuzione.

Capitolo 1

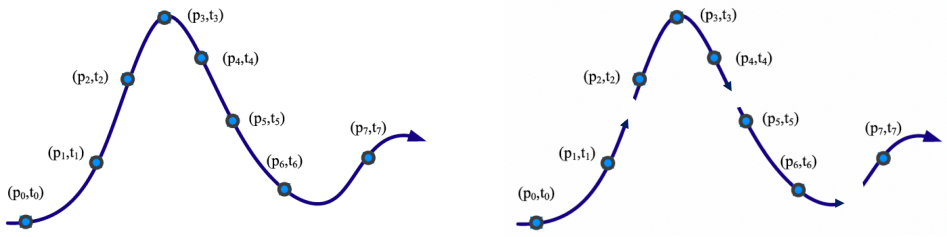
Mining di traiettorie

In questo capitolo viene descritto l'argomento affrontato nel progetto di tesi dando una visione generale sul problema. Verrà definito, motivata la sua importanza e saranno introdotte le realtà applicative in cui viene utilizzato.

Nella prima sezione viene fatta una panoramica del tipo di dato in questione, in cui si può trovare sia la definizione che la formalizzazione, unitamente al contesto a cui si riferisce e perché risulta essere importante. Nella sezione successiva invece si può trovare una panoramica sul clustering di traiettorie, in cui oltre a definire il problema, vengono analizzati i motivi per cui si usano queste tecniche sui dati di traiettoria, ed è stata scritta una breve discussione sugli obiettivi e le applicazioni in casi reali di tecniche di clustering su dati di traiettoria.

1.1 Dati di traiettoria

Negli ultimi anni è diventato facile generare dati di traiettoria relativi ad oggetti in movimento. Questo perchè sono disponibili sul mercato diverse tecnologie che generano dati relativi alla posizione (es. GPS (Global Positioning System), RFID) generalmente montati su smartphone, sensori, o rilevabili da post geo-taggati sui social network o beacon.



(a) Esempio di traiettoria (b) Esempio di sottotraiettoria

Figura 1.1: Traiettoria completa, e segmentazione in sotto traiettorie

L'insieme delle posizioni rilevate da un dispositivo per un certo utente, porta alla formazione di una traiettoria, la quale è una sequenza ordinata nel tempo di posizioni rilevate dai dispositivi.

Una traiettoria può essere formalizzata con la seguente notazione:

$$T = \{(p_0, t_0)(p_1, t_1)...(p_N, t_N)\}$$

per ogni rilevamento di una posizione p_j viene registrata anche una marca temporale t_j, la quale permette così di ordinare le varie posizioni nella sequenza complessiva $t_0 < t_1 < \ldots < t_N$, vedi figura 1.1a.

Le traiettorie possono essere rappresentate in diversi formati, i quali sono dipendenti dal contesto applicativo, dall'obiettivo e dal risultato che si vuole ottenere dall'analisi effettuata su di esse. Oltre ad essere considerate nel loro insieme, le traiettorie possono essere rappresentate come punti, segmenti oppure come sotto-traiettorie identificative di una certa porzione di spazio o di un insieme di posizioni incluse all'interno della traiettoria complessiva. Quando si parla di sotto-traiettoria o sub-trajectory, come in figura 1.1b, formalmente si intende una sotto sequenza di una traiettoria T, cioè una parte di punti consecutivi all'interno della traiettoria.

$$SubT = \{(p_{m_1}, t_{m_1})(p_{m_2}, t_{m_2})...(p_{m_k}, t_{m_k})\}, dove\ 1 \leq m_1 \leq m_2 \leq ... \leq m_k \leq N$$

Dato che i dati di traiettoria possono essere collezionati da diversi dispositivi, risulteranno essere di diversi formati e relativi a differenti oggetti in movimento, i quali possono essere di diverse fonti e categorie [10]:

1. **Persone**

   Le traiettorie in questo caso si riferiscono a persone che hanno registrato i loro movimenti nel mondo reale sotto forma di tracce GPS, e questo può avvenire sia in modalità attiva che passiva per un periodo arbitrario di tempo più o meno lungo.

   (a) **Registrazione attiva**

   I dati di questo tipo di fonte, riguardano persone che abilitano spontaneamente il servizio di localizzazione sui dispositivi al fine di ottenere tracce GPS e traiettorie, per questo viene definita registrazione attiva. Questa tipologia di dati riguarda generalmente persone in viaggio, che registrano la loro traccia a scopi di memorizzazione del viaggio stesso, oppure ciclisti o podisti che registrano le loro attività per successive analisi sportive e confronti con prestazioni pregresse. Su alcuni social network inoltre è possibile ricreare un percorso fatto da media geo-taggati in diversi punti.

   (b) **Registrazione passiva**

   È possibile ricostruire una traccia relativa al movimento di una persona senza che l'individuo stesso attivi appositamente il rilevatore GPS sul proprio dispositivo, in questo caso si parla di registrazione passiva. Uno dei casi in cui è possibile effettuare tale registrazione del percorso è data dalla sequenza di celle telefoniche agganciate dai dispositivi mobili con il corrispondente tempo in cui si è transitati nella zona associata, oppure quando si usa una carta di credito è possibile collegare le diverse posizioni in cui si è stati registrati durante un'altra azione.

2. **Veicoli**

   Una moltitudine di veicoli di trasporto sono al giorno d'oggi dotati di sistemi GPS che ne tracciano i percorsi o li identificano durante i loro viaggi, fra i quali ad esempio taxi, autobus, navi e aerei. I report relativi alle posizioni, unite al tempo, registrate dai dispositivi formano diverse quantità di dati di traiettoria che possono essere usate per diversi scopi

quali allocazione migliore di risorse, analisi del traffico o il miglioramenti dei trasporti.

3. **Movimenti di animali**
   Il comportamento degli animali negli ultimi anni è stato largamente studiato dai biologi e zoologi per capire il comportamento e gli spostamenti effettuati da gruppi di animali nelle diverse aree, a tale scopo diversi dispositivi GPS sono stati installati o dispositivi di prossimità per ricostruire traiettorie e ad analizzarne in seguito i movimenti.

4. **Mobilità di fenomeni naturali**
   Lo stesso principio base è applicato a fenomeni naturali, quali uragani, trombe d'aria e correnti oceaniche. Le traiettorie relative a questi fenomeni ne catturano il movimento e permettono di poter eseguire analisi approfondite sui cambiamenti e i comportamenti usuali di tali fenomeni.

1.1.1 Pulizia dei dati

I dati di traiettoria sono un tipo di dato molto variabile e eterogeneo, sia per loro definizione (ad esempio traiettorie di lunghezza diversa e/o campionate a intervalli di tempo variabili), sia per la diversa natura dei dispositivi che forniscono questo tipo di dato. Dato il diverso numero di fonti e la vasta quantità di dispositivi in grado di rilevare delle traiettorie, quando si eseguono applicazioni di data mining su questo tipo di dato, può essere utile eseguire una fase di pre-processing, cioè un'elaborazione del dato precedente l'analisi, al fine di migliorare la sua struttura e/o eliminare dati considerati poco utili in base al contesto in cui si sta operando. Fra le principali tecniche di pre-processing del dato di traiettoria troviamo:

- **Filtraggio dei punti di rumore**
  I dati raccolti non sono sempre perfetti, ma contengono spesso punti spuri o di rumore che possono influire sulle fasi successive di estrazione della conoscenza. Gli errori possono essere dovuti a diverse cause, riconducibili ai sensori o ad altri fattori che incidono sulla rilevazione della

posizione, e nel caso in cui l'errore non sia esagerato è possibile utilizzare algoritmi di map-matching, in altri casi invece può essere un problema derivare informazioni aggiuntive come velocità di una traiettoria, quindi la necessità di filtrare tali punti risulta fondamentale.

- **Segmentazione della traiettoria**
  Considerare le traiettorie nel loro insieme a volte può non essere utile e non portare a risultati concreti, infatti in alcuni scenari è preferibile dividere intere traiettorie in segmenti più piccoli in base a diversi criteri, cioè in sotto-traiettorie. Tali processi saranno descritti in seguito più nei dettagli in quanto molto utilizzati prima delle fasi di clustering.

- **Identificazione degli stay-points**
  Considerando come stay point, una zona o una località d'interesse (ad esempio: ristoranti, luoghi ricreativi, etc.), possono essere effettuate analisi in funzione di tali zone o di certe località. Una traiettoria viene trasformata in una sequenza di stay point, a rappresentare il movimento fra i luoghi visitati per compiere analisi su di essi.

- **Map-matching**
  In questa categoria rientrano gli algoritmi che forniscono un'approssimazione dei punti e un conseguente matching dei punti di traiettoria su una mappa. Consiste nel processo non banale di convertire sequenze di coordinate in sequenze di segmenti stradali. Portando il vantaggio di ottenere traiettorie che corrispondano alle strade senza avere punti esterni ad esse.

1.2 Clustering di traiettorie

I dati di traiettoria, la maggior parte delle volte contengono una grande quantità utile di informazione, sfruttabile in diversi contesti e applicazioni, questo indica che c'è il bisogno di analizzare ed estrarre il contenuto informativo di tale tipologia di dati.

In relazione a questi tipi e fonti di dato, esistono diverse applicazioni le quali possono dare un serio miglioramento e dei benefici alle persone comuni o alle industrie. Gli obiettivi principali che si pongono sono la scoperta di percorsi (nel caso di persone o veicoli) [4, 7], la predizione di una destinazione, oltre allo studio e l'analisi del comportamento, sia di persone [9], veicoli [8, 13] e fenomeni naturali [1, 2, 15]. Un settore della ricerca nell'ambito data mining, riguarda il trajectory clustering, il quale applica tecniche di clustering su dati GPS rappresentanti delle traiettorie.

L'obiettivo del clustering è trovare caratteristiche simili all'interno di un dataset, termine utilizzato per descrivere l'insieme dei dati disponibili. I dati vengono raggruppati per somiglianza, in base ad attributi comuni, formando dei gruppi contenenti dati il più possibile omogenei fra loro, massimizzando la similarità tra elementi nello stesso cluster e minimizzando la similarità tra cluster diversi. Le caratteristiche e il risultato di queste tecniche fanno si che il clustering venga utilizzato in svariate applicazioni negli ambiti di data analysis, pattern recognition, ricerche di marketing. Applicando queste teorie ai dati di traiettoria, l'obiettivo diventa quello di formare gruppi di traiettorie con caratteristiche simili.

Nelle sezioni seguenti vengono specificati i dettagli relativi ai diversi obiettivi che può avere il clustering di traiettorie, unitamente ad una panoramica sugli utilizzi reali di queste tecniche. Di seguito verranno descritte le diverse modalità e tecniche utilizzate per raggiungere gli obiettivi prefissati.

1.2.1 Obiettivi e utilizzi reali

Le applicazioni relative al clustering di dati di traiettoria, allo stato attuale dell'arte pongono l'attenzione su diversi obiettivi e realtà applicative. I cluster risultanti possono aiutare a fornire conoscenza aggiuntiva riferita al tipo di oggetto a cui si riferiscono le traiettorie.

L'obiettivo comune quando si fa clustering di traiettorie è quello di estrarre le informazioni contenute nelle traiettorie stesse e riorganizzarle in base alla motivazione applicativa per cui si sta compiendo l'analisi. Le applicazioni che

ne derivano possono riguardare svariati aspetti, da quelli riguardanti il traffico e le strade percorse da persone o veicoli, ad esempio il monitoraggio del traffico, a quelli riguardanti la pianificazione dei trasporti o l'identificazione di punti strategici. In questo caso l'obiettivo è ricavare dei gruppi di traiettorie in base a porzioni delle stesse traiettorie in comune fra loro, oppure in base alle zone attraversate o alle strade percorse: questo può portare alcuni benefici riguardo la gestione ed eventuali modifiche della viabilità, avendo come risultato del clustering l'insieme delle strade più percorse o più trafficate.

Può essere studiato anche lo spostamento delle persone all'interno di una città in diversi momenti avendo come indicazioni del clustering un insieme di percorsi comuni ad un certo gruppo di persone. Un esempio di applicazione reale di questa categoria riguarda la pianificazione del trasporto pubblico: è possibile eseguire un'ottimizzazione delle linee di autobus e mezzi urbani o della pianificazione dei luoghi in cui si trovano stazioni/terminali sapendo quali sono le strade ad alta densità di traffico o le zone più dense i cui si muovono le persone, aiutando così le autorità nel loro processo di decisione, sia per quanto riguarda il trasporto pubblico, sia per la gestione del traffico e l'evitare ingorghi cittadini. Un caso reale riguarda la ricerca dei passeggeri per i taxi: l'obiettivo è quello di scoprire la mobilità dei passeggeri e il comportamento dei taxi riguardo il carico e scarico delle persone. In base alle conoscenze estratte dai dati l'applicativo può suggerire velocemente agli autisti di taxi dove possono trovare i passeggeri.

Allo stesso modo, il conoscere le strade con flussi molto alti di veicoli o persone può aiutare a collocare in punti strategici pubblicità o negozi, quello che è chiamato "location-based advertising", in cui i punti vendita possono avere informazioni sul dove posizionare pubblicità o sconti tenendo conto delle strade più trafficate verso i propri negozi o delle strade principali a maggior percorrenza per avere più visibilità possibile.

I dati GPS, oltre a riferirsi a veicoli stradali o persone in movimento, possono riguardare anche imbarcazioni e navi commerciali: l'obiettivo in questo caso

è quello di formare dei cluster di traiettorie riferite alle linee di navigazione vicine alle operazioni portuali, in modo da migliorarne gli aspetti di sicurezza nelle fasi di entrata in zona portuale.

La ricerca e lo studio di pattern, cioè movimenti tipici, schemi ricorrenti, modelli, può essere vista anche come uno studio del comportamento di una certa categoria di persone, animali, o fenomeni naturali, e allo stesso modo il risultato del clustering può fornire una base di partenza per la predizione delle destinazioni di un certo oggetto in movimento, che sia questo un veicolo o un fenomeno ambientale come un uragano, o un fenomeno migratorio di animali, portando importanti benefici anche nella vita quotidiana. In questo ultimo caso, gli zoologi stanno studiando gli impatti a diversi livelli del traffico veicolare sui movimenti e sulla distribuzione sul territorio degli animali [2, 14].

Allo stesso modo, come accennato in precedenza, la ricerca ha riguardato anche fenomeni naturali, come uragani: i metereologi cercano di migliorare le loro previsioni sul tempo e il movimento degli uragani. Il processo di clustering delle traiettorie degli uragani aiuta ad identificare i comportamenti vicino alle coste o in mare prima dell'impatto, in modo da aiutare la previsione di dove avverrà l'impatto dell'uragano, risultando così un aiuto che può rivelarsi fondamentale in certi approcci decisionali [2, 15].

Le informazioni relative agli spostamenti o comportamenti comuni possono essere utilizzate anche per lo scopo opposto, cioè per identificare comportamenti anomali che non rispecchiano le caratteristiche usuali identificate dal processo di clustering.

1.2.2 Modalità di clustering

Il clustering di traiettorie si può dividere in due grandi macro-aree indipentemente dalla tipologia di algoritmo che verrà utilizzato per il clustering stesso.

- Clustering di traiettorie complete [1, 5]

- Clustering di sub-trajectories [2, 6, 7, 8, 9]

Il clustering di traiettorie complete individua esattamente le persone che fanno lo stesso percorso: le traiettorie vengono considerate nella loro interezza, e dato che queste sono eterogenee, i cluster risultanti potrebbero essere frammentati. Può essere utile quindi partizionare la traiettoria in sotto sequenze, per individuare dei cluster di sotto traiettorie, con l'obiettivo di scoprire il comportamento o il percorso comune fra gli oggetti a cui si riferiscono i dati. D'altro canto, considerare le traiettorie nella loro interezza permette di non perdere l'informazione complessiva sullo spostamento di ogni singolo oggetto, cosa che viene persa nel clustering di sotto-traiettorie. Nel caso in cui si voglia scoprire il percorso comune ad un certo gruppo di persone è utile dividere le traiettorie in sotto-traiettorie, nel caso in cui si voglia predirre il punto di arrivo o capire quali sono i percorsi effettuati da una zona all'altra è bene considerare la traiettoria nella sua interezza senza dividerla in segmenti indipendenti tra loro. La scelta di fare clustering di traiettorie complete o di sotto-traiettorie risulta quindi strettamente dipendente dall'obiettivo finale che si vuole ottenere e dal contesto applicativo.

L'eventuale segmentazione delle traiettorie può essere considerata parte del processo di pre-processing dei dati prima della fase di clustering, la quale può includere anche altre operazioni di pulizia dei dati, standardizzazione e map-matching. Per quanto riguarda le modalità di partizionamento delle traiettorie, possono essere effettuate in diverse modalità: in base alla rete stradale, tenendo conto del cambiamento di direzione, etc.

Per il clustering delle traiettorie, possono essere utilizzati anche altri parametri significativi, utili a confrontare fra loro le traiettorie per assegnarle ad un cluster piuttosto che ad un altro, oltre a coordinate spaziali e indicazioni sul tempo, si tiene soprattutto conto di velocità e direzione.

Le modalità di clustering di traiettorie le possiamo essenzialmente dividere in tre categorie:

- **Basato su un modello (Model based)**

  Le metodologie di clustering in questa categoria, modellano la traiettoria,

che sia intera o segmentata, per poi trovare un set di parametri volti a rappresentare i diversi clusters [5]. Per esempio è stato utilizzato un modello di regressione per clusterizzare le traiettorie di un ciclone [1], anche se nei diversi approcci vengono considerate le traiettorie nel loro insieme e non è possibile identificare eventuali porzioni di percorsi simili.

- **Basato sulla distanza (Distance based)**
  Gli algoritmi che fanno clustering basato sulla distanza generalmente utilizzano funzioni che calcolano la distanza fra le diverse traiettorie e poi con algoritmi generici di clustering, come per esempio K-Means [8, 12, 13], raggruppano le traiettorie in cluster.

- **Basato su densità e flusso (Flow and Density based)**
  Le metodologie di clustering basate sulla densità utilizzano una soglia riferita a tale parametro per poi verificarla fra le traiettorie al fine di distinguere i dati rilevanti dai dati di rumore. Risultano i più adatti a scoprire cluster di forme arbitrarie e filtrare i dati considerati come rumore. Oltre a considerare la densità fra le traiettorie, alcuni algoritmi utilizzano anche un concetto di flusso, più specificamente tendono a formare cluster unendo fra loro i segmenti o le traiettorie che rispettino una certa soglia di densità e abbiano un concetto di flusso di traffico continuo, al fine di creare unione di traiettorie o di segmenti che siano indicativi dei flussi di traffico.

Capitolo 2

Tecnologie utilizzate

Sono descritte in questo capitolo le tecnologie utilizzate durante la fase progettuale, con un particolare focus riguardo i big data, Hadoop e Spark.

2.1 Big Data

Quando si parla di big data, si utilizza questo termine per descrivere elevate quantità di dati, i quali sono così voluminosi e complessi che rendono le normali applicazioni di processing dei dati inadeguate, e allo stesso modo ci si riferisce con tale termine ai dataset la cui grandezza va oltre le tipiche capacità di un software di database che sia in grado di analizzare, gestire e salvare quantità enormi di dati, in tempi tollerabili per gli utenti.

Per quanto riguarda il volume, basta pensare che il 90% del volume dei dati mondiali è stato generato negli ultimi due anni [22]. Tale dato conferma che l'avvento di numerosi dispositivi informatici, non solo a livello di smartphone, ma anche di sensori in qualsiasi settore, dati satellitari, macchine informatizzate, la crescita delle smart city, e l'avvento dell'industria 4.0 con dispositivi IoT, hanno avuto come conseguenza la generazione di una quantità esponenziale di informazioni.

Possiamo definire un sistema "big data" quando aumenta il volume dei dati, e di conseguenza la velocità e il flusso di informazioni, che il sistema deve acquisire e gestire al secondo.

Per formalizzare il termine big data possiamo definirlo in termini di:

- **Volume**
  Non esiste una dimensione di riferimento, in quanto le macchine sono sempre più performanti e allo stesso tempo i dataset sempre più grandi. La mole di dati generati e considerabili big data va dai Terabyte (10^{12}), per arrivare nell'ordine di Zettabyte (10^{21}). La quantità di dati presente nel mondo, prevista circa nel 2020, è di 35 Zettabytes, cioè nell'ordine di miliardi di Terabyte.

- **Velocità**
  Riguarda la velocità con cui i dati vengono generati e la conseguente capacità di processare i dati in tempi ragionevoli per estrarre le informazioni ed ottenerne un vantaggio da esse. La raccolta e l'analisi possono avvenire a cadenze periodiche, in caso di applicazioni batch, oppure in tempo semi-reale nel caso di streaming in cui si processano dati in modo continuo a diverse finestre temporali, oppure totalmente in tempo reale in cui i dati vengono processati immediatamente per portare rapide conseguenze nell'applicativo.

- **Varietà**
  I dati prodotti sono generalmente dei più disparati formati, quindi possono essere già strutturati, semi-strutturati o senza struttura, e riguardare sia testo che immagini piuttosto che audio o video. Prima dell'avvento dei big data si teneva conto principalmente di dati strutturati, ma la diversità di dispositivi che li generano e le diverse fonti possibili portano ad avere una varietà ampia di dati da analizzare.

- **Veridicità**
  La qualità dei dati può variare enormemente e, se alla base i dati sono poco accurati, i risultati delle analisi possono essere incompleti o inconsistenti, è utile quindi data la varietà di sorgenti e la velocità alla quale i dati vengono prodotti che si riesca ad assegnare un indice di veridicità per distinguere dati con maggior qualità ed ottenere risultati migliori.

- **Valore**
  Quando si parla di valore, si indica la capacità dei big data e delle analisi che ne conseguono di generare un vantaggio competitivo piuttosto alto, traendo conoscenze intrinseche e approfondite che possono portare ad informazioni utili al business nel quale è utilizzato il sistema big data.

L'avvento dei big data ha portato a pensare nuove tecniche e modalità di analisi dei dati, e considerati i diversi formati e le disparate fonti da cui vengono generati, possiamo distinguere le azioni che vengono eseguite sui big data in due diverse categorie:

1. **Elaborazione e pre-processing**
   In cui i dati vengono collezionati e si eseguono operazioni di trasformazione pulizia e integrazione dei dati (incluse nelle fasi di ETL sui dati)

2. **Analytics**
   In cui sono compresi i processi e le azioni che hanno l'obiettivo di interpretare ed estrarre conoscenza dai dati. E' in questa categoria di processi che troviamo anche le applicazioni di tecniche di clustering sui big data.

Cambiamenti architetturali Gli ambienti di programmazione e calcolo pensati per interagire con i big data, basano la loro architettura su un insieme di cluster di computer, i quali sono connessi fra loro, e lavorano tutti insieme per ottenere il risultato finale. Si utilizza questo modello per ottenere scalabilità orizzontale, piuttosto che ottenere parallelismo utilizzando un supercomputer. Un'altra caratteristica oltre ad avere architettura distribuita, è quella di avere tolleranza ai guasti, di conseguenza le risorse sono replicate su diverse macchine all'interno del cluster, per ottenere una resistenza ai guasti e minimizzare la perdita di dati nel caso si presentino inconvenienti. Come ultimo cambiamento importante, il calcolo è diventato totalmente distribuito, in modo da riuscire a trarre vantaggio dalla potenza dell'insieme di computer che compongono il cluster.

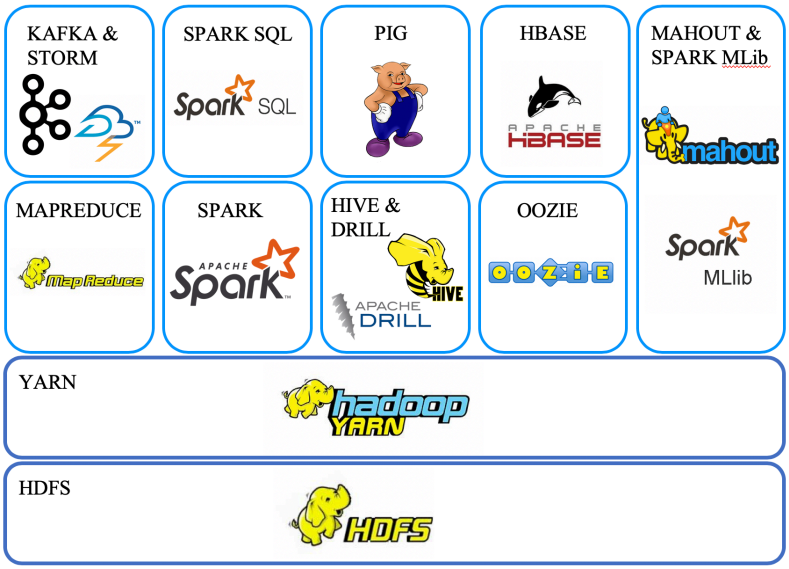


Figura 2.1: Componenti Hadoop

2.2 Hadoop

Hadoop è un framework che permette il calcolo distribuito di grandi dataset attraverso cluster di computer utilizzando semplici modelli di programmazione. La progettazione di Hadoop ha come focus quello di scalare da un singolo server a milioni di macchine, ognuna delle quali mette a disposizione computazione e spazio di archiviazione. Uno dei punti di forza di Hadoop riguarda il fatto che invece di far affidamento sull'hardware, per quanto riguarda l'individuazione e la gestione dei guasti, è stato progettato per spostare tali fasi a livello applicativo, in particolare salvando i file in modo ridondante e dividendo la computazione in più parti.

Il framework Hadoop include diversi moduli e progetti collegati ed esso, come visibile in figura 2.1. Fra i quali citiamo HDFS, YARN, MapReduce, Hive e Spark. Nelle sezioni seguenti verranno approfonditi i moduli appena citati, spiegandone le caratteristiche e le funzioni di ognuno.

2.2.1 HDFS

L'acronimo di Hadoop Distributed File System è riferito al file system distribuito di Hadoop, il quale è stato progettato per salvare file di grandi dimensioni, ed è in grado di essere eseguito su cluster formati da commodity hardware comprendenti macchine con prestazioni comuni senza la necessità di super computer a livello di costo e caratteristiche tecniche.

Le dimensioni tipiche di un file in HDFS sono dell'ordine di gigabytes e terabytes, e ci sono dei cluster Hadoop in funzione in grado di mantenere petabytes di dati. I punti di forza principali di HDFS riguardano principalmente:

- **Guasti Hardware**
  I guasti a livello hardware sono considerati come un avvenimento comune, piuttosto che un'eccezione. HDFS è progettato per essere formato da un vasto numero di macchine ognuna delle quali mantiene parti dei dati del file system, di conseguenza, l'identificazione di guasti e il loro ripristino in modo veloce ed automatico è una cosa fondamentale a livello architetturale per HDFS.

- **Accesso ai dati**
  Oltre al fatto di poter gestire grandi dataset come già citato, HDFS è progettato principalmente per processi batch invece che per utilizzi interattivi, e l'enfasi è posta sull'avere un alto throughput ai dati delle applicazioni invece di avere bassa latenza nell'accesso ai dati.

- **Gestione della computazione**
  Lo slogan che spiega questo punto di forza è "Spostare la computazione costa di meno rispetto a muovere i dati", dall'assunzione per cui a volte sia meglio spostare la computazione più vicino a dove sono salvati i dati piuttosto che spostare i dati nel punto in cui viene eseguita la computazione, riducendo così le congestioni a livello di rete e aumentando il throughput complessivo del sistema. Con un architettura di cluster computing, questa affermazione va tenuta in considerazione ancora di più avendo dataset di grandi dimensioni, e HDFS fornisce delle interfac-

ce per mettere in pratica queste assunzioni, note come principio di data locality.

NameNodes e DataNodes

I nodi all'interno di HDFS operano seguendo il pattern master-slave. Il master, chiamato NameNode, il quale si occupa di mantenere in modo persistente l'albero del filesystem e tutti i metadati riferiti ai file e alle cartelle, oltre a mantenere in memoria la posizione di ogni blocco per un dato file, cioè che è chiamato block pool. Il NameNode è inoltre responsabile dell'esecuzione delle richieste di operazioni come apertura, chiusura e cambio di nome di file e cartelle, e regola l'accesso ai file dai client.

I DataNodes, generalmente uno per ogni nodo nel cluster, si occupano dello salvataggio e del recupero (rispettivamente storing e retrieve) dei blocchi, cioè di una porzione di dati, e periodicamente, inviano al NameNode un report con l'elenco dei file di cui hanno il riferimento. I DataNodes si occupano inoltre di creare, cancellare eventuali blocchi sotto istruzione del NameNode.

I nodi sono organizzati in rack, i quali sono a loro volta organizzati in datacenters. Lo schema può essere visto come un albero, in cui i nodi sono le foglie, e il cluster è la radice, in questo modo Hadoop è in grado di calcolare la distanza fra due nodi come la distanza che hanno nell'albero, per organizzare meglio la computazione oltre al principio di data locality.

I due punti cruciali di HDFS riguardano i casi in cui è richiesta bassa latenza, o è presente una grande quantità di file con dimensioni piccole: HDFS potrebbe non essere adatto a gestirli, essendo progettato per big data.

Organizzazione dei dati

All'interno di HDFS, essendo progettato per supportare file di grandi dimensioni, generalmente le applicazioni scrivono i loro dati una volta sola, ma hanno la necessita di leggerli svariate volte. Un file dal momento in cui viene creato, non è più possibile modificarlo. Questa pratica semplifica il dover ga-

rantire coerenza fra i dati e permette di ottenere alto throughput in accesso ai dati.

I dati sono organizzati in blocchi, formalmente un blocco è la minima quantità di dato in scrittura o in lettura, e un file viene ripartizionato in blocchi perchè generalmente le sue dimensioni sono più grandi rispetto a quelle del disco, ed in questo modo, partizionando il file, si riesce a migliorare la sua gestione a livello di storage e si rende più semplice la replicazione dei dati. La dimensione dei blocchi su HDFS varia da 64 MB a 1 GB, e la grandezza di default è di 128MB.

Replicazione dei dati La replicazione dei dati avviene per migliorare le performance ed aumentare la robustezza del sistema. Tenendo conto della struttura dei nodi, ogni blocco viene replicato fra più DataNodes, e il NameNode mantiene la lista dei DataNodes che contengono il blocco stesso. Il fattore di replica di default è 3, il che significa che la prima replica è salvata nel nodo in cui il client ha scritto il comando, la seconda è chiamata anche off-rack, dato che il blocco viene replicato in un nodo all'interno di un rack diverso da quello precedente mentre la terza è all'interno dello stesso rack della seconda, ma in un nodo diverso.

Le repliche possono essere ribilanciate nel caso in cui vengano aggiunti nodi o questi diventino non disponibili. Hadoop riesce a sfruttare la topologia del custer e la replica dei blocchi per applicare il principio di data locality descritto in precedenza, più precisamente la gestione delle risorse viene effettuata seguendo tale schema in ordine di preferenza:

- Processo e dati nello stesso nodo
- Processo e dati all'interno dello stesso rack, in nodi differenti
- Processo e dati all'interno dello stesso datacenter, in rack differenti
- Processo e dati in diversi datacenter

2.2.2 Yarn

Yarn è un framework introdotto nella versione 2.0 di Hadoop, acronimo di Yet Another Resource Negotiator, con lo scopo di inserire una piattaforma che si occupi della gestione delle risorse disponibili nel cluster e dello scheduling delle operazioni da eseguire fra le risorse. Questa piattaforma non è tipicamente usata dal codice delle applicazioni, ma viene utilizzata dai framework di computazione distribuita descritti in seguito come MapReduce e Spark. L'idea alla base di Yarn è quella di separare le funzionalità di gestione delle risorse e quelle di scheduling e monitoring dei job in due daemon, cioè due diversi processi a lunga durata.

- Resource Manager
- Node Manager

Un Resource Manager, da qui in poi denominato RM, è globale, quindi ne è presente uno all'interno del cluster, ed ha la funzione di gestire e arbitrare le risorse fra le diverse applicazioni. Il Node Manager, da qui in poi NM, ha la responsabilità dei containers, monitora il loro utilizzo delle risorse e riporta i dati al RM. Un container è un'entità astratta, la quale viene usata per l'esecuzione su un processo specifico di un'applicazione con un dato set di risorse.

Per quanto riguarda la composizione del RM, esso è formato da due componenti principali: lo Scheduler e l'Application Manager. Lo Scheduler è il responsabile dell'allocazione delle risorse alle varie applicazioni in esecuzione, mentre l'Application Manager è il responsabile dell'accettazione dei job sottomessi e fornisce anche il servizio per eventualmente farlo ripartire nel caso di fallimento.

Scheduling Policies

L'allocazione e la gestione delle risorse può essere effettuata fra le diverse configurazioni di scheduling messe a disposizione da YARN. Fra le quali troviamo:

- **FIFO Scheduler**
  Il principio base è first-in-first-out, quindi le risorse vengono assegnate totalmente in base all'arrivo della sottomissione del job. Non necessita di configurazione, ma non è adatto per cluster condivisi.

- **Capacity Scheduler**
  E' la configurazione di default in YARN, lo scheduling avviene in base alla capacità. Viene riservata una quantità di memoria fissa ad ogni job, in modo che all'arrivo di un nuovo job sottomesso questo sia in grado di essere eseguito.

- **Fair Scheduler**
  L'assegnazione delle risorse viene effettuata dinamicamente: all'arrivo di una nuova sottomissione di un job, le risorse vengono ripartire fra il numero di job in esecuzione.

2.2.3 Map Reduce

Map Reduce è un sistema basato su Yarn per scrivere applicazioni che siano in grado di processare grandi quantità di dati in parallelo su un cluster di commodity hardware formato da un alto numero di nodi, ed è ciò su cui si basano la maggior parte dei sistemi di processing big data attuali.

Il problema classico di quando si processa un vasto numero di dati, è quello di iterare su un grande numero di record per estrarre informazioni da ognuno di essi, per poi aggregare i risultati e raggrupparli prima di generare un output finale.

L'idea alla base di Map Reduce, è quella di fornire due astrazioni per questi due tipi di operazioni, le quali prendono il nome di Map e Reduce. Solitamente un programma scritto utilizzando il paradigma Map Reduce, è formato dal codice per la parte di Map, e quello per la parte di Reduce e prende il nome di job Map Reduce. Ogni job viene diviso dal sistema in unità più piccole chiamate task, i quali sono schedulati utilizzando Yarn e messi in esecuzione nei vari nodi del cluster. Map Reduce opera su dati di tipo chiave-valore, ed il

tipo di uno e dell'altro può essere scelto dal programmatore durante la stesura dell'applicazione. Un job tipico di Map Reduce, inizialmente parte prendendo il dataset in input e lo divide in parti indipendenti tra loro, assegnando un task di Map ad ogni parte, i cui task vengono in modo totalmente parallelo, applicando la funzione di Map ad ogni elemento facente parte del dataset iniziale. L'output di ogni fase di Map viene ordinato per la chiave specificata, e salvato in disco locale, non in HDFS, altrimenti la replicazione di tali risultati sarebbe controproducente, ed infine inviato ai nodi in cui il task di Reduce è in esecuzione. I risultati di diversi mapper vengono uniti ed ordinati per chiave prima di applicargli la funzione di Reduce, e tutti i valori id una certa chiave vengono mandati allo stesso Reduce. Questo non implica che un Reduce possa gestire più chiavi seppur una per volta, quindi viene eseguita la funzione specificata per la fase di Reduce per combinare i valori di una data chiave. Il risultato della fase di Reduce, cioè le coppie chiave valore calcolate dalla funzione specificata, viene infine scritto in modo persistente su HDFS.

2.2.4 Hive

Hive è un software di datawarehouse il quale semplifica la scrittura, lettura e gestione di grandi dataset distribuiti, ed è costruito sulla base di Apache Hadoop. Fra le varie funzionalità principali indichiamo la possibilità di interrogare i dati in linguaggio Hive-QL, il quale utilizza la sintassi simil-SQL e l'esecuzione delle query avviene traducendo le query in job Map Reduce, Apache Tez, o job Spark.

Inoltre con Hive c'è la possibilità di poter salvare diversi formati di file, e non solo in formato "Hive", ma anche in CSV, Apache Parquet, Avro, e permette l'accesso ai file salvati direttamente in HDFS.

Oltre a fornire un semplice accesso ai dati via SQL, possono essere eseguiti anche task di tipo datawerhousing riguardo le fasi di ETL come estrazione, trasformazione e caricamento, oltre a reporting e analisi dei dati, definendo anche eventuali funzioni da parte dell'utente.

2.2.5 Spark

Spark è un motore di computazione generico e veloce compatibile con dati Hadoop, il quale fornisce modelli di programmazione semplici, efficaci e performanti, che supportano un'ampia gamma di applicazioni, dall'ETL, al machine learning, fino a calcoli SQL, processing di stream di dati e computazioni di grafi.

È progettato per eseguire sia batch processing, come Map Reduce, ma anche altre tipologie di lavoro come streaming o query interattive. Inoltre è particolarmente adatto per algoritmi di machine learning dato che supporta query iterative e veloci, e le sue primitive in-memory forniscono performance più di cento volte più veloci rispetto a Map Reduce. Spark può essere eseguito su cluster Hadoop attraverso Yarn, e utilizzando diverse fonti di dati, fra le quali Hive.

Limiti principali di MapReduce

Con il passare degli anni, sia a livello hardware che software, oltre che a livello di bisogno e richieste da parte degli utenti finali, ci sono state molti cambiamenti.

A livello hardware principalmente le macchine sono diventate multi-core, mentre prima erano solitamente single-core, oltre al fatto che la memoria RAM è vista come fonte principale di dati, mentre fino a qualche anno prima in questo ruolo era visto il disco.

Allo stesso tempo, a livello software, c'è stato l'avvento della programmazione funzionale e del modello NoSQL, oltre alla varietà e quantità di framework disponibili. Di pari passo allo sviluppo hardware è richiesta un'ottimizzazione dei software che sfrutti a pieno i diversi core presenti in ogni macchina. Unitamente ai miglioramenti di livello software, le maggiori compagnie del settore pongono sempre più attenzione sui big data, e si pone in generale più attenzione sulla velocità di computazione. Come accennato in precedenza è sempre più frequente la richiesta di applicazioni di diverso tipo,

anche in real time e applicate nei più disparati ambiti.

Considerando tali cambiamenti è possibile individuare alcune limitazioni che Map Reduce ha mostrato con l'avanzamento del tempo fra le quali:

- **Poco utilizzo delle nuove potenzialità hardware**
  Nel paradigma Map Reduce si sfrutta molto il disco, e il fatto che la ram sia diventata fonte primaria oltre all'avvento dei multi core, lasciano troppa pressione sul disco senza sfruttare adeguatamente gli altri componenti hardware.

- **Paradigma limitante**
  In Map Reduce, ogni programma deve essere pensato seguendo il paradigma di Map e Reduce, e con la richiesta di ambiti più disparati e algoritmi sempre più complessi questo si rivela un limite.

- **Non adatto per interattività e iteratività**
  Il paradigma Map Reduce è pensato per computazione batch, quindi non si presta al meglio per eventuali processi iterativi, o nel caso di applicazioni data mining interattive.

Caratteristiche di Spark

Oltre alle funzionalità citate in precedenza messe a disposizione da Spark, e considerando i limiti che Map Reduce ha evidenziato con il passare degli anni, possiamo individuare tre caratteristiche di Spark in grado di dargli un vantaggio e renderlo un'ottimo strumento per tale tipologia di computazione.

- **In-memory data caching**: non viene fatta troppa pressione sul disco, il quale viene scansionato una volta sola e i dati vengono scritti e letti dalla RAM

- **Lazy computations**: sul job vengono effettuate delle ottimizzazioni, fino al momento della sua effettiva esecuzione. Nelle sezioni successive con la spiegazione delle tipologie di comandi su Spark, verrà definito con più chiarezza il momento della effettiva esecuzione.

- **Efficient pipelining**: si cerca di evitare la scrittura su disco fino a che risulta possibile.

Astrazioni principali

Le astrazioni principali in Spark sono:

- **RDD**: Resilient Distributed Dataset
- **DAG**: Direct Acyclic Graph

RDD Un RDD è una collezione di dati. All'interno di Spark viene diviso in partizioni e salvato in memoria sui nodi del cluster. Può essere creato effettuando un caricamento di un dataset esterno, il quale può risiedere in diverse sorgenti (es: HDFS, Hive, file system locale, etc.) ed essere di diversi formati (es: file di testo, Avro, SequenceFiles, Parquet).

Le sue caratteristiche principali sono:

- **Immutabilità**
  Dal momento in cui viene creato un RDD non può essere modificato. Da una parte il fatto che sia immutabile rende più semplice la parallelizzazione evitando eventuali race conditions e senza dover aver bisogno di procedure di lock per eseguire le modifiche. D'altro canto occupa un maggior spazio. Quando si vuole eseguire una modifica su un RDD, si deve necessariamente creare un nuovo RDD sui cui salvare il risultato.

- **Lazy evaluation**
  Come accennato nel concetto di lazy computation, un RDD viene ottimizzato prima della sua effettiva esecuzione, ogni trasformazione effettuata su di esso non viene eseguita fino alla richiesta di un'azione. Per avere lazy evaluation è necessario che ci sia immutabilità e assenza di side effect, cioè che non sia possibile modificare una variabile o lo stato di un'oggetto al di fuori del proprio scope. Come vantaggio principale si ottiene quello di migliorare le performance e l'ottimizzazione del job stesso oltre a distribuire meglio il processo.

- **Cachability**
  Si trae un grosso vantaggio nelle performance cachando gli RDD, questo perché l'immutabilità sta a significare che i dati possano essere salvati in cache per un periodo di tempo piuttosto lungo e possano essere ricreati facilmente in caso di guasti.

- **Inferenza di Tipo**
  Il tipo di dato è inferito dal compilatore senza aver bisogno di specificarlo.

Possono essere eseguiti due tipi di operazioni sugli RDD:

- **Trasformazioni**
  Applicano un'operazione su un RDD, costruendone uno nuovo (es: map, reduceByKey, flatMap, filter, etc.)

- **Azioni**
  Calcolano un risultato il quale viene restituito al driver program o salvato in piattaforme esterne come HDFS (es: show, count, collect, saveAsTextFile, etc.)

Di default gli RDD non sono persistenti, questo significa che ogni volta che viene richiesta un'azione, vengono eseguite tutte le trasformazioni precedenti. Può essere comunque salvato l'RDD su RAM, disco o entrambi, anche se solitamente non c'è bisogno di salvare in memoria i risultati se l'RDD è utilizzato una sola volta.

Tutte le trasformazioni che si riferiscono ad un RDD non vengono eseguite fino al momento in cui non è richiesta un'azione, in modo da riuscire ad ottimizzare l'esecuzione nel complesso. Questo significa anche che fino al momento in cui non viene scatenata un'azione, i dati non vengono toccati dalle trasformazioni specificate. Le trasformazioni producono dei metadati e un cambiamento su di essi, i quali tengono conto delle partizioni e delle dipendenze riguardo all'RDD, cioè una lista di RDD precedenti che sono coinvolti nelle computazioni.

L'elenco delle trasformazioni che coinvolgono i vari RDD, produce un grafo, il quale può essere ottimizzato per l'esecuzione effettiva, chiamato lineage graph. Il piano di esecuzione effettivo può essere organizzato e ottimizzato aggregando operazioni diverse nel grafo e sfruttando il principio di data locality. Le dipendenze possono essere distinte in "narrow" e "wide", la prima sta a significare che ogni partizione di un RDD padre è utilizzato al massimo da una partizione dell'RDD figlio, mentre wide significa che ogni partizione di un RDD padre può essere usata più volte da diverse partizioni dell'RDD figlio. Le dipendenze narrow si riferiscono ad operazioni che possono essere eseguite in pipeline all'interno dello stesso nodo e quindi più ottimizzabili, mentre le dipendenze wide, per definizione solitamente richiedono uno shuffling (cioè un rimescolamento dei dati fra i vari nodi) dei dati e non possono essere eseguite in pipeline.

DAG Basandosi sull'applicazione definita e sul lineage graph, Spark calcola il piano d'esecuzione logico nella forma di un direct acyclic graph. Il DAG è una sequenza di computazioni che vengono eseguite sui dati. I nodi del grafo sono gli RDD, e gli archi sono operazioni eseguite su di esse. Come suggerisce parte del nome, "direct", il grafo è direzionato, cioè una trasformazione va da un RDD ad un altro, e il grafo è altresì aciclico quindi una trasformazione non può tornare ad una vecchia partizione.

Dopo la computazione del grafo, questo viene diviso in stage: il criterio per distinguere gli stage è dato dalle operazioni di shuffling o da eventuali partizioni salvate nella cache. Le dipendenze narrow saranno quindi raggruppate il più possibile all'interno dello stesso stage, e al verificarsi di operazioni wide o alla presenza di partizioni in cache viene definito un nuovo stage. Per ogni stage viene infine effettuata una decomposizione in task, i quali sono l'unità fondamentale dell'esecuzione. Si crea un task per ogni RDD e vengono schedulati e assegnate ai nodi sul principio della data locality. Nel caso di nodi lenti lo scheduler può eventualmente eseguire lo stesso task su nodi differenti in modo da concludere la computazione senza rimanere bloccato o allungando i tempi d'esecuzione.

Architettura e Ciclo d'esecuzione L'architettura di Spark è di tipo master-slave e non è dipendente dal tipo di cluster manager su cui gira Spark: il master è rappresentato dal driver il quale è unico per ogni applicazione ed ha il ruolo di coordinatore centrale, mentre gli slave sono i vari workers distribuiti che prendono il nome di executors. Ogni executor e il driver program sono dei processi indipendenti e il loro insieme è un'applicazione Spark.

Driver Program I compiti principali del driver sono quello di convertire il programma nei diversi task, quindi si occupa del calcolo del DAG e della sua conversione in un piano d'esecuzione fisico, oltre allo scheduling dei task agli executor, infatti ha una visione totale degli executors e dei task schedulati su di essi.

Executors Ogni executor è un processo con il compito di portare a termine l'esecuzione dei tasks ricevuti dal driver program utilizzando le risorse assegnate dal cluster manager. Ogni worker node può ospitare diversi executors, e un'applicazione spark solitamente ha svariati executors. Il ciclo di vita di un executors è solitamente pari all'intero tempo d'esecuzione dell'applicazione.

Cluster Manager E' il componente che si occupa dell'assegnamento e della gestione delle ricorse, e può essere o un manager standalone di Spark o qualsiasi gestore di risorse compatibile (come il già citato YARN). Nel caso si utilizzi Yarn il driver program corrisponde all'incirca all'applicazione, e ogni executor ad un container, i quali sono monitorati dal node manager.

Il funzionamento fra i tre componenti dell'architettura funziona solitamente nel seguente modo:

1. Il driver program crea lo Spark Context il quale si connette al cluster manager.
2. Il cluster manager alloca gli executors sui worker node basandosi sulle richieste pervenute dal driver program.
3. Lo Spark Context invia i task agli executors per essere eseguiti.

Modalità di deploy Spark mette a disposizione un'utility per il deploy di un'applicazione, tramite la quale è possibile settare diversi parametri fra i quali la memoria da utilizzare, i core della CPU, il numero di executors. Fra le diverse modalità di deploy si può scegliere fra:

- Cluster mode: in questo caso il driver program gira direttamente su un nodo del cluster
- Client mode: il driver viene eseguito su una macchina che non appartiene al cluster, mentre gli executors sono sempre all'interno del cluster
- Local mode: sia driver che executor vengono eseguiti sulla stessa macchina

Le modalità generalmente utilizzate sono le prime due, tenendo presente che nel caso di client mode, il driver debba essere il più vicino possibile agli esecutori per evitare il formarsi di eventuali problemi come il collo di bottiglia nello scambio di informazioni sulla rete. La local mode invece viene utilizzata in fasi di testing e debugging in cui non è necessario avere a disposizione l'intera architettura di cluster.

2.2.6 Spark SQL

Spark SQL è un modulo di Spark il quale ha lo scopo di effettuare data processing, e lavorare con dati strutturati e semi strutturati. Questo nasce dall'esigenza di effettuare fasi di ETL da diverse fonti di dati e su tipi di dati diversi, che abbiano o meno uno schema che li descrive. Oltre a questa motivazione è stato tenuto conto del fatto che c'è sempre più richiesta di effettuare analisi avanzate le quali sarebbero difficili da esprimere solo in sistemi relazionali, e una quantità elevata di sequenze di operazioni sui dati sono composte sia da query relazionali che codice procedurale. Spark SQL nasce da queste necessità ed è costruito su Spark in modo da permettere una perfetta integrazione fra API relazionali e procedurali.

Obiettivi di spark SQL:

- Fornire un supporto al processing relazionale sia nei programmi Spark (sugli RDD) che su fonti di dati esterne, utilizzando delle API comode ai programmatori
- Eseguire operazioni con elevate prestazioni utilizzando una logica DBMS
- Fornire supporto con nuove strutture dati sia con dati semi strutturati che database esterni
- Dare la possiblità di estensioni con tecniche avanzate di analytics

Fra le caratteristiche principali di Spark SQL è degno di nota il fatto di permettere:

- **Integrazione**
  Spark SQL consente di interrogare i dati strutturati all'interno dei programmi Spark, utilizzando SQL o una delle API apposite sui DataFrame. È utilizzabile in diversi linguaggi come Java, Scala, Python e R, e permette di combinare perfettamente query SQL con programmi in Spark, dando così la possiblità di integrare SQL con algoritmi di analisi dati.
- **Accesso unificato ai dati**
  È possibile caricare e interrogare i dati da diverse fonti possibili fra le quali i formati Hive, Avro, Parquet, ORC, JSON e JDBC, fornendo anche la possiblità di effettuare unioni e join fra fonti diverse.
- **Compatibilità con Hive**
  Spark SQL supporta la sintassi di HiveQL, e fornisce totale compatibilità con i dati Hive, le query ed eventuali UDF, permettendo così di accedere ai data warehouse esistenti su Hive.
- **Connettività Standard**
  È possibile effettuare una connessione attraverso JDBC o ODBC, utilizzano eventualmente una modalità server che con standard industriali che fornisce connettività verso JDBC e ODBC.

- **Scalabilità**
  Spark SQL include un ottimizzatore cost-based, lo storage colonnare e la generazione del codice per rendere veloci le query, sfrutta le proprietà del modello RDD e utilizza il motore di Spark, il quale fornisce anche supporto per eventuale fault tolerance, riuscendo a scalare job complessi.

SparkSQL fornisce una nuova astrazione dei dati chiamata Dataframe, la quale è basata sugli RDD di spark. I Dataframe sono definiti come una collezione distribuita di dati i quali sono organizzati in colonne, e concettualmente sono l'equivalente di tabelle relazionali, con ottime tecniche di ottimizzazione. Sono stati progettati specificamente per applicazioni big data prendendo esempio dai dataframe in R e pandas in python. Le caratteristiche fondamentali dei dataframe riguardano il fatto di:

1. Poter essere manipolati come gli RDD di Spark, ed è possibile creare dataframe sia da tabelle hive che da RDD esistenti, o come risultato di eventuali trasformazioni su dataframe già esistenti.

2. Supportare diversi formati di dati e sistemi di storage.

3. Capacità di processare dati da dimensioni ridotte (Kilobytes) a dimensioni elevate (Petabytes).

4. Ottimizzazione del codice.

Un Dataframe può essere visto come un RDD contenente degli oggetti riga, quindi è possibile chiamare le API di Spark citate in precedenza su di essi.

Altra caratteristica dei Dataframe è quella di essere "lazy", quindi non vengono effettuate esecuzioni fino al momento in cui vengono richieste delle azioni su di essi, oltre al fatto di essere immutabile, quindi ogni operazione eseguita su di essi dovrà generare un Dataframe diverso da quello su cui viene eseguita l'operazione. Le operazioni principali sono le query, che generano nuovi Dataframe, e le azioni, che eseguono le trasformazioni e restituiscono i dati al driver di Spark.

Capitolo 3

Stato dell'arte

In questo capitolo sono descritti gli approcci più significativi, presentandone gli aspetti chiave e le caratteristiche principali, portando anche alcuni confronti fra i diversi approcci unitamente ai vantaggi e agli svantaggi di ogni algoritmo.

Gli approcci sono stati distinti tenendo conto delle categorie descritte nel capitolo precedente: algoritmi density-based, flow-based, distance-based ed algoritmi distribuiti su piattaforma big data..

3.1 Algoritmi basati sulla densità

Gli algoritmi density based utilizzano una soglia di densità nel processo di formazione dei cluster. Fra i più interessanti e recenti citiamo TraClus, e un algoritmo derivato da evoluzioni di OPTICS.

3.1.1 Tra-POPTICS

Questo algoritmo [3] è un'estensione e un adattamento dell'algoritmo di clustering OPTICS prima in parallelo e poi anche per dati di traiettoria invece che punti.

OPTICS è un algoritmo density based, il quale è in grado di identificare cluster significativi fra dati a densità variabile, producendo un ordinamento lineare dei punti, tale che i punti che sono spazialmente vicini diventino vici-

ni anche nell'ordinamento prodotto. Nel lavoro effettuato, viene prodotta una versione parallela denominata POPTICS, che sfrutta le similarità fra OPTICS, e l'algoritmo MST (Minimum Spanning Tree), il quale calcola l'albero di copertura minimo fra un insieme di punti. I punti e la distanza di raggiungibilità di OPTICS sono considerati analoghi ai vertici e agli archi pesati che collegano i punti in MST. POPTICS divide l'intero dataset iniziale in porzioni, e calcola in parallelo l'MST locale di ogni porzione utilizzando l'algoritmo OPTICS. Infine viene eseguito un merging, cioè un'unione, dei vari alberi locali per ottenere l'albero globale.

Tale algoritmo non è previsto per dati di traiettoria, e viene adattato perché sia in grado, con lo stesso procedimento, di gestire anche dati di traiettoria, definendo una funzione apposita per calcolare la distanza fra le varie traiettorie rispetto ai singoli punti. Per ogni traiettoria verranno definiti i suoi ε-vicini, e infine calcolato l'albero globale delle distanze dal quale per una data soglia di raggiungibilità, i punti dell'ordine prodotto vengono assegnati ad un cluster piuttosto che ad un altro.

3.1.2 TraClus

TraClus [2] è un algoritmo di clustering di traiettorie che ha come scopo principale quello di trovare dei segmenti in comune fra le traiettorie, viene effettuato infatti clustering di sotto-traiettorie per individuare comportamenti comuni nella porzione di spazio a cui si riferisce il dataset. L'algoritmo è sostanzialmente diviso in due fasi:

1. Partizionamento delle traiettorie in un insieme di segmenti.

2. Raggruppamento dei segmenti prodotti dalla fase precedente basandosi sull'algoritmo DBSCAN [18].

Per quanto riguarda il partizionamento delle traiettorie, è basato su minimum description length [19], principio il cui concetto base spiega che ogni regolarità in un insieme di dati può essere usata per comprimerli, utilizzando meno

simboli di quelli che sono utilizzati per descrivere i dati. Il problema di segmentazione si trasforma in un problema di ottimizzazione, in cui si cerca il miglior partizionamento per una data traiettoria, cercando appunto un trade-off per ottenere un numero tale di segmenti adeguato i quali siano rappresentativi della traiettoria iniziale. In seguito utilizzando una misura di distanza apposita, viene eseguito il clustering basandosi sull'algoritmo citato catalogando i segmenti in un dato cluster o come rumore. Tale processo necessita di due parametri, ε e $minLns$, che rappresentano rispettivamente la distanza per la quale considerare i punti vicini, e il numero di dati minimo per considerare l'insieme dei vicini significativo, ed il processo può essere riassunto in tre fasi:

1. Calcolo del vicinato ($\varepsilon - neighborhood$) di ogni segmento non ancora classificato, ottenendo l'insieme dei vicini per un tale segmento entro una distanza specificata da ε. In questa fase vengono identificati i segmenti "core", cioè i segmenti per cui il numero di elementi presenti nel suo vicinato è maggiore di $minLns$. Nel caso un segmento sia identificato come core, si procede al secondo step, altrimenti viene considerato come rumore.

2. Per ogni elemento presente nell'insieme dei vicini del segmento considerato come "core", viene calcolato il suo vicinato. Nel caso in cui la cardinalità del vicinato di un segmento sia maggiore di $minLns$, il vicinato viene unito all'insieme dei vicini del segmento core, e il segmento viene aggiunto al cluster che si sta formando, nel caso in cui non sia già presente in altri cluster. All'esaurirsi della presenza di segmenti nell'insieme, l'espansione viene considerata conclusa. Tale procedimento viene ripetuto fino al momento in cui tutti i segmenti sono stati classificati.

3. Controllo della cardinalità, cioè del numero di traiettorie, di ogni cluster, eseguendo una fase finale di filtraggio dei cluster con pochi elementi.

In questo algoritmo il risultato è un insieme di cluster, in cui ogni gruppo è un set che rappresenta un insieme di traiettorie abbastanza vicine tra loro con ele-

vata densità. Per ogni cluster è infine calcolata una traiettoria rappresentativa del movimento dei suoi componenti.

3.1.3 OPTICS con multigranularità

L'obiettivo di questo algoritmo [6] è fornire una visualizzazione a diversi livelli di granularità in base ai parametri di densità specificati. Non viene prodotto un insieme di cluster, ma un augmented order dei dati, cioè un ordinamento dei dati iniziali, dal quale, in base alla granularità a cui si vuole visualizzare il risultato, è possibile estrarre i cluster come descritto in seguito. L'obiettivo primario rimane quello di riuscire ad analizzare gli spostamenti degli oggetti ed estrarre pattern di movimento, identificare cammini di movimenti frequenti, ponendo un focus particolare sul problema di identificare cluster a diversa granularità su traiettorie su larga scala.

Viene eseguita anche in questo algoritmo una segmentazione delle traiettorie, dato che si vanno a cercare dei movimenti frequenti, ed ogni traiettoria viene segmentata in parti in base ai corner point: punti in cui la direzione cambia repentinamente.

Il metodo di clustering, come accennato, è una versione modificata di OPTICS, e in questo caso non produce esplicitamente dei cluster, ma crea appunto un ordinamento di sotto-traiettorie, nel rispetto dei parametri epsilon (ε) e $minPts$ numero naturale, i quali rappresentano i valori riguardo l'intorno in cui considerare i vicini e la densità dell'intorno. I cluster con diverse densità possono essere estratti scorrendo il cluster-order e assegnando l'appartenenza al cluster in base a due parametri citati e in base a:

1. Core-distance: data una sotto-traiettoria e $minPts$ numero naturale, se il numero di vicini nell'intorno ε è minore di $minPts$ non è definita, altrimenti è la distanza dalla traiettoria ai suoi $minPts$ vicini.

2. Raggiungibilità: date due traiettorie dal dataset e $minPts$ un numero naturale, nel caso in cui il numero di neighbour sia minore del $minPts$

specificato la raggiungibilità non è definita, altrimenti è il valore massimo fra la core-distance e la distanza fra le due traiettorie.

Dopo di che i cluster con diverse densità possono essere estratti scorrendo il cluster-order e assegnando l'appartenenza al cluster in base ai parametri e ai valori di similarità appena descritti. Come misura di distanza fra le traiettorie viene utilizzata la distanza di Fréchet [21].

3.2 Algoritmi basati su densità e flusso

Per quanto riguarda gli algoritmi flow-based e flow-density based, oltre a considerare la densità durante il processo di clustering, utilizzano il concetto di flusso. Ad esempio: dopo aver mappato la traiettoria in una rete stradale, il clustering viene effettuato basandosi sui segmenti stradali e sul concetto di flusso di traffico continuo fra le varie strade al fine di creare una lista ordinata di segmenti stradali, il cui percorso prodotto sia indicativo del flusso di traffico e abbia continuità relativamente alla densità delle traiettorie e al flusso stesso. I cluster risultanti da questo tipo di algoritmo generalmente concatenano parti di traiettorie per ottenere la sequenza di segmenti che sia più rappresentativa di un certo gruppo di traiettorie. Sono molto adatti nel caso si vada alla ricerca dei percorsi più utilizzati o più trafficati in una rete stradale.

Un esempio di questo tipo di algoritmi di clustering è NEAT [7]. Questo algoritmo tiene conto dei vincoli fisici della rete stradale, delle prossimità fra le varie strade e del flusso di traffico tra segmenti consecutivi di strada per organizzare le traiettorie in cluster spaziali. Ha come obiettivo formare dei cluster che siano rappresentativi delle strade più percorse all'interno di un'area, e dopo una prima fase di partizionamento delle traiettorie, vengono collegati i segmenti fino a formare i cluster rappresentativi di ogni percorso.

Tale algoritmo è spiegato più dettagliatamente nel capitolo 4 dedicato, dove vengono approfondite sia la parte di segmentazione che di clustering.

3.3 Algoritmi basati sulla distanza

Questa categoria di algoritmi pone l'accento sulla definizione di misure di distanza apposite per confrontare le traiettorie, e poi generalmente vengono usati algoritmi classici di clustering, adattati in base alla misura di distanza definita.

3.3.1 DivClust

Questo algoritmo [8] ha l'obiettivo sempre di trovare comportamenti comuni di movimento, senza però avere nessuna conoscenza sulle informazioni geografiche dell'area in cui si trovano i dati. Un ulteriore obiettivo è quello di estrarre gruppi che abbiano lo stesso stile di movimento, quindi tenendo conto non solo del percorso ma anche della velocità e della direzione. Nasce da queste riflessioni la definizione del metodo di partizionamento, in cui vengono divise le traiettorie originali in segmenti di sotto-traiettorie, in base alle variazioni sia in velocità che direzione con una strategia slide window. Viene considerato un punto alla volta, e ad ogni aggiunta di punto, si controlla se vengono superati i due criteri di divisione: in caso affermativo viene collegato il primo punto con l'ultimo formando una linea chiamata replacement line (una segmento di collegamento fra i due punti rappresentante della sottoporzione del percorso), in caso contrario si riparte dall'ultimo punto che non superava i criteri. I due criteri assicurano che la replacement line abbia una variazione di direzione e velocità il più bassa possibile. L'input del clustering sarà quindi un insieme di linee che rappresentano un insieme di punti con velocità e direzione omogenea in una traiettoria, quindi non c'è il bisogno di progettare un algoritmo apposito per effettuare clustering, ma è stato scelto un metodo tipico che raggruppi le varie linee e ne fornisca una rappresentativa di ogni cluster.

La funzione di distanza definita per confrontare le traiettorie è formata da due parti: la prima misura la differenza spaziale fra le linee, la seconda invece riguarda la differenza temporale.

La fase di clustering è basata su su K-means: un approccio di clustering

il quale si occupa di partizionare l'insieme dei dati in un numero di gruppi scelto a priori in base agli attributi specificati e alla distanza utilizzata. In questo caso l'algoritmo di clustering utilizzato inizia il procedimento scegliendo in modo casuale un numero k di replacement lines, e fino all'assegnazione di tutte le linee prodotte il procedimento continua secondo l'algoritmo originario: generando prima nuove partizioni, assegnando ogni linea al cluster più vicino usando la misura di distanza apposita, e ricalcolando il centro di ogni cluster, fino all'esaurimento di ogni linea, cioè quando risultano tutte assegnate.

3.3.2 Fast ClusiVAT

Oltre agli approcci creati appositamente, e l'adattamento di algoritmi classici come K-means o DBSCAN, questo algoritmo, chiamato Fast-ClusiVat [9], propone una nuova misura di distanza fra una coppia di traiettorie, la quale risulta adatta anche per un alto numero di traiettorie sovrapposte in una fitta rete stradale. La misura di distanza in questione è chiamata trajDTW, in cui la distanza tra due archi è data dalla Dijkstra shortest path distance, che per due archi in una rete stradale è data dalla somma degli archi dell'albero più corto ottenuto per un grafo con numeri non negativi. Dato che la rete stradale è statica, vengono calcolate a priori le distanze tra tutti gli archi. La misura di distanza è bilanciata per traiettorie di lunghezza differenti in una rete stradale prevalentemente a forma di griglia con spazi ravvicinati e segmenti tendenti ad essere paralleli e perpendicolari, e viene definita sia tenendo conto della direzione delle traiettorie che senza. Non vengono eseguiti partizionamenti, ma è prevista una fase di pre-processing utilizzando un algoritmo di map-matching. Data una stima del numero di cluster nel dataset, si vuole scegliere un numero n di campioni che siano almeno equamente distribuiti fra i diversi cluster come le N traiettorie nel dataset, per ottenere quindi un campione rappresentativo di tutti i dati iniziali. Tale stima viene fatta partizionando del dataset in un numero k' di partizioni e viene effettuata la scelta delle traiettorie in modo random da ognuna delle k' partitioni per generare un insieme di traiettorie campione. In seguito viene applicato l'algoritmo Vat/iVat sulle traiettorie estratte.

Vat fornisce un'immagine che rappresenta la dissimilarità dei dati dalla quale è possibile ricavare il numero di cluster calcolato. Conclusa questa fase dell'algoritmo sono stati ottenuti dei cluster di n traiettorie campione, usando una misura che non tiene conto della direzione, quindi all'interno dei cluster ci sono traiettorie con percorsi simili ma potenzialmente direzioni opposte, di conseguenza viene effettuata una seconda applicazione dell'algoritmo usando però la misura direzionale. I cluster contengono quindi traiettorie che hanno percorsi simili e anche stessa direzione. In conclusione utilizzando la misura con la direzione, si assegnano le traiettorie non considerate precedentemente ad uno dei cluster ottenuti.

3.4 Algoritmi distribuiti

Allo stato dell'arte attuale non sono stati implementati una quantità elevata di sistemi che effettuano trajectory clustering utilizzando una grande quantità di dati. Sono presenti alcuni approcci, i quali ripropongono algoritmi di clustering ripensati in ottica big data utilizzando la piattaforma Map Reduce, sia con i vantaggi che ne consegue di riuscire a processare grandi quantità di dati, ma soprattutto con la difficoltà di dover riadattare l'algoritmo nelle fasi di mapping e di reducing. La tendenza è di utilizzare algoritmi classici di clustering ripensandoli in ottica big data.

I due approcci descritti di seguito sono molto simili fra loro, soprattutto nella fase di clustering in cui l'algoritmo K-means ripensato in ottica Map Reduce è pressoché lo stesso in entrambi gli approcci, in cui viene eseguita ogni iterazione di Map Reduce fino al convergere dei cluster. Le differenze sono evidenziabili nel tipo di misura di distanza effettuata per la comparazione dei dati, e nelle due diverse fasi di pre-processing dei dati prima dell'esecuzione di K-means per produrre l'output desiderato.

3.4.1 Approccio Map-Reduce per clustering di zone di traffico

In questo approccio [13] non vengono effettuate segmentazioni delle traiettorie, ma soltanto una rimozione non banale di outlier, cioè di punti di rumore, per evitare di considerare dati sporchi nella formazione dei clusters, per quanto riguarda la misura di distanza viene utilizzata o la Manhattan distance o la distanza euclidea in base al calcolo della curtosi, nel caso questa sia maggiore di una soglia viene considerata la prima distanza, la seconda altrimenti.

Le tre fasi di K-means su Map Reduce sono divise nel seguente modo:

1. **Map**
   Nella fasi di mapping viene calcolata la distanza fra ogni oggetto e ogni centro del cluster, assegnando ogni oggetto al cluster più vicino.

2. **Combine**
   Estrazione dei dati dell'output del map e unione dei dati appartenenti allo stesso centro del cluster.
   Calcolo della somma dei valori degli oggetti dello stesso cluster e salvataggio del numero di elementi per calcolare il mean value, cioè la media degli oggetti del cluster.
   In output alla fase successiva il local result di ogni cluster.

3. **Reduce**
   Aggregazione dei local result di tutti i cluster, e computazione del nuovo centro per ogni cluster.

Se il criterio di convergenza è soddisfatto viene restituito l'output altrimenti viene eseguita una nuova iterazione.

3.4.2 Clustering con Map-Reduce di traiettorie spazio temporali

Questo algoritmo [12] si pone l'obiettivo di effettuare clustering di traiettorie riferite a veicoli. Utilizza K-means diviso in tre fasi ripensandolo in Map

Reduce, ed esegue un calcolo ottimizzato della similarità fra traiettorie, riducendo il tempo di esecuzione quando le traiettorie sono molto lunghe.

Non viene effettuata una vera e propria segmentazione delle traiettorie, ma solo una fase di riduzione dei dati per migliorare l'accuratezza della misura di distanza. La riduzione di dati viene effettuata calcolando la vertical distance al fine di decidere se considerare tutti i punti o ridurne il numero. Il procedimento del calcolo della vertical distance consiste nel considerare tre punti ad ogni fase, e con una soglia specificata controllare se il punto intermedio può essere eliminato o meno. Il variare della soglia permette di effettuare una sorta di modifica della granularità, perché di conseguenza aumenterà il numero di punti o li ridurrà producendo traiettorie più precise o a granularità più ampia. L'utilizzo di una fase di riduzione dei dati è utile per migliorare l'efficienza della misura di distanza utilizzata (dynamic time warping [23]) la cui complessità è $O(l^2)$ con $l < n$, mentre con traiettorie lunghe e $l >> n$ la computazione migliora.

La tecnica di clustering è come detto K-means in tre fasi, ripensato in ottica Map Reduce le cui fasi sono state progettate nel seguente modo:

1. **Map**

   Divisione in gruppi o parti durante la fase di Map. Ricerca del cluster più vicino di ogni traiettoria, dando il centro del cluster ed esaminando la misura di similarità.

   L'output di questa fase contiene il cluster id in chiave, ed il valore associato è il cluster object che include il numero di traiettoria e la somma cumulativa dei punti delle coordinate di ogni traiettoria.

2. **Shuffle**

   Nella fase di shuffling si effettua l'ordinamento dei risultati intermedi, vengono accumulate le somme dei punti delle coordinate di ogni traiettoria con lo stesso cluster.

   In output, come da paradigma, la stessa chiave nello stesso reducer.

3. **Reduce**
   Nell'ultima fase è presente il calcolo della somma dei componenti delle coordinate di ogni traiettorie con lo stesso cluster, e il calcolo del nuovo centro del cluster.

3.5 Confronti

In questa sezione vengono confrontate le diverse fasi di ogni algoritmo presentato in precedenza, ponendo l'attenzione sui pro e contro di ogni metodologia.

3.5.1 Tecniche di segmentazione

Analizzando le tecniche di segmentazione delle traiettorie utilizzate dai diversi algoritmi, le possiamo riassumere in quattro categorie principali:

- **Conformazione stradale** [7, 9]
  La traiettoria è vista come sequenza di segmenti stradali.

- **Velocità e direzione** [8]
  I segmenti prodotti sono rappresentativi di una porzione di spazio con velocità e direzione simile.

- **Corner point** [6]
  La traiettoria viene segmentata nei punti in cui si rileva un cambio di direzione repentino.

- **Minimum description length** [2]
  Problema di ottimizzazione per rappresentare la traiettoria con il numero di punti adatto a descriverne il percorso.

Le tecniche di segmentazione delle traiettorie, sono strettamente collegate all'applicazione per cui viene usato l'algoritmo. Nel caso di TraClus [2] vengono creati dei segmenti come unità di clustering, dato che l'obiettivo è quello di

scoprire gruppi di sotto-traiettorie e unire percorsi simili di traiettorie diverse, che considerando le traiettorie nell'insieme non si sarebbero scoperti. Il partizionamento in quel caso è stato trasformato in un problema di ottimizzazione utilizzando la minimum description lenght per ottenere i segmenti più rappresentativi del movimento della traiettoria. In altri algoritmi invece le traiettorie vengono partizionate in base alla rete stradale, questo da una parte migliora l'accuratezza del partizionamento nel caso in cui si vogliano cercare percorsi simili, o i percorsi più comuni, ma implica di conoscere la rete stradale sottostante il proprio dataset, e magari utilizzare algoritmi che siano in grado di mappare ogni traiettoria sulle strade (algoritmi di map-matching) in fase di trasformazione dei dati per eliminare gli errori dovuti al rilevamento dei dispositivi GPS e aumentare così l'accuratezza del risultato. Altre tecniche di partizionamento, come visto, riguardano la segmentazione in base al cambio di direzione, o in base all'angolo che si riesce a formare fra un punto e l'altro [6]. Queste tecniche permettono di riuscire a creare delle sotto-traiettorie in base ai cambi di direzione senza conoscere la rete stradale. Da una parte è un vantaggio in quanto non c'è necessità di conoscere le strade sottostanti, ma d'altro canto potrebbe influire sulla lunghezza dei segmenti, i quali risulteranno a dimensioni variabili. Il metodo usato in DivClust [8] basato sulla velocità e direzione della traiettoria è stato comparato con quello utilizzato in TraClus con MDL [2]. Il risultato di tale comparazione in [8] ha mostrato che con la stessa quantità di dati, il primo ha più efficienza di processing, e il numero di linee ottenute dal partizionamento è pressoché identico.

In conclusione, il partizionamento delle traiettorie è usato abbastanza frequentemente prima della vera e propria fase di clustering. Ogni tecnica di partizionamento, come specificato, ha i suoi vantaggi e svantaggi, ed è strettamente collegata al contesto in cui viene utilizzata, e ovviamente nel caso in cui si consideri le traiettorie come punti, o nella loro interezza, il partizionamento non viene effettuato.

3.5.2 Algoritmi

In questa sottosezione vengono messe a confronto le caratteristiche dei alcuni algoritmi fra quelli presentati, e in tabella 3.1 sono raccolte le particolarità evidenziate e citate nei confronti descritti.

Tra-POPTICS (OPTICS) vs. TraClus (DBSCAN) TraClus [2] utilizza un algoritmo basato sulla densità per raggruppare le varie porzioni di traiettoria, ma non tiene conto del dominio temporale delle traiettorie. Come affrontato in [3], DBSCAN non riesce ad identificare cluster significativi nel caso ci sia densità variabile fra i dati, mentre OPTICS è in grado di affrontare questo problema e identificare comunque i cluster nel caso sia presente una densità variabile. Nel caso in cui ci sia densità variabile il valore di densità ε utilizzato in DBSCAN potrebbe non essere in grado di rappresentare i cluster, mentre in OPTICS vengono utilizzate anche altre due misure, come core-distance e distanza di raggiungibilità per la formazione dei cluster. D'altro canto, TraClus esegue un clustering di segmenti, e la densità intra-cluster fra segmenti può tendere ad essere molto alta rispetto a dei punti, e nel caso fosse stato utilizzato OPTICS, questo poteva portare a formare cluster poco distinguibili dalle parti di rumore. OPTICS quindi risulta più adatto per cluster di traiettorie nel caso queste vengano considerate come punti, ma non nel caso si considerino dei line-segment, nel quale risulta più efficace DBSCAN. OPTICS risulta comunque migliore nel caso l'obiettivo sia la scoperta di cluster robusti partendo da un dataset in cui ci sia densità variabile.

DivClust (K-Means) vs. TraClus (DBSCAN) L'algoritmo TraClus è stato usato largamente come parametro di confronto nei vari studi successivi ad esso. La comparazione effettuata in [8] porta alla luce alcuni aspetti di entrambi gli algoritmi.

L'obiettivo di DivClust è quello di generare cluster significativi e con una visualizzazione chiara, questo indica che al variare dei parametri di densità DBSCAN risulta non propriamente adatto a trovare cluster con proprietà spa-

Algoritmo	**Caratteristiche**
TraClus	• Adatto per clustering di line-segment • Ricerca di sub-trajectories comuni
Tra-POPTICS	• Migliore per dati a densità variabile • Adatto a traiettorie considerate come punti
DivClust	• Progettato per tener conto anche di proprietà spazio temporali consistenti
ClusiVat	• Clustering su strade parallele e perpendicolari

Tabella 3.1: Caratteristiche e confronti fra algoritmi

zio temporali consistenti, e forma un numero elevato di cluster tale da rendere difficile la comprensione. Nel caso non si considerino proprietà temporali o spaziali DBSCAN può risultare utile, dato che connette oggetti che siano raggiungibili a livello di densità in un gruppo. Ma nel caso si considerino tali proprietà DBSCAN può includere oggetti diversi fra loro nello stesso cluster, cosa che in DivClust è stata progettata e tenuta conto fin dall'inizio della definizione di partizionamento e misura di distanza. Per tali motivi è stato scelto di usare K-means piuttosto che DBSCAN. Nell'algoritmo K means va evidenziato il fatto che la scelta del numero k di cluster iniziale è non banale: scegliendo un valore troppo basso si ottengono cluster eterogenei, mentre scegliendo un valore troppo grande si ottiene un elevato numero di cluster [24].

In base all'obiettivo finale, cioè il tipo di cluster e il significato che deve avere ogni raggruppamento, risulta ancora una volta significativa la scelta del metodo di clustering e del metodo di partizionamento.

ClusiVAT vs. Altri I confronti fra ClusiVat e altri algoritmi quali DBSCAN, OPTICS o NEAT [9], pongono l'accento sul fatto che il primo algoritmo sia adatto soprattutto per strade tendenzialmente parallele e/o perpendicolari fra loro, il quale grazie anche alla misura di distanza apposita permette di raggiungere buoni risultati.

Capitolo 4

NEAT

In questo capitolo viene descritto nei dettagli l'algoritmo utilizzato come base di partenza per il progetto di tesi. È presente una descrizione dettagliata del funzionamento con un focus particolare su aspetti positivi e negativi al fine di evidenziare le motivazioni che hanno portato alle migliorie applicate.

4.1 Funzionamento generale

NEAT è un algoritmo di clustering flow-based, progettato nel 2012 [7] e migliorato con una pubblicazione seguente nel 2015 [16], il cui obiettivo è produrre cluster di traiettorie i quali descrivano dei flussi di traffico ad alta densità e continuità di movimento.

Gli scenari reali che hanno portato alla progettazione di questo algoritmo, riguardano sia la pianificazione e l'ottimizzazione del trasporto pubblico, che il posizionamento di pubblicità basate sulle location e sugli spostamenti in una rete stradale. Nel primo caso può essere utile migliorare i trasporti pubblici, ridurre il numero di fermate o posizionarle in punti strategici, sapendo quali sono le strade non solo con alta densità, ma anche con un'alta continuità di movimento. Nel secondo scenario invece, incrociando informazioni di marketing con informazioni relative a posizioni e movimenti tipici delle persone, si fornisce un aiuto nel posizionare pubblicità lungo le strade più percorse, o nel miglioramento del target di persone a cui inviare sconti e promozioni.

Considerando le strade più percorse in relazione al posizionamento del proprio punto vendita, se uno sconto viene inviato a persone che passano nella strada o in quelle in prossimità del proprio negozio, aumentano le possibilità che le persone si fermino, rispetto ad altre che ricevono sconti e fra i loro movimenti abituali non transitano mai nelle strade vicine al punto vendita.

In entrambi i casi è utile avere una conoscenza della conformazione delle strade a cui si riferiscono le traiettorie, di conseguenza, all'interno del processo di clustering viene considerata anche la rete stradale oltre all'insieme delle traiettorie.

Piuttosto che adattare algoritmi density-based come DBSCAN o OPTICS per raggruppare traiettorie simili come visto negli algoritmi descritti nei capitoli precedenti, viene utilizzato un approccio differente, perché i due algoritmi non sarebbero in grado di raggruppare in modo efficiente traiettorie strettamente legate ad un'area ad alta densità di traffico, ma risultano più adatti per raggruppare traiettorie il cui movimento non è strettamente vincolato alle strade, come eventi naturali o spostamenti di animali [16].

Per questi motivi oltre a tenere conto della conformazione stradale durante il processo di clustering, si tiene conto anche di diversi fattori come la densità di traffico, e la continuità di movimento oltre alla velocità alla quale si possono muovere gli oggetti per raggiungere l'obiettivo iniziale relativo al significato dei cluster ottenuti.

L'algoritmo segue tre step per la formazione dei cluster, rappresentate anche in figura 4.1:

1. **Computazione t-fragments e formazione di base clusters**
   Si considerano le intersezioni fra le strade come possibili punti di segmentazione di una traiettoria, quindi dato un insieme di traiettorie, queste vengono partizionate in sotto-traiettorie in base a tali punti, ottenendo per ogni traiettoria una serie di segmenti chiamati *t-fragments*. I segmenti vengono raggruppati in base al tratto di strada a cui si riferiscono creando una sorta di cluster locali a diversa densità denominati *base clu-*

ster. Ogni cluster locale è l'insieme dei segmenti di ogni traiettoria che passano per lo stesso segmento stradale, e l'insieme dei base cluster funge da base di partenza per la fase successiva di creazione dei cluster veri e propri.

2. **Raggruppamento in flow cluster**
   Nel secondo step si formano flussi di movimento ad alta densità e continuità di movimento. Per la loro costruzione ci si basa su continuità di flusso, densità, e sulla conformazione stradale, ed i gruppi ottenuti sono chiamati *flow cluster*.

3. **Raffinamento dei flow cluster**
   I risultati ottenuti vengono migliorati usando un metodo density-based per cogliere possibilità di aggregazione non sfruttate, utile soprattutto nel caso in cui il processing avvenga real time per sfruttare eventuali possibilità di raggruppamento ulteriore che altrimenti non verrebbero evidenziate.

4.2 Calcolo base clusters

Inizialmente le traiettorie appartenenti al dataset iniziale vengono processate tramite un algoritmo di map-matching, al fine di mappare i punti di ogni traiettoria sulla rete stradale sottostante, per questo passaggio come algoritmo è utilizzato SLAMM [17].

La rete stradale sottostante il dataset è definita come un grafo direzionato $G = (\nu, \varepsilon)$, formato da un insieme di nodi di giunzione $\nu = \{n_0, n_1, ..., n_{\mathrm{N}}\}$ e da un insieme di archi direzionati $\varepsilon = \{(segId, n_{\mathrm{i}}n_{\mathrm{j}})\}$. Un arco appartenente all'insieme rappresenta un segmento stradale che collega due punti di giunzione.

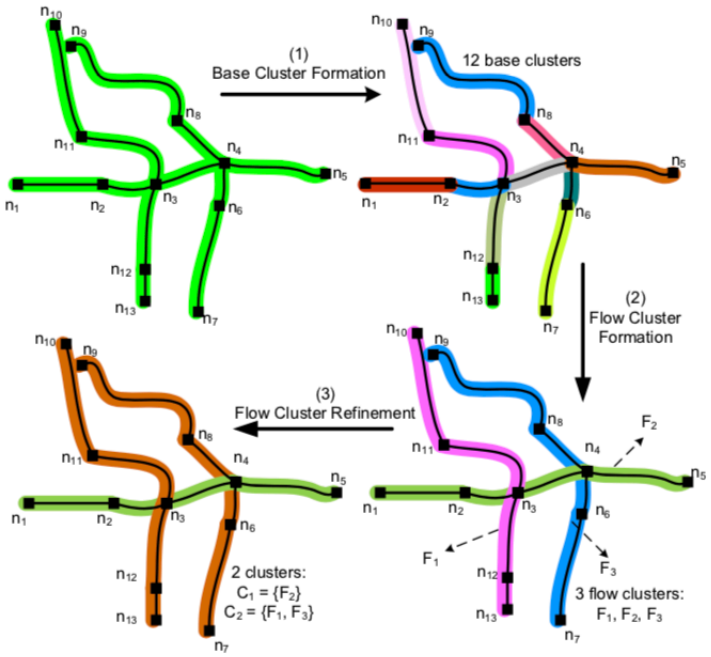


Figura 4.1: Esempio delle tre fasi di clustering previste dall'algoritmo. Immagine tratta da [16].

4.2.1 Segmentazione traiettorie: estrazione t-fragments

Per ogni traiettoria $TR = \{tr_{id}, l_0 l_1 ... ln\}$ appartenente al dataset, l'estrazione dei t-fragments avviene esaminando tutti i punti appartenenti alla traiettoria. Per ogni coppia di punti consecutivi, l_i e $l_{\mathrm{i}+1}$, si controlla se i corrispettivi segmenti stradali in cui sono stati mappati, sono diversi o meno. Se $segId(l_i) \neq segId(l_{i+1})$ significa che i due punti sono su due strade diverse, quindi viene inserito il punto di giunzione delle due strade fra i due punti della traiettoria. La traiettoria viene così trasformata in una sequenza di punti e nodi di giunzione, ed esaminando ogni punto in una data traiettoria, la sequenza di nodi di giunzione aggiunti è utile per segmentare la traiettoria in una sequenza di t-fragments. Un t-fragment di una traiettoria rappresenta una sotto-traiettoria, $l_\mathrm{k} l_{\mathrm{k}+1} ... l_{\mathrm{k}+\mathrm{m}}$ la quale consiste in $m+1$ punti consecutivi estratti da TR i quali si riferiscono allo stesso segmento stradale. La sequenza di t-fragments estratti da una traiettoria mantiene quindi le caratteristiche iniziali corrispondenti

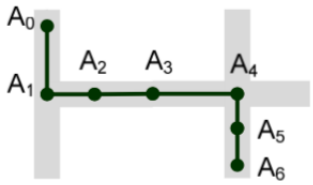


Figura 4.2: Esempio del partizionamento in t-fragments di una traiettoria: considerando i punti d'intersezione fra le strade i segmenti ottenuti saranno A_0A_1,A_1A_4,A_4A_6. Immagine tratta da [16].

al percorso effettuato nella traiettoria iniziale, ed è possibile visualizzare un esempio di segmentazione di una traiettoria in figura 4.2.

4.2.2 Formazione base-clusters

I t-fragments delle diverse traiettorie sullo stesso segmento stradale possono essere considerati vicini in termini di prossimità e questi possono mostrare la similarità fra i movimenti di diversi oggetti.

Esaminando i t-fragments estratti e raggruppandoli in base al valore che identifica il road segment corrispondente, è possibile ottenere gruppi diversi in cui ogni gruppo è identificato da un segmento stradale e contiene l'insieme dei t-fragment passanti per esso. Un base cluster è quindi definito come $S = \{tf_i | TR(tf_i) \in T, tf_i.segId = e.segId\}$, dove $TR(tf_i)$ indica la traiettoria dalla quale è estratto il t-fragment $tf_i \in S$, ed in cui $e.segId$ è l'identificatore del road segment, il quale deve appunto essere uguale all'identificatore del road segment dei vari t-fragments, i quali formano il base cluster corrispondente.

Per ogni base cluster viene quindi calcolata la densità, come il numero di t-fragments appartenenti ad esso, in modo da ottenere un ordinamento in base alla densità per i gruppi formati in questa prima fase.

Nella figura 4.3, è rappresentata graficamente la parte di creazione dei base clusters: dopo aver partizionato le traiettorie in t-fragments, raggruppandole in base ai segmenti stradali, otteniamo un insieme di base clusters

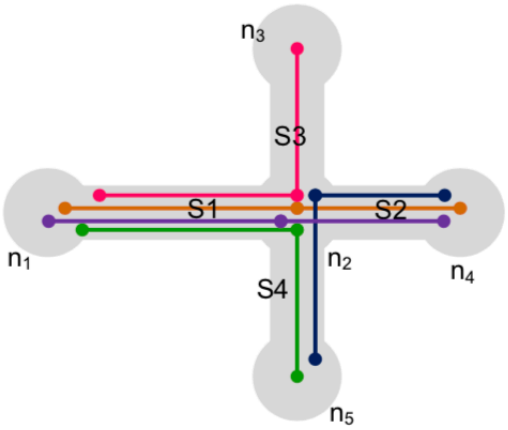


Figura 4.3: Esempio di base clusters. Immagine tratta da [16]

$B = \{S_1, S_2, S_3, S_4\}$. È possibile notare che ogni base cluster rappresenta un segmento stradale e ognuno contiene un insieme di t-fragment tutti sullo stesso segmento stradale, ad esempio il base cluster S_1 rappresenta il segmento stradale fra n_1 e n_2, e contiene quattro t-fragments.

4.3 Raggruppamento in flow cluster

Nella prima fase si produce un insieme di base cluster ordinati in base alla densità, cioè in base al numero di t-fragments che ogni base cluster contiene, e l'obiettivo della seconda fase, prendendo in input tale insieme, è quello di produrre dei flow cluster.

Un flow cluster è una lista ordinata di base clusters, rappresentabile come $F = \{S_0 S_1 ... S_N\}$ dove S_{i+1} appartiene all'insieme dei vicini di S_i e in cui l'elenco dei road segment che rappresentano ogni base cluster $e^{S_0} e^{S_1} ... e^{S_N}$ forma una sequenza di strade, la quale viene definita come route rappresentativa del flow cluster.

Per la formazione dei flow cluster l'algoritmo prevede di partire dal base cluster più denso chiamato "*dense-core*", come elemento iniziale del primo flow cluster. La scelta di partire da quello più denso è motivata da due ragioni: nel caso in cui venisse scelto un base cluster in modo casuale, si potrebbe

ottenere un flow cluster che descrive un flusso di traffico trascurabile o poco importante, il quale potrebbe essere eventualmente tagliato dai risultati, e in aggiunta a ciò il partire sempre dal più denso, permette che il procedimento di raggruppamento in flow-cluster sia deterministico in modo che, partendo da un dataset, il risultato prodotto sia sempre lo stesso.

Nel processo di formazione dei flow cluster vengono inoltre considerati dei vincoli spaziali e di effettiva continuità di motivimento. Per ogni coppia di base cluster S_i ed S_j possiamo definire il *netflow* $f(S_i, S_j)$ come il numero di traiettorie presenti in entrambi i base cluster. Considerando un base cluster S_i, un vertice n_u del segmento di strada che il base cluster rappresenta, e l'insieme B che contenente i base cluster: il vicinato di S_i in relazione a n_u, definito come $N_f(S_i, n_u)$, è il set di base cluster in cui ogni elemento ha almeno una traiettoria in comune, ed è formalmente definito come $N_f(S_i, n_u) = \{S_j | e^{S_i} \in L_{\mathrm{n_u}}, f(S_i, S_j) > 0\}$ in cui S_j è ogni base cluster appartenente al set B, e^{S_i} il suo road segment identificativo, ed $L_{\mathrm{n_u}}$ è il set di segmenti stradali che collegano il road segment identificativo del base cluster, identificabili perché hanno il punto di giunzione stradale n_u in comune.

Ogni vicino che soddisfa tali condizioni, garantisce che le due strade identificative dei base cluster siano effettivamente collegate nella rete stradale, e di conseguenza che ci sia continuità di flusso dal base cluster di provenienza a quello candidato perché è presente almeno una traiettoria in comune fra i due base cluster.

Supponendo di essere nel processo di espansione di un flow cluster F, e la sequenza di base cluster che contiene sia rappresentabile nella forma $\{S_a S_{a+1} ... S_{b-1} S_b\}$: il base cluster S_{b+1} che dovrà essere aggiunto al flow cluster dovrà far parte del vicinato di S_b nel rispetto del punto di intersezione che si sta considerando per l'espansione, più formalmente: $S_{b+1} \in N_f(S_b, n_u)$.

Fattori di merging Per ogni base cluster all'interno del vicinato dell'ultimo base cluster aggiunto al flow cluster, nel momento in cui viene analizzato per essere aggiunto o meno al flow cluster, si calcola un valore che ne identifica il

suo livello di merging. Tale valore è una somma pesata di tre fattori: *densità*, *flusso* e *velocità*.

$$SF(S_i, S_j) = w_q \times q + w_k \times k + w_v \times v$$

In cui S_i è l'ultimo base cluster aggiunto al flow cluster, e S_j un base cluster candidato, cioè contenuto nell'insieme dei vicini di S_i. I coefficienti w_q,w_k,w_v determimano il peso rispettivamente dei fattori di flusso q, densità k e velocità v. I pesi assegnati ai fattori, $w_q \geq 0$, $w_k \geq 0$,$w_v \geq 0$, devono soddisfare la condizione $w_q + w_k + w_v = 1$.

I tre fattori vengono calcolati per ogni candidato, e sono normalizzati tra $[0, 1]$ per rappresentare meglio la loro capacità relativa di estendere il flow cluster nel rispetto del concetto di alta densità e continuità di flusso.

Flusso Il fattore di flusso q rappresenta la continuità di movimento fra i due base cluster, e un alto valore sta ad indicare che il candidato mantiene una continuità di traffico alta tenendo conto del flow cluster che si sta formando. Può essere calcolato nel modo seguente:

$$q = \frac{f(S_i, S_j)}{|PTr(S_i)|}$$

cioè il numero di traiettorie presenti in entrambi i base clusters, rapportato alla cardinalità, cioè il numero di traiettorie, del base cluster per il quale si cerca il candidato.

Densità Il fattore di densità invece rappresenta la densità relativa del candidato rispetto al vicinato considerato il punto di giunzione in comune fra di essi, e si calcola secondo la seguente formula:

$$k = \frac{d(S_j)}{(d(S_i) + \sum_{S_m \epsilon N_f(S_i, n_u)} d(S_m))}$$

in cui d rappresenta la densità di un base cluster, cioè il numero di t-fragments al suo interno.

Velocità Il fattore di velocità serve a misurare quanto gli oggetti si muovono velocemente nel base cluster candidato in base al suo vicinato, al fine di formare flussi che possano avere più uniformità possibile a livello di velocità. La formula per calcolare tale fattore è descritta di seguito:

$$v = \frac{speed(S_j)}{\sum_{S_m \epsilon N_f(S_i, n_u)} speed(S_m)}$$

speed è il limite di velocità del segmento di strada a cui si riferisce il base cluster, e si divide quello del candidato per la sommatoria delle velocità di tutti i vicini dell'ultimo base cluster aggiunto al flow cluster.

Il fattore di flusso q, è utile per calcolare il mantenimento della continuità di movimento fra due base cluster, il fattore di densità k serve per formare cluster che siano identificativi di un numero di persone significativo, mentre il fattore di velocità v mira a mantenere uniformità di velocità nel flusso che si forma.

Bilanciamento dei pesi In base all'assegnamento dei pesi, il risultato della seconda fase varia in termini di significato. Generalmente i pesi per ogni fattore sono bilanciati, in modo tale che il base cluster aggiunto ad ogni iterazione sia il migliore in termini sia di numero di traiettorie che rappresenta, sia di continuità di movimento rispetto all'ultimo base cluster aggiunto e sia in termini di velocità per avere continuità fra le strade, come da obiettivo iniziale del processo di clustering. Nel caso si sbilancino i pesi con la combinazione $(w_q, w_k, w_v) = (0, 1, 0)$, cioè considerando solo il fattore di densità, si otterranno per esempio dei cluster i quali descrivono strade in cui c'è un'alta concentrazione di traffico.

Il vicino con valore di merging più alto verrà quindi aggiunto al flow cluster, e il flusso continua ad essere espanso fino al momento in cui ci sono candidati validi fra i vicini per essere aggiunti. All'esaurimento dei candidati il processo di formazione del flow cluster si ripete espandendosi dal vertice del segmento di

Algoritmo 1 Raggruppamento in flow cluster

Input:

1. $B = \{S_0, S_1, .., S_M\}$ ▷ Un set di base clusters
2. $G = (\nu, e)$ ▷ Una grafo rappresentante la rete stradale

Output: $W = \{F_1, F_2, .., F_Q\}$ ▷ Un insieme di flow cluster

```
flowId = 0                                   ▷ Identificatore del numero del flow cluster
W = ∅
while B ≠ ∅ do                               ▷ Fino a quando ci sono base cluster
    S = dense-core(B);                       ▷ Scegli il base cluster con il valore di densità più alto
    {n1, n2} = punti di giunzione(S, G);
    F_flowid = espansione flow-cluster da n1
    F_flowid = espansione flow-cluster da n2
    aggiunta F_flowid a W
    flowId++;
end while
```

partenza non utilizzato nella prima espansione, terminando con una condizione simile alla precedente. Quando entrambe le condizioni di stop sono raggiunte, cioè quando in entrambe le direzioni non ci sono più vicini da aggiungere, il flow cluster viene considerato completato, si sceglie nuovamente il base cluster più denso e si forma un nuovo flow cluster.

Il processo di formazione dei flow cluster termina nel momento in cui tutti i base cluster sono stati assegnati ad un flow cluster.

Considerando la figura 4.3, partendo dal base cluster S_1, il possibile flow cluster risultante è compreso fra le possibilità $\{S_1S_3\}\{S_1S_2\}\{S_1S_4\}$, e verrà scelta la combinazione con il candidato con valore di merging più alto. Oltre agli esempi precedenti possibili, un altro flow-cluster che potrebbe formarsi è $\{S_2S_4\}$ se ancora disponibili e non già utilizzati. E' importante notare che non potrebbe formarsi invece il flow cluster contenente $\{S_3S_4\}$ perché non ci sono traiettorie in comune fra i due base cluster. Lo pseudo codice riassuntivo della fase di raggruppamento in flow cluster è descritto in Algoritmo 1.

4.4 Raffinamento dei flow cluster

La terza fase prevista dall'algoritmo è progettata per sfruttare le opportunità ulteriori di merging che possono presentarsi dopo la seconda fase. Tali possibilità sono individuate tramite un approccio density-based per raggruppare i flow cluster. L'algoritmo density-based scelto è DBSCAN [18] e la distanza di Hausdorff è utilizzata per computare la distanza fra due flow cluster.

Questa fase è molto utile ed efficiente specialmente nel caso in cui il clustering venga eseguito in modo incrementale e distribuito. Le prime due fasi possono essere eseguite per ogni parte del dataset, indipendentemente dal momento in cui viene processato, e questa fase serve per unire eventuali flow-cluster simili che non potevano essere scoperti avendo processato diversi insiemi di traiettorie in momenti diversi, in modo da produrre un risultato compatto e completo.

4.5 Punti di forza

1. Ottenimento di cluster rappresentativi del modello stradale e indicativi dei flussi di traffico più densi

2. Definizione di astrazioni apposite per il processo di clustering: *t-fragments, base cluster e flow cluster*

3. Introduzione del concetto di vicinato per mantenere coerenza con la rete stradale a cui si riferisce il dataset e riconoscere le parti più critiche e interessanti durante il processo di clustering

4. Produzione di cluster basata su densità e flusso come continuità di movimento, invece che usando densità e misura di distanza, non particolarmente adatti in alcuni casi sul modello stradale.

5. Efficienza ed accuratezza rispetto ad altri algoritmi diffusi per clustering di traiettorie density-based.

Capitolo 5

Estensione ed implementazione

Partendo da un algoritmo il quale identifica dei flussi di traffico significativi, nel rispetto del movimento dei veicoli sulle strade, ed in grado di identificare le strade percorse più frequentemente, è stata aggiunta un'estensione che permette di eseguire il clustering a diversi livelli di granularità. In questo capitolo verrà presentata l'estensione dell'algoritmo flow based descritto nel capitolo 4, unitamente ai cambiamenti che questa ha portato a livello concettuale e di conseguenza vengono presentate le modifiche effettuate e i cambiamenti prodotti al fine di parallelizzare e distribuire il procedimento.

5.1 Clustering con multigranularità

L'algoritmo NEAT [7, 16] impone, per sua definizione, di conoscere la rete stradale sottostante, cosa che potrebbe non essere banale e risultare come un vincolo significativo, ma soprattutto non permette di eseguire analisi a diversi livelli di granularità. Nel caso si voglia formare cluster riguardanti una città, o una regione ad un livello più alto di granularità, l'algoritmo impone di conoscere e mappare come grafo tutta la rete stradale a cui si riferisce il dataset, ed a tale scopo è stata progettata e poi implementata un'estensione, la quale permette di rendere l'algoritmo più adattabile e completo.

È stato introdotto il concetto di multigranularità, per permettere di eseguire analisi e di conseguenza clustering a diversi livelli di dettaglio, senza dover

per forza conoscere la rete stradale sottostante, in modo che sia possibile definire l'unità fondamentale di clustering all'interno del framework in base alla granularità a cui vuole eseguire l'analisi.

Il dataset viene così mappato in una griglia formata da un insieme di celle definita come $G = \{C_0, C_1, C_2, ..., C_N\}$, e la cella diventa l'unità fondamentale di clustering al posto dei t-fragment previsti da NEAT. La dimensione della cella, cioè la porzione di spazio coperta, può essere definita in modo arbitrario in base alla granularità che si vuole utilizzare, e modificando a priori la dimensione delle celle si possono ottenere unità di cluster a grandezza variabile in base al valore specificato. Una cella di dimensione arbitraria, a livello spaziale ha otto vicini, uno per ogni punto cardinale. Data una griglia di celle G, per ogni cella, definiamo l'insieme dei suoi vicini a livello spaziale come $N_s(C) = \{dist(C, C_i) = 1, C_i \in G\}$ in cui $dist$ restituisce il numero di salti fra la cella C e C_i, il cui risultato dovrà essere pari a 1.

I vantaggi principali portati da questa estensione sono tre:

- Multigranularità: il clustering può essere effettuato a diversi livelli.
- Rimozione della necessità di avere conoscenza sulla rete stradale a cui si riferisce il dataset.
- Implementazione in ambito big data

Di conseguenza, avendo cambiato l'unita fondamentale di clustering, sono stati ridefiniti tutti gli elementi che ne conseguivano, dal concetto di vicinato alla definizione di un base cluster, e l'algoritmo è stato riadattato in funzione dell'estensione progettata, al fine di produrre cluster sempre significativi riguardo densità e flusso, permettendo eventuale multigranularità.

5.1.1 Segmentazione traiettorie

Le traiettorie vengono riorganizzate in celle: ogni punto viene mappato in una cella, ottenendo così la traiettoria come una sequenza di celle. Una cella

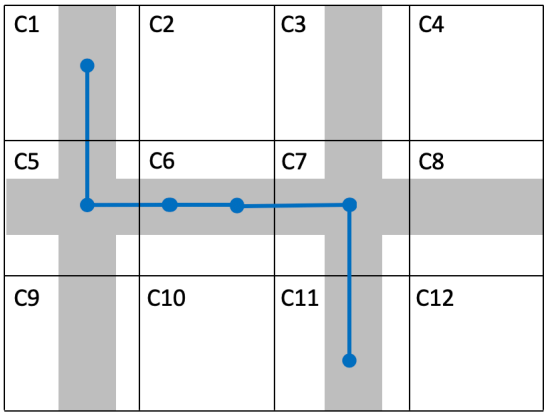
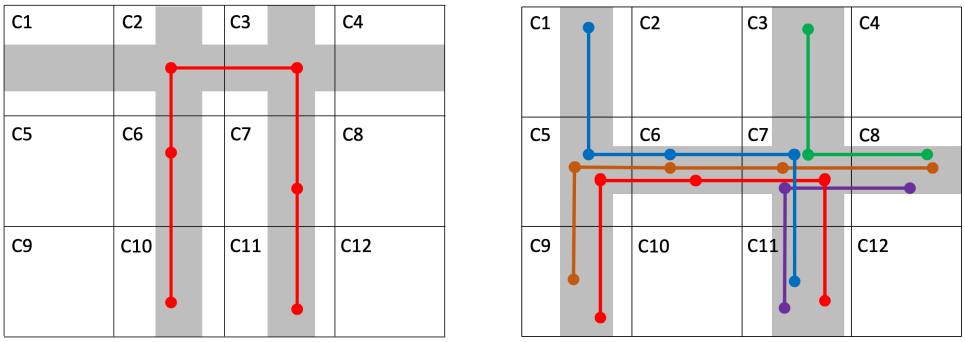


Figura 5.1: Esempio di segmentazione traiettoria in celle. Immagine tratta da [16] e riadattata in seguito.

può essere ripetuta più volte all'interno della traiettoria nel caso in cui siano presenti più punti, sia consecutivi che non, che ricadono all'interno della porzione di spazio coperta dalla cella. Viene mantenuta inoltre la sequenzialità temporale contenuta nel dato originale, dato che la trasformazione della traiettoria in sequenza di celle è una trasformazione dei punti solo a livello spaziale, mantenendo la sequenza a livello temporale inalterata. In figura 5.1 è riportato un esempio di segmentazione in celle di una traiettoria: la traiettoria può essere formalmente descritta come $TR = \{t_{id}, p_1^1\ p_5^2\ p_6^3\ p_6^4\ p_7^5\ p_{11}^6\}$ in cui il valore in pedice identifica la cella in cui viene mappato il punto, mentre il valore in apice rappresenta l'ordinamento dei punti all'interno della traiettoria.

5.1.2 Raggruppamento in base cluster

Un base cluster viene ridefinito nel seguente modo: dato un set di traiettorie, e C_i una cella della griglia, un base cluster è l'insieme delle traiettorie che contengono all'interno del loro percorso la cella C_i. Ogni cella può essere considerata un base cluster, viene rappresentata come C_i, ed è identificata da una coppia di coordinate, le quali sono univoche per ogni cella.



(a) Esempio di vicinato (b) Esempio di base e flow clusters

Figura 5.2: Esemplificazioni di possibili casistiche riguardo il calcolo del vicinato e la formazione dei cluster.

Cardinalità di un base cluster Definiamo come $PTr(C_i)$ una funzione che restituisce il set di traiettorie che contengono almeno una volta il base cluster C_i. La cardinalità di un base cluster è data da $|PT_r(C_i)|$.

Densità di un base cluster In questo caso viene ridefinita come il numero di traiettorie che passano per la cella stessa, quindi pari alla cardinalità. Di conseguenza il *dense-core* è dato dal base cluster a maggior densità, il quale ora rappresenta la cella all'interno della quale passa il maggior numero di traiettorie. Prendendo in considerazione l'immagine 5.2b il base cluster con più densità sarebbe la cella C_7, la quale ha cardinalità maggiore delle altre.

5.1.3 Definizione di vicinato

Flusso Definiamo la funzione *flusso* fra due base cluster $S(C_i)$ e $S(C_j)$, come $f(C_i, C_j, dir)$, in cui dir rappresenta la direzione di espansione. Il concetto di flusso calcolato con la seguente formula rappresenta il set di traiettorie che hanno viaggiato consecutivamente da una porzione di spazio ad una adiacente.

$$f(C_i, C_j, dir) = \begin{cases} \{T | p_i^t, p_j^{t'} \in T, t' - t = 1, C_j \in N_s(C_i)\}, se\ dir = avanti \\ \{T | p_i^t, p_j^{t'} \in T, t - t' = 1, C_j \in N_s(C_i)\}, altrimenti \end{cases}$$

Il flusso viene calcolato in modo diverso in base all'espansione che si sta effettuando. Nella prima espansione si controlla che ci sia continuità di movimento dall'ultimo base cluster aggiunto al flow cluster ai suoi vicini, per scegliere una direzione di movimento e mantenerla durante il processo di espansione. Nella seconda espansione invece si vuole mantenere la direzione scelta dalla prima, quindi si controlla che ci sia continuità di movimento da ogni base cluster vicino a quello per cui si cerca il vicinato.

Non avendo il vincolo della rete stradale, è importante verificare che una traiettoria passi da una cella ad una vicina ad essa consecutivamente, per mantenere coerenza e consistenza sia a livello spaziale che di flusso e di continuità di movimento.

Vicinato base cluster Durante il processo di formazione del flow cluster, un base cluster viene considerato candidato all'espansione se il flusso fra l'ultimo base cluster aggiunto e il candidato stesso è maggiore di zero. Formalmente possiamo definire, per l'ultimo base cluster aggiunto, l'insieme di base cluster candidati all'espansione come

$$N_f(C_i, dir) = \{C_j | f(C_i, C_j, dir) > 0, C_j \in G\}$$

in cui *dir* dipende da quale espansione si sta eseguendo. Ad esempio, nell'immagine 5.2a, considerando la cella C_{11}, l'insieme dei suoi vicini a livello di spazio e con una traiettoria in comune sarebbero $\{C_6, C_7, C_{10}\}$. Considerando però anche il fatto che ci debba essere effettivamente una traiettoria consecutiva fra C_{11} e i suoi vicini, l'insieme dei vicini considerati validi per una possibile aggiunta al flow-cluster viene ridotto a $\{C_7\}$, perché è l'unico fra i vicini il quale garantisce continuità di flusso e in cui c'è un vero spostamento da una cella all'altra.

5.1.4 Raggruppamento in flow cluster

La metodologia di raggruppamento in flow cluster avviene similmente a quella definita dall'algoritmo originale, tenendo conto della nuova definizione

di vicinato per il riconoscimento dei base cluster candidati. Anche utilizzando l'estensione con la griglia spaziale, scelto il base cluster iniziale rispettando il concetto di dense-core, ci si muove in direzione della cella più densa e con maggior continuità di flusso, continuando il processo iterativamente. Ad ogni iterazione, per l'ultimo base cluster aggiunto, si andrà a selezionare come nuovo elemento del flow cluster il vicino con il più alto valore di merging, fino al momento in cui per l'ultimo elemento non è presente nessun candidato valido. È importante notare che è stata scelta una direzione di movimento specifica nella prima espansione, la quale verrà mantenuta fino all'esaurirsi dei vicini in tale direzione.

Per quanto riguarda la seconda espansione invece, al fine di mantenere la direzione utilizzata alla prima espansione, come formalizzato in 5.1.3, si controlla il flusso in direzione inversa: non più dal base cluster di partenza verso quello candidato, ma viceversa, cioè da ogni candidato verso il base cluster di partenza. In questo modo il cluster ottenuto è rappresentativo di un percorso ad alta densità e con continuità di movimento in una direzione specifica.

Nella figura 5.2b, considerando C_7 come base cluster di partenza essendo quello a maggior densità, i possibili candidati come flow cluster ad esempio potrebbero essere $F = \{C_9C_5C_6C_7C_{11}\}$, oppure $F = \{C_9C_5C_6C_7C_8\}$: in base anche al valore di merging di C_{11} e C_8 verrà scelto l'uno piuttosto che l'altro.

Per quanto riguarda il calcolo del valore di merging per ogni candidato, sono stati mantenuti i fattori definiti originariamente con le rispettive formule, all'interno delle quali verranno usati i valori di densità, netflow, e cardinalità secondo le nuove definizioni date dall'estensione.

Cardinalità flow cluster Per ogni flow cluster possiamo definire la cardinalità $|PTr(F)|$ come il numero di celle contenute al suo interno.

5.2 Implementazione

In questa sezione vengono descritti i principali aspetti riguardanti l'implementazione dell'estensione dell'algoritmo descritto.

Oltre all'estensione applicata, l'algoritmo è stato adattato e di conseguenza implementato in ottica distribuita. In questo modo è possibile eseguire clustering di un dataset di traiettorie di grandi dimensioni e ottenere dei flussi non basati su un campione, ma su un elevato numero di dati più rappresentativo della realtà.

La parallelizzazione dell'algoritmo implica il fatto di dover ripensare il codice in ottica distribuita, e non in modalità sequenziale come previsto inizialmente, di conseguenza alcune fasi dell'algoritmo sono state modificate ed è stata cambiata la sequenza d'esecuzione di alcune computazioni. L'esecuzione della seconda fase del ragguppamento in flow cluster, è un processo ricorsivo ed ogni iterazione dipende dalla precedente, ma nonostante la sua struttura è possibile estrarre alcune operazioni al di fuori di questa fase parallelizzandone così il calcolo. Le operazioni eseguite in parallelo ed estratte dalla sua struttura principale riguardano il calcolo del vicinato per ogni cella, e la verifica dell'esistenza di una sequenza consecutiva fra due celle confinanti, vincolo necessario per essere considerato un vicino valido candidato al merging.

I motivi appena descritti hanno portato ad ottenere una diversa struttura dell'algoritmo come visibile in figura 5.3, e l'implementazione è stata di conseguenza divisa in una fase totalmente distribuita ed una centralizzata.

La terza fase prevista dall'algoritmo per il raffinamento dei flow cluster, non viene eseguita in quanto la computazione avverrà in modalità batch. In questo caso non si pone il problema di sfruttare eventuali merging di flussi simili che potevano risultare in caso di computazione di diverse porzioni di dati in momenti differenti, dato che l'intero dataset viene processato allo stesso momento.

5.2.1 Fase Distribuita

Nella prima fase dell'algoritmo, sono state implementate operazioni in modalità totalmente distribuita, per sfruttare la capacità di computazione disponibile e processare maggiori quantità di traiettorie.

Filtraggio traiettorie Le traiettorie vengono inizialmente filtrate, in modo da utilizzare l'algoritmo solo su dati consistenti e senza punti di rumore, sono rimosse infatti tutte le traiettorie che non rilevano movimento, ma tracciano le posizioni di persone ferme, e anche le traiettorie corte, le quali porterebbero ad un aumento della densità che potrebbe influire sulla bontà dei cluster trovati alla fine del processo.

Segmentazione traiettorie in celle Oltre al filtraggio, viene effettuata la trasformazione di ogni traiettoria da sequenza di punti ad una sequenza di celle. La grandezza delle celle può essere specificata a priori ed è stata utilizzata come unità di misura il metro, le traiettorie verranno segmentate di conseguenza, mappando ogni punto all'interno di una cella.

Calcolo vicinato e controllo continuità di flusso Il calcolo del vicinato viene eseguito in modalità distribuita invece che all'interno del processo di formazione di flow cluster, in modo da consegnare alla fase successiva direttamente per ogni base cluster i suoi vicini, tenendo conto anche del vincolo sulla continuità temporale fra celle vicine. Per ogni cella, con un operazione di join, si ricercano le celle che rispettano il vincolo di vicinato a livello spaziale, e per ogni coppia di celle, si verifica che sia presente almeno una traiettoria che abbia una sequenza consecutiva fra le due celle, ottenendo come risultato l'insieme delle celle confinanti che rispettano il vincolo di vicinato e che siano considerabili come candidate nella fase di merging.

Formazione base cluster Di conseguenza si esegue il raggruppamento delle varie celle per ottenere i base cluster. Per ogni cella si forma un base cluster,

Base Cluster	**Attributi**
latitudine, longitudine	• densità • |persone| • vicinato

Tabella 5.1: Struttura tabella base cluster

considerando le traiettorie che ricadono al suo interno, ed eseguendo un raggruppamento per ottenere anche l'insieme dei suoi vicini, calcolati dalla fase precedente.

Per ogni base cluster si calcola inoltre già il valore di densità come definito in precedenza, il quale servirà per ordinare i diversi base cluster e individuare il dense-core. La struttura dati prodotta da questa prima fase distribuita produce un insieme di base cluster con diversi parametri associati, come visibile in Tabella 5.1. Oltre agli attributi già citati, troviamo anche la cardinalità a livello di persone, intesa come il numero di persone transitanti in quella cella, il quale è diverso dal numero di traiettorie, assumendo il fatto che ad una persona possano corrispondere diverse traiettorie. Questo parametro non viene utilizzato esplicitamente nell'algoritmo, ma solo in fasi di valutazione e visualizzazione finale dei risultati ottenuti.

Ogni computazione eseguita nella fase distribuita è effettuata utilizzando SparkSQL, mappando l'intero dataset iniziale in un dataframe, ed eseguendo una serie di trasformazioni su di esso al fine di ottenere il risultato atteso da questa prima fase. Ogni trasformazione produrrà un nuovo dataframe, come da sua definizione, fino al concludersi della prima fase, in cui si mappa il risultato ottenuto in una struttura dati definita come mappa chiave valore, equivalente alla struttura visibile in Tabella 5.1 per iniziare la seconda fase.

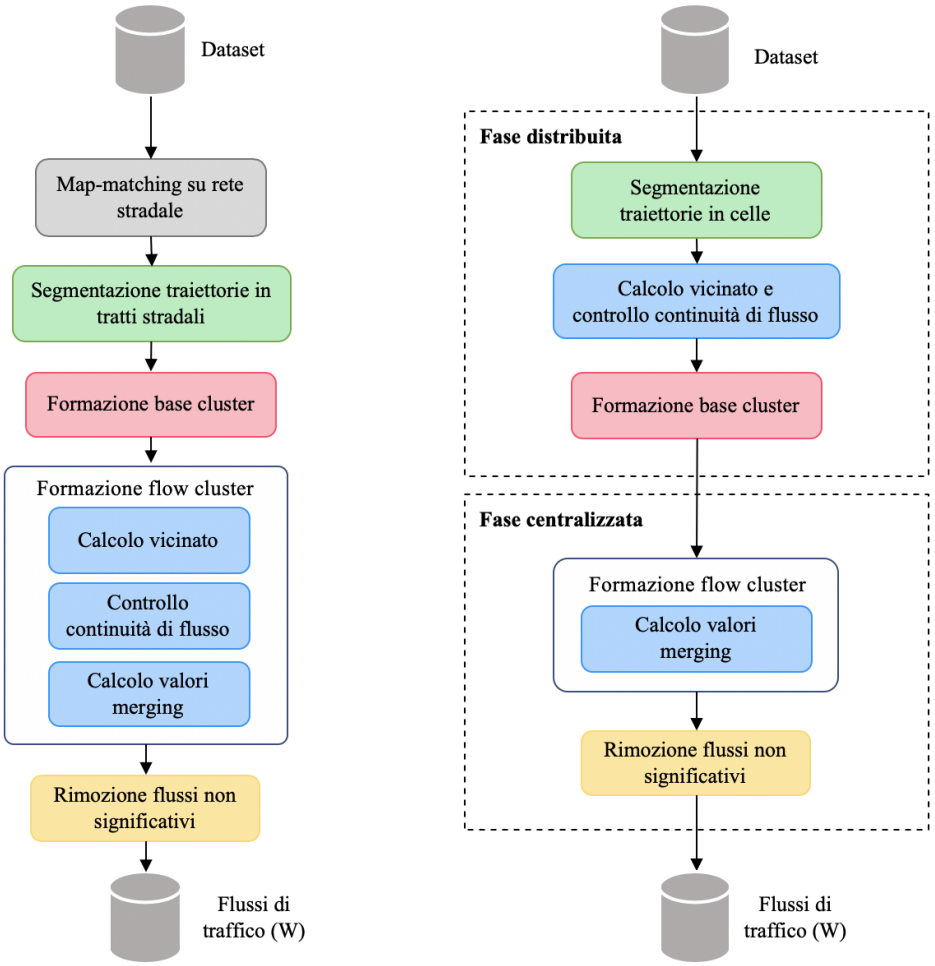
5.2.2 Fase Centralizzata

Le constatazioni riguardo la struttura ricorsiva e il fatto che ogni iterazione sia dipendente dalla precedente, durante il processo di raggruppamento in flow cluster, applicate sui Dataframe di SparkSQL che prevedono la creazione di un dataframe nuovo ad ogni trasformazione eseguita, stanno a significare che si creerà una catena di dipendenze tanto lunga quanto il numero di base clusters. Questo avviene perché ogni volta che si vorrà rimuovere il base cluster aggiunto al flow cluster dall'insieme dei base cluster, sarà necessario eseguire una trasformazione dell'insieme dei base cluster. Con una mole di dati big data, la dimensione dell'elenco delle dipendenze che si viene a creare supera il limite massimo supportato, creando problemi a livello di distribuzione. L'implementazione e la progettazione di una tecnica per la parallelizzazione di questa seconda fase, avrebbe previsto il calcolo dei flow cluster in porzioni diverse e totalmente parallele oltre ad una successiva fase di merging fra i cluster prodotti, snaturando così l'algoritmo iniziale. Queste motivazioni fanno si che la seconda fase dell'algoritmo, cioè la fase di raggruppamento in flow cluster venga eseguita in modalità centralizzata avendo così i dati tutti su una singola macchina.

L'algoritmo riguardante il raggruppamento in flow cluster delle celle prodotte dalla fase precedente è descritto in Algoritmo 2. È possibile notare che dato in input un set di base cluster prodotto dalla fase precedente, si ottiene in output l'insieme dei flow cluster prodotti. Il set di base cluster è organizzato come una mappa formata da un elenco di coppie chiave valore, con la stessa struttura visibile in Tabella 5.1, dal quale è possibile estrarre il base cluster più denso (riga 4). Dopo aver aggiunto il base cluster più denso al flow cluster che si sta formando e averlo rimosso dall'insieme totale dei base cluster (righe 5-6), è possibile procedere all'espansione del flow cluster nelle due direzioni (righe 7-8). Il processo di formazione dei base cluster continua fino al momento in cui tutti i base cluster sono stati utilizzati (riga 3) e nel momento in cui i flow cluster sono stati formati, si esegue una fase di filtraggio di eventuali flow cluster troppo corti rispetto alla media generale perché sono considerati non

significativi (righe 11-16).

La procedura di espansione del flusso non prevede più il calcolo del vicinato e il controllo del fatto che ci sia continuità di movimento, perché già calcolata in precedenza, quindi basta una semplice operazione di estrazione del vicinato (riga 18) per poi calcolare su ognuno di essi il valore di merging necessario (righe 20-22). Dopo aver calcolato i valori di merging è possibile ottenere quello con valore più alto (riga 23), il quale sarà il candidato migliore che rispecchia il flusso che si sta creando, di conseguenza viene aggiunto al flow cluster, prima di essere rimosso dall'insieme dei base cluster (righe 24-25), ed infine si continua l'espansione partendo dal base cluster appena aggiunto. La procedura, come previsto si fermerà nel caso in cui non ci siano vicini candidati validi per l'ultimo base cluster aggiunto al flow cluster (riga 19).



(a) NEAT (b) NEAT distribuito

Figura 5.3: Confronto della struttura fra i due approcci: il calcolo dei flow cluster è stato spezzato. Blocchi di colore uguale corrispondono ad operazioni concettualmente simili.

Algoritmo 2 Raggruppamento in flow cluster

Input:

$B = \{C_0, C_1, ..., C_M\}$ ▷ Un set di base cluster

$pesi = (w_q, w_k, w_v)$ ▷ Pesi per i fattori di merging

Output:

$W = \{F_0, F_1, .., F_Q\}$ ▷ Un set di flow cluster

1: $flowId = 0$ ▷ Identificatore del numero di flow cluster
2: $W = \emptyset$
3: **while** $B \neq \emptyset$ **do** ▷ Fino a che sono presenti base cluster
4: $C = densecore(B)$; ▷ Scegli il base cluster con densità più alta
5: aggiunta C a F_{flowId};
6: rimozione C da B;
7: F_{flowId} = espansioneFlusso$(C, dir, flowId, pesi)$;
8: F_{flowId} = espansioneFlusso$(C, dir, flowId, pesi)$;
9: $flowId++$;
10: **end while**

11: $avgCard$ = cardinalitàMedia(W); ▷ Cardinalità media dei flow cluster
12: **foreach** $F_i \in W$ **do** ▷ Per ogni flusso calcolato
13: **if** PTr$(F_i) < avgCard$ **then**
14: remove F_i from W ▷ Mantieni solo i flussi con cardinalità superiore alla media
15: **end if**
16: **end for**

17: **procedure** ESPANSIONEFLUSSO(C, dir, $flowId$, $pesi$)
18: N_f = estrazioneVicinato(C, dir)
19: **if** $N_f \neq \emptyset$ **then** ▷ Se esistono vicini validi
20: **foreach** $C_i \in N_f$ **do** ▷ Per ogni vicino
21: $SF(C_i)$ = calcoloValoreMerging$(C, C_i, pesi)$
22: **end for**
23: $SF(C_j) = \arg\max(SF)$ ▷ Scegli il vicino con valore più alto
24: aggiunta C_j a F_{flowId};
25: rimozione C_j da B;
26: F_{flowId} = espansioneFlusso$(C_j, dir, flowId, pesi)$
27: **end if**
28: **end procedure**

Capitolo 6

Testing

In questo capitolo viene analizzato il dataset utilizzato, presentandone le caratteristiche principali e di conseguenza sono descritti i risultati prodotti e le comparazioni fra i diversi flussi di traffico ottenuti. 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.

6.1 Dataset

Il dataset utilizzato contiene un insieme di traiettorie riferite a persone passanti per la città di Milano nell'arco di un anno. L'analisi viene effettuata solo sulle traiettorie circoscritte nell'area di Milano.

Le statistiche del dataset utilizzato sono presenti in Tabella 6.1, fra le quali troviamo il conteggio del numero di traiettorie pari a $12,5 \cdot 10^6$ per un totale di $2.95 \cdot 10^8$ punti. Il numero medio di punti all'interno di una traiettoria è 24, con un indice di dispersione di circa 98, di conseguenza saranno presenti traiettorie molto lunghe o corte rispetto alla media. Per quanto riguarda l'intervallo di campionamento della posizione, viene effettuato un rilevamento in media ogni 317 secondi, con una varianza di 445 secondi.

Pre-filtraggio Al fine di eliminare il più possibile dati considerati come rumore, sull'intero dataset viene eseguita a priori una fase di filtraggio. I criteri

Caratteristica	**Valore**
Punti GPS	$2.95 \cdot 10^8$
Traiettorie	$12,5 \cdot 10^6$
Media dei punti in una traiettoria	24 ± 98
Intervallo di campionamento medio posizione (secondi)	317 ± 445

Tabella 6.1: Statistiche dataset utilizzato

per il filtraggio applicati riguardano la lunghezza della traiettoria, e la velocità registrata durante il movimento, per analizzare solo traiettorie riferite a veicoli in movimento e di una lunghezza considerata significativa.

- **Velocità di movimento $\geq$ 20 km/h**
  Per evitare di considerare traiettorie formate da una sequenza di rilevamenti di posizioni ferme (ad esempio: persone all'interno di edifici o a piedi)

- **Lunghezza traiettoria $\geq$ 10 punti**
  Per evitare di considerare traiettorie di pochi punti che andrebbero ad aumentare la densità ma senza rappresentare un vero flusso di movimento, e che di conseguenza potrebbero inficiare sulla bontà del risultato.

6.2 Risultati e comparazioni

6.2.1 Multigranularità

Per quanto riguarda il concetto di multigranularità, a livello di test sono state effettuate diverse esecuzioni dell'algoritmo modificando la granularità dei flussi prodotti. Nei grafici in figura 6.1, è stata messa a confronto la grandezza delle celle con il numero di base cluster generati (6.1a), con il numero di flow cluster prodotti (6.1b) e con i tempi d'esecuzione dell'algoritmo (6.1c).

La dimensione più bassa della cella utilizzata è di 56×40 metri, identificata dal colore azzurro nei grafici, la quale produce il più elevato numero di

base clusters all'interno della mappa ed il più alto tempo d'esecuzione data la maggior quantità di dati. Il numero di flussi prodotto invece non risulta essere il più alto: la ragione è da ricercare nel fatto che il campionamento delle traiettorie non è sempre fine quanto la grandezza della cella, e verrà effettuata una segmentazione delle traiettorie troppo fitta, producendo una grande quantità di flussi con una lunghezza poco rilevante, i quali vengono eliminati effettuando la fase di filtraggio dei risultati descritta in sezione 5.2.2.

Allargando la dimensione delle celle a 222×160 metri, 333×240 metri e 556×399 metri, identificabili rispettivamente dai colori rosso, verde e bordeaux nei grafici, il numero di base cluster prodotti è inversamente proporzionale all'aumentare della dimensione della cella, così come il numero di flussi prodotti dato che si hanno meno celle da raggruppare, invece per quanto riguarda i tempi d'esecuzione tendono ad uniformarsi.

Il maggior numero di flussi si ottiene con una dimensione delle celle di 111×80 metri, partendo da un numero di base cluster pari a circa $23 \cdot 10^3$, visibili in figura 6.2. Questa dimensione è quella che riesce meglio a rappresentare le traiettorie come sequenza di celle, ed il tempo di esecuzione misurato è di 2 minuti e 43 secondi il quale è più simile al tempo registrato con celle di dimensioni elevate che al tempo d'esecuzione registrato utilizzando celle con dimensioni piccole. Per confrontare i risultati visuali con dimensioni diverse delle celle, in figura 6.3a è rappresentato il risultato ottenuto utilizzando celle di dimensione 222×160, ed in figura 6.3b sono rappresentati i cluster ottenuti celle di dimensione 56×40.

Più le celle hanno dimensione piccola più i cluster vengono mappati con maggior precisione sulle strade, mentre con celle a dimensione maggiore si perde precisione ottenendo flussi molto lunghi e poco rappresentativi del movimento reale all'interno di una città. Il fatto che con celle più strette ci siano meno flussi, motivato precedentemente, è visibile anche in figura 6.3b: la fase di filtraggio dei flow cluster in base alla lunghezza della media generale ha eliminato una notevole quantità di flussi poco significativi, i quali in linea generale risultano essere corti.

6.2.2 Risultati visuali

In ogni rappresentazione grafica dei risultati ottenuti, ogni flusso è identificato con un colore diverso e i base cluster che compongono ogni flusso sono collegati in sequenza per visualizzare il percorso prodotto. A livello grafico è possibile notare differenze tra i flussi in termini di spessore e di sfumatura di colore. Lo spessore indica il numero di traiettorie all'interno della cella, quindi un flusso più largo indica un percorso con una densità alta, rispetto a flussi più stretti, i quali si riferiscono ad un numero minore di traiettorie. La prima delle due espansioni è rappresentata da una sfumatura chiara, mentre la seconda espansione è rappresentata da una sfumatura scura, e l'insieme del gradiente di colore di ogni cluster è utile per identificare la direzione intrapresa, dalla sfumatura scura proseguendo verso la sfumatura chiara.

Nel dataset utilizzato non sono presenti dati riguardo il limite di velocità, come utilizzato nell'algoritmo originale. I pesi utilizzati durante il processi di clustering sono leggermente sbilanciati verso il fattore di flusso, rispettivamente con valori di 0.7 per il fattore di flusso e 0.3 per il fattore di densità, rispettando comunque il vincolo che la somma dei pesi sia uguale ad 1 in quanto il fattore di velocità non viene considerato.

La dimensione delle celle utilizzata per la generazione dei risultati proposti è di 111×80 metri, ed in figura 6.2 sono stati rappresentati i flussi ottenuti dal processo di clustering delle traiettorie.

Applicando ulteriori filtri all'insieme delle traiettorie, è possibile estrarre diverse sottoporzioni di dati, nello specifico per effettuare confronti qualitativi fra i flussi in diversi momenti della giornata o della settimana. Per ogni confronto vengono visualizzati i migliori 15 flussi, intesi come i primi 15 cluster. Prendendo i primi cluster creati si ha la certezza, data la definizione dell'algoritmo, che questi siano rappresentativi delle zone a maggiore densità di percorrenza all'interno del dataset.

Mattina - Sera Il confronto viene effettuato fra traiettorie registrate nell'arco della mattinata, con fascia oraria dalle 6 alle 12, paragonate alle traiet-

torie registrate nella fascia oraria dalle 18 alle 24. I flussi ottenuti tra questi due gruppi di traiettorie sono visibili in figura 6.4a per le traiettorie in fascia mattutina, ed in figura 6.4b per le traiettorie registrate in fascia serale.

Il cluster con più alta densità di traiettorie è identificato in entrambe le figure con una colorazione blu ed uno spessore ampio. Nel caso della mattina risulta essere quello che percorre lo svincolo autostradale di viale Certosa, mentre nel caso di traiettorie in fascia serale, il flusso più percorso risulta essere in zona stadio - ippodromo. Entrambe le tangenziali risultano essere tratti ad alta percorrenza, così come le arterie che dai quartieri periferici portano al centro della città. Analizzando il gradiente di ogni cluster, è possibile notare che la tangenziale ovest di Milano è percorsa in direzioni diverse nei due momenti della giornata, così come lo svincolo di viale Certosa, percorso in direzione Milano la sera, e in direzione opposta la mattina.

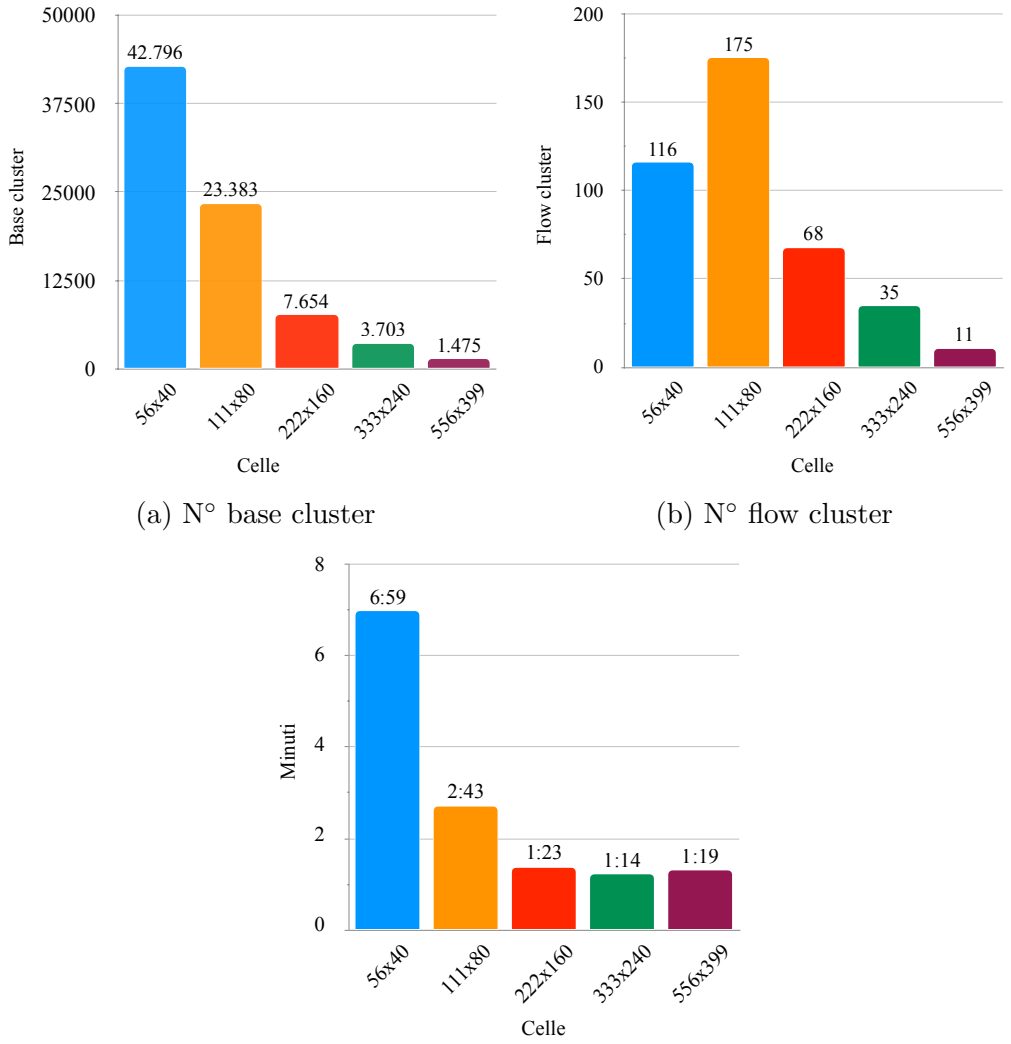
Giorni lavorativi - Weekend Un ulteriore confronto è stato effettuato comparando i flussi ottenuti per traiettorie registrate da lunedì a venerdì (in figura 6.5a), con flussi ottenuti da traiettorie registrate nei giorni sabato e domenica (in figura 6.5b).

Il flusso contenente la cella con densità più alta è sempre identificato da una colorazione blu e lo spessore più ampio, ed è lo stesso per entrambi i gruppi di giorni considerati. Fra i flussi più densi, nel weekend è presente una traccia consistente in zona Rho-Fiera, non presente invece nei giorni lavorativi. Allo stesso modo risultano visibili chiaramente una deviazione in direzione idroscalo e un raggruppamento notevole in zona stadio-ippodromo nei giorni del fine settimana come principale differenza fra i diversi momenti della settimana, probabilmente identificativi degli spostamenti verso zone ricreative e di ritrovo nei giorni non lavorativi.

6.3 Considerazioni e limiti

I cluster ottenuti sono rappresentativi di flussi di movimento ad alta densità all'interno della città di Milano, i quali rispondono agli scenari reali indicati

nel capitolo 4. Data la natura dell'algoritmo, ogni traccia trovata non prevede biforcazioni o un passaggio nella stessa zona più volte all'interno dello stesso percorso. Questo potrebbe essere un limite nel caso si voglia raggruppare persone con percorsi simili all'interno di una città. NEAT [7] esegue clustering di sotto-traiettorie, ed in questa tipologia di clustering si ignora il movimento complessivo di un veicolo o di una persona, perchè sotto-traiettorie della stessa persona possono appartenere a cluster differenti. È stata analizzata questa differenza studiando le motivazioni presentate in [20], in cui si paragona NEAT con un metodo di clustering di traiettorie intere chiamato TrajMob. In questo confronto viene posta l'attenzione sulla differenza fra questi due approcci: si evidenzia che NEAT è in grado di trovare cluster significativi in una rete stradale, a differenza di altri metodi noti di clustering di traiettorie basati solo sulla densità, ma viene perso il movimento complessivo di un singolo oggetto.



(a) N° base cluster

(b) N° flow cluster

(c) Tempi d'esecuzione

Figura 6.1: Grafici di confronto fra la grandezza delle celle utilizzata e il numero di base cluster e flow cluster ottenuti, oltre al confronto con i tempi d'esecuzione dell'algoritmo.

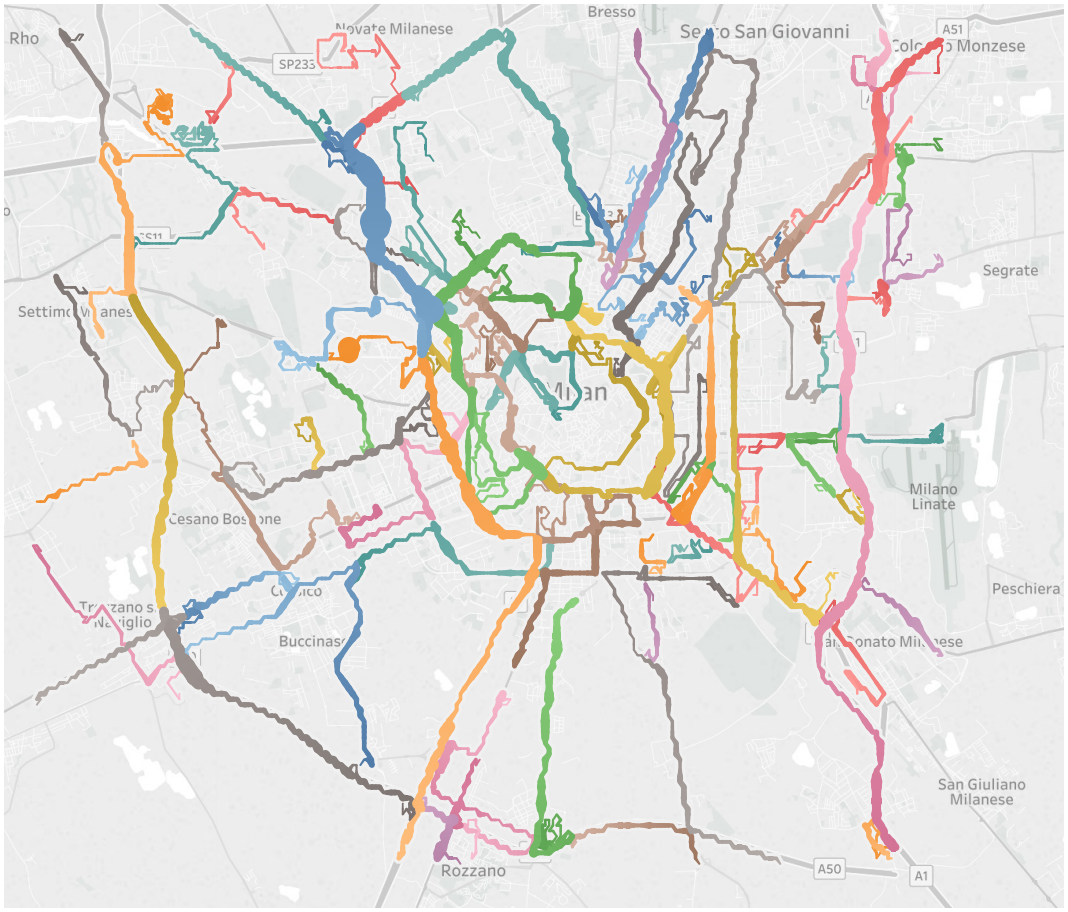
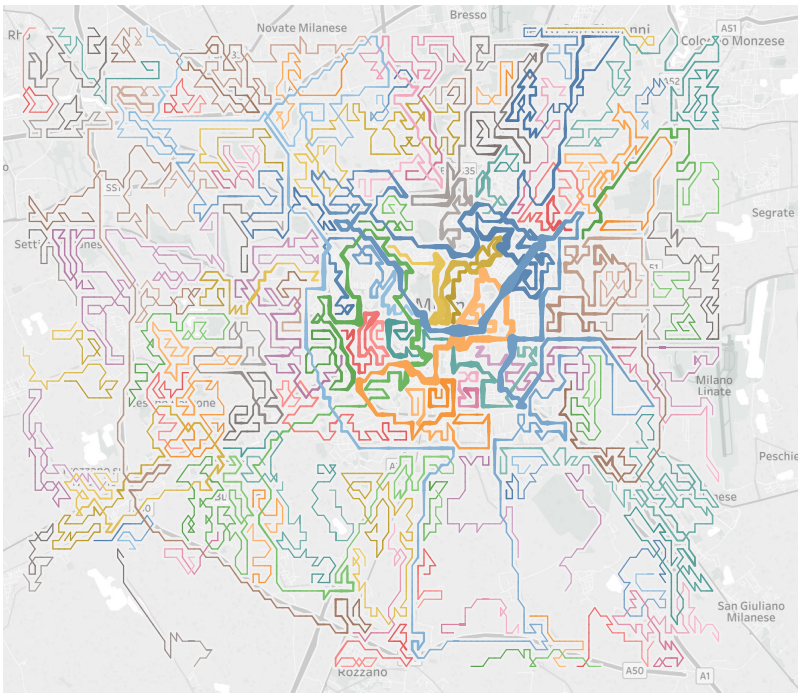
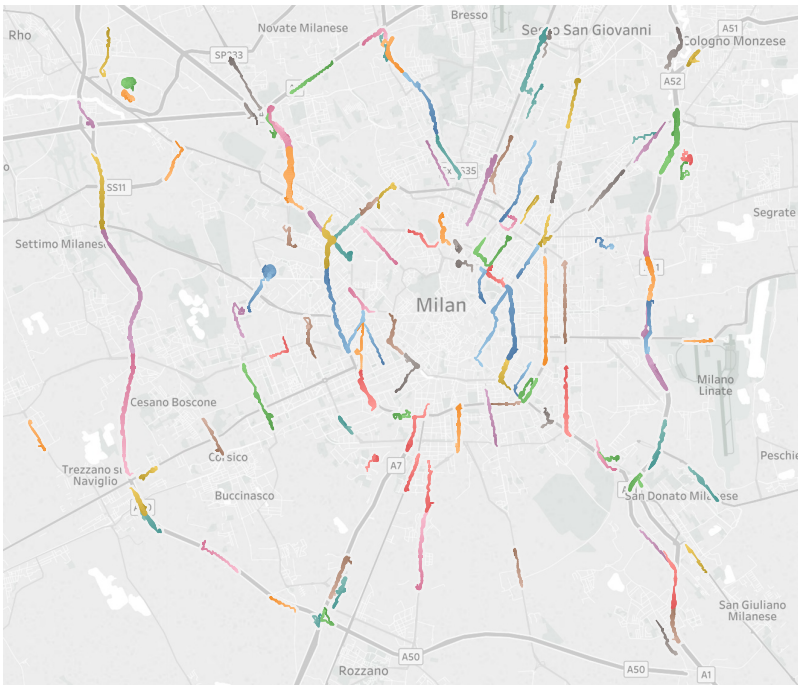


Figura 6.2: Rappresentazione complessiva dei flussi.

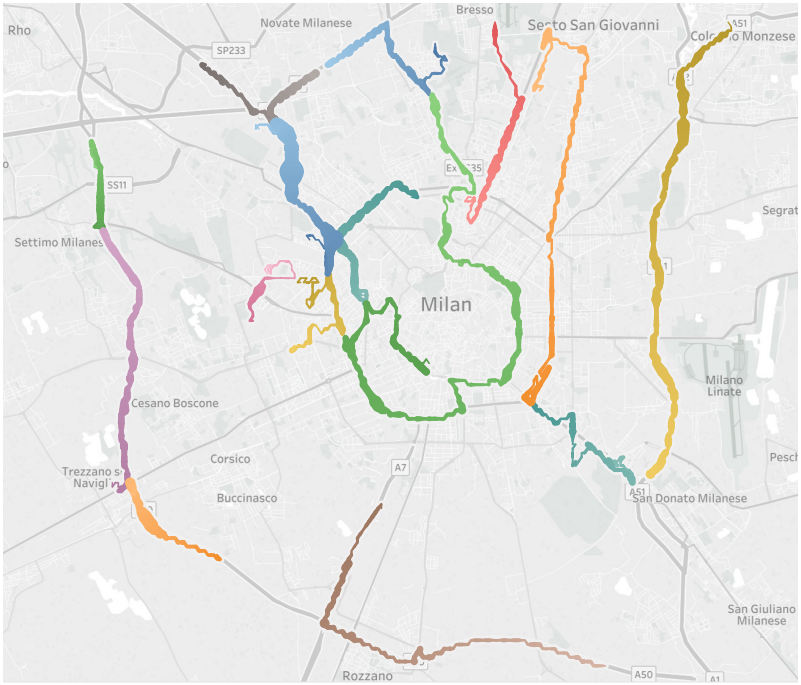


(a) Rappresentazione con celle di dimensione 222 × 160 metri.

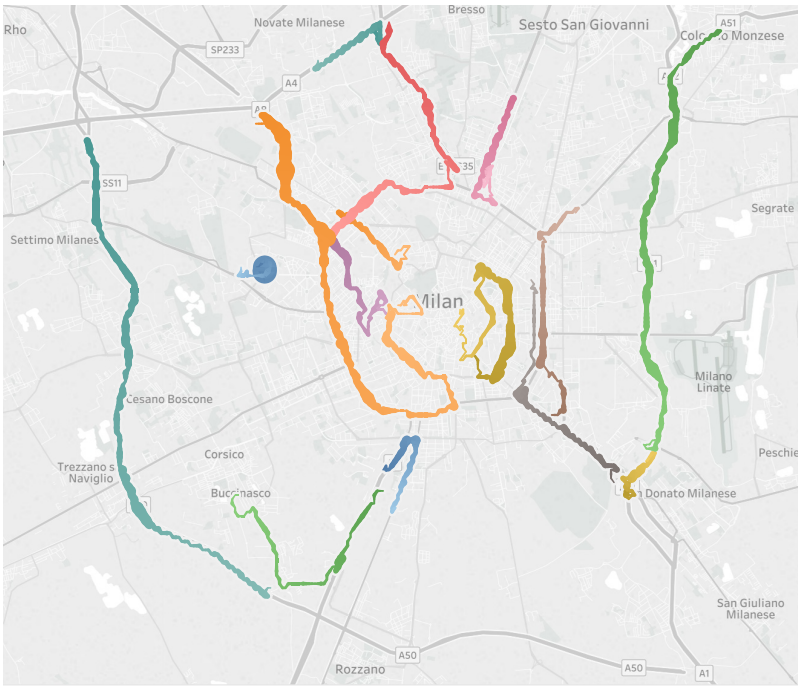


(b) Rappresentazione con celle di dimensione 56 × 40 metri.

Figura 6.3: Rappresentazione dei flussi calcolati sull'intero dataset a diverse granularità

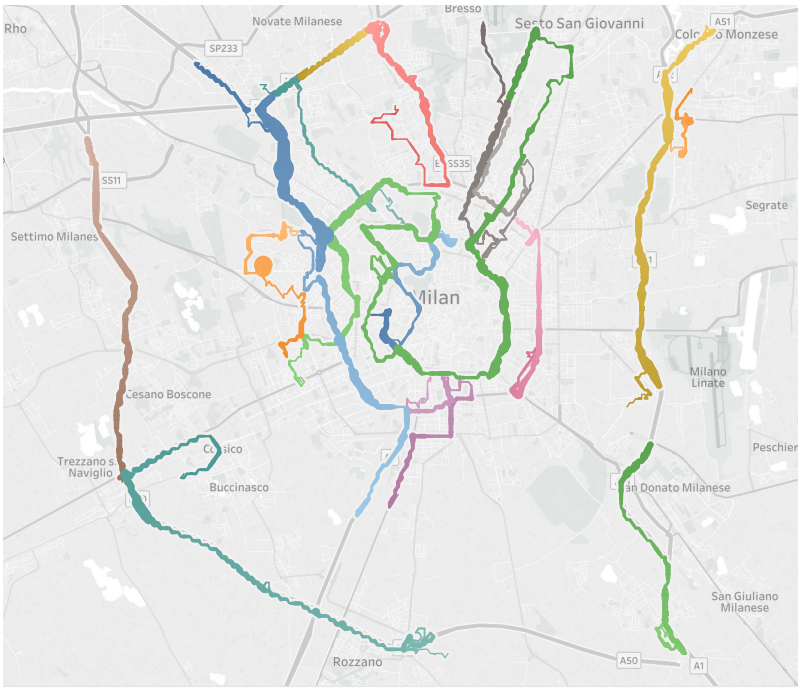


(a) Mattina (6-12)

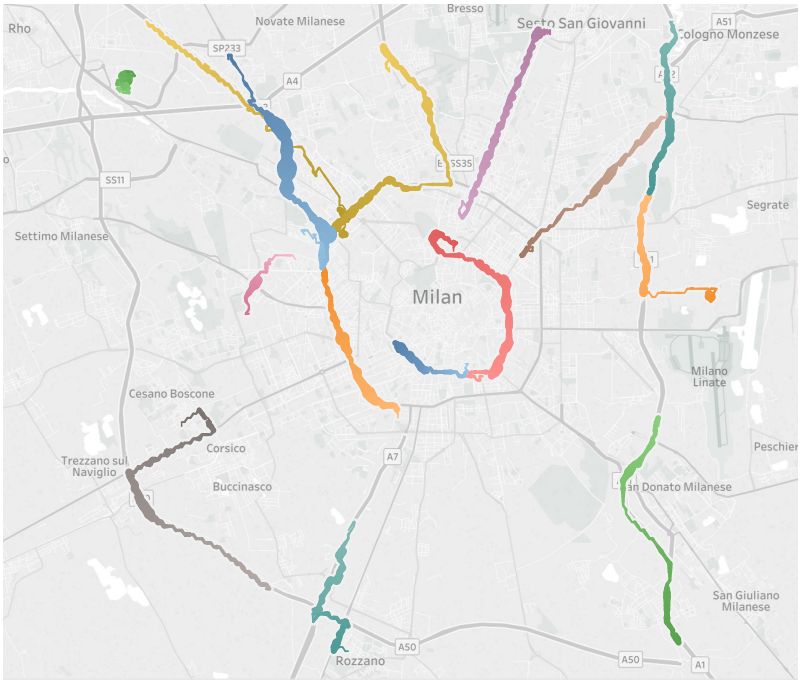


(b) Sera (18-24)

Figura 6.4: Rappresentazione grafica del confronto tra flussi ottenuti su porzioni di traiettorie riferite a momenti diversi della giornata.



(a) Giorni lavorativi



(b) Weekend

Figura 6.5: Rappresentazione grafica del confronto dei flussi ottenuti in diversi giorni della settimana.

Conclusioni

L'estensione applicata all'algoritmo scelto ha portato la possibilità di effettuare clustering a diversi livelli di granularità, rimuovendo un vincolo non banale di dover conoscere la rete stradale sottostante e dover eseguire sul dataset algoritmi per mappare le traiettorie sulle strade. E' stato introdotto il concetto di griglia, dividendo una porzione di spazio in celle di ugual dimensione. Più le celle avranno dimensione ridotta più si ottiene una mappatura diretta sulla rete stradale, ed il variare della dimensione della cella permette di ottenere unità di clustering più o meno grandi e di conseguenza risultati a diversi livelli di granularità.

La struttura iniziale dell'algoritmo è stata modificata ed adattata in ottica big data, in modo da permettere l'esecuzione distribuita della maggior parte delle operazioni, ed ottenere tempi d'esecuzione bassi su grandi quantità di dati come testimoniato dai risultati ottenuti.

I cluster prodotti dall'esecuzione dell'algoritmo sono flussi di traffico significativi sia a livello di traiettorie che rappresentano, sia a livello di continuità di movimento, ed è stata mantenuta coerenza a livello di direzione scelta durante il procedimento di espansione del flusso. I risultati ottenuti a livello qualitativo sono indicativi delle strade più percorse dai veicoli, ed evidenziano flussi compatibili con un ipotetico comportamento reale, come evidenziato nelle fasi di confronto dei risultati.

L'algoritmo mantiene una continuità di flusso durante la creazione del percorso, e di conseguenza non prevede eventuali biforcazioni del percorso o molteplici passaggi dalla stessa zona, e viene trascurata totalmente la direzionalità. Un flusso scelto in una direzione non può essere presente nella direzione oppo-

sta. Un altro limite può essere identificato nel fatto che non venga mantenuta la continuità del percorso complessivo di ogni oggetto, la motivazione di questa constatazione va ricercata nel fatto che viene effettuato clustering di sotto traiettorie, le quali per ogni persona potranno essere raggruppate in flussi diversi, perdendo così per ogni persona o veicolo il movimento complessivo.

I risultati ottenuti e i test effettuati evidenziano il fatto che i flussi prodotti rappresentano percorsi ad alta densità e continuità di movimento, e l'esecuzione dell'estensione prodotta avviene nell'ordine di un paio minuti su un dataset di elevate dimensioni, fornendo inoltre la possibilità di eseguire un procedimento di clustering di traiettorie il quale è in grado di identificare flussi di movimento significativi e coerenti con la rete stradale senza conoscere necessariamente la sua conformazione.

Sviluppi Futuri

L'algoritmo potrebbe essere reso ulteriormente completo dopo l'aggiunta del concetto di multigranularità e l'esecuzione in ambito big data. Un'estensione potrebbe essere effettuata prevedendo la possibilità di creare un flusso in direzione opposta ad uno già creato, producendo come risultati percorsi sovrapponibili e identificativi della direzione maggiormente utilizzata su una strada. Potrebbe essere utile anche modificare l'algoritmo in modo da prevedere eventuali biforcazioni del flusso nel caso in cui durante il processo di formazione del cluster siano presenti più direzioni che garantiscano un'elevata densità e continuità di movimento. In questo modo i flussi individuati sarebbero maggiormente indicativi del movimento completo delle persone, ed avvicinarsi ai risultati che si otterrebbero utilizzando un metodo di clustering di intere traiettorie.

Ringraziamenti

A conclusione del biennio di laurea magistrale e del lavoro di tesi, vorrei rivolgere un ringraziamento particolare al mio relatore, il professor M. Golfarelli, per l'opportunità concessa di svolgere la tesi in un ambito che stimola la mia curiosità, e per i consigli dati durante il lavoro effettuato. Un ringraziamento sentito va anche al mio co-relatore, il dott. M. Francia per avermi fornito supporto e preziosi consigli durante lo svolgimento del lavoro, oltre alla disponibilità in ogni occasione ce ne sia stato bisogno. Un ringraziamento va anche al dott. E. Gallinucci, presente all'interno del BIG group, per aver fornito il suo contributo e il suo tempo nei momenti di problem solving, oltre ai colleghi con i quali ho condiviso il percorso universitario e le fasi di elaborazione della tesi, in particolar modo a Federico per aver condiviso ogni momento universitario dal primo all'ultimo anno.

Un particolare ringraziamento è rivolto alla mia famiglia, per i consigli, la capacità di supportarmi ed ascoltarmi durante il percorso universitario, ed agli amici, per aver condiviso insieme i momenti che hanno caratterizzato questo percorso fino al raggiungimento del traguardo finale.

Mattia Oriani

Bibliografia

[1] Gaffney, S., Robertson, A., Smyth, P., Camargo, S., and Ghil, M., *Probabilistic Clustering of Extratropical Cyclones Using Regression Mixture Models*, Technical Report UCI-ICS 06-02, University of California, Irvine, Jan. 2006.

[2] Lee, J.G., Han, J., Whang, K.Y., *Trajectory clustering: a partition-and-group framework.*, ACM SIGMOD international conference on management of data, pp. 49–60, 2007.

[3] Z Deng, Y Hu, M Zhu, X Huang, B Du., *A scalable and fast OPTICS for clustering trajectory big data*, Cluster Computing (2015) 18: 549. https://doi.org/10.1007/s10586-014-0413-9

[4] Nanni, M., Pedreschi, D., *Time-focused clustering of trajectories of moving objects.*, J. Intell. Inf. Syst. 27, 267–289, 2006

[5] Lee, J.G., Han, J., Whang, K.Y., *Trajectory Clustering with Mixtures of Regression Models*, 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, San Diego, California, pp. 63–722, Ago. 1999.

[6] C Chang, B Zhou., *Multi-granularity Visualization of Trajectory Clusters Using Sub-trajectory Clustering*, IEEE International Conference on Data Mining Workshops, 2009

[7] B Han, L Liu, E Omiecinski., *NEAT: Road Network Aware Trajectory Clustering*, IEEE 32nd International Conference on Distributed Computing Systems, 2012

[8] HR Wu, MY Yeh, MS Chen., *Profiling moving objects by dividing and clustering trajectories spatiotemporally*, IEEE Transactions on Knowledge and Data Engineering (Vol. 25 , Issue: 11) ,Nov. 2013

[9] D. Kumar , H. Wu, S. Rajasegarar, C. Leckie, S. Krishnaswamy, M. Palaniswami., *Fast and Scalable Big Data Trajectory Clustering for Understanding Urban Mobility*, IEEE Transactions on Intelligent Transportation Systems (Volume: 19 , Issue: 11), Nov. 2018

[10] Y. Zheng., *Trajectory Data Mining: An Overview*, ACM Transactions on Intelligent Systems and Technology (TIST) - Volume 6 Issue 3, Mag. 2015

[11] Z Feng, Y Zhu., *A survey on trajectory data mining: Techniques and applications*, IEEE Access (Vol: 4), Page(s): 2056 - 2067, 2016

[12] C Hu, X Kang, N Luo, Q Zhao., *Parallel clustering of big data of spatio-temporal trajectory*, 2015 11th International Conference on Natural Computation (ICNC), Aug. 2015

[13] D Xia, B Wang, Y Li, Z Rong, Z Zhang., *An Efficient MapReduce-Based Parallel Clustering Algorithm for Distributed Traffic Subarea Division*, Discrete Dynamics in Nature and Society, 2015

[14] Wisdom, M. J., Cimon, N. J., Johnson, B. K., Garton, E. O., and Thomas, J. W., *Spatial Partitioning by Mule Deer and Elk in Relation to Traffic*, Trans. of the North American Wildlife and Natural Resources Conf., Washington, pp. 509–530, Mar. 2004

[15] Powell, M. D. and Aberson, S. D., *Accuracy of United States Tropical Cyclone Landfall Forecasts in the Atlantic Basin (1976-2000)*, Bull. of the American Meteorological Society, 2001

[16] B Han, L Liu, E Omiecinski., *Road-Network Aware Trajectory Clustering: Integrating Locality, Flow, and Density*, IEEE Transactions on Mobile Computing (Volume: 14 , Issue: 2), Feb. 2015

[17] M. Weber., *On map matching of wireless positioning data: a selective look-ahead approach,*, GIS, '10

[18] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise* In Proc. 2nd Int'l Conf. on Knowledge Discovery and Data Mining, Portland, Oregon, pp. 226–231, Ago. 1996.

[19] Grunwald, P., Myung, I. J., and Pitt, M., *Advances in Minimum Description Length: Theory and Applications* MIT Press, 2005.

[20] B Han, L Liu, E Omiecinski., *A Systematic Approach to Clustering Whole Trajectories of Mobile Objects in Road Networks*, IEEE Transactions on Knowledge and Data Engineering (Volume: 29 , Issue: 5), Mag. 2017

[21] T. Eiter and H. Mannila., *Computing discrete fréchet distance.* Tech. Report CD-TR 94/64,Information Systems Department, Technical University of Vienna, 1994

[22] SINTEF. (2013, May 22). Big Data, for better or worse: 90% of world's data generated over last two years. ScienceDaily. Retrieved March 11, 2019 from www.sciencedaily.com/releases/2013/05/130522085217.htm, (Ultima visita: Feb 2019)

[23] *Dynamic Time Warping. In: Information Retrieval for Music and Motion. Springer*, Berlin, Heidelberg, 2007

[24] Pang-Ning Tan, Michael Steinbach, Vipin Kumar., *Introduction to Data Mining.* Pearson International, 2006.