Alma Mater Studiorum — Università di Bologna
Campus di Cesena

Dipartimento di Informatica — Scienza e Ingegneria
Corso di Laurea magistrale in Ingegneria e Scienze Informatiche

Clustering di traiettorie in ambito Big Data

Tesi in Data Mining

Presentata da:
Federico Naldini
Matricola 0000852918

Relatore:
Prof. Matteo Golfarelli

Correlatore:
Dott. Matteo Francia

III sessione di laurea
Anno Accademico 2019–2020

Parole chiave

Trajectory mining

Trajectory clustering

Big data

Alla mia famiglia

Sommario

Uno dei trend più interessanti del momento è l'analisi e mining dei dati di traiettoria. Questa categoria di dati si compone principalmente delle tracce di movimento generate dalle più svariate categorie di dispositivi. Una traiettoria può essere interpretata come il cambiamento della posizione di un utente o oggetto nello spazio rispetto al tempo. Nell'ambito dell'analisi di traiettorie, le tecniche di clustering possono essere impiegate con diversi obbiettivi, come ad esempio la ricerca delle strade più frequentate o la profilazione degli utenti. Altrettante potenzialità sono racchiuse nella ricerca di itemset frequenti su dati di traiettoria. A metà tra questi due approcci si colloca l'analisi dei co-movements pattern. I pattern di co-movimento identificano quei gruppi di utenti che hanno viaggiato assieme per un certo periodo significativo di tempo. La ricerca di questi gruppi può estrarre diverse informazioni, come ad esempio le abitudini di un utente sulla base dei gruppi di appartenenza e dell'orario del giorno o ancora il mezzo di trasporto utilizzato da una certa categoria di utenti.

Obiettivo del lavoro di questa tesi è l'analisi di due algoritmi per la ricerca di pattern di co-movimento in ambito big data. Il primo è SPARE, framework descritto in letteratura, che permette di ricercare diversi pattern di movimento grazie a un mix delle tecniche di clustering di traiettorie e quelle di mining di itemset frequenti. L'altro algoritmo invece è CUTE, nuovo approccio definito e implementato in questo lavoro di tesi. CUTE si pone come framework di clustering sovrapposto basato su un insieme di dimensioni personalizzabili, che sfrutta le tecniche di colossal itemset mining per ricercare gruppi di movimento sulle dimensioni specificate. La struttura di CUTE è adattabile alla ricerca di pattern di co-movimento specificando le dimensioni spazio temporali come dimensioni su cui eseguire la ricerca.

Indice

Introduzione

La sempre più grande abbondanza di dispositivi in grado di monitorare la posizione degli oggetti ha portato a un'esplosione nelle quantità e qualità di dati di traiettorie. Enormi quantità di traiettorie, nella forma di sequenze spazio-temporali di punti, sono registrate ogni momento da chip di controllo di animali, dispositivi GPS su veicoli o su piattaforme wearable. Questa grande abbondanza di dati rende possibili diverse tipologie di analisi su dati di traiettoria, come ad esempio la pianificazione del traffico, l'analisi di movimenti di animali e profilazione di un utente sulla base dei suoi movimenti. Questi sono solo alcuni esempi di applicazioni del trajectory data mining. Scopo di questo ambito del data mining è l'applicazione di algoritmi per estrarre informazione da dati di traiettoria. La conoscenza così creata può poi essere impiegata per il miglioramento della qualità della vita del singolo individuo o a supporto delle decisioni in ambito governativo o industriale.

Fra gli scenari più interessanti dell'analisi di traiettorie è l'identificazione di pattern di co-movimento. Si definisce pattern di movimento un gruppo di oggetti che hanno viaggiato assieme per un certo periodo di tempo. Questa ricerca è direttamente collegata al clustering di traiettorie, il cui obbiettivo è suddividere un insieme di traiettorie in gruppi secondo un determinato criterio. Nel caso in cui il criterio di similarità si concentri sui luoghi visitati, allora sarà possibile dedurre dai raggruppamenti quali possono essere le strade o i luoghi più frequentati. Se invece l'analisi si pone sugli oggetti e i gruppi che formano nel tempo, allora sarà possibile identificare certe categorie di utenti che viaggiano assieme e i loro comportamenti comuni.

Il lavoro presentato in questa tesi può essere diviso in quattro passi. In primo luogo è stato condotto uno studio sullo stato dell'arte del trajectory data mining, soffermandosi sulla definizione di clustering e frequent itemset mining, prestando particolare attenzione alle loro applicazioni all'interno dell'analisi di traiettorie.

Successivamente è stata trattata la ricerca di pattern di co-movimento, con particolare attenzione al framework SPARE. Di questo framework, sviluppato in ottica big data, sono state analizzati a fondo funzionamento, potenzialità e limiti.

Una volta terminata questa analisi è stato condotto lo studio e la realizzazione di un nuovo framework, chiamato CUTE, per la ricerca di pattern di movimento. A differenza di quanto presente in letteratura, CUTE propone un nuovo approccio distribuito basato sulla divisione dello spazio di ricerca in celle sulla base di un sistema di riferimento arbitrario. Lo spazio di ricerca in questione viene composto da un numero custom di dimensioni. A questo

proposito non ci sono vincoli: possono essere impiegate solo dimensioni spazio-temporali, solo semantiche o un misto tra le due categorie. Su queste viene poi eseguita una ricerca mediante tecniche di mining di dati ad alta dimensionalità.

Infine sono stati confrontati i due framework sopracitati, analizzando pregi e svantaggi in determinate situazioni tramite un confronto sui risultati ottenuti.

Alla luce di questo, il documento è strutturato come segue: nel capitolo 1 vengono fornite la definizione di traiettoria e dei concetti ad essa collegati, successivamente è analizzato il problema del clustering. Questo viene poi declinato nelle applicazione relative ai dati di traiettoria. Ciò avviene analogamente anche per il frequent itemset mining. Nel capitolo 2 sono brevemente elencate le tecnologie utilizzate durante lo sviluppo dei due framework oggetto della tesi. Il capitolo 3 presenta il problema della ricerca dei pattern di movimento, successivamente espone il framework SPARE per l'individuazione di questi pattern. Il problema del mining di traiettorie basato su dati ad alta dimensionalità e la sua applicazione nell'algoritmo CUTE sono oggetto del capitolo 4. Il capitolo 5 mostra quali sono stati i test effettuati e i dataset utilizzati per questi, sono inoltre presenti le interpretazioni dei risultati tra i confronti di SPARE e CUTE. Nel capitolo 6 infine sono tratte alcune conclusioni sul lavoro svolto e potenziali punti di partenza per ricerche successive.

1 Problematica

Obbiettivo di questo primo capitolo è presentare gli ambiti dell'analisi di traiettorie toccati da questa tesi. Saranno presentati due approcci: in primo luogo verrà trattato il clustering, successivamente il frequent itemset mining. Per ciascuno di questi due ambiti verranno esposti le principali caratteristiche e le applicazioni nella ricerca di pattern di movimento.

1.1 Dati di traiettoria

Negli ultimi anni, la grande presenza di dispositivi in grado di catturare la posizione di un oggetto nel tempo e le sue variazioni hanno prodotto enormi quantità di dati. L'analisi di questi dati, resa possibile dalle moderne tecnologie Big Data, apre molteplici possibilità, come ad esempio la ricerca dei flussi di traffico all'interno di un territorio cittadino, oppure l'individuazione dei luoghi più visitati da una certa categoria di utenti, o ancora il riconoscimento di gruppi di oggetti che si muovono assieme all'interno di un certo spazio e tempo.

1.1.1 Definizione di traiettoria

Dato il grande numero di potenziali fonti per i dati, è necessario ricondurre questi ultimi a una formulazione comune così da poter sfruttare al meglio le loro potenzialità espressive. La rappresentazione più basilare consiste nel considerare una traiettoria come l'insieme delle posizioni spaziali dei punti che la compongono associando a ciascuno l'istante temporale in cui è stato registrato. Questa formulazione prende il nome di traiettoria grezza (*raw trajectory*), vedi figura 1.1. Andando a formalizzare quanto detto sopra:

Definizione 1.1.1 (Traiettoria)**.** Si definisce una traiettoria grezza tr una sequenza temporale di punti $p_t, p_{t'}, \dots, p_{t''}$ tale che ogni punto p_t è composto da una coppia di coordinate spaziali $(latitude, longitude)$ e un tempo t.

Tale formulazione può essere successivamente complicata, ad esempio adottando una scala unica tra le diverse traiettorie per lo spazio e una per il tempo, oppure aggiungendo ulteriori informazioni, come ad esempio la direzione o altri attributi dell'oggetto che la genera.

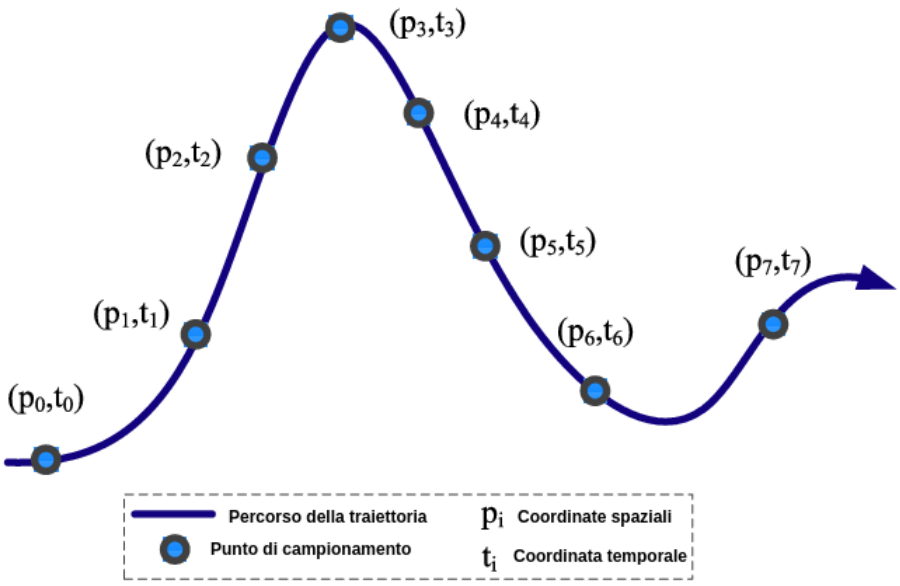


Figura 1.1: Esempio di traiettoria, Fonte: [1]

Per aumentare l'espressività della singola traiettoria, può essere utile definire il concetto di sotto-traiettoria, o *subtrajectory*. Intuitivamente una sotto-traiettoria non è altro che un segmento di una traiettoria relativo a un certo sottoinsieme dello spazio-tempo coperto da quest'ultima.

Definizione 1.1.2 (Sotto-traiettoria)**.** Date due traiettorie $tr_{\mathrm{i}}, tr_{\mathrm{j}}$ si definisce tr_{i} sotto-traiettoria di tr_{j} se $\forall p_{t_{\mathrm{i}}} \in tr_{\mathrm{i}}, p_{t_{\mathrm{i}}} \in tr_{\mathrm{j}}$.

A queste definizioni va aggiunto il concetto di sistema di riferimento: si definisce un sistema di riferimento un insieme di quanti completo e continuo all'interno di una regione spazio-temporale. I sistemi di riferimento sono fondamentali nel determinare le dimensioni spazio-temporali dei punti delle varie traiettorie, un esempio su tutti può essere una scala di riferimento espressa in coordinate polari. Tuttavia è possibile definire anche sistemi di riferimento che utilizzano metriche diverse dalle coordinate sopracitate, ottenendo così diversi effetti sulla rappresentazione dei dati. Una traiettoria rappresentata secondo le coordinate di un certo sistema di riferimento si definisce *trajectory abstraction* o astrazione di traiettoria:

Definizione 1.1.3 (Astrazione di traiettoria)**.** Data una traiettoria grezza tr e un sistema di quanti spazio-temporali $ST = \{q_1, \dots q_n\}$, un'astrazione di traiettoria è definibile come la sequenza di quanti $\{q_i, \dots q_j\}$ ottenuti esprimendo la traiettoria grezza tr sul sistema ST .

1.1.2 Metriche di distanza

Una volta definito che cos'è un dato di traiettoria, occorre mettere in chiaro che cosa distingue una traiettoria da un'altra. Per confrontare due traiettorie, sarebbe sbagliato

applicare le metriche di similarità per dati a bassa dimensionalità, poiché gli oggetti in movimento producono dati complessi e con particolari correlazioni tra le dimensioni. Il problema risulta quindi decisamente articolato: una buona metrica di similarità deve tenere conto non solo dei singoli punti, ma anche della traiettoria nella sua interezza. A questo va sommata la diversa lunghezza tra i due soggetti del confronto. In letteratura sono presenti diversi metriche che possono essere impiegate per confrontare due traiettorie:

La prima tra tutte le misure è la distanza euclidea. Grazie alla sua complessità lineare permette di gestire dati ad alta dimensionalità. Date due traiettorie tr_{i} e tr_{j} di lunghezza n e dimensioni p, la loro distanza euclidea si definisce come:

$$dist(tr_i, tr_j) = \frac{1}{n} \sum_{k=1}^{n} \sqrt{\sum_{m=1}^{p} (a_k^m - b_k^m)^2} \tag{1.1}$$

La metrica tuttavia non è esente da difetti: è molto sensibile al rumore e richiede che le traiettorie siano uguali per numero di punti e dimensioni, inoltre anche l'intervallo di campionamento temporale deve coincidere. Questi limiti sono abbastanza difficili da aggirare quando si processa un dataset reale.

Per superare questi problemi, sono disponibili diverse varianti della distanza euclidea: una su tutte è la *Principal Component Analysis Plus Euclidean Distance* (PCA + distance) [2]. Questa tecnica prima riduce le dimensioni spaziali ad una sola, successivamente esegue un'analisi PCA per convertire ogni traiettoria in k coefficienti; a questo punto viene calcolata la distanza euclidea tra le traiettorie così trasformate. L' equazione (1.2) mostra la formula per il calcolo della distanza PCA + distance: a_k^c e b_k^c rappresentano i k coefficienti nello spazio bidimensionale delle traiettorie tr_i,tr_j.

$$dist(tr_i, tr_j) = \sqrt{\sum_{m=1}^{p} (a_k^c - b_k^c)^2} \tag{1.2}$$

Questa variazione mantiene gli stessi punti di forza della versione base della metrica, in più consente una maggior resistenza al rumore.

La distanza di Hausdorff [3] è un alternativa a quella euclidea. Date due traiettorie, per ogni punto di una viene calcolato il più vicino punto dell'altra e la distanza tra i due. Il valore della distanza di Hausdorff corrisponde alla massima distanza calcolata nel passo precedente (equazione (1.3)).

$$\begin{aligned} D(tr_i, tr_j) &= \max\left(h(tr_i, tr_j), h(tr_j, tr_i)\right) \\ where\ h(tr_i, tr_j) &= \max_{a \in tr_i} \left(\min_{b \in tr_j} \left(dist(a, b)\right)\right) \end{aligned} \tag{1.3}$$

Nella formula h rappresenta la distanza di Hausdorff diretta, ovvero dai punti di una traiettoria verso l'altra; d invece sta per la distanza euclidea tra due punti. Il calcolo di entrambe le distanze dirette permette di gestire traiettorie con numero di punti differente tra

loro. Questa metrica risulta robusta rispetto all'influenza causata da particolari distribuzioni di punti, ma allo stesso tempo è sensibile al rumore.

Longest Common Sub Sequence [4] affronta il problema con un approccio diverso: invece che calcolare la distanza fra i punti delle due traiettorie, computa la più lunga sotto-sequenza comune ad entrambe. La lunghezza di quest'ultima determina la vicinanza: più il valore è alto, quindi la sotto-sequenza aumenta di dimensioni, più le due traiettorie sono vicine. Essendo impossibile una coincidenza assoluta dei punti tra due traiettorie sono definite due soglie, ϵ e σ che rispettivamente modellano la tolleranza rispetto all'asse x e y. L' equazione (1.4) definisce la metrica in termini formali:

$$D(tr_i, tr_j) = \begin{cases} 0 & n = m = 0 \\ 1 + LCSS_{\epsilon\sigma}(Head(tr_i), Head(tr_j)) & |a_i^x - a_j^x| \leq \epsilon \wedge |a_i^x - a_j^x| \leq \sigma \\ \max\left(LCSS_{\epsilon\ \sigma}(tr_i, Head(tr_j)),\right. & \\ \qquad \left. LCSS_{\epsilon\ \sigma}(Head(tr_i), tr_j)\right) & altrimenti \end{cases} \tag{1.4}$$

LCSS può essere calcolata in modo ricorsivo, inoltre consente una certa tolleranza rispetto alle deviazioni nei dati, ciò consente una buona efficienza nei dataset reali. Il maggior limite della metrica sta nella definizione dei parametri ϵ e σ in problemi complessi.

Dynamic time warping (DWT) [5] pone il focus sulla dimensione temporale rispetto a quelle spaziali, come accadeva nelle metriche precedenti. Lo scopo è trovare l'allineamento ottimo tra due traiettorie dati certi vincoli. DWT resiste bene alle differenti lunghezze tra traiettorie: obbiettivo della misura è infatti ricercare il percorso a cui assimilare le traiettorie che abbia il minor coefficiente di distorsione calcolato sulle trasformazioni subite dalle traiettorie.

$$D_d(tr_i, tr_j) = \begin{cases} 0 & n = m = 0 \\ \infty & n = 0 \ || \ m = 0 \\ dist(a_i^k - a_j^k) + \min \begin{cases} D_d(Rest(tr_i), Rest(tr_j))) \\ D_d(tr_i, Rest(tr_j)) \\ D_d(Rest(tr_i), tr_j) \end{cases} & altrimenti \end{cases} \tag{1.5}$$

L' equazione (1.5) definisce DWT: la distanza è calcolata in maniera ricorsiva sommando ad ogni passo la distanza degli elementi corrispondenti delle due sequenze. La distanza *dist* rappresenta la distanza euclidea tra due punti mentre *Rest* indica il segmento di traiettoria non ancora esplorato. Il termine di questo calcolo avviene quando entrambe le traiettorie sono state esplorate. DWT assicura il rispetto dell'ordine tra i punti delle traiettorie, inoltre introducendo un principio di scalabilità locale della dimensione temporale, riesce

a gestire scale temporali differenti tra le traiettorie. DWT tuttavia richiede continuità all'interno dei punti, è sensibile al rumore e a sotto-traiettorie molto distanti tra loro.

Infine si tratta della metrica di Fréchet [6], la quale considera in contemporanea sia la dimensione temporale che quella spaziale. Date due traiettorie di uguale lunghezza n, si calcola la distanza euclidea tra i punti aventi stessa posizione all'interno delle due traiettorie: il valore più alto corrisponde alla distanza di Fréchet. La formula per calcolare la distanza di Fréchet è mostrata nell' equazione (1.6):

$$D_f(tr_i, tr_j) = \min ||C|| \text{ dove } ||C|| = \max_{k=1}^{K}(dist(a_i^k, b_j^k)) \tag{1.6}$$

D_f rappresenta la misura della distanza, K è il valore minore di lunghezza tra tr_i e tr_j, a_i^k e b_j^k sono i k-esimi punti di tr_i,tr_j e infine $dist$ è la misura della distanza euclidea. Qualora le due traiettorie divergano come dimensioni, si eseguirà questo calcolo su tutte le possibili sotto-traiettorie di lunghezza n generabili dalla traiettoria più lunga. La distanza di Fréchet considera la traiettoria nella sua continuità, per questo motivo è molto sensibile agli outlier.

Le diverse misure della distanza hanno costi computazionali differenti tra di loro, che sono riassunti nella tabella 1.1.

Misura	Costo computazionale
Euclidea	O(n)
PCA + Euclidea	O(n)
Hausdorff	O($n * m$)
LCSS	O($n * m$)
DWT	O($n * m$)
Fréchet	O($n * m$)

Tabella 1.1: Costi computazionali delle metriche di similarità

1.2 Clustering

Il clustering, o analisi di raggruppamento, è una tecnica con lo scopo di aggregare i dati in *cluster*, o gruppi, tali che i dati all'interno di un gruppo siano più simili tra loro rispetto a quelli all'esterno [7], [8]. Questa tecnica è definita come non supervisionata, ovvero non è necessario avere una conoscenza sui gruppi presenti nel dataset.

Le principali categorie di clustering per elaborare dati statici a bassa dimensionalità sono cinque [9]: metodi basati sulle partizioni, sulla gerarchia, densità, griglia e infine modello.

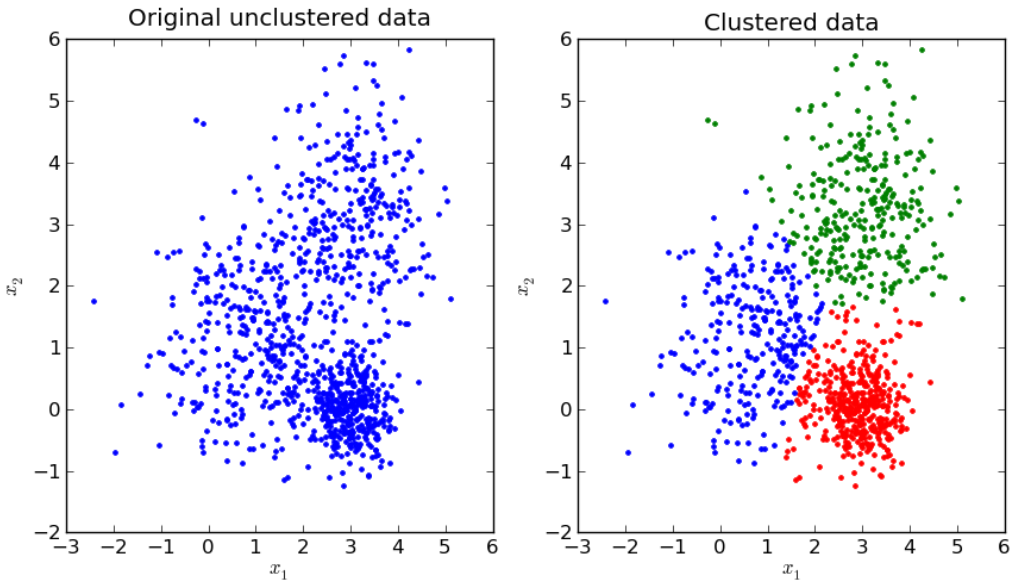


Figura 1.2: Cluster individuati da k-means, Fonte: [11]

Metodi basati sulle partizioni Il clustering partizionale consiste nell'individuare k punti e successivamente eseguire l'assegnazione di ogni elemento del dataset a uno di questi. In tal modo sono individuate k partizioni, che corrispondono ad altrettanti cluster. Questo metodo ha come vantaggio la semplicità e il ridotto numero di iperparametri, è infatti necessario specificare a priori solo il valore di k. Due esempi di algoritmi [10] possono essere *k-means* e *k-medioid*: entrambi partono dall'individuare k punti a caso all'interno del dataset, successivamente eseguono l'assegnazione dei restanti elementi, calcolano un nuovo centro del cluster in un caso, un medioide (punto che minimizza la dissimilarità rispetto agli altri elementi del cluster) nell'altro e ripetono le operazioni fino a giungere a una convergenza ai centri dei cluster. In figura figura 1.2 è possibile vedere il risultato di k-means su un dataset fissato. I limiti di questo approccio sono l'eccessiva rigidità delle regole applicate nella generazione dei cluster e la necessità di conoscere a prescindere il loro numero (k).

Metodi basati sulla gerarchia Passando ai metodi basati sulla gerarchia, questi prevedono una divisione del dataset definendo un criterio gerarchico tra i vari elementi. A seconda della modalità di divisione si possono individuare due categorie: se la gerarchia è definita combinando di volta in volta i singoli elementi, allora si parla di clustering gerarchico agglomerativo. Nel caso la ricerca avvenga dividendo ad ogni passo i cluster in sottoinsiemi, invece si tratta di clustering gerarchico divisivo. Gli algoritmi agglomerativi adottano un approccio *bottom-up*: partono considerando ogni punto come un gruppo ed a ogni passo fondono i cluster sulla base di determinate metriche di similarità. Tale passaggio viene ripetuto fino al soddisfacimento di una condizione. Al contrario gli algoritmi divisivi partono da un unico cluster, contenente tutti i punti del dataset, e dividono di volta in volta i cluster in cluster più piccoli. *Agglomerative Nesting (AGNES)* [12] e *Divisive*

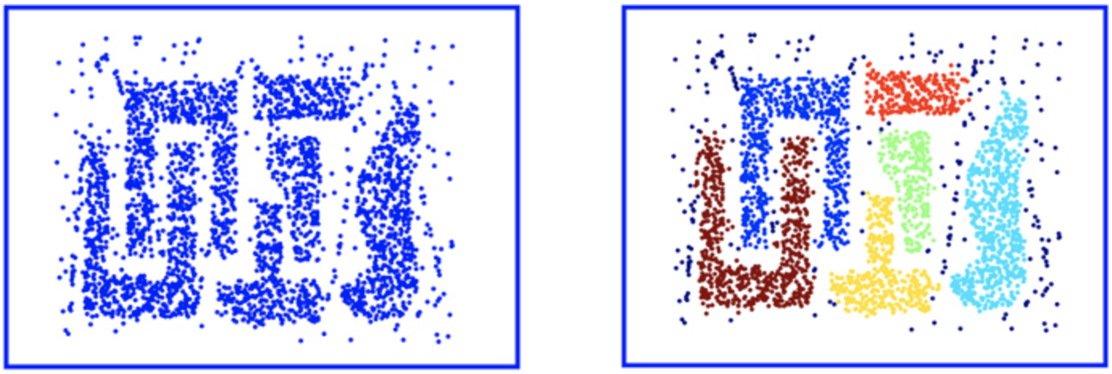


Figura 1.3: Cluster individuati da DBSCAN, Fonte: [16]

Analysis (DIANA) [13] sono due algoritmi che appartengono rispettivamente al clustering agglomerativo e a quello divisivo. Gli algoritmi di clustering gerarchico hanno il vantaggio di essere semplici, tuttavia risulta difficile definire i punti di divisione o aggregazione dei cluster. In più la natura irreversibile delle divisioni/aggregazioni rende complicato giungere a soluzione ottime.

Metodi basati sulla densità La categoria di algoritmi basati sulla densità ha alla base un'idea diversa rispetto a quanto detto fin d'ora. Data un'area spaziale, la cui densità è sopra una soglia di rilevanza, e un insieme di cluster, si assegnano i punti in questione al cluster più vicino. Il clustering basato sulla densità riesce così ad individuare raggruppamenti di ogni forma e dimensione. Due algoritmi emblematici della categoria sono *DBSCAN* [14] e *OPTICS* [15].

Density-Based Spatial Clustering of Applications with Noise, o *DBSCAN*, è un algoritmo di clustering che permette di individuare cluster di ogni forma e dimensione grazie alla connessione basata sulla densità dei singoli cluster. Dato un set di punti, questi vengono divisi in core point, density-reachable point e outlier secondo il seguente criterio: Il punto p si definisce core point se esistono almeno $minPts$ punti a distanza minore o uguale di ϵ, questi sono poi etichettati come direttamente raggiungibili da p. Un punto q è definito raggiungibile da p se esiste un'insieme di punti $p_1, \ldots, p_n$ dove $p_1 = p$ e $p_n = q$ dove ogni p_{i+1} è direttamente raggiungibile da p_i, quindi tutti i punti all'interno della sequenza sono core. Un punto non raggiungibile da nessun altro viene definito outlier. Se p è un core point, allora genera un cluster assieme a tutti i punto raggiungibili. Tutti i punti all'interno di un cluster devono mantenere una densità di almeno $minPts$ punti all'interno del proprio vicinato di raggio ϵ. I valori dei parametri ϵ e $minPts$ sono decisi senza una regola precisa, sta infatti all'esperienza dell'utente stabilire valori sensati e in questo risiede la maggiore debolezza di DBSCAN. In figura 1.3 è possibile vedere il risultato dell'algoritmo su un dataset avente cluster di forma non regolari.

Metodi basati sulla griglia Approccio ancora differente è quello dei metodi basati su griglia, che impiegano una griglia multi-risoluzione per suddividere lo spazio in un limitato numero di celle. Su ognuna di queste celle vengono eseguite in parallelo le operazioni di clustering, i cui risultati sono direttamente collegati alla compressione effettuata dalle dimensioni della singola cella. Il vantaggio principale di questa metodologia è la totale indipendenza dalle dimensioni del dataset: solo il numero di celle su ogni dimensione influenza il tempo di computazione. Un esempio di questa famiglia di algoritmi è *STING* [17], che divide lo spazio di ricerca in rettangoli ad alta dimensionalità, scomponendo poi ciascuno di questi in strutture a minore dimensionalità e conservando per ciascun livello un insieme di informazioni statistiche sugli attributi.

Metodi basati sul modello Ultima categoria trattata è quella degli algoritmi basati su modello. Questi assumono che per ogni cluster sia presente un modello matematico che lo descriva e ricercano l'insieme dei punti che meglio si adattano al modello selezionato. In primo luogo vengono calcolate le funzioni di densità che riflettono la distribuzione dei dati, successivamente tenta di adattare la distribuzione dei dati a un certo modello matematico. *COBWEB* [18] è un esempio di algoritmo che ricerca raggruppamenti sui dati tramite una gerarchia di concetti. Il clustering basato su modelli molto spesso fatica nel trovare la descrizione migliore per le relazioni tra i diversi punti del dataset.

1.2.1 Clustering di traiettorie

Uno degli obiettivi dell'analisi di dati di traiettoria è il *clustering* (raggruppamento) di traiettorie simili. Una traiettoria può essere considerata non solo come il tragitto percorso dall'oggetto che la genera, ma anche come l'insieme delle attività, ciascuna corrispondente a una posizione, quando a ogni posizione si collega un significato semantico.

Lo scopo del clustering di traiettorie è quindi di scoprire quali percorsi condividono una certa similarità e quali invece no. In tal modo è possibile identificare eventuali raggruppamenti di oggetti che hanno viaggiato assieme per una certa frazione del loro percorso, oppure individuare traiettorie differenti da tutte le altre.

Interpretando invece una traiettoria come l'insieme dei comportamenti di un oggetto, è possibile ad esempio dedurre quali possano essere attività comuni e quali invece no.

Come detto nella sezione 1.1.2, i dati di traiettoria sono molto più complessi rispetto ai dati solitamente utilizzati con gli algoritmi di clustering tradizionali. Occorre quindi definire modifiche di questi ultimi per riuscire a fare operazioni di clustering efficienti, in quanto sarebbe impossibile catturare tutta la complessità dell'informazione con tecniche pensate per dati a bassa dimensionalità.

Alla luce di quanto detto sopra, per il clustering di traiettorie sono individuabili i seguenti obbiettivi:

- **Supporto alla dimensionalità dei dati**. Obbiettivo del clustering di traiettorie è la ricerca di cluster tenendo conto di tutte le informazioni presenti sui dati. Ognuno di questi attributi dovrà essere considerato nel momento in cui verranno portate avanti le operazioni di divisione e raggruppamento delle traiettorie.

- **Definizione di una metrica di similarità tra traiettorie**. Come presentato nella sezione 1.1.2 il problema della similarità tra traiettorie è complesso e sono presenti diverse soluzioni. Scopo della ricerca è quello di individuare metriche che individuino le differenze tra le traiettorie in maniera affidabile e efficace.

- **Qualità dell'algoritmo**. L'algoritmo utilizzato nelle operazioni di clustering deve essere efficiente e scalabile, ad esempio impiegando apposite strutture dati per ridurre i tempi di accesso ai dati, oppure utilizzando le tecnologie di computazione Big Data per velocizzare l'esecuzione degli algoritmi.

Nonostante nessuno degli algoritmi di clustering tradizionali abbia tutte le caratteristiche espresse sopra, le idee alla loro base rimangono comunque valide in buona parte dei casi. Di conseguenza molti algoritmi pensati per i dati di traiettoria non sono che estensioni di quelli già noti in letteratura.

Gli algoritmi di clustering di traiettorie sono classificabili secondo la natura del dato in output e sulla tipologia di clustering. La natura del dato è direttamente collegata alle dimensioni considerate nelle operazioni di clustering: la ricerca può essere condotta considerando solo la componente spaziale o includendo anche quella temporale. La tipologia di clustering invece riguarda i cluster prodotti in output: algoritmi partizionanti produrranno cluster disgiunti, algoritmi di clustering sovrapposto cluster la cui intersezione non è vuota.

La tabella 1.2 riassume i principali algoritmi che saranno trattati nelle sezioni successive alla luce della classificazione appena introdotta.

Algoritmo	Natura dei dati	Tipologia di clustering
CB-SMoT	Spaziale	Sovrapposto
CACT	Spaziale	Sovrapposto
T-OPTICS	Spazio-temporale	Sovrapposto
TraceMob	Spaziale	Partizionante
DSC	Spazio-temporale	Sovrapposto

Tabella 1.2: Classificazione degli algoritmi di clustering di traiettorie trattati

1.2.2 Natura dei dati

Algoritmi spaziali

Le informazioni spaziali sono probabilmente la feature più importanti all'interno di un dato di traiettoria. Analizzando come un oggetto si muove e i luoghi che visita possono essere ricavate una vasta serie di informazioni. Negli anni vari algoritmi sono stati proposti per estrarre dai dati informazioni differenti tra loro.

Il primo ambito di ricerca sui dati spaziali riguarda i luoghi di maggior interesse, ovvero le posizioni in cui sono passati un certo numero di oggetti all'interno del dataset. A questa categoria appartiene il framework *CB-SMoT* (Clustering Based Stop and Moves of Trajectories) [19]. Questo algoritmo ricerca all'interno delle varie traiettorie considerando gli stop, ovvero segmenti in cui la velocità della traiettoria cala sotto una certa soglia o risulta uguale a zero. Successivamente questi segmenti sono raggruppati in cluster usando una versione modificata di DBSCAN. Tale versione dell'algoritmo si basa sulla velocità invece che sulla densità. Infine ogni cluster viene confrontato con la mappa dell'area coperta dalle traiettorie e viene associato a uno specifico punto o area. Questa associazione permette di interpretare meglio il risultato ottenuto dall'algoritmo.

Ancora è possibile estrarre da un insieme di dati di traiettorie, l'insieme delle strade più frequentate; ciò diverge dal primo ambito presentato poiché la ricerca di un percorso risulta più complessa rispetto a quella di un singolo punto: una strada infatti ha caratteristiche molto più complesse di una singola località, come ad esempio una continuità nello spazio tra i vari punti che la compongono. *CACT* [20] (Clustering and Aggregating Clues of Trajectories) è un possibile framework per ricercare percorsi che rappresentino i comportamenti di una certa categoria di utenti. L'idea dell'algoritmo è di definire una misura di similarità basata sugli indizi (*clue*): Un indizio è definibile come la vicinanza spazio-temporale di punti di traiettorie diverse che però condividono lo stesso comportamento. Tale indizio costituisce una corrispondenza parziale di comportamento. Sulla base della presenza di indizi simili, vengono costituiti cluster di traiettorie, che raggruppano queste ultime sulla base di un certo comportamento. I cluster così ottenuti sono però ancora percorsi parziali, per determinare percorsi completi è necessario un ulteriore passo di ricerca di indizi e fusione dei cluster.

I due ambiti appena descritti partono dalla stessa interpretazione dei dati, cercando di eseguire una separazione tra le varie traiettorie sulla base delle proprietà dei singoli punti. Un'alternativa a questa visione è presente nel clustering basato su forma (*Shape Based Clustering*), in cui i raggruppamenti sono basati sulla distribuzione dei punti piuttosto che sulle loro proprietà. Questo approccio non si limita ad analizzare solo la dimensione spaziale, ma include nel determinare la forma di una traiettoria anche la sua dimensione temporale.

Algoritmi spazio-temporali

Come detto nella definizione di traiettoria (sezione 1.1.1) le informazioni necessarie per definire un punto sono due: la componente spaziale e quella temporale.

Il tempo risulta più complesso da gestire rispetto allo spazio: è infatti praticamente impossibile definire una scala temporale univoca all'interno di un dataset. Essendo le traiettorie generate da diversi dispositivi GPS, è molto raro che questi condividano tra loro la frequenza di campionamento, rendendo quindi difficile definire un ordine assoluto all'interno dell'area temporale coperta dal dataset. Oltre a ciò, la possibile adozione di scale cicliche per l'analisi del tempo rende necessario introdurre ulteriore complessità negli algoritmi che supportano queste ricerche.

A differenza di quanto accade negli ambiti spaziali, la ricerca pone il suo accento sull'interpretazione del tempo e la conseguente formazione dei cluster piuttosto che solo su questo secondo ambito.

La maggior parte degli algoritmi riescono a gestire scale temporali assolute. Ad esempio *T-OPTICS* [21] è una variante di OPTICS che impiega una metrica di similarità adatta a individuare cluster considerando anche il tempo. L'idea alla base dell'algoritmo è di ricercare il miglior intervallo temporale l'algoritmo OPTICS, appositamente modificato per la ricerca di traiettorie, individua i risultati migliori. Sta all'utente specificare la lunghezza e il range dell'intervallo temporale: a seconda del periodo specificato i cluster individuati possono cambiare totalmente. Come però affermato in un'indagine [22] condotta nel 2013, esistono pochi framework in grado di gestire scale temporali cicliche e la ricerca di pattern periodici.

1.2.3 Tipologia di clustering

Algoritmi di clustering partizionante

Gli algoritmi di clustering, oltre alla dimensioni impiegate nella ricerca, possono essere classificati sulla base delle proprietà dei cluster prodotti. Tra tutte le proprietà una delle più interessanti è sicuramente la natura partizionante o sovrapposta: si definisce un algoritmo partizionante se, dato un punto, questo viene assegnato al massimo a un cluster, sovrapposto se questa cardinalità aumenta. Anche gli algoritmi di clustering di traiettorie possono essere classificati in partizionanti e sovrapposti.

Il clustering di traiettorie partizionante produce insiemi disgiunti, effettuando quindi una separazione totale tra i percorsi nel dataset. Ogni traiettoria è valutata nel suo complesso dal processo di clustering, ciò implica che vengano avvicinati i percorsi che sono più complessivamente simili mentre siano distanziati quelli che condividono solo brevi tratti in comune.

TraceMob (Transformation, partition clustering and cluster evaluation of moving objects) è un esempio di algoritmo partizionante. L'idea alla base del framework è di dividere lo

spazio coperto dalle traiettorie in celle di dimensioni $\alpha * \beta$. Questi parametri determinano la scala dell'analisi: valori alti producono aree grandi, adatte a territori in cui le traiettorie sono molto sparse, valori piccoli invece sono idonei per la ricerca in spazi ad alta densità. Successivamente viene calcolata la vicinanza tra tutti gli elementi del dataset: due traiettorie risulteranno vicine se in tutto il loro percorso sono sempre transitate in celle vicine. Una volta terminato il calcolo, ogni traiettoria viene proiettata come punto su uno spazio d-dimensionale. La funzione che si occupa di ciò è strutturata in modo da avvicinare le traiettorie simili e separare quelle diverse. Infine viene effettuato un clustering partizionante sui punti così generati. Tale operazione produrrà cluster di traiettorie vicine nella loro interezza, separando quelle che divergono in certi istanti.

Il clustering partizionante considera quindi le traiettorie nella loro totalità. Ciò può risultare vantaggioso in termini di performance, tuttavia questa modalità ignora le singole caratteristiche locali della traiettoria.

Algoritmi di clustering sovrapposti

Come affermato nella sezione 1.2.3, il clustering partizionante ignora le singolarità delle traiettorie. Per superare il problema descritto sopra, gli algoritmi di clustering sovrapposto effettuano una divisione di ogni traiettoria in sotto-traiettorie e effettuano un clustering partizionante su queste sotto-traiettorie.

Una famiglia di algoritmi basati su questa idea è *Partition and group based algorithm*. Il focus principale di questa categoria è l'individuazione dei segmenti e dei punti in cui "spezzare" la traiettoria originale. Per risolvere questo problema sono state ipotizzate diverse soluzioni, ad esempio il framework *DSC* (Distribuited Subtrajectory Clustering) [23] utilizza una metrica basata sul cambio di densità nell'intorno dei punti della traiettoria per determinare le divisioni. In una prima fase DSC compie una operazione di *self-join* su ogni traiettoria, ricercando tutte le traiettorie aventi uno spaziotempo comune di almeno k istanti con ogni singolo elemento del dataset. Questa misura viene effettuata tramite LCSS. Successivamente vengono determinati i punti di separazione di ogni traiettoria: ciò viene fatto analizzando il vicinato creato nella fase precedente e separando la traiettoria ogni volta che la densità o la composizione di quest'ultimo cambia in maniera significativa. Infine vengono i segmenti così originati sono sottoposti a una versione modificata di k-means: sono individuati R rappresentati nel dataset e ogni segmento viene attribuito ad un unico rappresentante. Alla fine di questo passo i cluster vengono raffinati così da eliminare segmenti ripetuti e fondere cluster simili. La figura 1.4 mostra un esempio di clustering effettuato da DSC: com'è possibile vedere ogni traiettoria viene divisa in più segmenti ed è assegnata a diversi cluster.

Il clustering sovrapposto permette di conservare le relazioni intra-cluster, tuttavia una frammentazione troppo fine può causare la perdita di *rare pattern*, ad esempio eventi significanti che accadono con una bassa frequenza [24], [25].

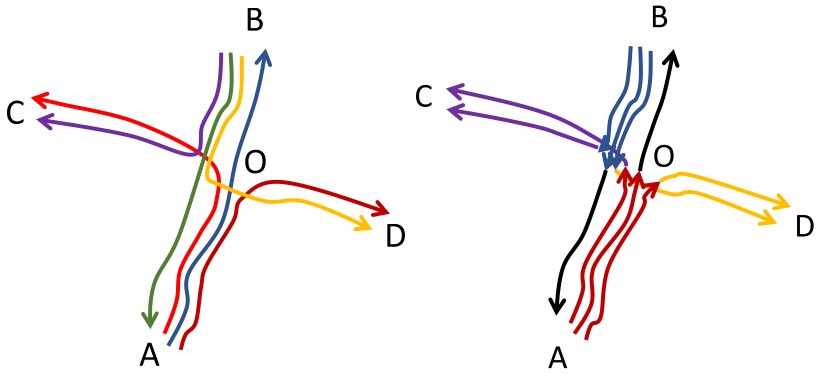


Figura 1.4: Traiettorie di partenza (a sinistra), Cluster individuati da DSC (a destra), Fonte: [23]

1.2.4 Applicazioni e limiti

Nelle sezioni precedenti sono state prese in considerazione le principali categorie di clustering di dati di traiettoria; queste sono state impiegate in diversi ambiti, a seconda delle singole caratteristiche della tecnica utilizzata.

Di seguito sono riportati i principali ambiti applicativi del clustering di traiettorie:

- **Analisi dei pattern di movimento degli oggetti.** L'applicazione più scontata di un raggruppamento di traiettorie: l'analisi di pattern di movimento mira ad individuare quali oggetti si sono mossi assieme secondo certe regole.

- **Previsione dei movimenti di un oggetto.** Sulla base delle caratteristiche di una traiettoria, è possibile predire quali saranno i futuri movimenti di un oggetto partendo dai suoi spostamenti passati: si attribuisce infatti il percorso fin d'ora eseguito a uno dei cluster individuati e considerando le sue caratteristiche si può prevedere con una certa accuratezza il percorso futuro dell'oggetto. La validità di questa previsione è direttamente collegata al numero di misure considerate durante le operazioni di clustering: maggiore il numero, più accurato il risultato. Importanza particolare ha il tempo: non considerandolo è di fatto impossibile eseguire questo tipo di ricerca.

- **Supporto alla pianificazione stradale e dei trasporti.** Come affermato nella sezione 1.2.2, determinati algoritmi possono individuare i punti più frequentati da certi utenti; è quindi possibile integrare queste informazioni in piani di controllo del traffico/ espansione delle infrastrutture urbane.

- **Ricerca di outlier.** Un outlier è per definizione un elemento che, dato un certo criterio di similarità, si discosta da tutti gli altri. Gli outlier possono essere interpretati come errori, ma anche come comportamenti che per determinati motivi divergono dalla norma. Sotto quest'ultima chiave di lettura divengono interessanti la loro ricerca e i motivi per cui differiscono dagli altri dati. Essendo poi il percorso di un

oggetto interpretabile come l'insieme dei suoi comportamenti, gli outlier individuano comportamenti inusuali e per questo possono destare grande interesse.

- **Deanonimizzazione dei dati.** Utilizzando tecniche di clustering su dati incerti, è possibile ricavare informazioni sugli oggetti collegati alle traiettorie precedentemente nascosti tramite tecniche di anonimizzazione.

Gli ambiti di utilizzo del clustering di traiettorie sono numerosi, tuttavia il margine di miglioramento è ancora abbondante. In primo luogo la maggior parte degli algoritmi non sfrutta tutto il potenziale semantico dei dati: molte tecniche impiegano solo le dimensioni spazio-temporali o un insieme ristretto di feature, scartando così informazioni che potrebbero ulteriormente migliorare la qualità dei risultati. Successivamente i risultati prodotti sono di difficile interpretazione, molto spesso accade che l'estrazione di conoscenza mostri relazioni banali o al contrario troppo complesse per essere spiegate. Infine manca ancora un'integrazione diffusa con le tecnologie di elaborazione Big Data, la maggior parte degli algoritmi infatti è pensata per computare in maniera centralizzata, rinunciando ai vantaggi del calcolo distribuito. In questo modo larghi dataset producono risultati di scarsa qualità in tempi alti. L'applicazione di queste strategie di computazione, sebbene complessa nella sua realizzazione, porterebbe notevoli vantaggi e supererebbe molte delle problematiche degli attuali algoritmi. I framework SPARE [26] e CUTE pongono questo supporto come uno dei loro punti di forza.

1.3 Frequent itemset mining

Nell'ambito dell'analisi dei dati, il *frequent itemset mining*, o ricerca di itemset frequenti, è uno dei task con maggiori ambiti di applicazione: può essere impiegato nei processi di classificazione, clustering o ricerca di outlier [27].

Il frequent itemset mining necessita di un insieme di transazioni per effettuare la propria ricerca, ogni transazione, o riga (*row*), è composta poi da un insieme di attributi, o feature, che identificano gli elementi all'interno della singola transazione. Occorre specificare che di una feature è considerata rilevante ai fini della ricerca solo la presenza o l'assenza e non un eventuale valore o altre proprietà collegate; queste potranno essere integrate mediante apposite tecniche di preprocessing. La definizione di transazione e item è formalizzata nella definizione 1.3.1

Definizione 1.3.1 (Transazione, item e itemset)**.** Dato un database, si definisce transazione t la singola riga della base di dati. Tale riga è caratterizzata da un identificatore e da un insieme di item o $feature$. Un item i, è un attributo binario, calcolato sulle colonne del database da cui sono estratte le transazioni. Si definisce infine un itemset I un set di item $i_1, \dots, i_n$.

Scopo del mining di itemset frequenti è di ricercare, dato un insieme di transazioni, ricercare le combinazioni di feature, o itemset, che risultano frequenti. Misura fondamentale per definire la frequenza è il supporto: tale metrica può essere descritta come il numero di transazioni che contengono un certo itemset (definizione 1.3.2).

Definizione 1.3.2 (Supporto)**.** Definito un insieme di transazioni T e un certo itemset I si definisce supporto $s(I)$ l'insieme $(t_1, \dots, t_n) \in T$ tale che $I \subseteq t_i.I \ \forall t_i \in (t_1, \dots, t_n)$. Il valore del supporto è la cardinalità dell'insieme $(t_1, \dots, t_n)$

All'interno della ricerca di itemset frequenti viene definito un limite inferiore al supporto per determinare l'interesse verso un certo itemset; questo parametro prende il nome di supporto minimo, o *minsup*. Definito ciò, è possibile esprimere il problema del mining di itemset frequenti con la seguente formulazione (definizione 1.3.3) :

Definizione 1.3.3 (Frequent Itemset Mining)**.** Definito T come l'insieme delle transazioni e F come quello delle feature complessive, il frequent itemset mining ricerca tutti gli itemset
$I = \{i_1, \dots, i_n\}, i_1, \dots, i_n \in F$ tali che definito il supporto $S = \{t_1, \dots, t_m\} \in T \ s.t \ \forall t_i \in \{t_1, \dots, t_m\} \ i \subseteq t_i$, $|S| \geq minsup$

Definito l'obbiettivo della ricerca, sono necessari due passaggi per realizzarla: il primo passo consiste nella generazione di tutti i possibili itemset, il secondo nel calcolo del supporto per ciascuno di questi e *pruning* (potatura) dei candidati non interessanti. Nonostante la definizione semplice, la complessità computazionale di queste fasi può esplodere: supponendo infatti di avere un set di feature contenente n diversi elementi, l'insieme di tutte le possibili combinazioni generabili è 2^n, rendendo di fatto molto costoso il processo di ricerca in presenza di dataset di larghe dimensioni.

1.3.1 Algoritmo Apriori

Apriori [28] è un algoritmo pensato per affrontare il problema della generazione e filtraggio dei candidati in maniera efficace e efficiente. Apriori utilizza una struttura di generazione a livelli in maniera iterativa: ad ogni livello sono ricercati pattern di dimensioni k aventi supporto maggiore di *minsupp*. Successivamente i pattern validi sono impiegati nella costruzione del livello successivo, mentre gli altri sono scartati. Intuitivamente, la ragione per cui accade ciò è che la frequenza di un set sarà sempre maggiore di quella di qualunque superset a cui questo appartiene.Se già un set non soddisfa il vincolo di frequenza, nessun suo superset potrà fare altrettanto. Questo principio è noto con il nome di monotonicità della generazione apriori. Ne segue che tutti gli itemset generati al livello $k+1$ saranno combinazioni di pattern validi al livello k. Questa strategia di generazione permette di scartare un grande numero di itemset e le loro successive combinazioni (figura 1.5).

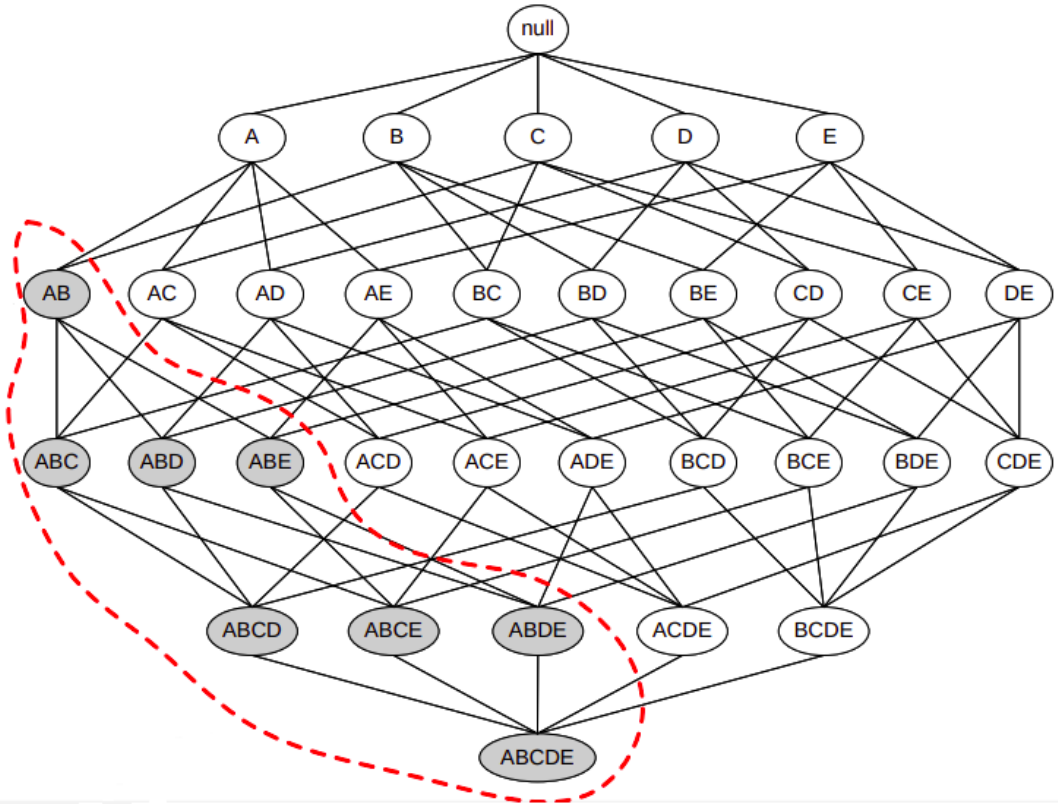


Figura 1.5: Esempio di struttura gerarchica generata dall'algoritmo Apriori, in questo caso l'itemset $\{a, b\}$ non risulta frequente, di conseguenza i suoi superset sono scartati,Fonte: [29]

i	r_i
1	a,b,c,l,o,s
2	a,d,e,h,p,l,r
3	a,c,e,h,o,q,t
4	a,e,f,h,p,r
5	b,d,f,g,l,q,s,t

Figura 1.6: Dataset contenente 5 transazioni, fissato $minsup = 2$ l'itemset (a, e, h) risulta chiuso e massimale, Fonte: [30]

All'interno dell'insieme degli itemset validi, è possibile eseguire un'ulteriore distinzione: si definiscono chiusi (*closed*) gli itemset aventi supporto strettamente maggiore di tutti i propri superset (definizione 1.3.4); al contrario un itemset non avente superset validi è definibile come massimale (definizione 1.3.5).

Definizione 1.3.4 (Itemset chiuso)**.** L'itemset I si definisce chiuso se $\forall J$ t.c $I \subset J,\ s(I) > s(J)$

Definizione 1.3.5 (Itemset massimale)**.** L'itemset I si definisce massimale se $\forall J$ t.c $I \subset J,\ s(J) < minsup$

Preso come esempio il dataset in figura 1.6, il pattern (a, e, h) risulta chiuso, in quanto nessuno dei suoi superset ha un supporto maggiore del suo. (a, e, h) inoltre può essere definito anche come pattern massimale, in quanto tutti i suoi superset hanno supporto minore di *minsup*. L'itemset (a, e) invece risulta frequente, ma né chiuso né massimale, in quanto $s((a, e)) = s((a, e, h))$.

L'esempio sopra mostra un'altra proprietà di chiusura e massimalità: ogni pattern massimale, come nell'esempio, può essere categorizzato come pattern chiuso. Un pattern massimale è infatti un itemset frequente i cui superset hanno supporto minore di *minsup*, tuttavia per definizione di frequenza $s(pattern) \geq minsup$. Ne segue che sarà sempre vero che il pattern in questione avrà supporto maggiore di tutti i suoi superset, rientrando così nella definizione di massimale. Al contrario non tutti pattern chiusi possono essere classificati come massimali. Non c'è infatti nessuna proprietà che garantisca che un superset di un itemset abbia una frequenza minore dell'itemset stesso, come ad esempio succede nel caso sopracitato.

In molti casi Apriori è efficace nel ridurre lo spazio di ricerca di itemset validi, tuttavia soffre di alcuni limiti: in primo luogo è molto probabile che il numero di candidati generati rimanga comunque alto, in quanto molti di questi sono generati in un primo momento e scartati dopo il calcolo del supporto. Proprio a quest'ultimo punto si collega un'altra problematica: la computazione del supporto richiede molte scansioni delle transazioni e

ad ogni livello devono essere verificati molti itemset. Prendendo ad esempio il dataset in figura 1.6, in cui sono presenti solo 5 righe e 15 item, sono generate e valutate tutte le combinazioni di item. Questo implica la valutazione di 2^15 possibili itemset. Ovviamente questo numero cala specificando soglie di supporto con valori alti. Tuttavia è probabile che soprattutto nei primi passi, il numero di itemset individuati esploda, peggiorando di conseguenza le performance dell'algoritmo.

1.3.2 Frequent itemset mining su dati di traiettoria

La ricerca di itemset frequenti può essere adattata ai dati di traiettoria. Ciò non è altro che un'evoluzione del mining di sequenze.

Il mining di pattern su un insieme di item è un caso particolare di mining di itemset frequenti. La principale modifica è l'introduzione del concetto di tempo e , conseguentemente, di sequenza. Una sequenza non è altro che un itemset i cui elementi sono ordinati rispetto al tempo. Ad esempio, l'itemset $< a, ab, c >$ può essere interpretato come una sequenza in cui i tre elementi si susseguono in istanti temporali contigui. Nella ricerca di sequenze, gli attributi sono espliciti. Nel caso dei dati di traiettoria però, ciò non risulta vero: ogni punto di traiettoria grezza è composto da latitudine, longitudine e istante temporale. Tali punti sono difficilmente coincidenti tra loro, ciò causa una difficoltà nell'estrazione delle feature. Per superare questa problematica, occorre ricorrere all'utilizzo di astrazioni di traiettoria: mappando il dataset su uno specifico sistema di riferimento, è possibile accomunare i punti negli stessi quanti.

Questa astrazione consente di superare certi limiti, tuttavia non è sufficiente a risolvere il problema. Gli algoritmi di riconoscimento di itemset frequenti o sequenze, oltre ad avere un alto costo computazionale, non sono in grado di processare la contiguità spaziale e la continuità temporale. Inoltre nel riconoscimento di sequenze sono richieste strette relazioni di adiacenza tra le feature, cosa non sempre possibile nelle traiettoria, a causa della loro intrinseca incertezza.

In letteratura sono state proposte diverse soluzioni per superare questi limiti: ad esempio per quanto riguarda la generazione delle sequenze è stato proposto l'utilizzo di una struttura ad albero per generare i pattern di movimento [31]. Per quanto riguarda invece le relazioni ignorate, sono stati proposti approcci per considerare la contiguità spaziale [32] e quella temporale [33].

Le applicazioni del mining di traiettorie presenti in letteratura riguardano l'estrazione dei percorsi più frequentati [34] e delle regioni più trafficate [35]. È interessante considerare come la ricerca di luoghi e percorsi frequentati sia ortogonale rispetto all'individuazione di gruppi di oggetti che hanno viaggiato assieme per un certo periodo di tempo. Approfondendo questo confronto, emerge che la definizione del problema è molto simile, ma cambia il ruolo di transazioni e feature. Nel caso di ricerca di luoghi frequenti in un dataset, tale frequenza è determinata sulla base degli oggetti che passano per un certo punto. Trattando invece di

analisi di oggetti che si sono mossi assieme, intuitivamente il supporto è definibile come l'insieme di posizioni condivise dal gruppo perché sia considerato frequente.

La ricerca di gruppi di oggetti non è affrontabile con le ordinarie tecniche di itemset mining. Il numero degli oggetti infatti supera di gran lunga quello delle posizioni visitate, di conseguenza l'insieme delle transazioni avrebbe cardinalità molto inferiore rispetto a quello delle feature. Tale condizione esplode il costo computazionale per la ricerca di itemset frequenti.

2 Tecnologie utilizzate

In questo capitolo sono presentate le principali tecnologie utilizzate durante lo sviluppo della tesi. In primo luogo verrà presentata la piattaforma Hadoop con una veloce panoramica sulle sue principali feature. Successivamente sarà trattato il framework Spark e le caratteristiche che lo hanno reso la soluzione con cui sono stati implementati gli algoritmi discussi in questa tesi.

2.1 Hadoop

Hadoop è un framework di natura open-source sviluppato da Apache. Obiettivo della piattaforma è la gestione, elaborazione e memorizzazione di grandi quantità di dati in modo scalabile.

Hadoop si pone come alternativa rispetto al modello *Massively Parallel Processor (MPP)*. In questo paradigma vengono impiegati più processori, ciascuno avente le proprie risorse di memoria e disco. Il lavoro complessivo è suddiviso in task. Ciascun task è poi eseguito da un processore a seconda delle politiche di schedulazione del sistema. Le architetture MPP sono composte da hardware e software proprietari.ù

Al contrario Hadoop è progettato per operare su commodity hardware e software open-source. Il commodity hardware è l'insieme di tutte le componenti hardware già in possesso di un utente. Tali componenti sono spesso generiche e senza particolari specifiche. Ciò permette di poter integrare anche risorse di natura e caratteristiche differenti tra loro, senza per forza dover mantenere un omogeneità all'interno del sistema. Questa diversità, oltre ad abbattere i costi, consente tempi di manutenzione e sostituzione più veloci.

Il framework Hadoop si compone di quattro moduli principali:

- **HDFS**. *File system* distribuito con alto throughput e replicazione a livello di blocco.
- **MapReduce** Framework per la creazione di applicazioni in grado di processare grandi moli di dati. MapReduce è pensato per astrarre la complessità della gestione parallela della computazione. Per l'utente è necessario solamente specificare le fasi di Map (elaborazione parallela sui dati) e di reduce (aggregazione dei risultati).
- **Common**. Questo modulo contiene librerie e strumenti di utilità per la gestione del sistema.

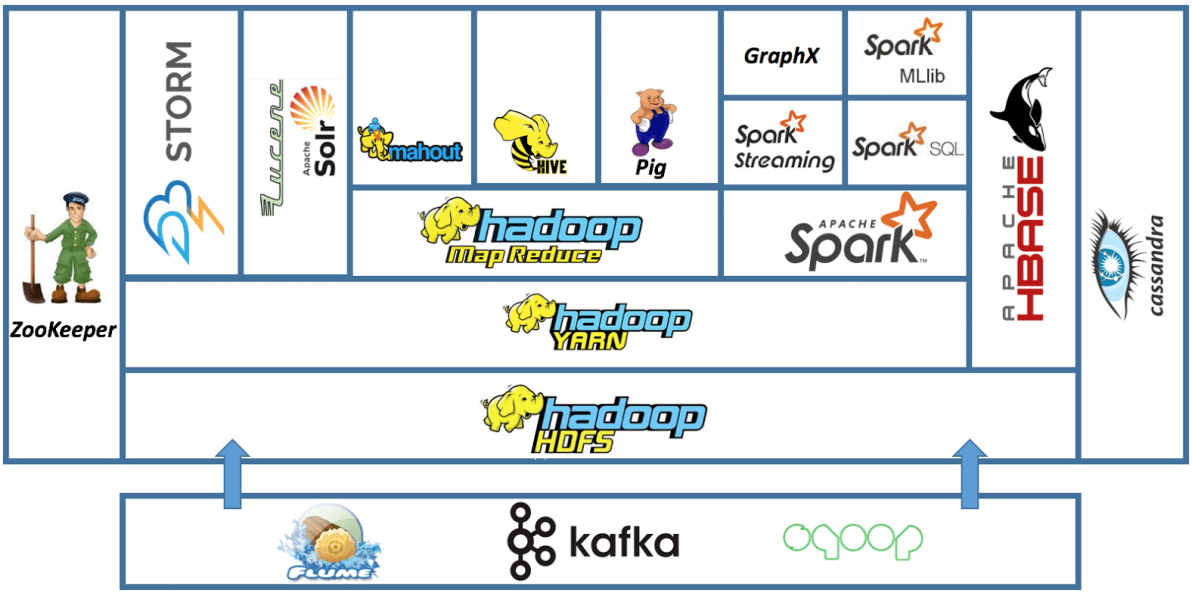


Figura 2.1: Panoramica dell'ecosistema Hadoop, Fonte: [36]

- **YARN**. Framework per la gestione delle risorse all'intero del cluster Hadoop.

Oltre a questi sono disponibili ulteriori moduli accessori, illustrati in figura 2.1.

2.1.1 HDFS

HDFS, o Hadoop Distribuited File System, è il file system distribuito proprietario di Hadoop. A differenza della maggior parte dei file system, Hadoop è progettato per operare con file di grandi dimensioni, introducendo politiche di ridondanza sui dati. Il file system è stato progettato per operare in batch, ovvero raggruppando prima i dati, piuttosto che in streaming. Questo tipo di accesso permette di ottenere un alto throughput, a discapito di una bassa latenza. Alla luce di ciò, le applicazioni adottano un modello di accesso ai dati *write-once-read-many*.

I principali punti forza di Hadoop sono:

- **Computazione distribuita**. Nel momento in cui si tratta di computazione su grandi moli di dati, i costi per spostare i dati attraverso un'infrastruttura di rete possono peggiorare le performance del sistema. Per risolvere questo problema, Hadoop mette a disposizione delle interfacce per spostare la computazione negli stessi nodi, o quantomeno vicino, dove risiedono i dati da elaborare. Questo principio è noto col nome di data locality.

- **Resilienza ai guasti**. I guasti a livello hardware vanno considerati come naturali nella vita del sistema. Sotto questa prospettiva, HDFS mette in campo tecniche di identificazione dei guasti e di ripristino veloce e automatico dei processi. Grazie anche a questo meccanismo, è possibile gestire hardware differenti tra di loro.

- **File system ottimizzato per file di grandi dimensioni**. HDFS definisce word, o blocchi in cui viene suddivisa e indicizzata la memoria del sistema, di dimensioni comprese tra 64 megabyte e un gigabyte. Blocchi di queste dimensioni gestiscono file nell'ordine di gigabyte o terabyte limitando il numero di blocchi in cui vengono divisi. Ciò implica una maggior coesione tra i dati, informazioni vicine e quindi con un alto grado di correlazione sono con buona probabilità salvate all'interno dello stesso blocco.

Passando poi all'architettura di HDFS, i nodi all'interno del sistema operano secondo un pattern master-slave. Il master, chiamato NameNode, mantiene in modo persistente l'albero rappresentante il file system, insieme ai metadati collegati a cartelle e file. Un altro compito del NameNode è mantenere il riferimento alla posizione di tutte le partizioni, o blocchi, di un certo file all'interno degli slave. Infine il master si occupa delle operazioni di input/output sui file, come ad esempio le richieste di apertura, chiusura e cambio di nome.

Gli slave invece sono chiamati DataNode hanno il compito di salvataggio fisico e recupero dei blocchi di ogni file nel cluster. I DataNode comunicano periodicamente con il NameNode, inviando l'elenco dei file e blocchi salvati sul singolo nodo. Il master al contrario invia ai DataNodes comandi per la creazione e cancellazione di blocchi, che gli slave provvedono ad eseguire.

I singoli nodi sono poi organizzati in rack, che a loro volta sono organizzati in datacenter. Questa struttura è rappresentata come un albero, avente nelle foglie i singoli nodi e nella radice il cluster. La distanza tra due nodi, fondamentale per il principio di data locality, viene calcolata quindi come la distanza che i due nodi hanno nell'albero.

La struttura del cluster viene sfruttata anche durante il processo di ridondanza dei dati. La replicazione dei dati migliora le performance di accesso ai dati e aumenta la robustezza del sistema. La ridondanza avviene a livello di blocco: il NameNode mantiene in memoria la lista dei DataNode contenenti un certo blocco. Il fattore di replica di default è impostato a 3. Queste repliche sono salvate in nodi differenti del sistema aventi posizioni diverse. La prima viene salvata sul nodo del client che ha lanciato il comando, la seconda in un nodo appartenente a un rack diverso del primo nodo, il terzo infine in un nodo nello stesso rack del secondo.

Le repliche possono essere ribilanciate in caso di problemi con un certo nodo o con un cambiamento nel cluster, ad esempio l'aggiunta di nuovi nodi. Hadoop utilizza la topologia del cluster in combinazione con la replica dei blocchi per applicare il principio della data locality. In primo luogo Hadoop cerca di mantenere la località a livello di nodo, poi a livello di rack, datacenter e infine cluster.

HDFS non ha performance efficienti quando viene richiesta una bassa latenza o i singoli file sono di dimensioni piccole. La ragione è che HDFS è progettato per computazioni in ambito Big Data, in cui queste situazioni sono rare.

2.1.2 YARN

YARN (Yet Another Resource Negotiator) è il gestore di risorse introdotto con Hadoop 2.0. Originalmente sviluppato per migliorare le performance di MapReduce, YARN risulta abbastanza generico per adattarsi anche ad altri paradigmi. Compito di questo gestore di risorse è negoziare le componenti necessarie per eseguire una certa applicazione e gestire la computazione distribuita. YARN rimane trasparente all'utente: viene usato solamente da framework e mai direttamente dal codice dell'utente.

L'idea alla base di YARN è la separazione tra la gestione dei job e quella delle risorse. Questa divisione è ulteriormente accentuata dalla presenza di due demoni che si occupano di questi aspetti. YARN si compone di due processi demone:

- **Resource Manager (RM)**. Demone globale, ne esiste solo uno per cluster, gestisce le risorse tra le varie applicazioni. Si compone di *Scheduler* e *Application Manager*. Il primo si occupa di allocare le risorse per le varie applicazioni, mentre il secondo è responsabile di accettare i job e eventualmente farli ripartire in caso di errori.

- **Node Manager (NM)**. Uno per ogni nodo slave, si occupa di eseguire i container. Un container è un'entità usata per eseguire un processo con un insieme limitato di risorse. Il Node Manager monitora l'esecuzione dei processi, il loro utilizzo di risorse e riporta i dati sul Resource Manager.

Nel momento in cui viene inviata al sistema la richiesta dell'esecuzione di una nuova applicazione, il RM cerca tra i vari NM uno che possa lanciare l'*Application Master Process*. A questo punto l'Application Manager negozia il primo container per eseguire l'AMP, che poi provvederà a richiedere le risorse necessarie. Questo processo risulta altamente scalabile: RM infatti delega ai singoli AMP la richiesta delle risorse e di nuovi container. In caso di fallimento inoltre è l'Application Master a occuparsi del ripristino.

YARN risulta quindi un framework aperto, che supporta altri paradigmi oltre a Map-Reduce, inoltre si integra bene con i meccanismi nativi di HDFS.

2.1.3 Hive

Hive è un modulo compatibile con Hadoop pensato per la lettura, scrittura e in generale gestione di dataset distribuiti. Hive è costruito sulla base di Hadoop.

Funzionalità cardine del framework è la possibilità di interrogare i dati in un linguaggio proprietario Hive-QL. Questo linguaggio si basa su una sintassi simil-SQL che produce query che vengono trasformate in serie di job Map-Reduce o Spark.

Altra feature rilevante è il supporto a molteplici tipi di dati. Tra questi sono presenti CSV, Apache Parquet e Avro. Il framework poi permette di accedere direttamente ai file salvati in HDFS.

2.2 Spark

Apache Spark è un framework per il calcolo distribuito e general purpouse. Spark nasce dalle trasformazioni che gli hardware e i software hanno subito nel corso del tempo e dalle nuove esigenze emerse. Negli ultimi anni sono disponibili macchine con CPU aventi sempre più core. Oltre a questo è stato registrato anche un aumento della memoria ram nelle macchine. Questo incremento ha permesso alla ram di divenire la memoria principale durante le computazioni, sostituendo il disco anche nella gestione di grandi moli di dati. Dal punto di vista del software invece, è avvenuta un evoluzione che ha dato maggior rilevanza ai paradigmi funzionali rispetto a quelli ad oggetti. Allo stesso modo i sistemi NoSQL, orientati a velocità e disponibilità hanno messo in ombra i classici sistemi SQL, orientati alla consistenza del dato. Anche nel mondo Big Data sono emerse nuove possibilità ed esigenze: la necessità di poter processare dati in streaming, favorire approcci veloci e l'integrazione con nuovi ambiti della data science, come ad esempio il machine learning sono alcuni esempi.

Alla luce di questi cambiamenti, Map-Reduce è divenuto limitante. Questo paradigma infatti non si presta a tutti i tipi di elaborazione, essendo pensato per un approccio batch e basato sul disco. Spark si pone come integrazione di Map-Reduce a queste nuove esigenze. È in grado infatti di eseguire sia processi batch che streaming o query interattive. Le sue primitive, basate sull'accesso a dati memorizzati in RAM, consentono performance cento volte più veloci di Map-Reduce. Spark inoltre mantiene l'assoluta compatibilità con Hadoop tramite YARN e supporta varie fonti di dati, come ad esempio Hive.

2.2.1 Architettura

Spark si basa su due principali astrazioni: RDD e DAG.

RDD, o *Resilient Distribuited Dataset*, è la rappresentazione del framework Spark di una collezione di dati di qualunque natura. Questa collezione è divisa in partizioni, ciascuna salvata su un nodo. Un RDD può essere generato da qualunque fonte, ad esempio la query su un database esterno o la lettura di un file di input.

Le caratteristiche di un RDD sono le seguenti:

- **Type inference**. Determina i tipi di dato durante la compilazione, senza bisogno di specificarli in fase di scrittura del codice. Questo è reso possibile dal linguaggio funzionale Scala, in cui è scritto il framework.

- **Cachability**. Gli RDD non sono di default salvati in memoria, ma sono processati e poi scartati. Per migliorare le performance, è possibile eseguire caching salvando in memoria o su disco un RDD. Ciò consente di evitare di doverlo ricalcolare ad ogni utilizzo.

- **Immutabilità**. Un RDD è per definizione immutabile. Qualunque operazione di trasformazione produce un nuovo RDD. Ciò semplifica la gestione di corse critiche senza bisogno di aggiungere nessun meccanismo di accesso alle risorse.

- **Laziness**. Le operazioni su un RDD possono essere di due tipi: trasformazioni o azioni. Le trasformazioni modificano i dati all'interno di un RDD, costruendone di fatto uno nuovo. Le azioni calcolano un risultato da restituire al driver o salvare su piattaforme esterne (ad esempio database Hive). Ogni operazione effettuata su un RDD non è eseguita fino alla richiesta di un azione. Le trasformazioni infatti producono metadati che vengono utilizzati nel momento in cui tramite un azione viene scatenata la catena di trasformazioni.

La proprietà di laziness pone le basi per la definizione di DAG. Durante la computazione, a un RDD saranno associate più trasformazioni: queste possono essere rappresentate come un grafo. Questo è chiamato lineage graph e costituisce la base per la definizione del piano di esecuzione. Sulla base delle operazioni da eseguire e delle partizioni coinvolte, viene definito un piano di ottimizzazione. Questo aggregherà operazioni diverse e sfrutterà il principio della data locality.

Una volta terminata la fase di ottimizzazione, il piano di esecuzione viene presentato nella forma di DAG (Directed Acyclic Graph). DAG è un grafo i cui nodi sono RDD e gli archi le trasformazioni che portano da un RDD all'altro. Il grafo è direzionato e sono assenti cicli al suo interno.

Determinato il DAG, questo viene diviso in stage secondo il seguente principio: si raggruppano operazioni fino a giungere a una trasformazione che richieda un processo di shuffle dei dati. Uno shuffle è un operazione di collezione e ridistribuzione dei dati necessaria per certe istruzioni, come ad esempio un raggruppamento dei dati su una certa chiave. A questo punto lo stage termina e ne viene creato uno nuovo. Ogni stage è poi diviso in task in numero pari a quello delle partizioni moltiplicato per il numero delle trasformazioni da eseguire. Ogni task è assegnato a un nodo con il principio della data locality, avendo però attenzione di non sovraccaricare il sistema. In caso di problemi, un task può essere schedulato su nodi differenti in modo da prevenire blocchi e ritardi.

Trattando infine dell'architettura di Spark, il paradigma adottato è ancora una volta master-slave. Di seguito sono trattati i principali componenti dell'architettura di Spark:

- **Driver**. Master dell'architettura, unico nell'applicazione. Gestisce le fasi della creazione e ottimizzazione di DAG, inoltre assegna i task ai vari executor.

- **Executor**. Processo avente il compito di eseguire su ogni core un task. Ad ogni applicazione possono essere assegnati più executor, ciascuno avente una certa quantità di RAM e un numero fissato di CPU.

- **Cluster Manager**. Componente che si occupa delle risorse. Può essere un processo stanalone o un resource manager compatibile con Spark, come ad esempio YARN.

2.2.2 SparkSQL

Spark SQL è un modulo costruito sopra Spark. Obbiettivo di Spark SQL è quello di poter integrare l'elaborazione di dati strutturati e semi-strutturati tramite primitive simili al linguaggio SQL. La necessità alla base di Spark SQL è stata l'integrazione di computazione procedurale e query SQL su database di grandi dimensioni. Spark SQL nasce quindi come modulo che integra le API procedurali con il modello relazionale. Altri punti fondamentali per Spark SQL sono l'efficienza nelle operazioni e il supporto a diversi database esterni.

Posti questi obbiettivi, il framework Spark SQL fornisce:

- **Integrazione**. Spark SQL permette di interrogare strutture dati all'interno delle applicazioni Spark. Queste interrogazioni possono essere fatte o tramite API su RDD o con query SQL. Il framework è disponibile per i linguaggi in cui è disponibile Spark.
- **Compatibilità con Hive**. Viene supportata la sintassi di HiveQL ed è totale la compatibilità con i dati, le query e le user defined function su Hive.
- **Connettività Standard**. Vengono supportati gli standard industriali JDBC e ODBC.
- **Accesso Uniforme ai dati**. Sono supportate diverse fonti di dati, come ad esempio Hive, Avro, JSON o Parquet. Questi dati sono trattati nella stessa maniera e, una volta caricati, possono essere integrati tra di loro.
- **Scalabilità**. Spark SQL combina la laziness del modello RDD di Spark con una gestione colonnare e un ottimizzatore interno di query per migliorare i risultati.

La caratteristica principale di Spark SQL è la definizione di una nuova astrazione, chiamata Dataframe. Un Dataframe è una collezione di dati distribuita organizzata in colonne, rendendolo di fatto equivalente a una tabella relazionale. Questa astrazione è basata su RDD, ne conserva quindi tutte le proprietà e modalità di computazione.

3 Pattern di co-movimento

3.1 Definizione di pattern di co-movimento

Un'analisi cruciale che può essere eseguita su un dataset di traiettorie è sicuramente la ricerca di oggetti che si muovono assieme. Come detto nella sezione 1.3.2, il mining di traiettorie pone l'attenzione sulla sul percorso e non sugli oggetti che ne fanno parte. L'analisi dei gruppi di oggetti è molto simile nelle modalità a quella dei percorsi/luoghi frequenti, tuttavia le potenzialità informative sono totalmente diverse. Analizzando i gruppi di oggetti si può vedere come questi variano nello spazio e nel tempo, ad esempio vedendo in quali gruppi un oggetto viaggia e per quanto. Questa analisi può portare a diversi risultati, ad esempio sulla base di regolarità di certi gruppi di viaggio è possibile dedurre informazioni sulle modalità di viaggio di un utente. Ancora la ricerca di gruppi di oggetti in movimento può fornire informazioni sul comportamento comune e quindi la natura di un certo gruppo: ad esempio un gruppo di molti utenti che viaggia insieme per un lungo periodo di tempo nel perimetro di una città potrebbe essere collegato a un tour turistico, al contrario un gruppo numeroso che viaggia assieme per un breve periodo potrebbe indicare un insieme di persone che stanno viaggiando su un mezzo pubblico.

Un *co-movement* pattern [37] individua un gruppo di oggetti che si sono mossi assieme per un certo tempo. L'appartenenza a tale gruppo è determinata solitamente dalla vicinanza nello spazio. La ricerca di questi pattern di movimento include diversi parametri che definiscono le caratteristiche dei gruppi individuati. Varie tipologie di cluster sono state definite in letteratura sulla base dell'adozione e configurazione di questi parametri.

Prima di scendere nel dettaglio sui singoli pattern, occorre definire gli elementi principali del problema. Innanzitutto dato un dataset di traiettorie $TR_{db} = \{tr_1, \dots, tr_n\}$, da questo viene derivato il dataset degli oggetti che hanno generato quelle traiettorie, $O_{db} = \{o_1, \dots, o_m\}, m \leq n$ e un dataset contenente tutti i possibili istanti temporali del primo dataset $T_{db} = \{t_1, \dots, t_k\}$. In generale, nella ricerca di *co-movement* pattern si ricerca un gruppo di oggetti $O = \{o_1, \dots, o_p\}, O \subseteq O_{db}$ tale che il corrispondente $T = \{t_1, \dots, t_j\} T \subseteq T_{sb}$, definito come l'insieme degli istanti in cui gli oggetti di O sono vicini, goda di certe proprietà, come ad esempio una lunghezza minima, oppure una continuità all'interno del tempo. Due parametri comuni a tutti i pattern di movimento sono m, che individua una soglia minima per la dimensione di O e k, limite inferiore al numero di istanti temporali in

cui il gruppo in questione è considerato vicino. La tabella 3.1 riassume i principali vincoli che saranno adoperati nella ricerca di pattern.

Parametro	Vincolo espresso
m	dimensione minima del gruppo
k	minimo numero di istanti temporali in cui il gruppo è stato assieme
l	lunghezza minima di ogni sotto-sequenza continua di T

Tabella 3.1: Parametri per la ricerca di pattern di co-movimento e il loro significato

I pattern di movimento possono essere suddivisi in due categorie a seconda dell'algoritmo di clustering utilizzato per riconoscere quali punti siano vicini e quali no. Se viene impiegato un clustering basato sulla distanza, allora si parla di *distance based pattern* (pattern basati sulla distanza), mentre se viene utilizzato un algoritmo basato sulla densità, allora si parla di *density based pattern*. Per semplicità nelle tipologie di pattern presentate d'ora in poi si fa riferimento a pattern basati sulla densità.

Il primo pattern di movimento individuabile alla luce dei vincoli è *swarm* [38]. In maniera informale, swarm ricerca gruppi di m oggetti che hanno viaggiato assieme per almeno k istanti temporali senza porre alcun ulteriore vincolo. Uno swarm può essere definito come segue:

Definizione 3.1.1 (Swarm)**.** Una coppia $\{O, T\}$ si definisce swarm se:

$$\begin{cases} \forall t \in T, \exists c \; t.c \; O \in c \\ |O| \geq m \\ |T| \geq k \end{cases}$$

La definizione 3.1.1 formalizza i seguenti vincoli: ad ogni istante di T gli oggetti di O devono appartenere a uno stesso cluster, il gruppo deve essere poi rilevante dal punto di vista degli elementi (m) e del tempo trascorso (k). Per quanto riguarda il primo vincolo, è possibile utilizzare varie metriche per fare clustering sulla dimensione spaziale del dataset, tuttavia l'algoritmo più utilizzato per la ricerca di swarm è DBSCAN (sezione 1.2).

Il concetto di swarm può essere ulteriormente raffinato in quello di *closed swarm*: un closed swarm intuitivamente si definisce allo stesso modo di uno swarm, ma con l'ulteriore vincolo di considerare la massima sequenza di istanti in cui gli oggetti dentro O risultano vicini tra di loro (chiusura rispetto al tempo) oppure il numero massimo di oggetti dato un certo T (chiusura rispetto agli oggetti). Formalmente, ciò è espresso nella definizione 3.1.2

Definizione 3.1.2 (Closed Swarm)**.** Una coppia $\{O, T\}$ si definisce closed swarm se:

$$\begin{cases} \{O, T\} \; risulta \; swarm \\ \nexists O' \; t.c. \; \{O \cup O', T\} risulta \; swarm \\ \nexists T' \; t.c. \; \{O, T \cup T'\} risulta \; swarm \end{cases}$$

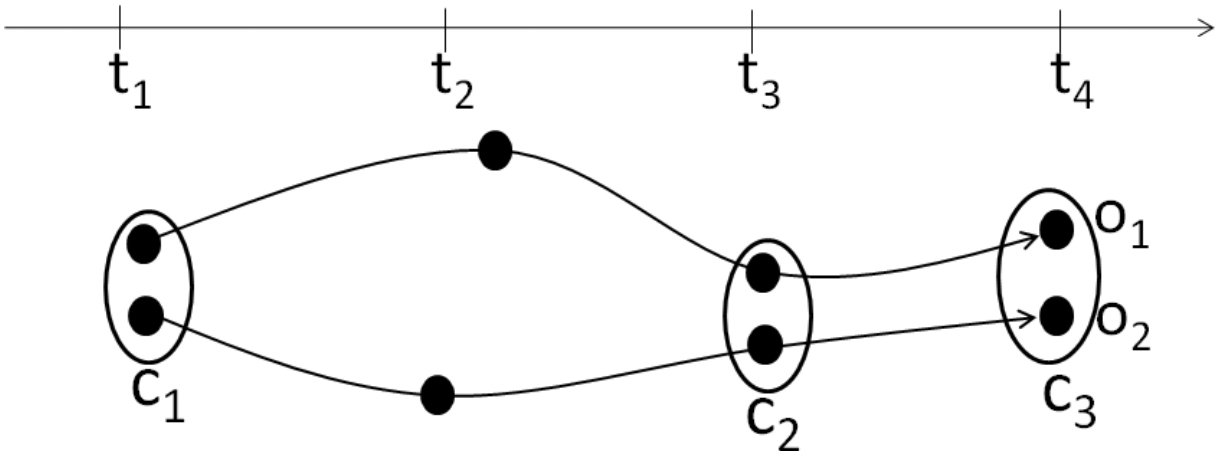


Figura 3.1: Ricerca di swarm su un dataset fissati $m = 2$ e $k = 3$, Fonte:[40]

Com'è possibile vedere in figura 3.1 (Fonte: [39]), la ricerca di swarm produce riconosce il gruppo $\{o_1, o_2\}$ in quanto i due oggetti risultano avere viaggiato vicini per almeno tre istanti temporali (t_1, t_2, t_4).

Entrambi i pattern appena presentati rilassano al massimo i vincoli sul tempo, accettando gruppi aventi istanti temporali parecchio distanti gli uni dagli altri. Un tipo di analisi che aggiunge un rigido vincolo sulla continuità degli istanti temporali è la ricerca di *convoy* [41]: un convoy per definizione è un raggruppamento di oggetti O in cui tutti gli istanti in T sono consecutivi, mantenendo i precedenti vincoli sulle dimensioni del gruppo e sul tempo trascorso assieme.

Definizione 3.1.3 (Convoy)**.** Una coppia $\{O, T\}$ si definisce convoy se:

$$\begin{cases} \{O, T\}\ risulta\ swarm \\ \forall t_i \in T, t_{i+1} = t_i + 1 \end{cases}$$

Convoy utilizza un algoritmo basato su densità, come ad esempio DBSCAN, per determinare la vicinanza o meno di due oggetti in base all'appartenenza a un certo cluster ad ogni t_i. Qualora si voglia adottare un algoritmo basato sulla distanza per determinare la vicinanza, allora i pattern individuati sono chiamati *flock* [42] pattern. La figura 3.2 (Fonte: [39]) mostra un esempio di ricerca di convoy: fissati $m = 2$ e $k = 3$, risulta come pattern valido $\{o_1, o_2\}$, $\{o_1, o_2, o_3\}$ viene scartato in quanto non soddisfa il vincolo di continuità sugli istanti temporali.

Convoy e swarm rappresentano i due casi limite per quanto riguarda la rigidità dei vincoli temporali: swarm rilassa totalmente la continuità mentre convoy esige una rigidità assoluta nella sequenza degli istanti temporali. Una via di mezzo tra i due estremi è individuata nel pattern *group* [43]. Un pattern group individua un insieme di convoy disgiunti nel tempo relativi allo stesso gruppo di oggetti O; interpretando ogni convoy come un singolo punto temporale t_s, un pattern group può essere visto come uno swarm di convoy. Ognuno dei singoli punti così individuati dovrà risultare come valido convoy, si definisce il parametro l

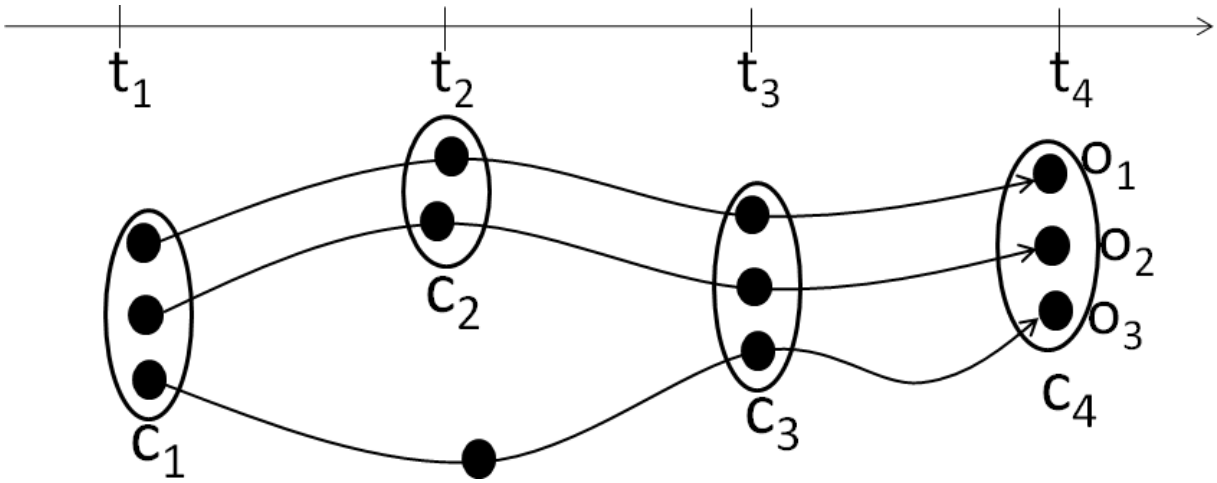


Figura 3.2: Ricerca di convoy su un dataset fissati $m = 2$ e $k = 3$, Fonte:[40]

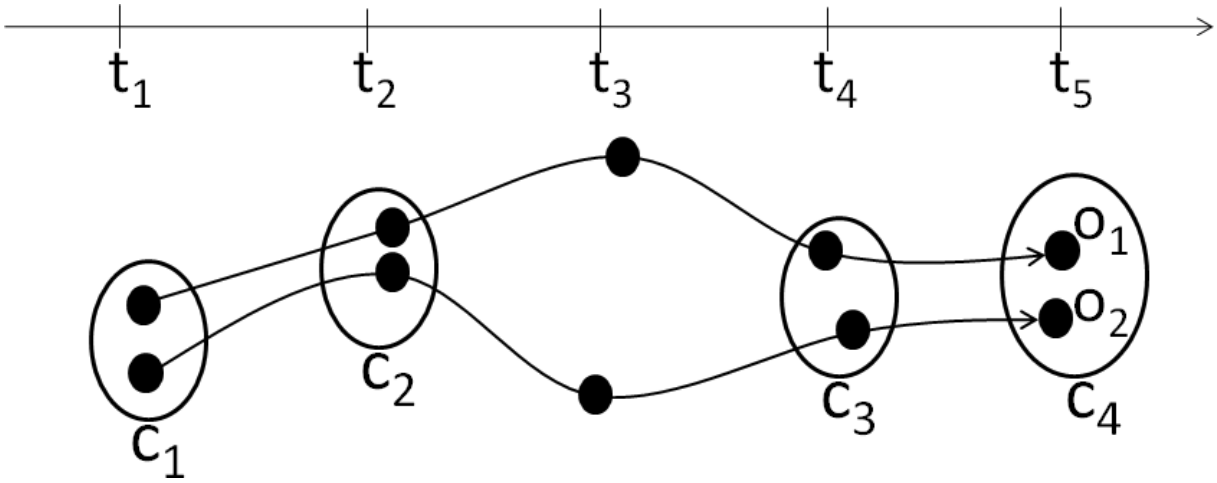


Figura 3.3: Ricerca di group su un dataset fissati $m = 2$ e $l = 2$, Fonte:[40]

come la lunghezza di istanti condivisi in ognuno dei punti individuati. Il valore di k per lo swarm così individuato indica il numero minimo di convoy necessari per il riconoscimento di group, solitamente tale valore è fissato a 1.

Definizione 3.1.4 (Group)**.** Definito T come l'insieme totale di istanti in cui gli oggetti di O sono vicini, T_s come l'insieme delle sotto-sequenze continue e disgiunte $T' \subseteq T$ una coppia $\{O, T_s\}$ si definisce group se:

$$\begin{cases} \{O, T_s\}\ risulta\ swarm \\ \forall t_s \in T_s,\ |t_s| \geq l \end{cases}$$

La figura 3.3 contiene un esempio di ricerca di pattern group, com'è possibile vedere il parametro k non è considerato nella ricerca.

Group introduce la possibilità di ricercare gruppi con una continuità rilassata, tuttavia non pone nessun vincolo sul numero complessivo di istanti necessari per considerare un pattern interessante. Per integrare questo parametro, è stato definito il pattern *platoon* [44]. Questo approccio interpreta diversamente il valore di k rispetto a quanto fatto in group:

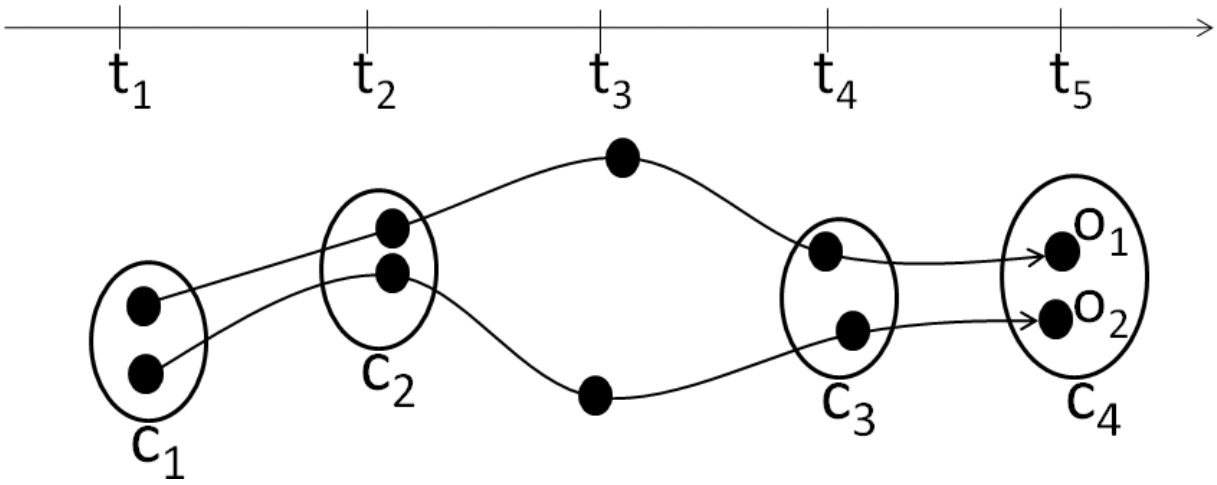


Figura 3.4: Ricerca di Platoon con $m = 2$, $k = 3$, $l = 2$, Fonte:[40]

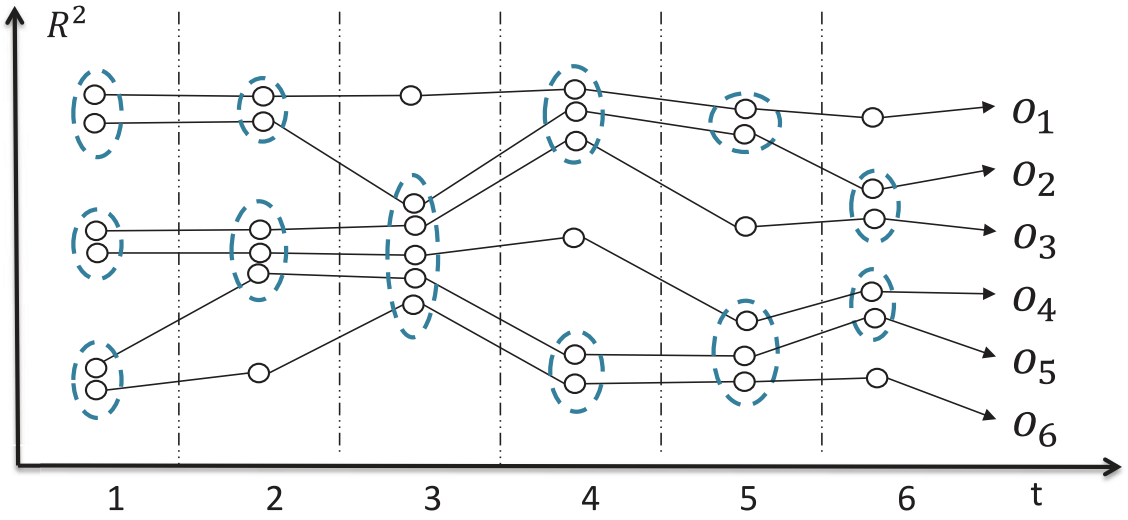
se in quest'ultimo approccio k indicava il numero di convoy necessari per individuare un raggruppamento valido, ora k indica il numero di istanti assoluti necessari per un gruppo valido. In termini formali, il pattern platoon può essere definito come segue:

Definizione 3.1.5 (Platoon)**.** Definito T come l'insieme totale di istanti in cui gli oggetti di O sono vicini, T_s come l'insieme delle sotto-sequenze continue e disgiunte $T' \subseteq T$ una coppia $\{O, T\}$ si definisce platoon se:

$$\begin{cases} \{O, T\}\ risulta\ swarm \\ \forall T' \in T_s,\ |T'| \geq l \end{cases}$$

Com'è possibile vedere nella figura 3.4, in questo caso i risultati di platoon e group coincidono. Tuttavia non è sempre così: confrontando l'applicazione dei pattern appena descritti a uno stesso dataset, la figura 3.5 mostra quali sono le differenze, a parità di valori di m, k e l, tra i diversi pattern di movimento. Dalla figura in questione emergono chiaramente le differenze tra i gruppi individuati dai vari pattern: ad esempio group riconosce $\{o_3, o_4, o_5\}$ negli istanti $\{1, 2\}$ poiché la lunghezza di tale sotto-sequenza risulta uguale ad l; platoon invece spezza il raggruppamento in $\{o_3, o_4\}, \{o_4, o_5\}$ poiché quest'ultimo non avrebbe rispettato il vincolo sulla lunghezza minima degli istanti k. Convoy e Swarm invece riconoscono i pattern aventi una continuità assoluta in un caso, totalmente assente nell'altro. Entrambi ignorano il valore del parametro l.

La tabella 3.2 riassume i pattern di co-movimento appena presentati in relazione ai vincoli espressi.



Flock (M=2, K=3)	$\langle o_3, o_4 : 1,2,3\rangle$ $\langle o_5, o_6 : 3,4,5\rangle$
Convoy (M=2, K=3)	$\langle o_3, o_4 : 1,2,3\rangle$ $\langle o_5, o_6 : 3,4,5\rangle$
Group (M=2, L=2)	$\langle o_1, o_2 : 1,2,4,5\rangle$ $\langle o_3, o_4, o_5 : 2,3\rangle$ $\langle o_5, o_6 : 3,4,5\rangle$
Platoon (M=2,K=3,L=2)	$\langle o_1, o_2 : 1,2,4,5\rangle$ $\langle o_3, o_4 : 1,2,3\rangle$ $\langle o_4, o_5 : 2,3,5,6\rangle$ $\langle o_5, o_6 : 3,4,5\rangle$
Swarm (M=2,K=3)	$\langle o_1, o_2 : 1,2,4,5\rangle$ $\langle o_3, o_4 : 1,2,3\rangle$ $\langle o_4, o_5 : 2,3,5,6\rangle$ $\langle o_5, o_6 : 1,3,4,5\rangle$ $\langle o_2, o_3 : 3,4,6\rangle$

Figura 3.5: Ricerca dei principali pattern di movimento su un dataset fissati $m = 2$, $k = 3$ e $l = 2$, Fonte: [26]

Pattern	Dimensione del gruppo	Numero di istanti	Continuità
swarm	m	k	-
closed swarm	m	k	-
convoy	m	k	globale
group	m	-	locale
platoon	m	k	locale

Tabella 3.2: Parametri per la ricerca di pattern di co-movimento e il loro significato

3.2 L'algoritmo SPARE

Come può essere visto nella sezione precedente, sono presenti varie tipologie di pattern di co-movimento. Platoon risulta il più generale tra tutti quelli appena esposti, essendo in grado di rappresentare gli altri con certe combinazioni specifiche di parametri. I vari pattern quindi non sono totalmente separati tra di loro, ma possono essere ricondotti a una formulazione comune.

Un problema che affligge tutti i pattern di movimento in cui la continuità nel tempo è locale è l'anomalia *loose-connection*, o della connessione interrotta. Questa anomalia si presenta quando un gruppo di oggetti viaggia assieme per brevi periodi continui intervallati da lunghe distanze in cui il gruppo si rompe. La figura 3.6 mostra un esempio di anomalia:

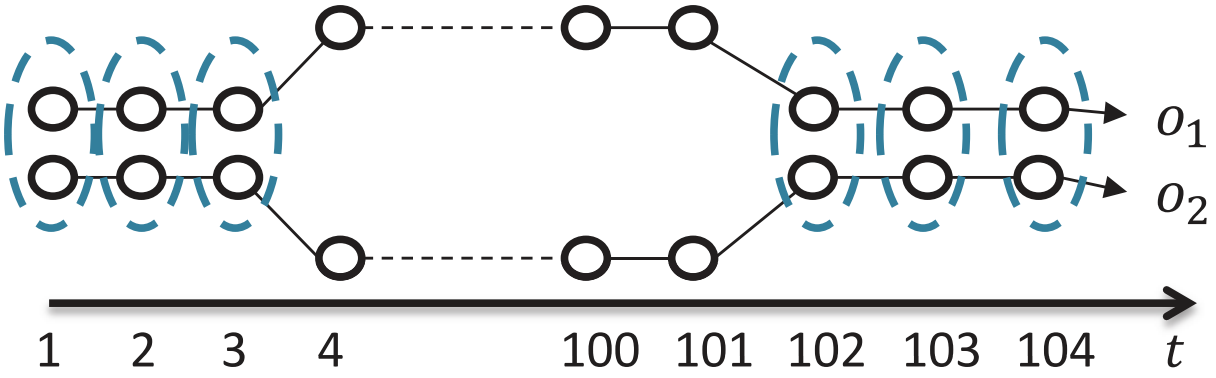


Figura 3.6: Anomalia di connessione interrotta nel pattern o_1, o_2: costituiscono un platoon valido ma le sotto-sequenze sono a 98 istanti l'una dall'altra, Fonte: [26]

gli oggetti o_1, o_2 viaggiano assieme negli istanti $1, 2, 3$ e $102, 103, 104$. Questo gruppo costituirebbe un platoon valido impostando $m = 2, k = 4, l = 3$, tuttavia i quasi cento istanti temporali tra le due sotto-sequenze continue portano a pensare che la relazione tra queste sia debole, essendo molto distanti nel tempo.

GCMP, o General Co-Movement Pattern, è un astrazione per modellare varie tipologie di pattern di movimento. Questa si occupa non solo di modellare i principali pattern di movimento con una tecnica univoca, ma anche di risolvere l'anomalia della connessione interrotta.

A questo proposito viene aggiunto il parametro g, o gap temporale, agli altri parametri individuati nella sezione 3.1. Dati due istanti temporali t_1, t_2, g rappresenta la massima distanza in termini di istanti tra questi. Questo ulteriore vincolo permette di limitare l'anomalia della connessione interrotta: ad esempio nel caso descritto in figura 3.6 basta specificare $g = 10$ per scartare il pattern.

GCMP è quindi in grado di modellare pattern di movimento aventi le seguenti proprietà:

- **Chiusura**. Dato un gruppo di oggetti O in una sequenza temporale T, ogni oggetto o_i risulta vicino ad ogni altro oggetto di O ad ogni istante temporale $t_i \in T$.
- **Importanza**. Definito un valore di m, il numero di oggetti in O deve essere maggiore o uguale a m.
- **Durata**. Definito un numero minimo di istanti temporali k per considerare un gruppo interessante, $|T| \geq k$.
- **Consecutività**. Definito l come il valore minimo di consecutività locale, ogni sotto-sequenza continua di T deve risultare maggiore o uguale a l. Una sequenza che rispetta questa proprietà viene detta *L-consecutive*, o consecutiva rispetto a l.
- **Connessione**. Definito g come il massimo gap tra due punti nel tempo, deve valere che $\forall t_i \in T, d_t(t_i, t_{i+1}) \leq g$. Una sequenza che rispetta questa proprietà viene detta *G-connected*, o connessa rispetto a g.

GCMP è quindi in grado di modellare tutti i pattern di co-movimento fin d'ora descritti variando i parametri come descritto nella tabella 3.3:

Pattern	M	K	L	G
swarm	m	k	1	∞
closed swarm	m	k	1	∞
convoy	m	k	k	1
group	m	1	l	∞
platoon	m	k	l	g

Tabella 3.3: Configurazioni di parametri di GCMP per la ricerca dei principali pattern di co-movimento

SPARE (Star Partitioning and ApRiori Enumerator) è un algoritmo per la ricerca di GCMP. Questo framework realizza la ricerca di questi pattern suddividendo lo spazio-tempo in istanti temporali e individuando ad ogni istante quali traiettorie risultino vicine e quali no. Successivamente genera degli itemset utilizzando gli oggetti come item e fondendoli assieme. Intuitivamente due item saranno considerati interessanti e quindi fusi in un unico itemset se questi hanno compaiono assieme in k istanti tali che la continuità locale sia almeno di l elementi e il massimo gap non sia superiore a g. Una volta individuati questi itemset validi, l'algoritmo procede a ricercare gli itemset di almeno m elementi con le tecniche del frequent itemset mining.

Nell'ambito del lavoro di questa tesi, l'approccio di SPARE risulta interessante per due principali motivi: in primo luogo SPARE utilizza una tecnica di generazione e ricerca dei gruppi di movimento a metà tra il clustering e il frequent itemset mining. Questo approccio risulta sicuramente innovativo rispetto a quanto visto fino ad ora in letteratura. Successivamente SPARE è implementato in maniera distribuita: questo implica che possa processare enormi moli di dati in tempistiche ragionevoli.

Una volta presentate le ragioni per cui è stato scelto questo algoritmo, vengono illustrate le tre fasi di SPARE:

1. **Snapshot Generation**. Dividendo il tempo in singoli istanti, esegue un clustering sulle posizioni spaziali di ogni oggetto ad ogni istante.

2. **Star Partitioning**. Gli snaphot della fase precedente sono fusi in un grafo a stella, che successivamente viene suddiviso in ulteriori sotto-grafi sulla base dei nodi.

3. **Apriori Enumerator**. Sfruttando una definizione custom di monotonicità basata sul tempo, esegue la trasformazione dei grafi in item. Successivamente conduce una ricerca di itemset frequenti e massimali, utilizzando il principio di *forward closure*.

La figura 3.7 riassume i passaggi dell'algoritmo SPARE. Ciascuno di questi sarà analizzato nei dettagli nelle sezioni successive di questo documento.

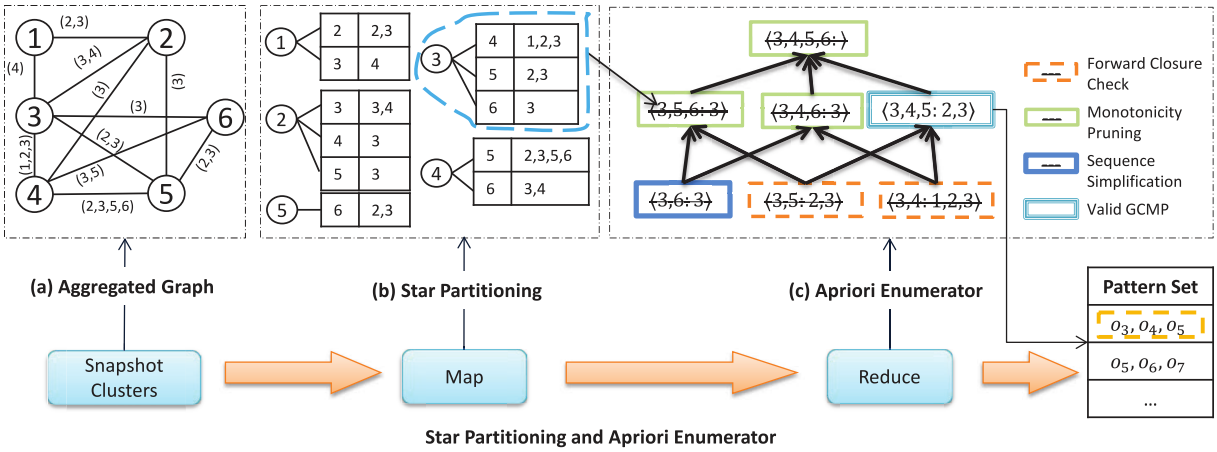


Figura 3.7: Fasi di SPARE: [26]

3.2.1 Snapshot Clusters

La prima fase di SPARE è la generazione degli Snapshot. Durante questo passo viene valutata la dimensione spaziale delle traiettorie in relazione alla dimensione temporale. Scopo della fase è individuare ad ogni istante quali oggetti sono vicini e quali no, in modo da creare una sequenza di cluster che descrivono l'evolvere dei gruppi nel tempo.

Unico passo di preprocessing necessario prima di questa fase è la definizione di una scala temporale univoca all'interno del dataset. Tutti i punti devono infatti essere rapportati a una scala di tempo omogenea, altrimenti i passi successivi non sarebbero possibili. Il primo passo consiste quindi nella definizione di una sequenza di istanti univoci chiamata T_{db}. Tutti i punti di ogni traiettoria saranno espressi rispetto a questa scala.

Una volta determinata la scala temporale, si scompone ogni traiettoria nei suoi singoli punti. Questi punti vengono poi raggruppati sulla base dell'istante temporale in cui accadono. Per ogni istante temporale nella scala sarà determinato un insieme di punti che indicano la posizione delle rispettive traiettorie al tempo $t_i \in T_{db}$.

A questo punto è necessario valutare quali punti siano vicini e quali no. Come detto nella sezione 3.1, algoritmi di clustering diversi generano pattern di movimento diversi (density o distance based). SPARE supporta la ricerca di entrambi i pattern: sta all'utente scegliere quale algoritmo impiegare. Questa scelta risulta trasparente alle successive fasi di itemset mining.

Questo è l'unico passo dove vengono considerate le dimensioni spaziali. Intuitivamente, ogni traiettoria avrà al massimo una posizione per ogni istante temporale. Il processo di clustering risulterà quindi partizionante, così da prevenire eventuali ambiguità nelle fasi successive.

Questa operazione produce una serie di raggruppamenti di cluster. Uno snapshot S_t non è altro che una coppia $\langle t, S(t)\rangle$ dove t rappresenta un instante temporale e $S(t)$ l'insieme dei cluster individuati a quello specifico istante.

Gli snapshot sono l'output di questa prima fase.

3.2.2 Star partitioning

Scopo di questa seconda fase è trasformare gli snapshot in itemset di due elementi che verranno poi processati nella fase successiva. Durante la fase di star partitioning vengono aggregati gli snapshot generati nella fase precedente in maniera tale da avere per ogni coppia di oggetti l'insieme degli istanti temporali in cui hanno viaggiato vicini. Successivamente per ciascun oggetto si valuterà in maniera univoca tutte le possibili combinazioni tra questo e gli altri oggetti, generando così un insieme di coppie, o itemset 2-dimensionali, collegate alla sequenza di istanti in cui questi risultano vicini.

Dato uno snapshot S_t questo può essere rappresentato con un grafo non direzionato G_t. Tale grafico avrà nei nodi gli identificatori degli oggetti, tra nodi sarà presente un arco se quei due oggetti appartengono allo stesso cluster al tempo t. È possibile definire G_t per ogni snapshot S_t.

Il singolo grafo tuttavia è poco espressivo, contiene infatti i dati relativi solo a un singolo istante temporale. Si definisce quindi G_A o grafo aggregato una struttura composta dai singolo G_t e che descrive l'evoluzione dei cluster nel tempo. G_A contiene gli stessi nodi di ogni G_t. Gli archi invece sono formulati nel seguente modo: esiste un arco tra due nodi se, all'interno di tutti i cluster presenti in tutti gli snaphot individuati, questi elementi compaiono assieme in almeno un cluster. Oltre a questo sull'arco viene salvata la sequenza di istanti in cui i due oggetti compaiono nello stesso cluster.

Questo grafo così creato viene poi diviso in partizioni. Per fare ciò si utilizza una struttura a stella, opportunamente modificata per evitare potenziali ripetizioni di coppie di nodi. I nodi, o vertici, vengono infatti numerati sulla base di un ordinamento globale. Una struttura così definita viene chiamata *directed star*, o stella diretta (definizione 3.2.1)

Definizione 3.2.1 (Stella diretta)**.** Dato un vertice con ID globale s si definisce una stella diretta Sr_s l'insieme dei vertici direttamente raggiungibili, o vicinato, aventi ID globale $t > s$, definito s come ID della stella.

G_A viene diviso in n stelle dirette, dove n è il numero di nodi nel sistema. Grazie all'ordine globale dei vertici, ogni arco è contenuto in una sola partizione del grafo iniziale. Ogni stella viene poi processata in maniera autonoma dalla fase di Apriori Enumeration

Questa operazione ha però un limite, ovvero il criterio di ordinamento. Un criterio di ordinamento poco efficace porta a generare stelle sbilanciate. Stelle con pochi archi terminano presto la computazione, mentre il sistema deve attendere che anche le stelle più grandi finiscano di essere computate. Occorre quindi utilizzare un criterio di ordinamento efficace.

Dati n nodi, sono possibili $n!$ ordinamenti. Euristicamente si dimostra che un ordinamento basato sull'identificatore dei nodi porta a una suddivisione abbastanza bilanciata, di conseguenza SPARE non introduce nessun criterio di ordinamento particolare.

3.2.3 Apriori enumerator

Ultimo passo dell'algoritmo: date in ingresso le varie stelle dirette prodotte al passo precedente, produce gruppi di movimento come itemset. Intuitivamente questa fase riconosce tra le stelle della fase precedente quali coppie di nodi possono essere considerate come valide. Una coppia di nodi sarà valida se la sequenza collegate risulterà significativa rispetto a k, l e g. Tutte le coppie valide saranno poi fuse in itemset, intersecando ogni volta le sequenze temporali e assicurandosi che ad ogni passo l'intersezione rimanga valida. Questo processo di esplorazione produce itemset massimali aventi dimensione maggiore di m.

Passando poi al procedimento, si parte dalle stelle generate alla fine della fase precedente. Ogni stella viene scomposta in coppie nella forma $\langle(o_v, o_n) : (t_1, \dots, t_n)\rangle$. o_v è il vertice della stella, o_n è uno dei nodi della stella, $(t_1, \dots, t_n)$ sono gli istanti temporali sull'arco (o_v, o_n).

Eseguendo una generazione e ricerca di itemset, si può pensare di applicare l'algoritmo Apriori (sezione 1.3.1). Tuttavia non è possibile applicare questo algoritmo. Il principio della monotonicità tra un set e un suo superset non è valido nel caso degli itemset individuati da SPARE.

Si supponga di considerare due candidati $C_1 = \langle(o_1, o_2) : 1, 2, 3, 6\rangle$ e $C_2 = \langle(o_1, o_3) : 1, 2, 3, 7\rangle$. Fissati $m = 2, k = 3, l = 2, g = 1$, né C_1 né C_2 risultano itemset validi, violano infatti sia il vincolo su l che su g. In C_1, $maxgap = (6-3) > g$ mentre per $C2$, $maxgap = (7-3) > g$. Per quanto riguarda l invece $C_1 = \{C_1' = \langle(o_1, o_2) : 1, 2, 3\rangle, C_1'' = \langle(o_1, o_2) : 6\rangle\}$, risulta evidente che $|C_1''| < l$. Altrettanto vale per $C_2 = \{C_2' = \langle(o_1, o_3) : 1, 2, 3\rangle, C_2'' = \langle(o_1, o_3) : 7\rangle\}$, anche qui $|C_2''| < l$. Definito $C_3 = C_1 \cup C_2 = \langle(o_1, o_2, o_3) : 1, 2, 3\rangle$, questo itemset risulta valido rispetto a tutti e quattro i parametri. C_3 quindi è un superset valido di due set non validi: ciò viola la definizione classica di monotonicità.

Occorre quindi definire una nuova nozione di monotonicità. Per giungere a questo obbiettivo sono necessari tre nuovi concetti: *Maximal G-Connected sequence*, *Decomposable sequence* e infine *Sequence semplification*.

Data una sequenza di istanti temporali T, la massima sequenza connessa rispetto a g (MGS) è la più lunga sotto-sequenza T' tale che non T' risulti connessa rispetto a g e non esista $T'' \supset T'$ che sia a sua volta connessa rispetto a g. Ogni MGS deve essere quindi connessa rispetto a g e massima nel numero di elementi. Ogni sequenza può essere divisa in n MGS. Ciascuna di queste MGS è disgiunta dalle altre se non per il primo o l'ultimo elemento. Inoltre l'unione di tutte le MGS produce la sequenza originale. Infine ogni MGS di una sotto-sequenza può essere considerata come sotto-sequenza di una MGS della sequenza originale. Prese ad esempio le sequenze $T' = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ e $T'' = (1, 2, 4, 5, 6)$ con $g = 2$. T' ha due MGS: $T_a' = (1, 2, 4, 5, 6)$ e $T_b' = (9, 10, 11, 13)$. T'' ha una sola MGS corrispondente alla sequenza stessa. T'' è sotto-sequenza di T', quindi ogni MGS di T'' dovrà essere sotto-sequenza di una MGS di T'. In questo caso, T'' e T_a' coincidono,

dimostrando la validità della proprietà.

Una volta definito il concetto di MGS, si può introdurre quello di sequenza decomponibile (*Decomposable sequence*). Una sequenza decomponibile è una sequenza che ha al suo interno almeno una sotto-sequenza valida rispetto a k, l e g. Una sequenza si definisce decomponibile se, presa una qualunque delle sue MGS, questa risulta consecutiva rispetto a l e avente lunghezza maggiore o uguale a k. Prese ad esempio la sequenza $T' = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ con $k = 5$, $l = 2$ e $g = 2$, si considerano le due MGS $T'_a = (1, 2, 4, 5, 6)$ e $T'_b = (9, 10, 11, 13)$. T'_a risulta sia consecutiva rispetto a l che lunga k elementi, di conseguenza T' può essere definito come sequenza decomponibile.

Infine si definisce semplificazione di sequenza (*Sequence simplification*). Data un sequenza T si definisce sequenza semplificata o $Sim(T)$ la sotto-sequenza di T ottenuta applicando due trasformazioni:

- **f-step**. Rimuove i segmenti di T che non rispettano la consecutività rispetto a l.
- **g-step**. Tra le MGS individuate sul risultato di f-step, scarta quelle che hanno dimensione minore di k.

Ad esempio dato $T = (1, 2, 4, 5, 6, 9, 10, 11, 13)$ e $l = 2, k = 3, g = 2$. Alla fine di f-step $T = (1, 2, 4, 5, 6, 9, 10, 11)$, 13 viene scartato in quanto facente parte di una sotto-sequenza di lunghezza uno. Questa nuova sequenza ha due MGS: $(1, 2, 4, 5, 6)$ e $(9, 10, 11)$. Il secondo di questi due elementi viene scartato in quanto avente dimensioni minori di k. La sequenza semplificata risulta quindi $(1, 2, 4, 5, 6)$.

Il processo di semplificazione di una sequenza può portare alla creazione di una sequenza nulla, qualora non siano verificate certe condizioni su l e k. Per definizione, la sequenza vuota non risulta decomponibile. Alla luce di ciò è possibile esprimere la seguente proprietà: per qualunque T supeset di una sequenza decomponibile vale che $Sim(T) \neq \emptyset$. Banalmente, per una sequenza decomponibile è impossibile che $Sim(T) \neq \emptyset$, ciò vale anche per ogni suo superset.

Una volta introdotto quest'ultimo concetto è possibile formulare la nuova definizione di monotonicità (definizione 3.2.2):

Definizione 3.2.2 (Monotonicità)**.** Dato un candidato pattern $P = \langle O : T \rangle$, se $Sim(P.T) = \emptyset$ allora ogni pattern P' tale che $P' \supseteq P$ può essere scartato.

La definizione si basa sulla considerazione che qualunque superset di P è composto dall'unione degli oggetti e dall'intersezione delle sequenze temporali. Risulta a questo punto scontato affermare che qualunque sotto-sequenza di una sequenza tale che $Sim(T) \neq \emptyset$ a sua volta produce una sequenza vuota durante il processo di semplificazione.

Definita la nuova monotonicità, lo pseudocodice della fasi di Apriori Enumerator è presentato nel algoritmo 1.

Algoritmo 1 *AprioriEnumerator*

Richiede: Sr_s

1: $C \leftarrow \emptyset$
2: **for each** archi $c =< o_i \cup o_j, T_i \cap T_j >$ in Sr_s **do** ▷ Per ogni 2-itemset.
3: **if** $Sim(T_i \cap T_j) \neq \emptyset$ **then** ▷ Se la semplificazione dell'intersezione delle sequenze risulta valida
4: $C \leftarrow C \cup c$ ▷ La tupla viene aggiunta a C
5: $level \leftarrow 2$
6: **while** $C \neq \emptyset$ **do**
7: **for each** $c_1 \in C$ **do**
8: **for each** $c_2 \in C \wedge |c_1.O \cup c_2.O| = level$ **do** ▷ Per tutti i validi n-itemset
9: $c' =< c_1.O \cup c_2.O, c_1.T \cap c_2.T >$ ▷ Vengono generati tutti gli n+1 itemset
10: **if** $Sim(c_1.T \cap c_2.T) \neq \emptyset$ **then** ▷ Se la semplificazione dell'intersezione delle sequenze risulta valida
11: $C \leftarrow C \cup c$ ▷ La tupla viene aggiunta a C
12: **if** Nessun c' viene aggiunto a C e c_1 è un pattern valido **then**
13: **return** c_1 ▷ c_1 viene restituito in output in quanto massimale
14: $O_u \leftarrow$ unione di tutti i $c.O$ in C
15: $T_u \leftarrow$ unione di tutti i $c.T$ in C
16: **if** $Sim(T_u) \neq \emptyset$ **then** ▷ Controllo della forward closure
17: **return** $< O_u, T_u >$ ▷ L'intersezione tra tutti i possibili itemset in C è massimale, quindi l'esplorazione è terminata

SPARE parte dalla valutazione degli itemset contenenti due elementi. Questi vengono filtrati secondo il principio di monotonicità descritto sopra.

Successivamente comincia l'esplorazione dello spazio di ricerca: ogni itemset valido n-dimensionale viene fuso con tutti gli altri tali da formare itemset $n+1$-dimensionali. Questi sono poi valutati con la stessa logica usata sopra: i validi vengono mantenuti, gli altri sono scartati. Nel caso un itemset durante questa fase non produca nessun superset valido, questo viene valutato come output. Se risulta rilevante rispetto a m e k viene aggiunto all'output, altrimenti viene scartato.

Una volta terminato di computare i candidati del livello successivo, viene eseguito un controllo sulla $forwardclosure$. Questa operazione consiste nel fondere assieme tutti gli itemset da valutare e testare se il candidato così costruito sia un itemset valido e rilevante. Se ciò risulta vero, allora questo può essere restituito in output in quanto massimale, altrimenti i candidati singoli sono valutati con la stessa logica descritta sopra.

4 Colossal Trajectory Mining

Le tecniche presentate nei capitoli precedenti rappresentano lo stato dell'arte per quanto riguarda il clustering di traiettorie. In particolare la ricerca dei pattern di movimento presenta un interessante mix tra l'approccio clustering e mining di itemset frequenti. Il framework SPARE in particolare implementa la ricerca di pattern di co-movimento generici su piattaforma big data.

Tra le possibilità inesplorate della letteratura ci sono le tecniche di mining ad alta dimensionalità. Queste sono state applicate ai dati di traiettoria ma mai per ricercare pattern di co-movimento. Per questo scopo è stato realizzato e successivamente adattato l'algoritmo CUTE. CUTE consente di ricercare pattern di movimento su un numero custom di dimensioni mediante un approccio di mining ad alta dimensionalità. CUTE inoltre è implementato sul framework spark (sezione 2.2).

In questo capitolo sarà presentato prima l'approccio del mining di itemset ad alta dimensionalità applicato ai dati di traiettoria (*Colossal Trajectory Mining*). In primo luogo verrà esposto il problema del *Colossal Itemset Mining*, o ricerca di itemset su dati ad alta dimensionalità. Successivamente si tratterà dell'algoritmo di Carpenter e delle applicazioni/limiti di questo approccio in letteratura.

Una volta fatto questo, si passerà alla formulazione dell'algoritmo CUTE e del contributo apportato da questo nuovo approccio nell'ambito del clustering dei dati di traiettoria.

Di CUTE in primo luogo verranno presentate le esigenze che hanno portato alla stesura dell'algoritmo e delle novità dell'approccio rispetto allo stato dell'arte, successivamente saranno presentate le fasi dell'algoritmo e i dettagli implementativi.

4.1 Colossal Itemset Mining

Negli ultimi anni, la maggiore disponibilità di risorse, le nuove tecniche di elaborazione dati e piattaforme Big Data ha reso possibile gestire dataset di dimensioni sempre maggiori. A crescere però non è solo il numero dei dati, ma anche la loro dimensionalità. Un caso su tutti è l'ambito bio-informatico: i dataset di espressione dei geni raggiungono 10.000-100.000 attributi per 100-1000 righe. Questi dataset sono una sfida per le tecniche di frequent itemset mining per la seguente ragione: il numero di itemset candidati cresce esponenzialmente rispetto alle dimensioni del set di attributi. Ciò implica che con elevati numeri di attributi, la generazione con le tecniche classiche, come Apriori [28], diventi

proibitiva. Conseguentemente al numero dei candidati aumenta esponenzialmente anche il numero degli itemset frequenti. Sono generati infatti, per ogni set frequenti, tutti i suoi possibili subset, in quanto loro stessi frequenti.

Il *colossal itemset mining* si pone come un'estensione del frequent itemset mining volta ad approcciare i dataset con le caratteristiche descritte sopra. Questa espansione riguarda l'utilizzo di dati a elevata dimensionalità [45] all'interno delle operazioni di mining. Il principale cambiamento che permette al colossal itemset mining di essere efficiente nell'esplorazione dei possibili itemset sono le proprietà ricercate sopra questi ultimi. Gli itemset prodotti sono infatti o chiusi (definizione 1.3.4) o massimali (definizione 1.3.5). Generando solo pattern di queste due categorie lo spazio di ricerca cala, sebbene non calino le dimensioni degli itemset individuati. Affrontando la ricerca con le tecniche di mining tradizionali, come ad esempio Apriori (sezione 1.3.1), il numero di itemset generati e valutati nel caso peggiore è 2^n, dove n è il numero di attributi del dataset. Analogamente, dato un itemset chiuso di m elementi, le ricerche di mining tradizionali richiedono di generare quel pattern e i 2^m figli frequenti per definizione di monotonicità. Gli algoritmi di mining colossale non esplorano questi 2^m figli frequenti e mantengono solo l'itemset chiuso. Il numero di pattern individuati e i tempi necessari calano quindi drasticamente, a scapito della granularità della ricerca.

La ricerca di questi pattern è limitativa, ma può produrre comunque risultati validi. Nei casi in cui il dataset abbia un numero elevato di attributi, molto spesso sono ritenuti "interessanti' solo i gruppi di attributi particolarmente lunghi. La ragione è che spesso questi pattern hanno un maggior significato rispetto agli itemset più corti [45]. Ad esempio nell'ambito della bioinformatica sequenze genetiche più lunghe e complesse sono più indicative per determinare la predisposizione a certe malattie. Al contrario sequenze brevi hanno spesso relazioni più deboli con potenziali problematiche.

4.1.1 Estrazione di itemset colossali:Carpenter

Carpenter [30] è uno dei principali framework per la ricerca di itemset su dataset ad alta dimensionalità.

Scopo dell'algoritmo è la ricerca di itemset colossali chiusi su dati ad alta dimensionalità. Questo algoritmo individua insiemi di feature frequenti generando una versione trasposta del dataset originale, chiamata TT, e un albero, chiamato *row enumeration tree* avente nei nodi tutti i possibili set di transazioni; successivamente questo viene esplorato utilizzando una ricerca *depth-first*.

Il punto di partenza di Carpenter consiste in un insieme di transazioni D che contiene F attributi possibili. Su questo sono definite due funzioni: si definisce funzione di calcolo del supporto di un set di attributi $\mathcal{R}$ la funzione che dato un insieme di attributi F' calcola il numero totale di righe che li contiene. Si noti che $\mathcal{R}$ è analogo alla funzione s, che dato un itemset ne calcola il supporto, presentata nella sezione 1.3. Analogamente si definisce

i	r_i
1	a,b,c,l,o,s
2	a,d,e,h,p,l,r
3	a,c,e,h,o,q,t
4	a,e,f,h,p,r
5	b,d,f,g,l,q,s,t

(a) Example Table

f_j	$\mathcal{R}(f_j)$
a	1,2,3,4
b	1,5
c	1,3
d	2,5
e	2,3,4
f	4,5
g	5
h	2,3,4
l	1,2,5
o	1,3
p	2,4
q	3,5
r	2,4
s	1,5
t	3,5

(b) Transposed Table, TT

Figura 4.1: Esempio di tabella originale (a) e tabella trasposta (b), Fonte: [30]

funzione di calcolo del supporto comune ad un insieme di righe $\mathcal{F}$ la funzione che dato un set di righe R' calcola il massimo insieme di item comune tra queste. Preso come riferimento il dataset in figura 4.1, dato $F' = (a, e, h)$, $\mathcal{R}(F') = 2, 3, 4$. Dato invece il set di righe $R' = (2, 3)$, il set di feature corrispondenti risulta uguale a $\mathcal{F}(R') = a, e, h$.

Una volta definite queste due funzioni di supporto, vine definita una tabella trasposta TT sulla base del dataset originale. In questa tabella le righe rappresentano le possibili feature e sono denominate tuple, mentre nelle colonne sono presenti gli id transazioni originali del dataset. Un esempio di questa tabella è disponibile nella figura 4.1.

A differenza dei classici algoritmi di itemset mining, che esplorano lo spazio di ricerca tramite un enumerazione degli attributi, Carpenter esegue la ricerca sulla base di un enumerazione delle righe. Questo approccio, nel caso di dataset ad alta dimensionalità, riduce lo spazio di ricerca. Preso come esempio il dataset in figura 4.1 (figura a), nella sezione 1.3.1 si era dimostrato come una ricerca con l'algoritmo Apriori potesse potenzialmente generare 32768 candidati da valutare. Una ricerca basata su righe implica considerare tutte le possibili combinazioni di righe. Avendo il dataset in questione 5 righe, il numero di itemset da valutare sarà nel caso peggiore di 32, di tre ordini di grandezza inferiore rispetto a quello ottenuto con Apriori.

La figura 4.2 mostra il row enumeration tree, contenente il set completo di tutte le possibili combinazioni tra righe. Questo albero è costruito enumerando tutte le possibili combinazioni di righe e disponendole in modo tale che ogni elemento di livello uno contenga nel suo sotto-albero tutte le possibili combinazioni tra se stesso e tutti gli elementi aventi ID maggiore del proprio.

Per trovare i pattern chiusi e frequenti, Carpenter esegue una ricerca depth first, ovvero in cui viene stabilito un ordine lessicografico tra le transazioni e partendo da quelle avente indice più basso si esplora completamente il ramo delle possibilità collegate alla singola transazione. Ogni elemento viene esplorato considerando solo gli elementi successivi, mai i precedenti. In questo modo ogni possibile combinazione di righe è considerata una e una

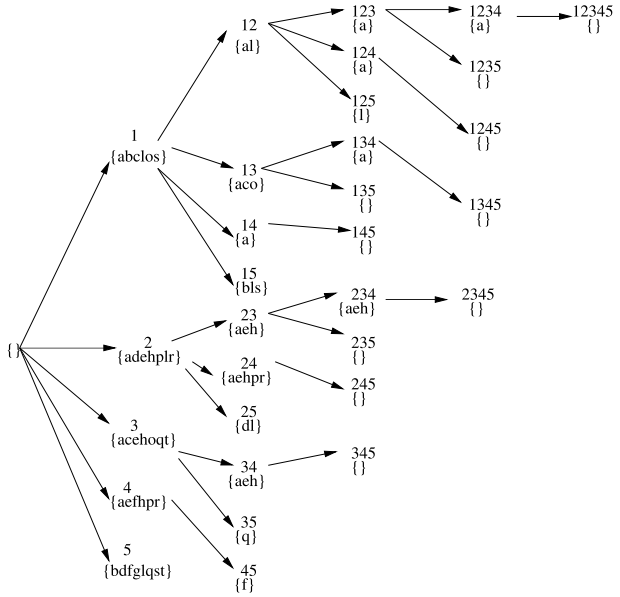


Figura 4.2: Esempio di Row Enumeration Tree, Fonte: [30]

sola volta. Ad ogni combinazione di righe è associato uno e un solo itemset chiuso, l'insieme di tutti gli itemset chiusi viene coperto considerando tutte le possibili combinazioni di righe.

Per eseguire la ricerca di itemset, poste queste condizioni, sarebbe sufficiente generare tutti le possibili combinazioni di righe, generare il loro set di attributi comuni applicando $\mathcal{F}$ e valutare la frequenza di questo itemset. Ciò però non sarebbe efficiente in quanto non verrebbe applicata la proprietà monotona del supporto. Occorre quindi definire una strategia di pruning per ridurre lo spazio di ricerca.

Sia X un sottoinsieme di righe. Data la tabella trasposta TT si definisce tabella trasposta condizionata a X, denominata $TT|_X$ un subset di righe di TT tale che:

1. Per ogni tupla x che contiene tutti gli elementi di X, esiste una tupla corrispondente in $TT|_X$.

2. Ogni tupla di $TT|_X$ contiene solo elementi tali che il loro identificatore di riga è maggiore di qualunque elemento presente in X.

La figura 4.3 è un esempio di tabella trasposta per le transazioni $(2, 3)$ del dataset in figura 4.1

Terminate le premesse, viene presentato il flusso dell'algoritmo di Carpenter (algoritmo 2).

f_j	$\mathcal{R}(f_j)$
a	4
e	4
h	4

Figura 4.3: Tabella trasposta $TT_{(2,3)}$ per l'insieme di transazioni $(2, 3)$, Fonte: [30]

Algoritmo 2 Algoritmo di Carpenter

Richiede: F: insieme degli attributi totali sul dataset, TT: tabella trasposta, $minsup$: soglia del supporto minimo
Restituisce: FCP: Pattern frequenti e chiusi
1: $FCP \leftarrow \emptyset$; ▷ Inizializza l'output
2: Dichiara R come l'insieme delle transazioni nel dataset originale ordinate per id;
3: $MinePattern(TT|_{\emptyset}, R, FCP)$;
4: **return** FCP

La logica di espansione e pruning di un itemset è contenuta all'interno del metodo $MinePattern$ (algoritmo 3). In questi passi infatti viene valutata la creazione di un superset di un insieme di righe e la sua eventuale frequenza. Durante il problema, sono definiti come I un itemset da valutare, X l'insieme delle righe che supportano I e R l'insieme delle righe da esplorare. Nel passo 1 dell'algoritmo viene calcolato il supporto di ogni riga $r_i \in R$ all'interno della tabella trasposta TT_X. A questo punto sono valutati in sequenza tre vincoli di pruning, il cui scopo è mantenere gli itemset chiusi non ancora esplorati:

1. **Valutazione del superset dell'attuale set di transazioni** (passo 2). In questo passo viene ricercato il superset di X aggiungendo elementi di R tali che questi compaiano in almeno una riga di TT_X. Considerando tutti gli $r_i, ..., r_n$ che soddisfano questa condizione, viene definito un superset di X $EX = X \cup r_i \cup ... \cup r_n$. A questo punto se un superset di X non raggiunge la soglia minima di supporto, allora nessun itemset associato a una qualunque combinazione di queste righe potrà essere frequente. Preso come esempio il dataset in figura 4.1 e posto $minsup = 3$, si consideri $X = \{3\}$ e $R = \{4, 5\}$. Da questo X viene generato un solo superset $EX = \{3, 4\}$, in quanto 5 non compare in nessuna delle transazioni di TT_3. Essendo $|EX| < 3$, il processo legato all'esplorazione di questo ramo viene interrotta. È intuibile dalla figura 4.2 che questo superset non potrà mai generare nessun itemset valido di 3 elementi.

   Formalmente si ricercano dentro R tutte le transazioni che compaiono in almeno una tupla di $TT|_X$. Questo insieme viene definito come U. Se la cardinalità di $X \cup U$

risulta minore di *minsup*, allora non potrà essere generato nessun itemset frequente nel superset, quindi l'esplorazione potrà essere interrotta. In caso contrario $R' = U$. Riferendosi all'esempio di prima $U = 4$, $|X| + |U| = 2 < 3$, quindi è verificato quanto intuito sopra.

2. **Eliminazione delle righe presenti in ogni tupla** (passo 3). Successivamente sono eliminate dall'insieme delle espansioni quelle righe che compaiono in ogni tupla della tabella trasposta $TT'|_x$. Questo potrebbe inizialmente sembrare un controsenso, eliminare una riga in cui I può essere espanso senza perdere elementi sembrerebbe uno sbaglio. Proprio per questo motivo in realtà queste righe vanno scartate. Scopo della creazione dei superset non è il conteggio del supporto di pattern già esistenti, quanto più l'individuare nuovi itemset non ancora valutati. Preso ad esempio $X = (2, 3), R = (4, 5)$, $\mathcal{F}(2, 3) = \mathcal{F}(2, 3, 4)$ quindi la valutazione di $(2, 3, 4)$ non è necessaria, in quanto produrrebbe un itemset già visitato.

   In termini formali, le righe presenti in ogni tupla di $TT'|_x$ sono raccolte nell'insieme Y, successivamente $R' = R' - Y$.

3. **Pruning degli itemset frequenti già esplorati** (passo 4). Ultimo passo di pruning, l'itemset che sta venendo considerato è già presente in FCP, allora si può interrompere la ricerca su quel ramo. Intuitivamente, la natura depth-first dell'algoritmo implica che se un itemset è già presente in FCP, questo sia già stato generato in un passaggio precedente dell'algoritmo. Questo passo di ricerca ha considerato transazioni con id inferiore a quello delle transazioni correnti, quindi sicuramente ha incluso anche quelle attualmente considerate nel calcolo del supporto. Ne segue che ogni possibile superset di queste righe è già stata valutata. Di conseguenza l'esplorazione del ramo termina.

   Si consideri $X = (3, 4) R = (5)$ nel dataset in figura 4.1, $\mathcal{F}(X) = (a, e, h)$. Nel momento in cui viene analizzato X, (a, e, h) risulta già presente in FCP, in quanto generato da $X' = (2, 3), R' = (4, 5)$. Fissato questo, è possibile aggiungere che il superset $EX = (3, 4, 5)$ risulta già valutata in $EX' = (2, 3, 4, 5)$. Questo perché se EX contenesse qualche itemset valido, questo sarebbe già individuato in EX', in quanto le righe in X' hanno id minore o uguale di quelle in X e individuano lo stesso pattern.

Una volta che un candidato supera i tre vincoli di pruning, può essere valutato come frequente e espanso. Nel passo 5 se l'unione tra X e Y risulta frequente, I viene aggiunto a FCP, questo passaggio garantisce la validità del secondo vincolo. Infine nel passo 6, per ogni riga in R' si esegue ricorsivamente $MinePattern$ aggiungendo a Xr_i e calcolando $TT'|_X|_{r_i}$

Algoritmo 3 $MinePattern(TT'|_X, R', FCP)$

1: Calcolo del supporto per ogni riga $r_i \in R'$ su TT'. Inizializzazione di $Y = \emptyset$;
2: Sia $U \subset R'$ un set di righe che compare almeno in una tupla di TT'_X. Se $|U| + |X| \leq minsup$ **return** altrimenti $R' = U$. ▷ Primo passo di pruning
3: Sia Y il set di righe tali che $\forall r_i \in Y, r_i \in t_j \forall t_j \in TT'|_X$, $R = R - Y$ ▷ Secondo passo di pruning
4: Se $\mathcal{F}(X) \in FCP$, **return** ▷ Terzo passo di pruning
5: **if** $|X| + |Y| \geq minsup$ **then** ▷ Se l'itemset è frequente rispetto alla soglia di supporto
6: $FCP \leftarrow \mathcal{F}(X) \cup FCP$ ▷ La tupla viene aggiunta a FCP
7: **for each** $r_i \in R'$ **do** ▷ Per ogni elemento esplorabile
8: $R' = R' - r_i$
9: $MinePattern(TT'|_X|_{r_i}, R', FCP)$ ▷ Passo ricorsivo dell'algoritmo

Al termine dell'esecuzione dell'algoritmo sono riportati all'utente tutti gli itemset così individuati.

4.1.2 Limiti per i dati di traiettoria

Il colossal itemset mining trova diverse applicazioni nel mining di itemset. Lo stesso algoritmo di Carpenter ha diversi utilizzi nella ricerca di dati legati al mondo della biologia o medicina. In [30] vengono presentati i risultati su dataset collegati ai carcinomi a polmoni, ovaie e leucemia. Questi risultati sono frutto del confronto tra Carpenter e altri due algoritmi di mining di itemset chiusi. In media Carpenter risulta molto più veloce e i suoi risultati sono concordi con quanto ottenuto dagli altri due algoritmi.

Parlando poi di traiettorie, è noto che queste non sono semplici sequenze, come affermato nella sezione 1.3.2. I dati di traiettoria hanno una natura spazio-temporali che può essere utilizzata per definire tipi specifici di pattern (sezione 3.1) e un pruning più efficiente. Questo implica che i dati di traiettoria abbiano tra loro strette relazioni spazio-temporali difficili da modellare, come ad esempio vincoli di continuità temporale o contiguità spaziale. Gli algoritmi di colossal itemset mining presenti in letteratura [46], [47] non sono pensati per sfruttare questi vincoli. Sarebbe teoricamente possibile applicare questi vincoli a posteriori, ma sarebbe molto poco efficiente. Questi infatti sono in grado di ridurre consistentemente lo spazio di ricerca dei possibili itemset.

Un altro problema comune negli algoritmi di colossal itemset mining è la loro natura centralizzata. Ad esempio Carpenter è realizzato come algoritmo centralizzato: per evitare infatti di generare itemset già valutati in iterazione precedenti FCP viene mantenuto in un registro globale. Tradurre questo problema in ambito distribuito pone numerose sfide. Alcuni algoritmi in letteratura [46], [48] realizzano un'implementazione distribuita.

Per concludere, il colossal itemset mining è un approccio interessante per analizzare dati di traiettoria, tuttavia rispetto a quanto presente in letteratura si rende necessario integrare:

- Dataset avente transazioni di ordini di grandezza inferiori al numero delle feature.
- Pruning basato su criteri spazio-temporali oltre che sulla frequenza.
- Gestione della continuità nel tempo e nello spazio.
- Implementazione distribuita.

4.2 L'algoritmo CUTE

CUTE, o ClUstering TrajectoriEs, è un algoritmo di clustering sovrapposto il cui obbiettivo è l'analisi di gruppi di oggetti in movimento. Questa ricerca viene condotta considerando un insieme custom, nel numero e nella natura, di dimensioni. Ad esempio è possibile considerare solamente la dimensioni collegate allo spazio-tempo delle traiettorie oppure solo dimensioni semantiche (come ad esempio, le municipalità di una città), o ancora possono essere selezionati alcuni elementi da un gruppo e altri dall'altro. CUTE può essere utilizzato per l'estrazione di pattern di co-movimento, considerando solamente la dimensione spaziale e quella temporale.

Lo scenario reale per la realizzazione di questo algoritmo è stata l'analisi condotta su un insieme di traiettorie generate a Milano. All'interno di questo studio, i pattern di movimento sono stati analizzati a diversi livelli. Una prima analisi è stata condotta dividendo la superficie della città in piccole celle:questa ricerca ha rivelato i pattern di movimento del traffico, individuando quali potessero essere le strade maggiormente frequentate. Successivamente è stato realizzato uno studio basato sul vicinato: questo ha individuato i flussi di spostamento per specifiche categorie di utenti. Infine una ricerca basata sulle municipalità ha mostrato quali fossero le aree della città più visitate da differenti gruppi di individui.

Questi casistiche hanno mostrato come al variare delle dimensioni considerate e della finezza della ricerca i risultati abbiano una natura totalmente diversa. Lo scopo di CUTE è dunque di estrarre i pattern di movimento che avvengono con una certa soglia di frequenza. CUTE si pone inoltre come obbiettivo l'integrazione di dimensioni custom e di scale personalizzabili così da riuscire a condurre più tipi di ricerche sfruttando lo stesso framework.

CUTE è etichettabile come algoritmo di *colossal trajectory mining*. L'idea alla base del colossal trajectory mining, o mining di traiettorie su larga scala, interseca entrambi gli ambiti del clustering di traiettorie (sezione 1.2.1) e del colossal itemset mining. Questa intersezione permette di sfruttare le potenzialità di entrambi gli approcci. Scopo del mining di traiettorie colossali infatti è di individuare gruppi o cluster utilizzando i principi del colossal itemset mining.

In letteratura sono presenti esempi di tecniche a metà tra il mining di itemset frequenti e il clustering. In particolare la ricerca di pattern di co-movimento presenta approcci

ibridi di questo genere. SPARE (sezione 3.2) e *GeT Move* [40] implementano la ricerca di gruppi mischiando l'approccio basato su frequent itemset mining e il clustering. Entrambi questi framework discretizzano il tempo in intervalli (*bucket*) di dimensione finita. Su questi poi applicano un clustering basato sulla densità o distanza. Infine ricercano pattern di movimento mappando i vari oggetti come item e fondendoli in itemset sulla base del principio apriori.

Rispetto a quanto presentato fin d'ora, CUTE si pone come framework generico con le seguenti caratteristiche:

1. **Ricerca di itemset su dataset ad alta dimensionalità**. Rispetto allo stato dell'arte, CUTE integra i principi e l'efficienza del mining di pattern colossali a traiettorie. Queste tecniche sono impiegate per individuare gruppi di oggetti definibili come pattern di co-movimento.

2. **Sistema di riferimento generico**. A differenza degli altri algoritmi, CUTE permette di utilizzare un insieme di dimensioni a scelta dell'utente per la ricerca di gruppi. Ad esempio è possibile impostare una ricerca esclusivamente basata sulle municipalità o la composizione del vicinato, senza includere le dimensioni spazio-temporali. Oppure si può espandere la dimensione spazio temporale aggiungendo dati come l'altitudine oppure misure come velocità o accelerazione. Inoltre sono supportate dimensioni non esclusivamente monotone, come ad esempio i giorni della settimana.

3. **Continuità rispetto al sistema di riferimento**. Nella ricerca di pattern di movimento è possibile specificare vincoli di contiguità su tutte le dimensioni impiegate.

4. **Efficienza nel pruning**. La strategia di pruning impiegata permette di ridurre lo spazio di ricerca dell'algoritmo sulla base delle dimensioni utilizzate nel sistema di riferimento.

5. **Implementazione distribuita**. CUTE mette a disposizione una soluzione distribuita per il problema del colossal trajectory mining compatibile con dataset costruiti su problemi del mondo reale.

L'algoritmo segue tre passi per la formazione dei cluster, come anche rappresentato in figura 4.4

1. **Mapping delle traiettorie**.

   In questo primo passo vengono analizzate le traiettorie presenti nel dataset. Da queste viene determinata la regione di movimento rispetto alle dimensioni impiegate. Successivamente quest'area viene divisa in un insieme di celle omogenee per dimensioni. A questo punto ad ogni traiettoria viene assegnato un insieme di celle secondo

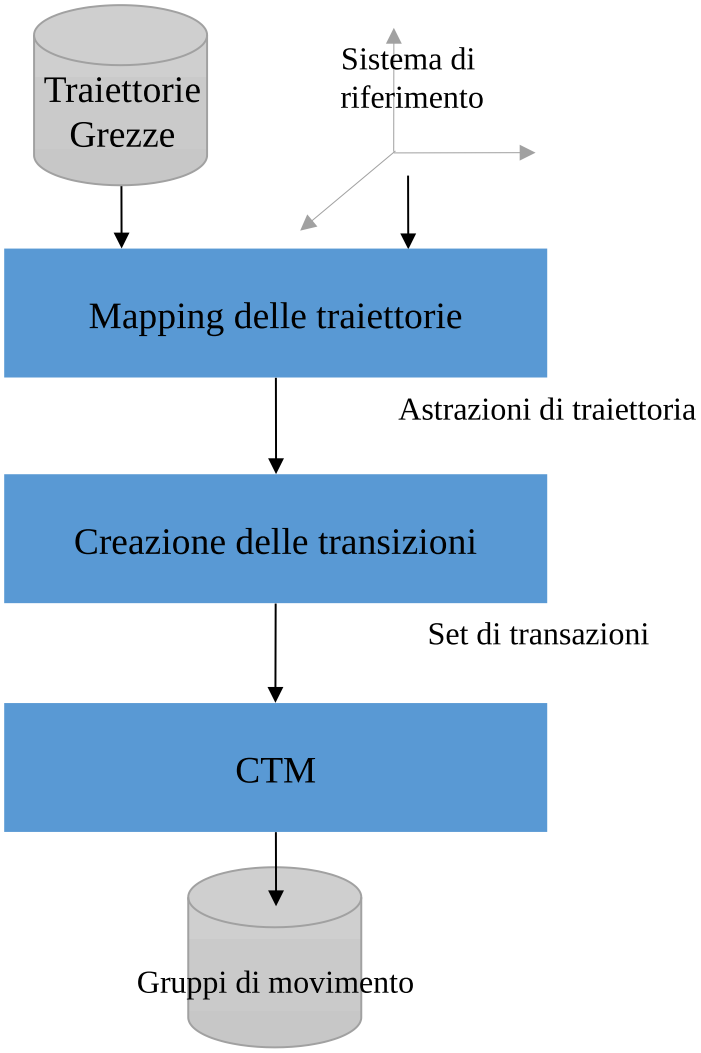


Figura 4.4: Rappresentazione grafica delle tre fasi dell'algoritmo CUTE

il seguente principio: una cella è attribuita ad una traiettoria quando quest'ultima ha almeno un punto che ricade entro i confini della cella.

2. **Creazione delle transazioni**.

   Durante questa fase vengono poste le basi per il mining di traiettorie: scopo di questa parte dell'algoritmo è infatti andare a generare l'insieme delle transazioni su cui verrà eseguita la ricerca di pattern. Ciò avviene considerando l'output della fase precedente e ribaltando la relazione traiettoria-cella.

3. **Colossal Trajectory Mining**.

   Ultima e più importante passaggio dell'algoritmo, produce in uscita i pattern di movimento. Dato l'output della fase precedente, ovvero un insieme di transazioni appositamente creato, esegue una ricerca di itemset su dati ad alta dimensionalità.

4.2.1 Input e parametri

Prima di concentrarsi sui singoli passi di CUTE, occorre definire la formulazione dell'algoritmo. Innanzitutto si pone necessario riprendere i concetti di itemset, supporto e transazione descritti nella sezione 1.3 e di traiettoria grezza e astrazione di traiettoria presentati nella sezione 1.1.1.

Rispetto a questi concetti, la formulazione di CUTE è descritta dai seguenti parametri:

1. **Sistema di riferimento n-dimensionale**. Dato lo spazio di ricerca definito da n dimensioni, questo viene diviso in ipercubi sulla base del sistema di riferimento impiegato. Ogni ipercubo così individuato è definito quanto. Le traiettorie saranno inizialmente espresse come insiemi di quanti, successivamente ogni quanto diverrà una transazione durante la fase di itemset mining.

2. **Funzione di distanza tra quanti**. Dati due quanti q_1, q_2 espressi su k dimensioni, $d_q(q_1, q_2)$ determina la distanza tra i due considerando tutte le possibili dimensioni. Il risultato di questa operazione è un vettore k dimensionale.

3. **Soglia di distanza**. Dati due quanti q_1, q_2 espressi su k dimensioni e una funzione di distanza d_q si definisce δ come un vettore di k valori ciascuno relativo a una dimensione. q_1, q_2 si dicono vicini se per ciascuna dimensione la loro distanza è minore della soglia presente in δ.

4. **Dimensione minima**. Questo parametro γ riguarda il numero minimo di item all'interno di ogni itemset restituito in output alla fine dell'algoritmo.

5. **Supporto minimo**. La frequenza minima α oltre cui ogni itemset deve apparire all'interno dell'insieme delle transazioni.

Questa definizione è generica e include qualunque possibile dimensione.

Rispetto al lavoro svolto in questo documento di tesi, è opportuno fornire la formulazione di CUTE per il riconoscimento di pattern di movimento sulla base delle dimensioni spazio-temporali. I quanti definiti sono quindi cubi a tre dimensioni: latitudine, longitudine e tempo. Inoltre, per facilitare il confronto successivo con SPARE, vengono formulate separatamente la dimensione spaziale e quella temporale. Tuttavia questa è solo una procedura di forma, l'algoritmo non altera la sua originale formulazione né le sue logiche di funzionamento. Di seguito vengono presentati i parametri di CUTE adattato alla ricerca di pattern di movimento:

1. **Sistema di riferimento tridimensionale**. Nella ricerca di pattern di co-movimento sono impiegate tre dimensioni, due spaziali e una temporale: la latitudine la longitudine ed il tempo.
   - **Lato spaziale del quanto** s. Definisce la dimensione di latitudine e longitudine coperta da ogni quanto.
   - **Lato temporale del quanto** t. Definisce la durata temporale di ogni quanto.
2. **Funzione di distanza tra quanti**. Vengono impiegate due differenti funzioni di distanza:
   - **Funzione di distanza spaziale tra quanti** d_s. Dati due quanti q_1, q_2, $d_s(q_1, q_2)$ determina la distanza spaziale tra i due.
   - **Funzione di distanza temporali tra quanti** d_t. Dati due quanti q_1, q_2, $d_t(q_1, q_2)$ determina la distanza temporale tra due quanti.
3. **Soglia di distanza**. Anche qui le soglie temporali vengono distinte:
   - **Soglia di distanza spaziale** ϵ. Dati due punti p_1, p_2 e una funzione di distanza spaziale d_s, si definisce ϵ come la soglia sotto la quale due punti si considerano vicini.
   - **Soglia di distanza temporale** τ. Analogamente a quanto detto sopra, il parametro τ permette di specificare una soglia massima per la distanza tra due punti nel tempo.

Con questa formulazione, risulta possibile per CUTE affrontare il problema della ricerca di co-movement pattern, descritto nella sezione 3.1. Scendendo nel dettaglio, γ esprime la dimensione minima di un gruppo considerato interessante, mentre α il numero minimo di istanti spazio-temporali in cui tutti i membri del gruppo sono considerati vicini. ϵ rappresenta il criterio di vicinanza spaziale tra due punti, τ invece permette la ricerca dei diversi pattern di movimento: con $\tau = 1$ si ha un vincolo stringente sulla continuità temporale, andando quindi a ricercare pattern flock, dall'altra parte con un valore uguale

a ∞, si rilassa al massimo la continuità temporale, ottenendo così dei risultati classificabili come swarm. Infine con qualunque valore intermedio tra 1 e ∞, CUTE esegue la ricerca di pattern platoon degeneri, ovvero con vincolo l posto uguale a 1 e $\tau = g$.

4.2.2 Mapping del dataset

La prima fase dell'algoritmo si occupa dato un dataset avente traiettorie n-dimensionali di generare un set di quanti distinti e di esprimere le traiettorie come combinazioni di questi. Lo scopo di questa fase è quello di esprimere le traiettorie in un sistema di riferimento custom univoco per tutto il dataset. Tramite questa trasformazione dovrà essere possibile determinare la vicinanza di due traiettorie in maniera univoca. Inoltre i quanti di cui dovrà essere composto questo sistema dovranno essere di molto inferiori rispetto al numero di traiettorie. Questa compressione dell'informazione sarà fondamentale per le fasi successive, in particolare per quella di ricerca degli itemset.

Punto focale di questa prima fase è il sistema di riferimento fornito in input. Questo specifica quante e quali dimensioni considerare all'interno del problema. All'interno del sistema è presente l'insieme dei valori che la singola dimensione può assumere. La definizione dei quanti avviene quindi sulla base dei singoli valori possibili di ogni dimensione. Date n dimensioni utilizzate, lo spazio di ricerca composto da tutte le traiettorie proiettate sulle n dimensioni è suddiviso in un insieme di quanti. Un quanto per definizione è rappresentabile come un ipercubo n dimensionale avente un id univoco all'interno dell'insieme. Ogni quanto è caratterizzato da uno specifico valore per ciascuna delle n dimensioni e intuitivamente copre una parte dello spazio totale di ricerca. Una traiettoria può essere espressa utilizzando il set di quanti sopra definito. Ogni punto di una traiettoria infatti individua uno specifico quanto. Le traiettorie passano così dall'essere insiemi di punti a insiemi di quanti univoci per tutto il set di traiettorie. Intuitivamente due punti risulteranno vicini quando apparterranno allo stesso quanto. Una volta terminata la conversione, la prima fase dell'algoritmo può dirsi conclusa.

Trattando del caso specifico della ricerca dei pattern di co-movimento, quanto detto sopra viene declinato come segue: Si utilizza un sistema di riferimento tridimensionale, basato su latitudine, longitudine e tempo. I quanti sono definiti mediante un mapping su griglia, che produce cubi omogenei. Questi cubi sono chiamati celle.

Ogni cella avrà una componente spaziale e una temporale. L'area spaziale coperta da una cella è determinata dal parametro s ed è espressa in metri rispetto alle coordinate polari di ogni punto.

Per quanto riguarda la dimensione temporale t invece sono prese in considerazione nello specifico due scale cicliche: ore del giorno, giorni della settimana. Rispetto a una metrica di tempo monotona basata su data e ora di ogni singolo punto, come ad esempio in SPARE, una scala circolare ha due vantaggi: il primo riguarda il supporto a pattern ciclici che verrebbero altrimenti ignorati e in secondo luogo il ridotto range di valori della scala (da 0

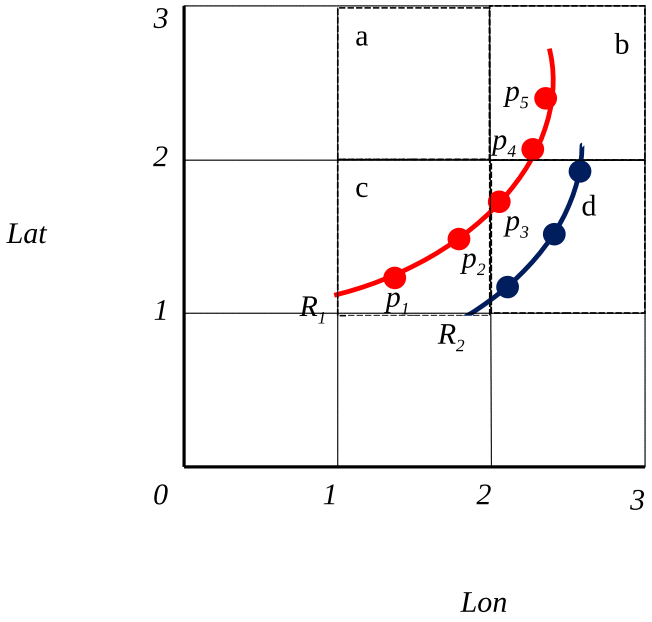


Figura 4.5: Mapping di due traiettorie, R_1, R_2in un sistema di riferimento basato su celle. Sugli assi x e y sono espressi latitudine e longitudine, il tempo invece è espresso in pedice ai punti

a 23 in caso di scala giornaliera, da 0 a 6 in quella settimanale) previene l'esplosione nel numero di celle al crescere del dataset.

Una volta determinata il lato spaziale delle celle e la loro durata temporale, viene generato l'insieme delle celle C. Ottenuto questo, è necessario per concludere questa prima fase esprimere ogni traiettoria nella nuova formulazione. Così facendo una traiettoria non sarà più definita come la composizione di diversi punti isolati nel tempo, ma come un insieme di celle. La figura 4.5 mostra un esempio di conversione in un sistema di riferimento basato su celle:

Prendendo in considerazione le traiettorie R_1, R_2, è possibile vedere come i punti p_1, p_2 appartengano alla cella c, p_3 a d mentre p_4, p_5 a b; analogamente ogni punto della traiettoria R_2 sia attribuibile a d. Il set di celle del problema sarà quindi composto dalle seguenti celle: $\langle b, c, d \rangle$ mentre le due traiettorie saranno espresse in funzione del nuovo sistema di riferimento come segue:

- R_1. $\langle b, c, d \rangle$
- R_2. $\langle d \rangle$

Va sottolineato infine che, sebbene appaia nell'immagine, la cella a non viene effettivamente generata, poiché nessuna traiettoria ha un punto entro i suoi confini.

Una volta che le traiettorie sono state espresse nelle nuove coordinate, questo primo passo dell'algoritmo giunge a termine.

La scelta delle dimensioni di ogni cella influenza molto i passaggi successivi. Celle piccole produrranno traiettorie lunghe, estendendo il possibile spazio di ricerca di ogni cluster.

Tanto più ampio lo spazio di ricerca, tanto più a lungo dura la sua esplorazione, tuttavia è possibile eseguire una ricerca più fine su di esso. Lati spaziali più ampli e dimensioni temporali ridotte generano celle più grandi, con uno spazio di ricerca più ridotto e risultati più grezzi.

4.2.3 Creazione delle transazioni

Nella prima fase viene prodotto un dataset nella forma $< tr_{id}, (q_1, ..., q_n) >$, dove tr_{id} rappresenta l'id di una certa traiettoria, mentre $q_1, ..., q_n$ l'insieme di quanti in cui tale percorso è stato convertito. Scopo di questo secondo passo di CUTE è quindi modificare il dataset ottenuto dalla prima fase in modo da renderlo valido per la ricerca di pattern di movimento. Intuitivamente il dataset in output dalla prima fase non è ancora adatto ad essere sottoposto a mining, poiché in questa situazione l'insieme delle transazioni è composto dalle traiettorie, mentre quello delle feature dai quanti. Un'operazione di ricerca di itemset frequenti va ad individuare tra le feature quelle che compaiono assieme con almeno una certa frequenza, quindi in questo caso il risultato dell'algoritmo sarebbe vari raggruppamenti di quanti. Questi insiemi possono essere sicuramente utili ad individuare percorsi frequenti e trafficati, ma non sono in grado di descrivere gruppi di movimenti comuni ad un certo insieme di oggetti. Come detto nella sezione 1.3.2, per ottenere la ricerca di gruppi di oggetti bisognerebbe disporre di un dataset avente come transazioni i quanti e come feature le traiettorie.

Per ottenere un dataset adatto alla ricerca di gruppi, occorre trasporre il ruolo di quanti e traiettorie: prendendo come ipotesi di trasporre la relazione tra traiettoria e insieme di quanti che la compongono, si ottiene la versione trasposta del dataset originale. La versione trasposta del database sarà quindi composta dai singoli quanti del problema che ora saranno identificati come transazioni, mentre il ruolo delle traiettorie sarà di feature. Così facendo ad ogni quanto sarà associato l'insieme delle traiettorie in cui compare. In questa situazione la ricerca di itemset frequenti produce raggruppamenti di traiettorie che hanno una frequenza maggiore a una certa soglia di supporto. Questa frequenza può essere interpretata come il numero minimo di quanti condiviso tra tutti i componenti di un gruppo.

Trasponendo il dataset è quindi possibile eseguire una ricerca di gruppi di movimento, ma questa operazione ha anche un'altra importante proprietà. Analizzando il rapporto tra il numero di quanti generati nella prima fase e di traiettorie in tutto il dataset, è corretto affermare che la quantità di questi superi di gran lunga l'altra. Di conseguenza il dataset che considera come transazioni le celle e come feature le traiettorie sarà etichettabile come dataset ad alta dimensionalità, adatto quindi a essere processato con le tecniche proprie del colossal itemset mining.

Queste operazioni sono eseguite alla lettera nella ricerca di pattern di movimento: ogni cella individuata diviene così una transazione del dataset utilizzato nella fase successiva.

4.2.4 Ricerca di gruppi di movimento

Ultimo e più importante passaggio dell'algoritmo: dato il dataset prodotto in output dalla fase precedente, viene eseguita una ricerca di gruppi di movimento utilizzando i principi del colossal trajectory mining. L'intuizione dietro questa fase si compone dei seguenti passi. Per iniziare si considera che ogni quanto corrisponde a un set di traiettorie e quindi a un gruppo. Sono scartati i gruppi che non rispettano i vincoli di dimensione. Successivamente per ogni gruppo si ricerca il "percorso più lungo' che rispetti i vincoli di continuità su tutte le dimensioni. Se questo è valido è aggiunto all'output, altrimenti viene scartato.

Scendendo più nel dettaglio, l'idea alla base di quest'ultima fase si ispira a quanto visto nell'algoritmo di Carpenter (sezione 4.1.1): vengono ripresi e adattati i concetti di row enumeration tree e depth first search, mantenendo la proprietà di chiusura negli itemset individuati. Il flusso di CUTE è fortemente ispirato a quello di Carpenter. Molti passi sono comuni tra i due algoritmi. CUTE tuttavia si pone come evoluzione distribuita di Carpenter pensata per i dati di traiettoria. Di conseguenza sono aggiunte e/o modificate alcune strutture.

Il primo cambiamento consiste nella definizione del vicinato di ogni quanto. Per ciascun quanto q, si definisce il suo vicinato come l'insieme dei quanti per cui vale che $d_q(q, q_i) \leq \delta$ per tutti i valori nel vettore δ. Formulando il problema dal punto di vista dei pattern di co-movimento, data una cella c, si definisce vicinato di N_c l'insieme di tutte le celle $c_1, \ldots, c_n$ tali che $d_s(c, c_i) \leq \epsilon \wedge d_t(c, c_i) \leq \tau \wedge c.id < c_i.id$. Il vicinato risulta fondamentale nel momento in cui si conduce una ricerca specificando una continuità nelle dimensioni di riferimento. In particolare per la ricerca di pattern di co-movimento la continuità è espressa tramite i vincoli ϵ e τ. Il vincolo sull'id della cella serve ad evitare ridondanze nel vicinato, ogni vicinato infatti contiene solo una parte delle celle effettivamente vicine a quella considerata. Ciò però non costituisce un problema, poiché la cella che si sta considerando comparirà nel vicinato di quelle aventi id minore, rendendo così nulla la perdita di informazione. In questo modo vengono preservate le proprietà della ricerca depth first. Grazie alle limitazioni effettuate sui parametri spazio-temporali e sull'id della cella è possibile ridurre lo spazio di ricerca del problema. L'introduzione del concetto di vicinato consente a CUTE di superare i limiti del colossal itemset mining riguardo alla prossimità spazio-temporale delle traiettorie (sezione 4.1.2).

Una volta determinato il vicinato per ogni cella, vengono individuati i possibili itemset da ricercare. Analogamente a quanto accade nell'algoritmo di Carpenter, gli itemset iniziali sono individuati sulla base della singola transazione. Nel caso di CUTE per ogni cella sarà quindi definito un pattern come l'insieme di tutte le traiettorie che sono passate in quella cella. Già nel momento della generazione di questi gruppi viene effettuata una prima operazione di pruning: i pattern la cui lunghezza non sia maggiore o uguale a γ sono scartati in quanto non interessanti. Gli elementi che superano questa prima fase vengono

poi trasformati in tuple analogamente a come accade nell'algoritmo di Carpenter: per ciascuno di essi è definito il set di transazioni X che al momento supportano l'itemset e l'insieme delle transazioni da esplorare in futuro R. Vale la pena sottolineare che per considerare ogni possibile set di transazioni una volta sola, R conterrà solo celle aventi id superiore a quelle in X, analogamente a quanto accade per il vicinato.

Successivamente occorre definire una metrica per valutare la possibilità di una tupla di essere ulteriormente espandibile in termini di supporto. Questa metrica definisce ad ogni iterazione quali itemset processare ulteriormente e quali invece possono essere valutati per essere aggiunti all'output del processo. L'algoritmo termina la sua computazione quando non è rimasto più nessun itemset da espandere e tutti i risultati sono stati collezionati. Lo pseudocodice presente nell' algoritmo 4 descrive il funzionamento dell'algoritmo CUTE.

Algoritmo 4 CUTE

Richiede: $\mathcal{T}$: insieme delle transazioni, α: supporto minimo, γ: cardinalità minima, ϵ: soglia di vicinanza spaziale, τ: range temporale ,p: numero delle partizioni
Restituisce: $\mathcal{C}$: gruppi chiusi rilevati rispetto a α, γ.

1: Broadcast $\mathcal{T}$ ▷ Rende $\mathcal{T}$ disponibile su tutti gli executors
2: Compute N as $\{(I_q\{q\}, \{q', q' \in S, q < q', d_{\mathrm{s}}(q, q') <= \epsilon, d_{\mathrm{t}}(q, q') <= \tau\}), I_q \in \mathcal{T}\}$
3: Broadcast N ▷ Rende N disponibile su tutti gli executors
4: $C \leftarrow \{(I_q, true, \{q\}, \{q', q' \in S, q < q', false), I_q \in \mathcal{T}\}$
▷ Tuple nella forma (itemset considerato, flag di estensione, supporto, transazioni da esplorare, flag di salvataggio)
▷ Esplora tutte le transazioni aventi id > max(supporto).id
5: **while** Itemsets with $flag = true$ exist **do**
6: Esegue uno shuffle su C per creare p partizioni bilanciate.
7: $C \leftarrow C \cup \bigcup_{(I,f,X,R,tf) \in C} Extend(I, f, X, R, tf, \alpha, \beta, \gamma)$
▷ Ricerca tutte le possibili espansioni di un itemset
8: **return** C

Punto focale dell'algoritmo è il metodo *Extend* il cui pseudocodice è fornito nell' algoritmo 5 Qui infatti viene eseguita la creazione, ricerca e filtraggio degli itemset a partire dai gruppi individuati nel primo passo di questa fase. L'idea alla base di questo passo dell'algoritmo è generare i gruppi di movimento per costruzione. Questa operazione richiede di "inseguire' il pattern lungo il vicinato. In poche parole, partendo da un gruppo potenzialmente valido, ad ogni passo il supporto viene "espanso' con l'aggiunta del vicinato e i quanti che contengono il pattern sono aggiunti al supporto.

Nello specifico, come è successo per la generazione degli itemset iniziali, anche qui l'idea alla base rimane la stessa dell'algoritmo di Carpenter. Per ogni tupla sono individuate le celle di R che supportano lo stesso itemset I e che appartengono al vicinato di una delle celle di X, questo insieme sarà definito Y. A differenza di quanto accade in Carpenter, Y è filtrato sulla base del vicinato delle celle in X. A questo punto Y viene sommato a X, creando il nuovo set di transazioni $X \cup Y$ e rimosso da R, generando un nuovo

spazio di ricerca R'. Successivamente si deve considerare il vicinato di tutte le celle in Y, potrebbe infatti capitare che qualcuna delle celle in questione supporti I, in tal caso tale cella sarà aggiunta al passo successivo dell'algoritmo. Se ciò non accade, vuol dire che l'insieme X non può essere ulteriormente espanso in celle con id maggiore senza violare i vincoli sullo spazio-tempo. Verificato se la tupla può essere espansa o meno, R' viene intersecato con il vicinato di $X \cup Y$, in modo da considerare solamente espansioni valide rispetto ai vincoli spazio-temporali. Una volta determinato $ExpandedR$ viene effettuata la fase di generazione di nuovi itemset vera e propria: per ciascuna cella q in $ExpandedR$ avente almeno un item in comune con I viene generata una tupla nella seguente forma: $(I \cap q.I, X \cup q, ExpandedR \backslash q)$. Questa tupla sarà poi processata nell'iterazione successiva dell'algoritmo. Infine per tutte le tuple processate durante l'attuale passo dell'algoritmo sarà verificato l'effettivo interesse mediante un apposito processo di filtraggio, che manterrà gli itemset validi e andrà a scartare quelli che non soddisfano le condizioni di chiusura, supporto, dimensione o coesione.

L'algoritmo giunge al termine quando non sono più presenti tuple da valutare come da espandere. A questo punto l'insieme degli itemset frequenti può essere restituito come output della funzione oppure salvato su tabella remota.

Fino ad ora è stato dato per assodato che ϵ e τ fossero definiti, potrebbe però capitare di rilassare contemporaneamente entrambi i vincoli. Nella ricerca di swarm, la struttura dell'algoritmo non subisce radicali trasformazioni, l'unica differenza sta nel fatto che il vicinato di una cella corrisponde esattamente a R. Alla luce di ciò i passaggi e vincoli basati sul vicinato e la sua composizione vengono ignorati, in quanto sarebbero ridondanti o superflui.

4.2.5 Dettagli implementativi

In questa sezione si tratterà di come i passaggi precedenti dell'algoritmo sono implementati e delle problematiche connesse.

Il principale problema da affrontare durante l'implementazione è stata la necessità di progettare strutture adatta a una computazione su Spark. Rispetto a quanto visto in Carpenter occorre dunque mettere in campo strategie particolari per adattare l'algoritmo a una piattaforma distribuita.

La fase di Mapping di traiettorie (sezione 4.2.2) viene eseguita per lo più utilizzando query SQL per determinare le varie tabelle. In primo luogo si determina la tabella delle celle, successivamente viene effettuato un join tra quest'ultima e la tabella delle traiettorie originali. Questa operazione ha il fine di assegnare a ogni traiettoria l'insieme delle sue celle. Ultima tabella di supporto generata è quella del vicinato. Il vicinato viene generato eseguendo un self join all'interno della tabella delle celle, calcolando per ogni coppia di celle la loro distanza spaziale, temporale e tra le varie dimensioni. Tutte queste operazioni sono eseguite con Spark-SQL. Terminata la costruzione delle tabelle accessorie, vengono

Algoritmo 5 Extend

Richiede: I: itemset ricercato, f: flag per l'estensione, X: insieme delle transazioni che supportano I, R: transazioni da esplorare, α: supporto minimo, γ: cardinalità minima, N: vicinato, ReturnResult:flag per salvare una tupla interessante

Restituisce: L: set di itemset frequenti e chiusi.

1: $L \leftarrow \emptyset$ ▷ Accumulatore di itemset
2: **if** $!I.toStore$ **then** ▷ Se la tupla corrente è valutata come da non salvare
3: **if** f **then** ▷ Se l'itemset deve essere ulteriormente espanso
4: $Y \leftarrow sup(I) \cap R$ ▷ Insieme delle transazioni in R che supportano I
5: **if** $N.isDefined$ **then** ▷ Se sono definiti dei vincoli spazio-temporali
6: $Y \leftarrow Y \cap allNeighbor(X)$ ▷ Vengono salvati in Y solo i valori spazio-temporali idonei
7: $X' \leftarrow X \cup Y$ ▷ ... e vengono aggiunti al supporto
8: $R' \leftarrow R\ Y$ ▷ ... e tolti da R
9: **if** $N.isDefined$ **then**
10: ExpandedR = $(R' \cap allNeighbors(XplusY))\ XplusY$ ▷ Se in tutte le celle da esplorare
11: **if** $((ExpandedR \cap sup(I)) = \emptyset)$ **then** ▷ Nessuno contiene I per intero
12: $L \leftarrow L \cup \{(I, false, XplusY, R', false)\}$ ▷ La tupla corrente si trova sul percorso più lungo
13: **for each** $I_q \in \mathcal{T} \wedge q \in ExpandedR$ **do** ▷ Processa tutte le possibili espansioni dell'itemset.
14: $I' \leftarrow I \cap I_q$ ▷ Genera il nuovo itemset intersecando con I_q
15: $X'' \leftarrow X' \cup q$ ▷ Supporto del nuovo itemset
16: $R' \leftarrow R'\ q$ ▷ Aggiorna lo spazio di ricerca rimanente
17: $L \leftarrow L \cup \{(I'', true, X'', R'', false)\}$
18: **for each** $l \in L$ **do**
19: **if** $(!filter(l) \vee l.X == \emptyset)$ **then** ▷ Se la tupla non soddisfa i criteri di bontà
20: $L \setminus l$ ▷ La tupla è rimossa da R
21: **else** ▷ La tupla è già stata processata e non è da salvare...
22: $L \leftarrow L \cup \{(I, false, \{\}, \{\}, false)\}$ ▷ ...viene sostituita da una tupla nulla
23: **else** ▷ La tupla è da salvare ed è già stata processata...
24: $L \leftarrow L \cup I$
25: **if** ReturnResult **then** ▷ Se i risultati sono restituiti in output dall'algoritmo...
26: **for each** $l \in L$ **do**
27: **if** $!l.f \wedge l.X.size >= \alpha$ **then** ▷ Se la tupla considerata è idonea
28: $L \setminus l$ ▷ La tupla viene rimossa
29: $L \cup (l.cluster, l.f, l.X, l, R, true)$ ▷ e riaggiunta con il flag *toStore* settato a true
30: **else**
31: **for each** $l \in L$ **do**
32: **if** $!l.f \wedge l.X.size >= \alpha$ **then**
33: store(l) ▷ In caso contrario la tupla viene salvata su tabella remota

create la tabella delle transazioni e quella trasposta. Queste tabelle sono salvate come RDD e trasmesse in broadcast ai vari executors, in modo da avere sempre disponibile l'accesso a queste tabelle nelle varie fasi della computazione distribuita.

Nella fase di astrazione delle traiettorie, (sezione 4.2.3) l'RDD contenente le coppie cella-traiettoria viene trasformato in modo da apparire nella forma cella - lista di traiettorie. Questa operazione viene gestita con le primitive di Spark e non comporta accorgimenti particolari.

La fase di mining di gruppi (sezione 4.2.4) è infine la più complicata e delicata. L'RDD cella-traiettorie della fase precedente viene trasformato nel primo livello di ricerca dell'algoritmo di Carpenter. L'RDD così trasformato si compone di tuple nella forma: $I, extend, X, R$. Per esprimere i set I, X, R e in generale tutti i set contenenti transazioni o attribuiti si utilizza la classe `RoaringBitMap` ([49]). Una RoaringBitMap è un set di n bit che riesce a esprimere un set di n numeri interi. Questa implementazione risulta efficace e efficiente, poiché permette di rappresentare i set dell'algoritmo in maniera efficiente e efficace nelle operazioni.

Trattando infine dei risultati, l'algoritmo si comporta in maniera diversa a seconda della configurazione specificata dall'utente. I gruppi possono essere restituiti a video e salvati su tabella nel caso il loro numero diventi troppo grande per essere processati in maniera efficace. In questo caso ad ogni passo dell'algoritmo i risultati sono processati, elaborati e poi scartati. Nel caso invece si necessiti di avere i gruppi in una struttura dati per eseguire ulteriori calcoli, allora il processo cambia. Tramite il flag *toStore* viene valutato se una tupla è già valida per essere restituita in output oppure no. A differenza di quanto accade nel caso precedente, i gruppi validi non vengono elaborati ad ogni ciclo, ma vengono collezionati alla fine dell'esecuzione. Per eseguire quest'operazione senza intaccare la struttura dell'algoritmo, le tuple valide vengono etichettate con il flag *toStore* come vero. In questo modo possono rimanere nell'RDD senza essere processate in ulteriori operazioni, se non un ultima fase di filtraggio che elimina tutti i gruppi non validi.

4.2.6 Pruning

Una delle principali sfide nel distribuire l'algoritmo di carpenter è la traduzione dei tre vincoli di pruning, pensati per funzionare in maniera centralizzata. I primi due sono facilmente riproducibili, ma il terzo è più complicato. La ragione è che prevede una struttura di memoria centralizzata dove inserire di volta in volta gli itemset valutati. Ovviamente questo non è realizzabile facilmente in contesto distribuito, in quanto l'accesso in scrittura a risorse centralizzate pone diversi problemi, come ad esempio le corse critiche. Occorre quindi formulare una nuova definizione per individuare quando un itemset sia già stato valutato o meno. Intuitivamente, il supporto di un itemset può essere interpretato come un percorso di quanti. Scopo della ricerca di itemset chiusi è di trovare gli itemset aventi maggior supporto: alla luce di quanto detto sopra, il supporto più grande corrisponde al

percorso più lungo. Avendo le celle un identificatore ordinato, ogni supporto può essere interpretato come un percorso che parte dal quanto con id minore e arriva in quello con supporto maggiore. Alla luce di ciò, terzo criterio di pruning di carpenter può essere riformulato come: "una tupla deve essere mantenuta se questa sta costruendo il percorso più lungo per un certo itemset, altrimenti questa deve essere scartata'.

Analizzando questo nuovo criterio, il vincolo può essere descritto nei seguenti punti:

- Non devono esistere celle con ID minore all'inizio della sequenza che contengono quest'ultima nel loro vicinato. Intuitivamente se ciò accadesse, vorrebbe dire che il percorso massimo non comincia dall'attuale prima cella. In tal caso infatti esisterebbe un'altra cella che genererà lo stesso percorso che sta venendo considerato ma con supporto totale maggiore. Per evitare una ripetizione è necessario quindi scartare l'attuale tupla.

- La somma del supporto totale e delle potenziali celle che contengono l'itemset I deve avere valore maggiore di α. Se ciò non fosse vero, I non potrebbe mai raggiungere la soglia di supporto per essere considerato frequente, di conseguenza sarebbe inutile espandere la tupla nelle iterazioni successive.

- La somma del supporto totale, delle potenziali celle che contengono l'itemset I da esplorare e delle celle antecedenti alla testa della sequenza valide per il supporto devono coprire tutte le celle che supportano I. In caso contrario, non sarebbero mai considerate potenziali celle per l'espansione.

La funzione di filtraggio viene formalizzata nell' algoritmo 6

Algoritmo 6 Filter

Richiede: $\mathcal{T}$: Insieme delle transazioni, N: Vicinato, α: soglia di supporto, γ: soglia di cardinalità.

Restituisce: l è frequente e potenzialmente chiuso.

```
1: if l.lcluster.size >= γ ∧ l.X.size + l.R.size >= α then ▷ La tupla ha una cardinalità valida e un supporto potenzialmente buono
2:     totalSupport = sup(l.lcluster)
3:     if !N.isEmpty then ▷ Se il vicinato è definito, la tupla deve risultare valida rispetto ai vincoli spazio-temporali
4:         minCellID = l.X.min
5:         OutlierCells ← ∅
6:         for each cell ∈ totalSupport do
7:             if cell.ID >= min.ID ∨ minCellID ∉ neighborhood(cell) then
8:                 OutlierCells ∪ cell
9:         if (totalSupport l.X l.R OutlierCells) == ∅ then ▷ Controlla che tutte le celle continue siano dentro X o R
10:            return true
11:    else ▷ Altrimenti controlla che X + R contenga tutte le celle che supportano I
12:        if min(totalSupport) ∈ l.X ∧ (totalSupport l.X l.R) == ∅ then
13:            return true
return false
```

I vincoli sopra espressi permettono quindi di esprimere un filtro analogo a quello espresso nell'algoritmo di Carpenter.

5 Test sperimentali

In questo capitolo vengono presentati i risultati ottenuti dall'algoritmo CUTE. In primo luogo sono analizzati i dataset utilizzati nei test svolti. Successivamente sono mostrati i risultati ottenuti da CUTE per quanto riguarda i cluster individuati e le performance in relazione al tempo. Infine è discusso il confronto tra CUTE e SPARE: sono presentate le differenze tra gli approcci e i risultati ottenuti.

I test sono stati eseguiti su un cluster composto da 15 nodi, ognuno dei quali equipaggiato con un 8-core i7 CPU 3.60Hz e 32GB di RAM e interconnessi da una ethernet Gigabit.

5.1 Dataset

Nel processo di testing e confronto tra i due algoritmi sono stati impiegati due diversi dataset.

Il primo dataset utilizzato è Geolife ([50]). Questo dataset è composto da un insieme di traiettorie raccolte in Asia, in particolare in prossimità della città di Pechino. In Geolife sono presenti 17158 traiettorie distinte, composte da punti contenenti latitudine, longitudine e altitudine. Il 91% delle traiettorie nel dataset sono generate da dispositivi aventi tempo di campionamento compreso tra 1 e 5 secondi. Il tempo totale coperto da tutte le traiettorie nel dataset è circa quattro anni.

Il secondo dataset impiegato è composto da un insieme di traiettorie registrate nella città di Oldenburg. Questo conta 1000000 traiettorie, a differenza di quanto detto prima, i punti non sono espressi in coordinate polari, ma in coordinate cartesiane rispetto a un sistema di riferimento custom. Anche per quanto riguarda la dimensione temporale, il dataset viene fornito con punti aventi una componente temporale relativa a una sequenza di istanti. La sequenza in questione conta 245 istanti distinti.

La tabella 5.1 riassume le caratteristiche principali dei dataset impiegati nei test. Per quanto riguarda il numero di traiettorie, Geolife viene utilizzato sempre nella sua interezza, Oldenburg invece partizionato diversamente a seconda delle situazioni di impiego.

Dataset	Spazio	Tempo	Punti	Traiettorie
Geolife	Coordinate polari	yyyy-mm-gg hh:mm:ss	18891115	17158
Oldenburg	Coordinate cartesiane	Sequenziale	64895840	1000000

Tabella 5.1: Riassunto delle caratteristiche principali dei due dataset utilizzati

5.2 CUTE

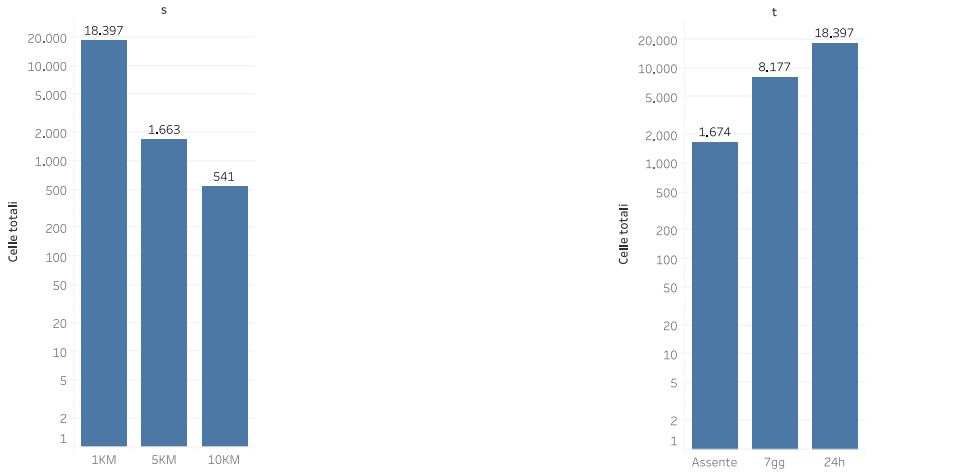
Di seguito sono presentati tutti i test eseguiti su CUTE. Questi test sono stati condotti, salvo specificato diversamente, su un insieme di 9 executors, aventi ciascuno 3 cores. Ad ogni executor sono stati assegnati 12 gigabyte di RAM.

5.2.1 Cluster individuati

Scopo dell'algoritmo CUTE è la ricerca di pattern di movimento aventi caratteristiche di rilevanza rispetto al percorso e dimensione del gruppo in movimento. Essendo CUTE un framework generico, deve essere possibile distinguere le varie tipologie di pattern di movimento (swarm, flock, platoon) e verificare le variazioni nel numero degli itemset. Intuitivamente, una ricerca di swarm produrrà molti più itemset validi di una di flock, per via dei vincoli più rilassati. Allo stesso modo i diversi livelli di grana delle celle dovranno produrre risultati diversi a seconda delle soglie di supporto e dimensioni. Per condurre questi test, il dataset utilizzato è stato Geolife, in quanto compatibile con le scale settimanale e giornaliera.

Primo e fondamentale ambito di testing è stato il sistema di riferimento e di conseguenza la granularità di ogni cella. Questo parametro risulta particolarmente interessante: in un algoritmo di row enumeration, come CUTE o Carpenter, il numero di transazioni determina la complessità dell'algoritmo. Come detto nella sezione 4.1, nel primo passo della ricerca dei gruppi, ogni cella corrisponde a una transazione. A seconda del valore dei parametri s e t, ovvero l'intervallo spaziale e quello temporale, il volume di ogni cella e il loro numero complessivo cambia. Le figura 5.1a mostra il numero di celle al variare del lato s tra $1, 5, 10$KM. All'aumentare del lato spaziale delle celle, la compressione dello spazio di ricerca aumenta, diminuendo quindi il numero di celle. La figura 5.1b mostra invece il rapporto tra t che varia su $24h, 7gg$ e in assenza di dimensione temporale e le celle. Com'è possibile vedere, il numero delle celle a parità di spazio non cresce esponenzialmente, ma di un fattore moltiplicativo collegato ai possibili valori dell'intervallo temporale.

Su ognuna delle configurazioni sopra espresse è stata eseguita una ricerca di itemset rispetto alle variazioni dei parametri in gioco presentati nella sezione 4.2.1. I valori espressi per questi parametri sono riassunti nella tabella 5.2. Nei successivi test, dove non specificato, ai parametri è assegnato il valore di default (in grassetto nella tabella 5.2).



(a) s tra $1, 5, 10$ KM

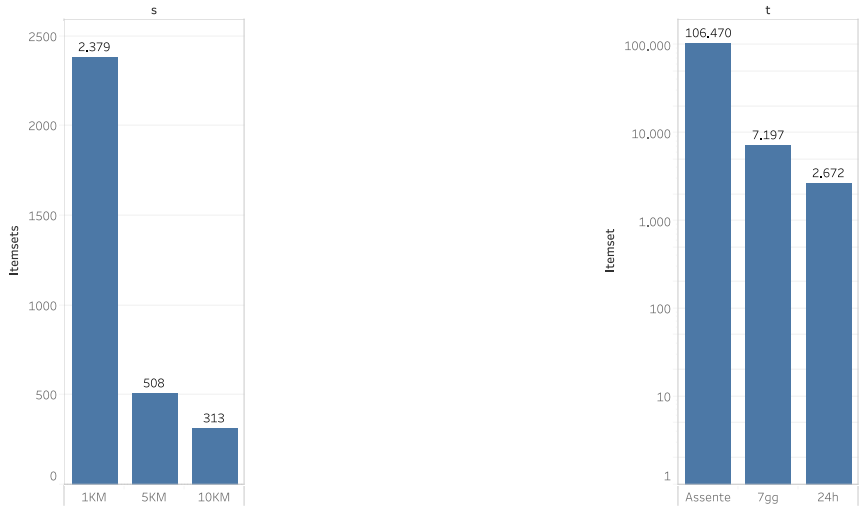
(b) t tra 24h, 7gg e assenza di tempo

Figura 5.1: Celle alla variazione di s a sinistra, t a destra

Param.	Significato	Valori
s	Lato spaziale cella	**1KM**, 5KM, 10KM
t	Lato temporale cella	**24h**, 7gg, assenza di tempo
minsize	dimensione dei gruppi	5, **10**, 15
minsup	supporto minimo	5, **10**, 15
ϵ	soglia spaziale	2, **5**, 10, ∞
τ	soglia temporale	1, **2**, ∞

Tabella 5.2: Parametri e il loro valore di default (in grassetto)

In particolare il focus di questi test è capire come varia il numero dei cluster individuati all'interno del problema. I valori di supporto e dimensione minima coincidono e sono stati determinati empiricamente. Il range di ϵ varia tra $(2, 5, 10, \infty)$ mentre quello di τ tra $(1, 2, \infty)$. Mentre i valori di ϵ sono stati scelti sperimentalmente, quelli di τ sono stati selezionati sulla base del range delle scale temporali utilizzate. In particolare le scale giornaliere e settimanale presentano una natura ciclica. Una scala ciclica non ha un valore massimo assoluto e un minimo assoluto, ma prevede che il valore successivo al massimo sia il minimo. Questa circolarità impatta la misura della distanza tra due punti. Questa è definita come la dimensione del più piccolo intervallo tra due punti: può essere quindi la minima distanza calcolata dal maggiore al minore o viceversa. Ciò comporta che all'interno di una scala ciclica, i valori siano molto più vicini rispetto a una monotona. Quindi è molto facile che considerando valori alti di τ il risultato tenda a essere quello di $\tau = \infty$. Ciò vale soprattutto per la scala settimanale: essendo solamente 7 i giorni della settimana e la scala ciclica per definizione, la distanza massima tra due qualunque elementi è al massimo 3. Questo implica che l'unico valore plausibile per eseguire una ricerca di platoon su questa scala sia 2. τ determina inoltre la categoria di co-movement pattern individuato, ad esempio su scala settimanale vale che:

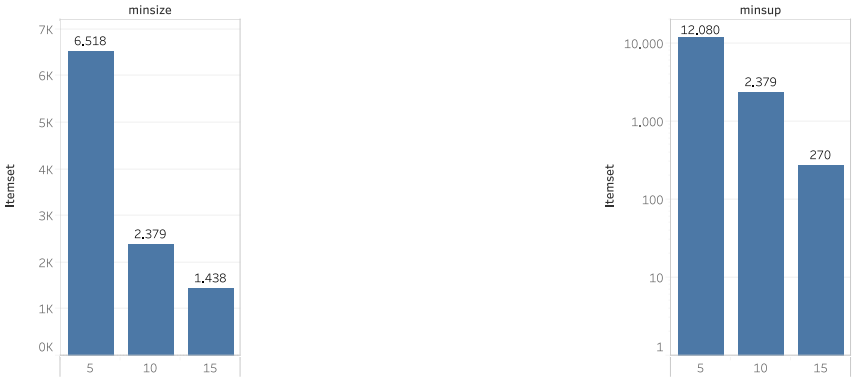


(a) s tra $1, 5, 10$ KM (b) t tra 24h, 7gg e assenza di tempo

Figura 5.2: Numero di itemset alla variazione di s a sinistra, t a destra

- $\tau = 1$: Flock
- $\tau = 2$: Platoon g-connected
- $\tau = \infty$: Swarm

Le figure 5.2a e 5.2b mostrano le variazioni del numero di itemset su s e t. Per quanto riguarda le variazioni nel lato spaziale delle celle s, i risultati sono coerenti con quanto atteso: all'aumentare dell'area coperta dalle celle, calano il numero di itemset individuati (figura 5.2a). Ciò è coerente con la natura del problema: se ogni cella corrisponde a una transazione, al calare del numero di transazioni a parità di supporto calano sia il numero di itemset frequenti che il tempo necessario per eseguire la ricerca. Per t, i risultati presentati sono stati ottenuti ponendo $\tau = \infty$, questo perché non avrebbe avuto senso considerare la continuità nel tempo considerando una scala come l'assenza di tempo che esclude questa dimensione (figura 5.2b). Alla luce di ciò, la dimensione temporale t presenta invece un trend opposto a s per quanto riguarda gli itemset individuati: l'impiego della scala giornaliera rispetto a quella settimanale individua meno itemset sia in caso di vincoli su τ che in sua assenza. L'assenza di una dimensione temporale invece produce risultati di circa un ordine di grandezza superiori rispetto agli altri due. Questo fenomeno può essere direttamente collegato all'aumento del numero di celle a seguito dell'adozione di diverse scale temporali. L'introduzione di una nuova dimensione produce infatti un'ulteriore suddivisione dello spazio di ricerca: un pattern valido in uno spazio n-dimensionale può non risultarlo più in uno $n + 1$ dimensionale, poiché il gruppo può non risultare vicino nella nuova dimensione. Questa frammentazione sarà tanto più netta tanti più valori possibili avrà la scala della nuova dimensione. Applicando quanto detto sopra alle scale temporali, l'assenza di tempo non produce nessuna ulteriore divisione, la scala settimanale e quella temporale aggiungono una cardinalità di rispettivamente 7 e 24 intervalli. La



(a) *minsize* tra $5, 10, 15$ (b) *minsup* tra $5, 10, 15$

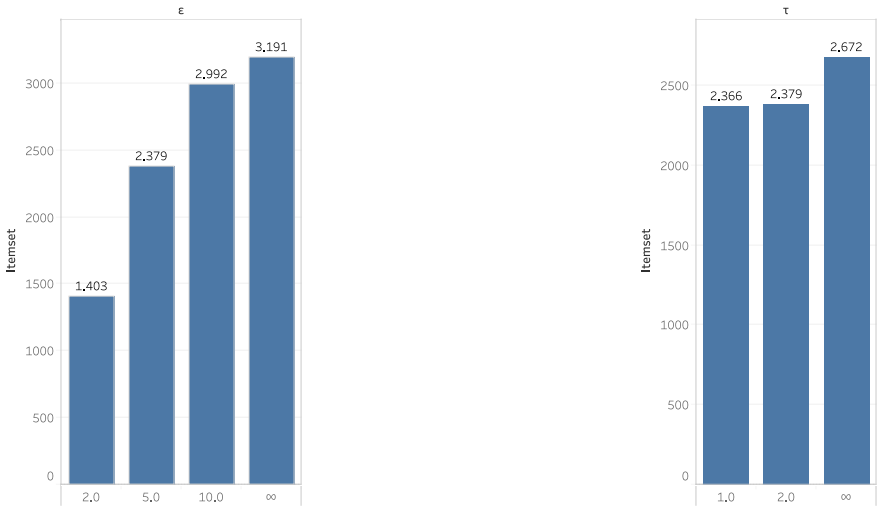
Figura 5.3: Numero di itemset alla variazione di *minsize* a sinistra e *minsupp* a destra

frammentazione dello spazio di ricerca causa quindi la differenza nei risultati delle tre scale.

Le figure 5.3a e 5.3b individuano le variazioni di pattern riconosciuti al variare di supporto e dimensione minima. Minsup e minsize (α, γ) sono fatti variare tra $5, 10, 15$. Entrambi i parametri si comportano come atteso: all'aumentare del valore cala il numero di itemset individuati. La ricerca di gruppi di dimensioni maggiori o che abbiano un elevato tempo di viaggio riduce il numero di itemset validi.

Infine i test condotti su ϵ e τ mostrano il diverso numero di pattern individuati stringendo o rilassando i vincoli di contiguità. Per quanto riguarda ϵ, la figura 5.4a mostra come il numero di pattern individuati cali al diminuire del valore di soglia. Questa tendenza rallenta solo nel caso dei valori 10 e ∞ dove i risultati quasi coincidono. I risultati ottenuti su questo parametro sono coerenti con quanto atteso: vincoli più stretti producono meno itemset. La coincidenza tra il valore 10 e ∞ può essere motivata dall'ampiezza del raggio di vicinanza coperto rispetto all'area complessiva del problema. Un raggio di vicinato pari a 10KM include probabilmente la maggior parte delle celle individuate nello spazio, di conseguenza i suoi risultati tendono a quelli ottenuti rilassando il vincolo.

Per quanto riguarda τ, è possibile riscontrare come la ricerca di flock, platoon e convoy non produca le differenze attese nei risultati. Su Geolife le differenze riguardano solo poche tuple tra le tre ricerche. Questi risultati sono mostrati nella figura 5.4b. Questo comportamento è a prima vista inusuale: i vincoli sullo spazio si dimostrano efficaci nella riduzione dello spazio di ricerca mentre quelli sul tempo sono praticamente trascurabili. Tuttavia occorre considerare la diversità fra la scala spaziale e quella temporale in termini di possibili elementi: quest'ultima infatti ha un range decisamente inferiore rispetto a qualunque scala temporale utilizzata nel corso dei test. Nel caso di scala giornaliera ad esempio i valori possibili sono 24, mentre per quella settimanale sono 7. Come detto nella presentazione di Geolife, i singoli punti hanno frequenze di campionamento molto più elevate della granularità temporale del sistema di riferimento. Quindi sono poche le traiettorie che hanno una continuità rispetto alle ore del giorno, ancora meno rispetto ai



(a) ϵ tra $2, 5, 10, \infty$ (b) τ tra $1, 2, \infty$

Figura 5.4: Numero di itemset alla variazione di ϵ a sinistra e τ a destra

giorni della settimana.

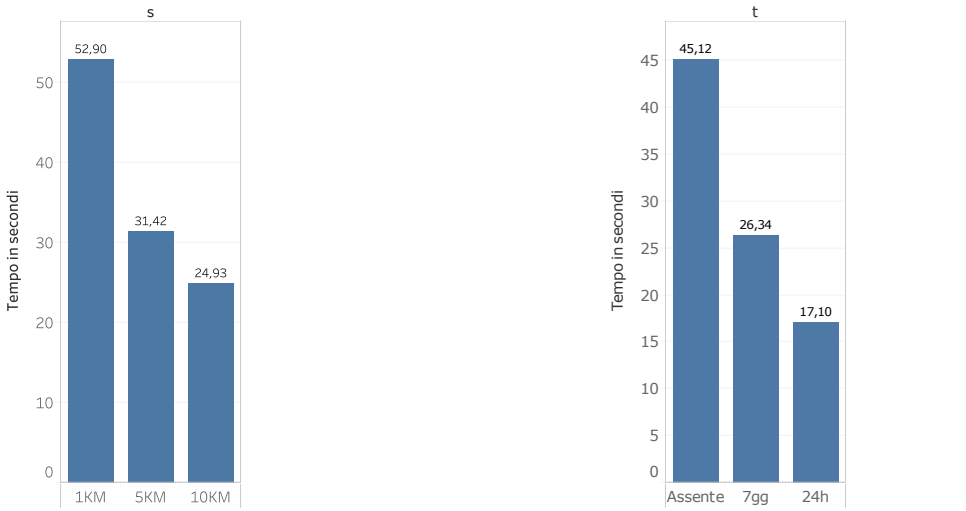
5.2.2 Performance

In parallelo sono state verificate le performance di CUTE valutando diverse situazioni. In primo luogo, è stato valutato il tempo di esecuzione rispetto alle configurazioni già presentate nella sezione 5.2.1. Successivamente sono stati eseguiti test per valutare le tempistiche al variare del numero di risorse fisiche impiegate, ovvero executor e core. In tutti i test il tempo è sempre registrato in millisecondi.

Nel cronometrare la durata dell'algoritmo, non è considerata la fase di creazione delle tabelle di supporto. Questo perché le tabelle in questione possono potenzialmente essere create separatamente da CUTE e poi utilizzate dall'algoritmo. Inoltre a parità di parametri s e t una stessa tabella può essere utilizzata da più esecuzioni dell'algoritmo.

Partendo dalle valutazioni basate sull'alterazione dei parametri dell'algoritmo, la figura 5.5 mostrano il tempo di calcolo rispetto alle variazioni dei lati spazio-temporali della cella. Al crescere di s (figura 5.5a), cala il tempo di esecuzione. Intuitivamente ciò è collegato al numero di celle: celle più grandi e meno numerose corrispondono a meno transazioni da valutare durante la ricerca. Per t (figura 5.5b) la situazione è diversa. Il trend è inversamente proporzionale al numero di celle: al calare di queste, aumentano i tempi di esecuzione. Nonostante quindi il numero di celle sia maggiore all'inizio con $t = 24h$, la minor lunghezza delle transazioni porta a generare un minor numero di gruppi da esplorare (come evidenziato anche negli itemset individuati) e causa quindi una diminuzione nei tempi di esecuzione.

Variando *minsize* (figura 5.6a) è possibile vedere un trend di diminuzione dei tempi di esecuzione al crescere del parametro. Questo è coerente con quanto atteso: la ricerca di



(a) s tra $1, 5, 10$ KM (b) t tra 24h, 7gg e assenza di tempo

Figura 5.5: Tempo di computazione alla variazione di s a sinistra, t a destra

gruppi più numerosi comporta il pruning dei piccoli gruppi e risparmia tempo sul tempo di esecuzione globale. Considerando invece *minsup* (figura 5.6b), vale lo stesso discorso. Le ragioni sono analoghe a quanto detto per *minsize*.

Trattando poi dei parametri di continuità che riducono lo spazio di ricerca, ϵ (figura 5.7a) presenta per entrambi i dataset una certa regolarità: al crescere del suo valore, aumenta il tempo di computazione. Questa regolarità dipende dalla riduzione dello spazio di ricerca effettuato dai vincoli. Valori più rigidi comportano una maggior riduzione dello spazio di ricerca, che si traduce in un pruning più forte

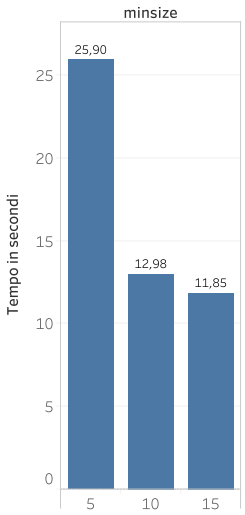
Su τ (figura 5.7b) la situazione è più eterogenea: non è possibile individuare un trend definito. Ancora una volta le cause sono da ricercare nel debole pruning di τ, la presenza di questo vincolo infatti può in ceri casi rallentare l'esecuzione, in quanto giunge a risultati assimilabili a quelli ottenuti rilassando il vincolo. Questa ricerca avviene in maniera più lenta, per via delle operazioni aggiuntive sul controllo della continuità.

Passando ai test di scalabilità, questi sono stati condotti su un sottoinsieme dei parametri utilizzati nella sezione precedente. Ciascuna configurazione di parametri è stata testata variando rispettivamente il numero di executor e quello di core, mantenendo come altri parametri i valori di default specificati nella tabella 5.2. La tabella 5.3 contiene i valori possibili per core e executor.

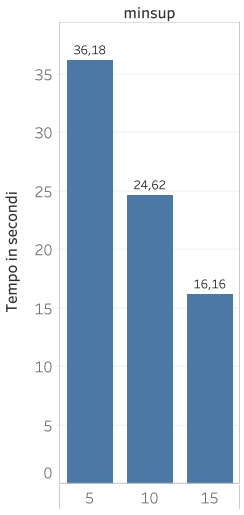
Parametro	Valori
core	2,**3**,4
executor	5,**7**,9

Tabella 5.3: Valori per core e executor (default in grassetto)

Le figure 5.8a e 5.8b mostrano i tempi di esecuzione alla variazione di questi parametri.

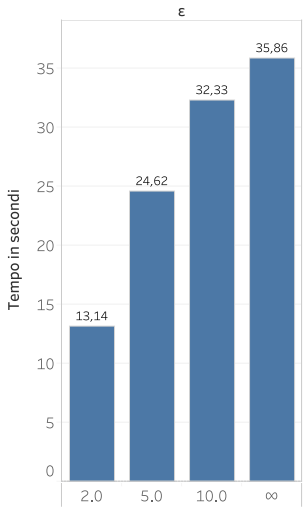


(a) *minsize* tra $5, 10, 15$

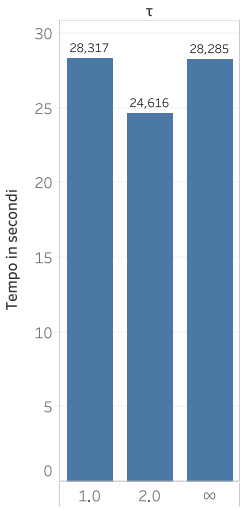


(b) *minsup* tra $5, 10, 15$

Figura 5.6: Tempo di computazione alla variazione di *minsize* a sinistra e *minsupp* a destra

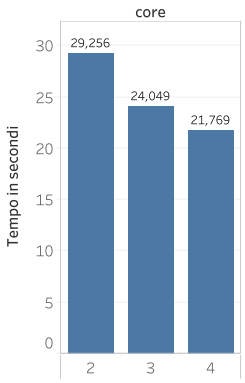


(a) ϵ tra $2, 5, 10, \infty$

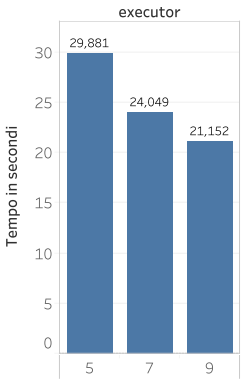


(b) τ tra $1, 2, \infty$

Figura 5.7: Tempo di computazione alla variazione di ϵ a sinistra e τ a destra



(a) core tra $2, 3, 4$



(b) executor tra $5, 7, 9$

Figura 5.8: Tempo di computazione alla variazione di core a sinistra e executor a destra

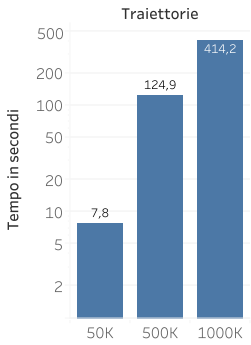


Figura 5.9: Tempi di esecuzione in secondi variando il numero di traiettorie tra 50K,500K,1000K

Com'è possibile notare, la tendenza generale espressa dai grafici è una diminuzione dei tempi di esecuzione all'aumentare dell'impiego di risorse.

Infine per quanto riguarda l'ambito delle performance, sono stati eseguiti alcuni test di scalabilità rispetto al numero di traiettorie. In particolare per questi è stato utilizzato Oldenburg, in quanto avente un numero di traiettorie molto superiore a Geolife. La tabella 5.4 contiene i parametri di CUTE utilizzati durante queste esecuzioni.

Parametro	Valore
traiettorie	50K,500K,1.000K
s	1000
t	10
$minsize$	100
$minsupp$	10
ϵ	2
τ	1
core	3
executor	9

Tabella 5.4: Valori dei parametri di CUTE utilizzati durante i test sul numero di traiettorie su Oldenburg

I risultati indicano come a parità di risorse, CUTE riesca a gestire in tempi ragionevoli anche un numero di transazioni molto elevato (figura 5.9). Nel caso peggiore infatti sono necessarie appena 10 minuti per eseguire la ricerca su un milione di traiettorie. Questo dipende da diversi fattori. In primo luogo il numero di celle, e di conseguenza quello di transazioni, rimane invariato durante le varie esecuzioni. Ciò che cambia è quindi la lunghezza media di ogni transazione, che aumenta all'aumentare del numero di traiettorie. Insiemi di transazioni di diversi ordini di grandezza vengono processati in diversi ordini temporali di grandezza. Oltre a ciò, questi risultati possono essere collegati a valori molto stringenti per i parametri collegati al pruning, dimostrando così la loro efficacia nel ridurre lo spazio di ricerca.

5.2.3 Considerazioni

Alla luce di quanto mostrato riguardo a itemset individuati e performance, si possono trarre le seguenti considerazioni.

In primo luogo i risultati ottenuti confermano l'importanza della suddivisione in quanti sulla base del sistema di riferimento. Questo è un aspetto delicato. La grana del sistema di riferimento è direttamente collegata con il tipo di pattern che si vuole ricercare. Differenti ricerche infatti possono portare a differenti suddivisioni dello spazio di ricerca. Ad esempio una ricerca che consideri la rete stradale di una città avrà sempre molti più quanti di una che considera i quartieri o municipalità di una città. Occorre quindi valutare il comportamento rispetto alla variazione di transazioni in input e al sistema di riferimento. Per simulare questo scenario, sono state utilizzate più griglie che hanno permesso di costruire di volta in volta un numero di transazioni arbitrario.

Dai test svolti è emerso che il comportamento dell'algoritmo al variare del numero di celle può essere riassunto come segue: Per ogni dataset, esiste un sistema di riferimento ideale avente una certa granularità. Nell'utilizzo di griglie a grana più grossa, a parità di dimensione e supporto minimo, calano le transazioni di esplorare. Di conseguenza calano gli itemset validi (per via del supporto) e i tempi di esecuzione. Usando griglie a grana troppo fine, il numero di transazioni aumenta, ma diminuisce la lunghezza di ogni transazione. In questo caso è la frammentazione a ridurre il numero di elementi in ogni transazione, diminuendo quindi il supporto e la dimensione media degli itemset individuati. Questo comporta un maggiore pruning e di conseguenza una riduzione nei tempi di calcolo e negli itemset trovati. In entrambi questi due casi, il sistema di riferimento risulta scarsamente compatibile con le caratteristiche spazio-temporali del dataset.

Per quanto riguarda i parametri di supporto *minsup* e dimensione *minsize*, il comportamento dell'algoritmo è in linea con quanto atteso in ogni situazione. L'aumentare dei due parametri comporta una riduzione degli itemset validi. Il pruning di questi itemset diminuisce il volume della ricerca e di conseguenza il costo computazionale totale.

Altro punto focale emerso dai test è la debolezza nel vincolo di continuità temporale negli itemset individuati: rispetto a quello spaziale infatti il pruning effettuato è molto più debole. Come già affermato precedentemente, questo dipende dalla finezza della scala impiegata. Per quanto riguarda la componente spaziale, è improbabile che traiettorie vicine si muovano su celle lontane della mappa. Per il tempo invece, come affermato nella sezione 5.2.1, le scale delle ore/giorni risultano poco compatibili con l'alto tempo di campionamento delle traiettorie. Sempre collegato ai vincoli di continuità e alla riduzione dello spazio di ricerca, non sempre accade che una riduzione dello spazio di ricerca comporti un calo nei tempi di esecuzione. Come affermato nella sezione 4.2.4, il rilassamento dei vincoli sulla continuità nello spazio tempo rende superflue alcune operazione e rende più snella la computazione. Se quindi il pruning effettuato da questi vincoli è debole, lo spazio di ricerca non sarà ridotto e sarà esplorato più lentamente per via dei vincoli. Oltre a

questo, dal punto di vista implementativo la ricerca di swarm produce meno job da eseguire. Ogni job spark infatti aggiunge una costo computazionale alla ricerca, in quanto richiede un certo overhead in termini di tempo e risorse per essere creato.

CUTE, come mostrato dai test di scalabilità, ottiene tempi ragionevoli con diverse configurazioni di risorse. Ciò è possibile grazie al numero contenuto di celle individuate, che risulta sempre molto minore del numero di traiettorie. Qualora si adottasse una scala temporale assoluta o si riducesse di molto l'area spaziale coperta dalla singola cella, le performance di CUTE calerebbero notevolmente. La ragione dietro questo calo sarebbe attribuibile all'esplosione del numero di celle. Con un numero alto di celle, non solo la creazione delle tabelle di supporto aumenta esponenzialmente, ma vengono meno i principi del mining su dataset colossali, in quanto il numero di transazioni (celle) supera quello di item (traiettorie).

5.3 CUTE vs SPARE

Terminata l'analisi di CUTE in termini di itemset e tempo di computazione, si passa al confronto tra CUTE e SPARE. Gli algoritmi approcciano uno stesso problema, la ricerca di pattern di co-movimento, con due tecniche differenti tra di loro. Questo comporta che durante un confronto, non abbia senso parlare di tempistiche: CUTE e SPARE sono intrinsecamente diversi e la velocità non può essere un buon metro di confronto. Se il tempo di esecuzione non può essere utilizzato, allora l'intero confronto sarà impostato sulla base dei pattern ottenuti tra i due algoritmi.

5.3.1 Confronto tra CUTE e SPARE

CUTE e SPARE entrambi eseguono una ricerca di pattern di co-movimento. In tutti e due sono presenti i concetti di lunghezza minima del percorso (k, α), dimensione minima di un gruppo (m, γ) e massimo gap temporale tra i punti del percorso (g, τ).

Parlando poi delle idee di funzionamento dei due algoritmi, in entrambi si utilizzano i concetti di itemset,pruning e monotonicità. In CUTE ciò viene realizzato dividendo lo spazio di ricerca in celle, assegnando a ogni cella l'insieme delle traiettorie che transitano nei suoi confini spazio-temporali e eseguendo poi una variante distribuita dell'algoritmo di Carpenter per ricercare cluster di traiettorie. SPARE invece utilizza un approccio basato su itemset solo nella seconda fase, quella relativa al tempo e formula una propria definizione di monotonicità, diversa dalla frequenza, basata su due parametri l e g. Nella prima fase utilizza un clustering spaziale per definire ad ogni istante quali punti sono vicini e quali no.

SPARE prende come ipotesi che per ogni istante della scala temporale, ogni traiettoria abbia al massimo una posizione: questo diventa limitante quando si vuole considerare traiettorie generate da diverse scale di campionamento, poiché l'algoritmo riduce tutte le

traiettorie a una scala comune. Questo processo di semplificazione comporta una perdita di informazioni.

Anche su CUTE è necessario compiere un'operazione di assimilazione a una stessa scala temporale, tuttavia CUTE pone l'enfasi della sua divisione nella creazione di celle con un certo volume nello spazio-tempo. Dal punto di vista temporale, ogni cella non esprime un istante, ma una durata. Una traiettoria passa da una cella se transita nei suoi confini anche solo per un istante. Questo implica che una traiettoria possa muoversi su un range di celle aventi stesso istante temporale. In questo modo CUTE minimizza la perdita di informazione legata all'adozione di una scala temporale univoca su diverse traiettorie.

Dal punto di vista della vicinanza nello spazio, SPARE è più flessibile sulle forme dei cluster, potendo utilizzare vari algoritmi differenti per la creazione di questi. Al contrario CUTE ha confini spaziali più rigidi, determinati da una divisione regolare dello spazio. SPARE però considera la vicinanza solo in termini relativi, trascurando i movimenti eseguiti dal gruppo tra due istanti temporali. CUTE invece permette di esprimere vincoli sulla contiguità spaziale (ϵ) tra le varie celle percorse da un gruppo.

Parlando poi della dimensione temporale, CUTE presenta una gestione omogenea rispetto alla dimensione spaziale, processando quindi spazio e tempo assieme, tuttavia l'algoritmo mostra i suoi limiti in corrispondenza di bucket temporali nell'ordine dei secondi o pochi minuti, a causa dell'aumento del numero di celle. SPARE invece ha buone performance anche con bucket temporali nell'ordine dei secondi, inoltre permette una maggiore espressività nei vincoli temporali grazie al vincolo l. Dall'altro lato però CUTE è in grado di gestire scale temporale cicliche come i giorni della settimana o le ore del giorno, mentre SPARE è limitato a una scala temporale monotona.

CUTE consente inoltre di processare altre dimensioni senza nessuna ulteriore espansione, mentre SPARE è pensato esclusivamente solo per processare dati spazio-temporali.

Trattando infine i risultati, SPARE genera cluster basati su itemset frequenti massimali, CUTE invece su frequenti chiusi. CUTE quindi, a parità di configurazione, genererà sempre un numero maggiore di risultati rispetto a SPARE.

5.3.2 Risultati a parità di configurazione

Come detto nella sezione precedente, CUTE individua itemset chiusi, SPARE massimali. Per quanto molti passi nei due approcci divergano, è possibile in determinate condizioni ottenere gli stessi risultati tra i due algoritmi. Ciò però avviene solo in casistiche particolari, nella maggior parte delle situazioni succede che i risultai divergano per alcune caratteristiche intrinseche dei due algoritmi:

- *Itemset chiusi* $\supseteq$ *Itemset massimali.* È fisiologico, salvo rare eccezioni, che il numero di itemset chiusi in un dataset sia anche di ordini di grandezza più grande di quello dei massimali.

- *Le metriche spaziali divergono.* L'impiego di due metriche spaziali differenti porta a risultati che in assoluto possono divergere per alcune tuple.

Per quanto i due algoritmi possano avere elementi in comune, queste particolarità devono essere tenute in conto durante il confronto dei risultati. Confrontare il numero di itemset identici produrrebbe quindi scarsi risultati. Serve perciò definire una metrica di confronto che tenga conto di queste due caratteristiche: non deve penalizzare eccessivamente la differenza nel numero dei risultati e deve misurare la similarità tra gli itemset tenendo conto che questi possono divergere per l'assenza o presenza di qualche elemento.

Alla luce di questo, è stata formulata la seguente misura di similarità S. Per confrontare i due insiemi di itemset si utilizza una variazione dello Stable Marriage Problem [51]. Dati due set di stesse dimensioni, questa tecnica permette di assegnare ogni elemento di un set a uno dell'altro sulla base di un criterio di similarità custom. Questo permette di ottenere una soluzione sub-ottima al problema dell'assegnamento, assicurandosi che non esistano due elementi appartenenti a insiemi diversi che abbiano una similarità maggiore tra di loro rispetto che coi rispettivi partner (definizione 5.3.1).

Definizione 5.3.1 (Stable Marriage Problem)**.** Dati due insiemi di oggetti N e M, si definisce problema del matrimonio stabile il problema che ricerca l'insieme delle coppie $(n_i \in N, m_j \in M)$ tali che definita una metrica di similarità s non esistano in contemporanea due coppie $(n_p, m_p), (n_q, m_q)$ tali che $n_p, n_q \in N, m_p, m_q \in M$ per cui vale che:

$$\begin{cases} s(n_p, m_j) > s(n_i, m_j) \wedge s(n_p, m_j) > s(n_p, m_p) \\ s(n_i, m_q) > s(n_i, m_j) \wedge s(n_i, m_q) > s(n_q, m_q) \end{cases}$$

Definita la modalità di assegnamento, si rende necessario individuare una metrica di confronto tra i singoli itemset. Questa è definita via coefficiente di Jaccard. Questo calcola la similarità tra due set come il numero di elementi condivisi tra i due diviso l'unione di tutti gli elementi presenti in entrambi (definizione 5.3.2).

Definizione 5.3.2 (Coefficiente di Jaccard)**.** Dati due set di item N e M, si definisce il coefficiente di Jaccard tra i due insiemi come:

$$Jaccard(N, M) = \frac{|N \cap M|}{|N \cup M|}$$

Una volta definita questa metrica, la similarità totale S tra i risultati di CUTE e SPARE viene calcolata nel seguente modo: in primo luogo viene eseguito l'assegnamento di ogni cluster individuato su un algoritmo a quello di un altro sulla base del valore di Jaccard tra i due. Questa operazione segue le regole del problema del matrimonio stabile, di conseguenza produce una soluzione sub-ottima. A questo punto viene calcolata la somma totale dei valori di similarità tra ogni coppia e questo risultato viene poi diviso per il numero delle coppie totali. Il risultato di questa operazione determina la similarità tra due insiemi.

Definizione 5.3.3 (Similarità tra due set di cluster)**.** Dato un set di cluster di CUTE C, uno di SPARE M e insieme di coppie $CM = \{(c_1, m_1), ..., (c_n, m_n)\}$, generato mediante risoluzione del problema del matrimonio stabile utilizzando $Jaccard$ come metrica di similarità, tali che $c_i \in C \wedge m_i \in M$ si definisce la similarità S con la seguente formula:

$$S(C, M) = \frac{\sum_{k=1}^{n} Jaccard(c_k, m_k)}{n} \tag{5.1}$$

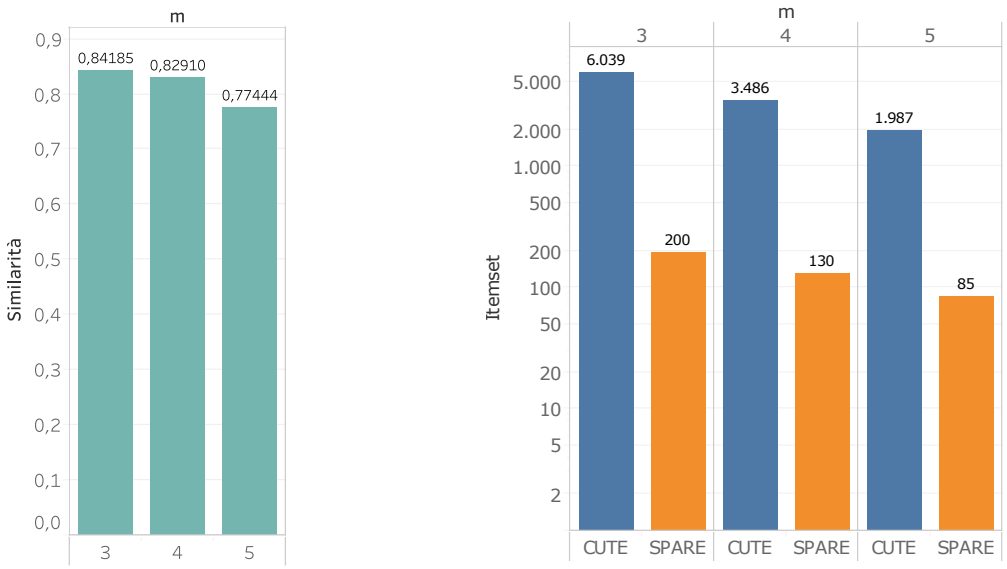
Occorre però un'ulteriore precisazione. Come detto sopra il problema del matrimonio stabile funziona solo se entrambi gli insiemi hanno lo stesso numero di elementi. CUTE e SPARE invece divergono per numero di itemset: in particolare CUTE individuerà in media molti più risultati di SPARE. Per risolvere questo problema, sono state definite due variazioni della metrica: in una gli elementi in più sono scartati (S_{minus}) mentre nell'altra (S_{plus}) contribuiscono alla misura con un valore di similarità pari a 0. Tra queste due metriche ci è sembrato più rilevante cercare di massimizzare (S_{minus}), in quanto misura che penalizza in maniera minore la differenza di cardinalità tra massimali e chiusi.

Definita la procedura di confronto, è stato scelto come dataset utilizzato Oldenburg, campionato ogni 5 istanti temporali. Questa configurazione garantisce che il numero di celle create da CUTE aumenta in maniera controllata, allo stesso tempo il dataset rimane abbastanza compatto per ottenere risultati significativi rispetto a SPARE.

I primi test sono stati condotti sul dataset nella sua interezza, a parità di risorse fornite. CUTE sin da subito è stato in grado di gestire la complessità del problema nel suo insieme, mentre SPARE no. In numerose occasioni, le risorse fornite all'applicazione non sono state sufficienti per permettere all'algoritmo di terminare con successo.

Alla luce di questi fallimenti, è stato ridotto il numero di traiettorie impiegate. Sono stati così condotti esperimenti su $1000, 2000$ e infine 3000 traiettorie. Anche in queste condizioni, SPARE ha avuto difficoltà nell'eseguire la ricerca con determinate configurazioni. Inoltre la differenza tra i risultati dei due algoritmi risultava decisamente grande: in alcune situazioni SPARE individuava 1-2 elementi, mentre CUTE sui 1000-2000.

Sono state impiegate allora tecniche e accorgimenti per riuscire a ottenere risultati significativi. Il primo passo è stato individuare una configurazione di $\epsilon, minPts$ che permettesse a SPARE di individuare un numero consistente di itemset (> 50) su più configurazioni. Successivamente è stato lanciato diverse volte CUTE, variando il lato spaziale delle celle su valori ragionevoli e direttamente collegati a ϵ. Di seguito (figure da 5.10 a 5.13) è illustrato il confronto tra SPARE e CUTE. La tabella 5.5 riassume le configurazioni di parametri in gioco.



(a) Similarità S_{minus}

(b) Itemset individuati su CUTE e SPARE

Figura 5.10: Similarità a sinistra, itemset a destra al variare della dimensione minima dei gruppi m

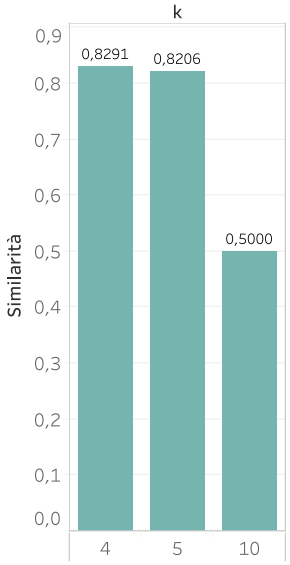
Parametro	Algoritmo	Valori
s	CUTE	$1.5, \mathbf{1.75}, 2$ KM
ϵ	SPARE	**2** KM
$minPoints$	SPARE	**5**
m	SPARE e CUTE	$3, \mathbf{4}, 5$
k	SPARE e CUTE	$4, \mathbf{5}, 10$
g	SPARE e CUTE	$2, \mathbf{5}, 1000$

Tabella 5.5: Valori dei parametri in gioco durante il confronto (default in grassetto)

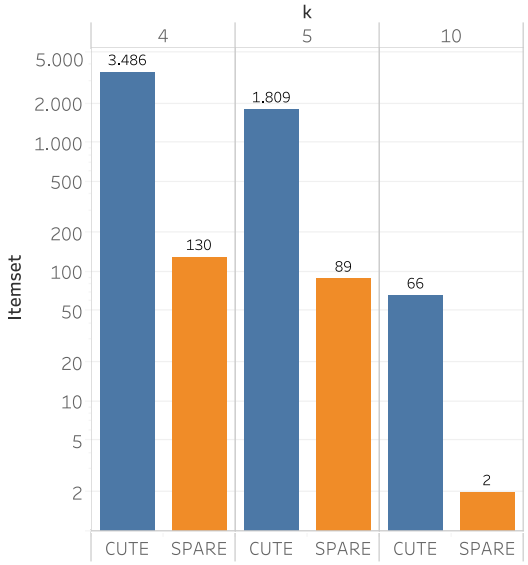
Al variare di m (figura 5.10), è possibile vedere come all'aumentare delle dimensioni dei gruppi, la similarità totale tenda a calare. Discorso analogo vale per k: al suo aumentare la similarità totale cala. Questo può essere giustificato sulla base delle divergenze nel determinare la vicinanza spaziale tra i due algoritmi. Impostando infatti vincoli più rigidi, verranno riconosciuti itemset con maggiori dimensioni e supporto. All'aumento di queste due misure corrisponde un incremento della possibilità che la composizione dei gruppi individuati tra i due algoritmi cambi.

Trattando invece di g figura 5.12, al rilassamento della continuità aumenta la similarità. Anche questo è collegato con quanto detto sopra: pattern swarm contenenti gli stessi oggetti possono essere individuati in situazioni differenti. È più raro che ciò possa accadere invece con pattern continui nel tempo, come ad esempio flock.

Infine al variare di s si può vedere come il valore di similarità migliore si ottenga con il valore medio di s. La spiegazione di ciò può essere ricondotta alla distribuzione dei dati nello spazio. Purtroppo è difficile determinare con precisione il valore ottimale di s, se non

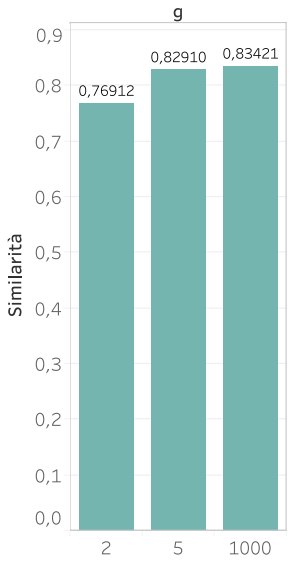


(a) Similarità S_{minus}

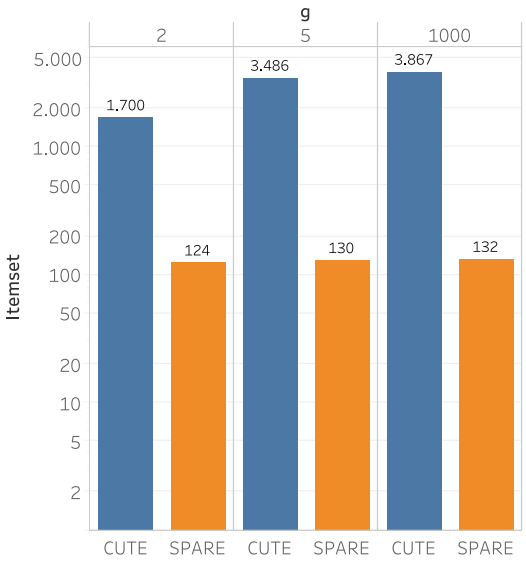


(b) Itemset individuati su CUTE e SPARE

Figura 5.11: Similarità a sinistra, itemset a destra al variare del supporto minimo k

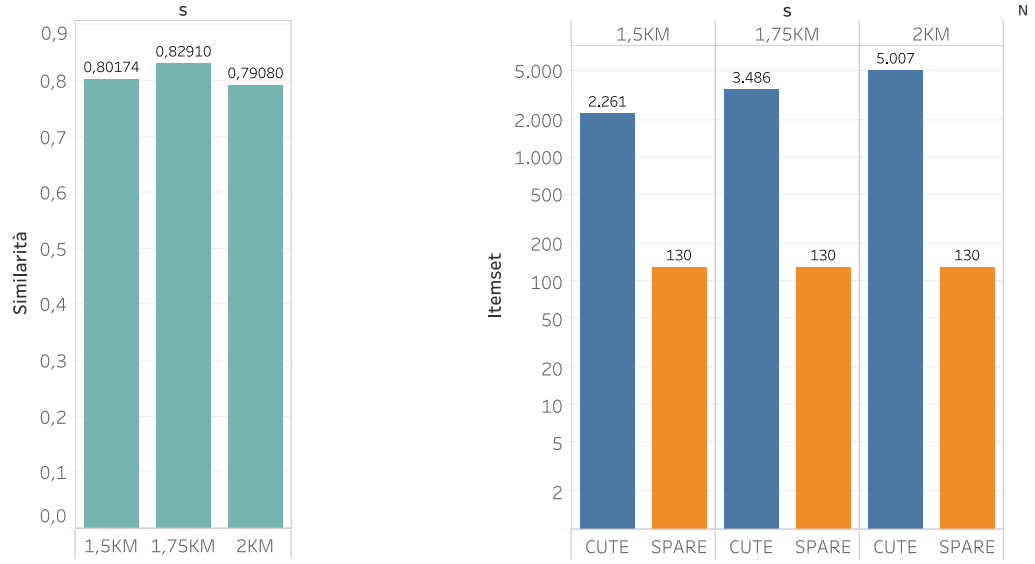


(a) Similarità S_{minus}



(b) Itemset individuati su CUTE e SPARE

Figura 5.12: Similarità a sinistra, itemset a destra al variare della continuità sul tempo g



(a) Similarità S_{minus} (b) Itemset individuati su CUTE e SPARE

Figura 5.13: Similarità a sinistra, itemset a destra al variare dell'area spaziale coperta dalle celle s

determinandolo sperimentalmente. Nonostante questo, 1.75 risulta un buon valore: quasi il 50% degli itemset di SPARE sono coperti da CUTE e la similarità complessiva è dell'80%.

5.3.3 Interpretazione dei risultati

Anche utilizzando definizioni di similarità personalizzate sul problema, i risultati fra i due cluster sono comunque molto diversi fra di loro.

Sicuramente entrambi gli algoritmi formalizzano la ricerca di pattern di co-movimento con alcuni elementi in comune, tuttavia l'approccio alla ricerca e la natura stessa degli itemset individuati porta a escludere una casistica reale in cui gli algoritmi individuano esattamente gli stessi cluster. È però possibile tentare di massimizzare l'intersezione tra i risultati di questi due algoritmi, sempre tenendo presente che le cardinalità di CUTE sono di molto superiori a quelle di SPARE.

In teoria a parità di risultati sul clustering spaziale dovrebbe garantire quanto meno la totale sovrapposizione totale dei risultati di SPARE su quelli di CUTE. Questo perché il tempo viene processato nello stesso modo dai due algoritmi, mentre invece lo spazio no. Tuttavia trovare questo punto di intersezione è praticamente impossibile, SPARE valuta i cluster ad ogni istante temporale, definendo potenzialmente una configurazione spaziale diversa ad ogni istante. La divisione effettuata da CUTE invece è fissa in tutti gli istanti temporali.

Sia CUTE che SPARE dispongono poi di vincoli che l'altro non non è in grado di coprire. Questi devono essere necessariamente rilassati durante il confronto, per non far divergere ulteriormente i risultati. Ciò però implica che il potenziale espressivo di questi parametri sia perso nella ricerca.

In definitiva CUTE e SPARE pongono due approcci che possono convergere parzialmente nella ricerca di pattern di co-movimento. Allo stesso tempo tuttavia portano un contributo unico alla ricerca che viene minimizzato quando si vuole far convergere i risultati.

6 Conclusioni e sviluppi futuri

Il contributo fornito da questa tesi si spinge nell'analisi approfondita del clustering di traiettorie e del frequent itemset mining. Durante questa analisi è stata individuata una categoria a metà tra il clustering e il frequent trajectory mining: la ricerca di pattern di co-movimento. Sono stati indagati quindi i principali algoritmi di analisi di questi pattern. Tra tutti il framework SPARE è risultato particolarmente interessante, per via della sua implementazione distribuita. Inoltre SPARE formalizza il concetto e la ricerca di pattern di co-movimento generici (GCMP), pattern che tramite la configurazione di alcuni parametri possono esprimere i principali pattern di co-movimento. SPARE tuttavia risulta limitato nel momento in cui si vuole espandere la dimensionalità dei dati oppure si considerano vincoli di contiguità nello spazio.

In questa tesi è stato implementato il framework CUTE, algoritmo generico per la ricerca di gruppi di traiettorie su un insieme di dimensioni custom in ambito big data. Questo è stato adattato e configurato per la ricerca di co-movement pattern. CUTE è stato poi testato approfonditamente su vari dataset per valutare i suoi risultati e le sue performance. Traendo qualche conclusione dal lavoro svolto, è possibile dire che la ricerca di pattern di movimento è una categoria ampia e gli algoritmi presenti in letteratura faticano a coprila nel suo intero. Esistono infatti una serie di possibilità e definizioni di pattern che difficilmente sono totalmente compatibili tra di loro. CUTE lavora in questa direzione, tentando di coprire tutti i pattern di movimento presenti nella categoria. La chiave per questo sta nell'approccio di CUTE al problema della ricerca: in primo luogo la possibilità di esprimere un sistema di riferimento custom consente di modellare la ricerca di un pattern su un qualunque insieme di dimensioni, monotone o meno. Successivamente la natura generica dei pattern restituiti in output consente di ricercare più categorie sulla base della configurazione di parametri specificata. In conclusione CUTE è un framework decisamente valido per cercare di affrontare il problema della ricerca di pattern di co-movimento nella sua totalità.

CUTE però non si limita alla ricerca di pattern di co-movimento, ma può essere espanso alla ricerca di altre categorie di gruppi. Durante gli esperimento compiuti, non sono mai considerate dimensioni diverse dallo spazio e dal tempo. In futuro sarebbe sicuramente utile provare ad integrare altre dimensioni nella ricerca e vedere i risultati estratti dall'algoritmo. Un'altra possibile direzione riguarda l'aggiunta di ulteriori meccaniche di pruning per gli itemset individuati. Com'è possibile notare dai risultati, il numero di cluster per

parametri ragionevoli è comunque molto alto. Qualora sia necessaria una ricerca più fine, si potrebbe pensare ad esempio di introdurre una nuova metrica, la coesione, per valutare la bontà di un itemset. La coesione esprimerebbe il rapporto tra i movimenti di un gruppo e i movimenti dei singoli elementi che lo compongono. Cluster abbastanza coesi rappresenterebbero oggetti che hanno viaggiato assieme per buona parte del loro percorso singolo. Al contrario cluster con un basso valore di coesione sarebbero composti da oggetti che hanno viaggiato assieme una breve parte del loro percorso totale, di conseguenza sarebbero meno interessanti.

Ringraziamenti

A conclusione dei miei studi, che io vedo come un percorso unico dall'inizio del corso triennale fino a questo momento, è per me doveroso ringraziare chi mi è stato accanto e di supporto in questo percorso. Un primo ringraziamento va al mio relatore, il professor M.Golfarelli, per avermi dato l'opportunità di vedere più da vicino un mondo che durante i miei studi non avevo mai approcciato: quello della ricerca. Un altro ringraziamento importante va ai ragazzi del gruppo del BIG group: Anna, Nicola e il professor E.Gallinucci che con il loro supporto e competenza hanno saputo consigliarmi nei momenti di dubbio. Vorrei ringraziarli inoltre per il clima di accoglienza e amicizia che ho respirato in questi mesi di lavoro. Il ringraziamento più grande però va al mio co-relatore, il dottor M.Francia, che è stato per me costante supporto e punto di riferimento in tutti i momenti di questa tesi, nonostante i 16.000 KM e le 9 ore di fuso orario.

Un particolare ringraziamento alla mia famiglia, per il costante supporto e l'ascolto durante il mio periodo universitario. Un grande grazie va anche a Niccolò, Luca, Lorenzo, Riccardo, Jacopo, Nicholas e Gjulio, per aver condiviso la totalità del mio percorso universitario, nei mille progetti, nei momenti belli e soprattutto in quelli difficili. Per finire, mi sento di ringraziare tutti gli altri amici che sono stati assolutamente fondamentali e necessari per raggiungere questo traguardo finale. Grazie a tutti.

Federico Naldini

Bibliografia

[1] Z. Feng e Y. Zhu, «A Survey on Trajectory Data Mining: Techniques and Applications,» *IEEE Access*, vol. 4, pp. 2056–2067, 2016.

[2] Z. Zhang, K. Huang e T. Tan, «Comparison of similarity measures for trajectory clustering in outdoor surveillance scenes,» in *18th International Conference on Pattern Recognition (ICPR'06)*, IEEE, vol. 3, 2006, pp. 1135–1138.

[3] J. Chen, R. Wang, L. Liu e J. Song, «Clustering of trajectories based on Hausdorff distance,» in *2011 International Conference on Electronics, Communications and Control (ICECC)*, IEEE, 2011, pp. 1940–1944.

[4] C. Rick, «Efficient computation of all longest common subsequences,» in *Scandinavian Workshop on Algorithm Theory*, Springer, 2000, pp. 407–418.

[5] L. Chen, M. T. Özsu e V. Oria, «Robust and fast similarity search for moving object trajectories,» in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, 2005, pp. 491–502.

[6] V. Khoshaein, «Trajectory clustering using a variation of Fréchet distance,» *Citeseer*, 2013.

[7] T. W. Liao, «Clustering of time series data—a survey,» *Pattern recognition*, vol. 38, n. 11, pp. 1857–1874, 2005.

[8] G. Zazzaro, «Data Mining: esplorando le miniere alla ricerca della conoscenza nascosta. Clustering con l'algoritmo k-means,» vol. 9, pp. 37–45, mag. 2009.

[9] J. Han, J. Pei e M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.

[10] P. Arora, S. Varshney et al., «Analysis of k-means and k-medoids algorithm for big data,» *Procedia Computer Science*, vol. 78, pp. 507–512, 2016.

[11] *K-Means Data Clustering - Towards Data Science*, `https://towardsdatascience.com/k-means-data-clustering-bce3335d2203`, (Accessed on 02/21/2020).

[12] L. Kaufman e P. J. Rousseeuw, «Agglomerative nesting (program AGNES),» *Finding Groups in Data: An Introduction to Cluster Analysis*, pp. 199–252, 2008.

[13] L. Kaufman e P. Rousseeuw, «Divisive analysis (program diana),» *Finding Groups in Data*, pp. 253–279, 2008.

[14] M. Ester, H.-P. Kriegel, J. Sander, X. Xu et al., «A density-based algorithm for discovering clusters in large spatial databases with noise.,» in *Kdd*, vol. 96, 1996, pp. 226–231.

[15] M. Ankerst, M. M. Breunig, H.-P. Kriegel e J. Sander, «OPTICS: ordering points to identify the clustering structure,» *ACM Sigmod record*, vol. 28, n. 2, pp. 49–60, 1999.

[16] *DBSCAN: What is it? When to Use it? How to use it. - Evan Lutins - Medium*, `https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818`, (Accessed on 02/21/2020).

[17] W. Wang, J. Yang, R. Muntz et al., «STING: A statistical information grid approach to spatial data mining,» in *VLDB*, vol. 97, 1997, pp. 186–195.

[18] D. H. Fisher, «Knowledge acquisition via incremental conceptual clustering,» *Machine learning*, vol. 2, n. 2, pp. 139–172, 1987.

[19] A. T. Palma, V. Bogorny, B. Kuijpers e L. O. Alvares, «A clustering-based approach for discovering interesting places in trajectories,» in *Proceedings of the 2008 ACM symposium on Applied computing*, 2008, pp. 863–868.

[20] C.-C. Hung, W.-C. Peng e W.-C. Lee, «Clustering and aggregating clues of trajectories for mining trajectory patterns and routes,» *The VLDB Journal*, vol. 24, n. 2, pp. 169–192, 2015.

[21] M. Nanni e D. Pedreschi, «Time-focused clustering of trajectories of moving objects,» *Journal of Intelligent Information Systems*, vol. 27, n. 3, pp. 267–289, 2006.

[22] S. Mitsch, A. Müller, W. Retschitzegger, A. Salfinger e W. Schwinger, «A survey on clustering techniques for situation awareness,» in *Asia-Pacific Web Conference*, Springer, 2013, pp. 815–826.

[23] P. Tampakis, N. Pelekis, C. Doulkeridis e Y. Theodoridis, «Scalable Distributed Subtrajectory Clustering,» *arXiv preprint arXiv:1906.06956*, 2019.

[24] Y. S. Koh e S. D. Ravana, «Unsupervised Rare Pattern Mining: A Survey,» *TKDD*, vol. 10, n. 4, 45:1–45:29, 2016.

[25] Y. Huang, J. Pei e H. Xiong, «Mining Co-Location Patterns with Rare Events from Spatial Data Sets,» *GeoInformatica*, vol. 10, n. 3, pp. 239–260, 2006.

[26] Q. Fan, D. Zhang, H. Wu e K. Tan, «A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories,» *PVLDB*, vol. 10, n. 4, pp. 313–324, 2016.

[27] C.-H. Chee, J. Jaafar, A. Izzatdin, M. H. Hasan e W. Yeoh, «Algorithms for frequent itemset mining: a literature review,» *Artificial Intelligence Review*, mar. 2018. DOI: `10.1007/s10462-018-9629-z`.

[28] R. C. Agarwal, C. C. Aggarwal e V. Prasad, «A tree projection algorithm for generation of frequent item sets,» *Journal of parallel and Distributed Computing*, vol. 61, n. 3, pp. 350–371, 2001.

[29] *bias.csr.unibo.it/golfarelli/DataMining/MaterialeDidattico/2017/11-RegoleAssociative.pdf*, `http://bias.csr.unibo.it/golfarelli/DataMining/MaterialeDidattico/2017/11-RegoleAssociative.pdf`, (Accessed on 02/22/2020).

[30] F. Pan, G. Cong, A. K. Tung, J. Yang e M. J. Zaki, «Carpenter: Finding closed patterns in long biological datasets,» in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2003, pp. 637–642.

[31] H. Cao, N. Mamoulis e D. W. Cheung, «Mining frequent spatio-temporal sequential patterns,» in *Fifth IEEE International Conference on Data Mining (ICDM'05)*, IEEE, 2005, 8–pp.

[32] L. Chen, M. Lv, Q. Ye, G. Chen e J. Woodward, «A personal route prediction system based on trajectory data mining,» *Information Sciences*, vol. 181, n. 7, pp. 1264–1284, 2011.

[33] M. Lv, Y. Li, Z. Yuan e Q. Wang, «Route pattern mining from personal trajectory data,» *J. Inf. Sci. Eng.*, vol. 31, n. 1, pp. 147–164, 2015.

[34] M. Qiu e D. Pi, «Mining frequent trajectory patterns in road network based on similar trajectory,» in *International Conference on Intelligent Data Engineering and Automated Learning*, Springer, 2016, pp. 46–57.

[35] L. Zheng, D. Xia, X. Zhao, L. Tan, H. Li, L. Chen e W. Liu, «Spatial–temporal travel pattern mining using massive taxi trajectory data,» *Physica A: Statistical Mechanics and its Applications*, vol. 501, pp. 24–41, 2018.

[36] *Introduction To Apache Hadoop - Architecture, Ecosystem*, `https://intellipaat.com/blog/tutorial/hadoop-tutorial/introduction-hadoop/`, (Accessed on 02/25/2020).

[37] Y. Zheng, «Trajectory data mining: an overview,» *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 6, n. 3, pp. 1–41, 2015.

[38] Z. Li, B. Ding, J. Han e R. Kays, «Swarm: Mining relaxed temporal moving object clusters,» *Proceedings of the VLDB Endowment*, vol. 3, n. 1-2, pp. 723–734, 2010.

[39] N. Phan, P. Poncelet e M. Teisseire, «All in one: Mining multiple movement patterns,» *International Journal of Information Technology & Decision Making*, vol. 15, n. 05, pp. 1115–1156, 2016.

[40] N. Phan, P. Poncelet e M. Teisseire, «All in One: Mining Multiple Movement Patterns,» *International Journal of Information Technology and Decision Making*, vol. 15, n. 5, pp. 1115–1156, 2016.

[41] H. Jeung, H. T. Shen e X. Zhou, «Convoy queries in spatio-temporal databases,» in *2008 IEEE 24th International Conference on Data Engineering*, IEEE, 2008, pp. 1457–1459.

[42] M. Benkert, J. Gudmundsson, F. Hübner e T. Wolle, «Reporting flock patterns,» *Computational Geometry*, vol. 41, n. 3, pp. 111–125, 2008.

[43] Y. Wang, E.-P. Lim e S.-Y. Hwang, «Efficient mining of group patterns from user movement data,» *Data & Knowledge Engineering*, vol. 57, n. 3, pp. 240–282, 2006.

[44] Y. Li, J. Bailey e L. Kulik, «Efficient mining of platoon patterns in trajectory databases,» *Data & Knowledge Engineering*, vol. 100, pp. 167–187, 2015.

[45] F. Zhu, X. Yan, J. Han, S. Y. Philip e H. Cheng, «Mining colossal frequent patterns by core pattern fusion,» in *2007 IEEE 23rd international conference on Data Engineering*, IEEE, 2007, pp. 706–715.

[46] D. Apiletti, E. Baralis, T. Cerquitelli, P. Garza, F. Pulvirenti e P. Michiardi, «A Parallel MapReduce Algorithm to Efficiently Support Itemset Mining on High Dimensional Data,» *Big Data Research*, vol. 10, pp. 53–69, 2017.

[47] F. Pan, G. Cong, A. K. H. Tung, J. Yang e M. J. Zaki, «Carpenter: finding closed patterns in long biological datasets,» in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, 2003, pp. 637–642.

[48] M. K. Vanahalli e N. Patil, «An efficient parallel row enumerated algorithm for mining frequent colossal closed itemsets from high dimensional datasets,» *Information Sciences*, vol. 496, pp. 343–362, 2019.

[49] *GitHub - RoaringBitmap/RoaringBitmap: A better compressed bitset in Java*, `https://github.com/RoaringBitmap/RoaringBitmap`, (Accessed on 02/25/2020).

[50] *Geolife GPS trajectory dataset - User Guide - Microsoft Research*, `https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/`, (Accessed on 03/03/2020).

[51] D. G. McVitie e L. B. Wilson, «The stable marriage problem,» *Communications of the ACM*, vol. 14, n. 7, pp. 486–490, 1971.

[52] Z. Feng e Y. Zhu, «A survey on trajectory data mining: Techniques and applications,» *IEEE Access*, vol. 4, pp. 2056–2067, 2016.

[53] S. Tsumoto e S. Hirano, «Behavior grouping based on trajectory mining,» in *Social Computing and Behavioral Modeling*, Springer, 2009, pp. 1–8.

[54] Y. Yanagisawa, J.-i. Akahani e T. Satoh, «Shape-based similarity query for trajectory of mobile objects,» in *International Conference on Mobile Data Management*, Springer, 2003, pp. 63–77.

[55] Y. Yanagisawa e T. Satoh, «Clustering multidimensional trajectories based on shape and velocity,» in *22nd International Conference on Data Engineering Workshops (ICDEW'06)*, IEEE, 2006, pp. 12–12.

[56] J.-G. Lee, J. Han e K.-Y. Whang, «Trajectory clustering: a partition-and-group framework,» in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, 2007, pp. 593–604.

[57] Y. Zheng, Q. Li, Y. Chen, X. Xie e W.-Y. Ma, «Understanding mobility based on GPS data,» in *Proceedings of the 10th international conference on Ubiquitous computing*, 2008, pp. 312–321.

[58] J. J.-C. Ying, W.-C. Lee, T.-C. Weng e V. S. Tseng, «Semantic trajectory mining for location prediction,» in *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2011, pp. 34–43.

[59] *K-Means Clustering – What it is and How it Works – Learn by Marketing*, `http://www.learnbymarketing.com/methods/k-means-clustering/`, (Accessed on 02/21/2020).