Università degli Studi di Pisa

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE E NATURALI
Corso di Laurea Specialistica in Tecnologie Informatiche

Tesi di laurea specialistica

Indicizzazione di dati di movimento per il clustering basato su densità

Candidato:
Antonino Mistretta

Relatori:
Prof. Dino Pedreschi
Dr. Mirco Nanni

Controrelatore:
Prof. Stefano Chessa

Anno Accademico 2012-2013

Abstract

The technologies of mobile communications pervade our society and wireless networks sense the movement of people, generating large volumes of mobility data, such as mobile phone call records and Global Positioning System (GPS) tracks. The analytical power of massive collections of trajectory data may contribute to show the complexity of human mobility. In this thesis we propose a novel index-based method for range query over trajectory data, specifically tailored for a density-based clustering algorithm, that significantly improves performances over alternative indexing and querying strategies. Performance analysis of the range query method are provided, both stand alone and integrated in the clustering schema, with results on scalability, sensibility to parameters and behaviour over changing time windows.

Riassunto

Le tecnologie di comunicazione mobile pervadono la nostra società e le reti wireless rilevano il movimento di persone, generando grandi volumi di dati di mobilità, come ad esempio i record delle chiamate di telefonia mobile e le tracce Global Positioning System (GPS). Il potere analitico di queste grandi raccolte di dati può contribuire a mostrare la complessità della mobilità umana. In questa tesi si propone un nuovo metodo index-based per range query sui dati di traiettoria, studiato appositamente per un algoritmo di clustering basato su densità, che migliora sensibilmente le prestazioni di indexing e delle strategie di interrogazione. Sono state eseguite analisi delle prestazioni del metodo di range query, sia stand-alone sia integrata nello schema di clustering, con risultati su scalabilità, sensibilità ai parametri e comportamento nel cambiamento delle finestre temporali.

Capitolo 1

Introduzione

Negli ultimi anni il numero di dispositivi mobili, dotati di sensore GPS, è cresciuto notevolmente, e con esso anche la mole di informazioni georeferenziate. La raccolta di questi dati (relativi a posizioni di persone, animali o veicoli) produce un'informazione che descrive gli spostamenti dei soggetti in esame nell'intervallo di tempo considerato. La gestione e l'analisi dei dati di movimento ha prodotto da un lato un grande interesse nella comunità scientifica e dall'altro nuove problematiche legate alla modellazione e alle performance. L'interesse è una diretta conseguenza dell'importanza dei dati di movimento; infatti grazie ad essi, e alle tecnologie a loro supporto, è possibile studiare fenomeni che in passato erano di difficile comprensione. Un possibile campo di applicazione potrebbe essere l'analisi del traffico veicolare di un grosso centro urbano, al fine di migliorarne la viabilità.

In queste tesi affrontiamo il problema della gestione dei dati di movimento, focalizzando l'attenzione sui metodi di indicizzazione e sull'esecuzione di range query sui dati di traiettoria. In particolare, l'obiettivo della tesi è quello di proporre un metodo di indicizzazione alternativo, appositamente studiato per l'algoritmo di clustering Trajectory-Optics presentato in [1]. Trajectory-OPTICS è un algoritmo di clustering basato su densità, e su un semplice concetto di distanza tra traiettorie. Sempre in [1] viene presentata una variante, Time Focused-OPTICS, appositamente pensata per ritrovare un intervallo di tempo ottimale con l'obiettivo di migliorare la qualità del clustering di traiettorie. La ricerca dell'intervallo ottimale avviene con un algoritmo greedy di ricerca locale che, ad ogni iterazione, prova a migliorare il risultato ottenuto dalla precedente spostando l'intervallo temporale ed eseguendo l'algoritmo Trajectory-OPTICS sulla porzione di traiettorie appartenenti a quell'intervallo. A questo scopo ho analizzato il problema dell'indicizzazione di traiettorie, identificando una famiglia di indici da utilizzare e implementando un algoritmo di ricerca chiamato SimilarityRangeSearch.

L'indice individuato è un evoluzione spazio-temporale di un indice spaziale largamente utilizzato in ambito GIS e CAD: l'indice R-Tree ideato da Gutmann in [4]. L'algoritmo di ricerca è stato pensato per utilizare alcune tecniche di pruning definite in [2] e per supportare efficientemente interrogazioni su una porzione temporale del dataset. La prima fase implementativa è stata seguita da una serie di test e bug fixing. Successivamente sono stati eseguiti una serie di test per verificare il comportamento dell'algoritmo al variare di alcuni parametri. Infine sono stati eseguiti ulteriori test per verificare l'effettivo miglioramente/peggioramento delle prestazioni dell'algoritmo. Dai risultati ottenuti, è emerso un miglioramento delle prestazioni dell'algorimto Time Focused-OPTICS, giustificato soprattutto dal fatto che lo spostamento della finestra temporale non richiede il re-indexing delle traiettorie.

1.1 Organizzazione tesi

La tesi è strutturata su tre diversi capitoli. Nel capitolo Contesto e Stato dell'Arte (capitolo 2) verranno descritti i dati di mobilità e analizzate le tecnologie che permetto la lora raccolta. Successivamente verranno esposti i concetti di traiettoria e trajectory database, approfondendo gli aspetti legati alla modellazione e all'indicizzazione di traiettorie al fine di supportare una tipologia particolare di interrogazioni spazio-temporali, le range query. In seguito, parleremo di Ddata Mining e di Knowledge Discovery in Database applicato ai dati di movimento e noto come Geographic Knowledge Discovery. Concluderemo il capitolo dello stato dell'arte descrivendo alcuni dettagli delle tecniche di clustering density based applicati alle traiettorie e dando una descrizione dell'algoritmo Trajectory-OPTICS e della variante Time Focused.

Successivamente verrà descritto in dettaglio il lavoro svolto, che consiste nell'implementazione di un indice spaziale basato su R-Tree, per il supporto efficiente di query trajectory. In particolare, affronteremo il problema di ritrovare l'insieme di traiettorie che hannno una dissimilitudine, rispetto ad una query trajectory data, minore di un dato range. Verranno descritti l'algoritmo di ricerca e le tecniche di ottimizzazione applicate.
Infine, nel paragrafo 4.2 verranno esposti i test eseguiti e i risultati ottenuti.

Capitolo 2

Contesto e Stato dell'arte

In questa capitolo esporremo il concetto di moving object e mobility data (Paragrafo 2.1), descrivendo le tecnologie che hanno permesso la loro diffusione.
Successivamente verrà introdotto il concetto di traiettoria, e di Trajectory Database (Paragrafo 2.2), estensione naturale dei tradizionali database, focalizzando l'attenzione principalmente su: strutture dati utilizzate per gestire i dati spazio-temporali (Paragrafo 2.3), e i tipi di interrogazione a cui questi sistemi possono essere sottoposti. Infine parleremo di data mining, knowledge discovery in database e l'applicazione di queste tecniche ai dati spazio-temporali, noto come Geographic Knowledge Discovery (GKDD).

Da sempre, l'analisi di fenomeni relativi ad insiemi di oggetti che si muovono nello spazio (in letteratura Moving Object [10]), ha catturato l'interesse di molti studiosi. Alcuni esempi possono essere:

- analisi dei flussi migratori (animali, persone);
- studio e ottimizzazione del traffico stradale (persone, mezzi di trasporto);
- individuazione di modelli comportamentali (persone, animali).

In passato, a causa della difficoltà nel reperire queste informazioni, non è sempre stato possibile analizzare facilmente questi fenomeni. Negli ultimi 10/15 anni, abbiamo assistito a una diffusione massiccia di dispositivi mobili in grado di rilevare la posizione rispetto ad un sistema di riferimento, tramite l'utilizzo di tecnologie come: GPS (satellite), GSM, e altre tecnologie wireless [1, p. 74]. Grazie a questo, la disponibilità di dati di mobilità è notevolmente aumentata, e, assieme ad essi, l'interesse della comunità scientifica. Nel paragrafo successivo verranno descritte brevemente alcune di queste tecnolgie,

concentrandoci soprattutto sui dati forniti.
In seguito introdurremo il concetto di Trajectory Database approfondendo alcuni aspetti progettuali e realizzativi:

- in data model (Paragrafo 2.2.1) vedremo come questi dati sono stati modellati;
- in query language (Paragrafo 2.2.2) verranno descritti i tipi diversi di interrogazioni che possono essere fatte su dati spazio-temporali;
- in indexing (Paragrafo 2.3) verrannno mostrate le strutture dati utilizzate per immagazzinare e, successivamente, ritrovare queste informazioni.

Concluderemo il capitolo parlando del processo Knowledge Discovery in Database e dell'evoluzione dovuta all'impatto dei dati geografici denominato Geographic Knowledge Discoverey [11] (Paragrafo 2.4).

2.1 Tecnologie di localizzazione e Dati di mobilità

Con il termine mobility data identifichiamo i dati che tracciano il movimento di persone e veicoli nel tempo. Come già anticipato, la provenienza di questi dati è da attribuirsi principalmente alla diffusione di dispositivi mobili, dotati di tecnologie che supportano la geolocalizzazione.

Una di queste tecnologie è la rete cellulare (GSM/UMTS), che permette la localizzazione di un dispositivo connesso con livelli di accuratezza differenti in base al metodo usato [1]. La rete cellulare è costituita da un insieme di antenne disseminate sul territorio. Ciascuna antenna definisce una sua area di copertura detta cella, che può sovrapporsi ad altre celle. Il sistema, per scopi di routing, individua la presenza di un dispositivo all'interno di una cella, e mantiene aggiornata quest'informazione in apposite banche dati (Visit/Home Location Register). Da queste informazioni, con l'utilizzo di vari metodi (Cell Identify, E-ODT, A-GPS) è possibile calcolare un'approssimazione della posizione del dispositivo. Il livello di accuratezza varia in base al metodo utilizzato.

Un'altra fonte di mobility data sono le tecnologie di geolocalizzazione di veicoli basata su satellite. In [6], vengono individuate due famiglie di servizi di localizzazione (Location Based Services): terminanti nel dispositivo e originate dal dispositivo. Appartengono alla prima famiglia tutti i servizi i

cui dati vengono ottenuti e mantenuti dal dispositivo. Alla seconda famiglia appartengono quei servizi che per ragioni di sicurezza eseguono attività di monitoraggio e salvataggio. In questo caso i dati sono mantenuti dal centro di sorveglianza. I più comuni, appartengono alla prima famiglia che comprende tutti i dispositivi dotati di un sensore GPS. Essi utilizzano i segnali ricevuti da almeno 3 satelliti per determinare la posizione corrente, con un'accurattezza proporzionale al numero di satelliti contattati.

In entrambi i casi, si riesce ad ottenere dei dati che in generale avranno le seguenti informazioni: istante di tempo, posizione espressa in un sistema di riferimento geografico e l'indentificativo del device. Quindi, tracciare il movimento di un dispositivo, consiste nel collezionare queste informazioni nel tempo, ottenendo così la sua traiettoria.

2.2 Trajectory Database

I trajectory database, sono un'estensione dei tradizionali database. Essi, infatti, sono stati pensati per gestire un nuovo tipo di dato, caratterizzato dall'unione di due informazioni: posizione (rispetto ad un sistema di riferimento) e tempo. In passato la ricerca ha affrontato separatamente le informazioni spazio-temporali. La ricerca sui database spaziali si è concentrata nel supportare interrogazioni per ottenere oggetti associati ad una data geometria (per esempio: ritrovare tutte le entry che intersecano una data geometria). I trajectory database possono essere visti come dei database di oggetti in movimento (moving object).
Possiamo vedere la traiettoria T, tracciata da un oggetto in movimento p, come una funzione che mappa ad ogni istante di tempo t_i, la posizione p_i assunta dall'oggetto in quell'istante di tempo. Per questioni pratiche, vedremo rappresentate le traiettorie come una sequenza discreta di punti nello spazio al variare del tempo t. Quindi:

$$T = \{P_1, ..., P_i, ..., P_n\} \text{ dove } P_i = (p_i, t_i) \quad \forall i \in 1, ..., n \tag{2.1}$$

dove: P_i è una ennupla contenente sia le dimensioni spaziali che quella temporale.

Un trajectory database, è un sistema ottimizzato per immagazzinare molteplici traiettorie di oggetti, e per ritrovarli. Questi sistemi vengono utilizzati per supportare i processi di analisi storica e/o predittiva riguardanti vari fenomeni, come ad esempio lo studio della congestione del traffico stradale o l'analisi dei movimenti migratori di animali.
Nelle sezioni successive descriveremo alcuni dettagli di questi sistemi. Infatti

vedremo: come queste traiettorie sono state modellate, il tipo di query che riescono a processare, e le strutture dati utilizzate.

2.2.1 Data Models

In questo paragrafo, definiamo il concetto di traiettoria, e esponiamo lo stato dell'arte della modellazione di traiettorie. Il movimento di un oggetto (es.: automobile, persona, ...), visto come un punto materiale che si muove nello spazio, descrive una curva che è detta traiettoria. Possiamo vedere la traiettoria T, tracciata da un oggetto in movimento p, come una funzione che mappa ad ogni istante di tempo t_i, la posizione p_i assunta dall'oggetto in quell'istante di tempo. Nella modellazione di un insieme di oggetti

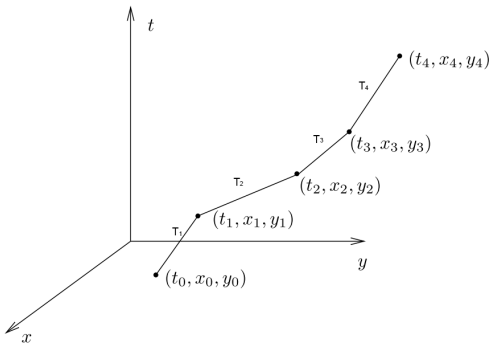


Figura 2.1: Esempio di Traiettoria: due dimensioni spaziali + dimensione temporale

in movimento, scelto un sistema di riferimento, la traiettoria di un oggetto può essere vista, come già anticipato nel paragrafo 2.2, come un curva, o, in alternativa, come una polilyne nello spazio-tempo (Figura 2.1). Nel primo caso, la traiettoria è costituita da un insieme infinito di punti, mentre nel secondo caso la traiettoria è costituita da un insieme finito di segmenti che approssimano la curva tracciata dal movimento dell'oggetto.

$$T = \{T_1, ..., T_i, ..., T_n\} \text{ dove } T_i = (P_i, P_{i+1}) \quad \forall i \in 1, ..., n-1 \tag{2.2}$$

dove T_i è il segmento che unisce due posizioni temporalmente consecutive dell'oggetto.
L'accuratezza della traiettoria, dipende dal numero di punti della traiettoria, dalle tecnologie di geolocalizzazione e dai metodi usati per ricostruire la traiettoria in caso di campionamenti mancanti. Per quest'ultimo problema

possono essere utilizzate tecniche che vanno dalla semplice interpolazione all'utilizzo di funzioni spline [10].
La modellazione di dati relativi a oggetti in movimento può essere suddivisa in due macro-aree: modellazione per l'analisi di dati storici, e modellazione per l'analisi in tempo reale e predittiva. Qui ci concentremo sulla prima: i modelli per dati storici. Il vantaggio di questo approccio è che i dati sono noti a priori, e in alcuni casi, come vedremo nel paragrafo 2.3.2, ci permette di applicare alcune ottimizzazioni.
Nel secondo approccio, uno dei principali problemi da affrontare, è quello di mantenere aggiornati i dati in tempo reale. Infatti aggiornamenti frequenti, soprattutto per un numero elevato di elementi, possono risultare critici per il sistema; dall'altro lato, se la frequenza degli aggiornamenti è bassa, gli errori relativi alla posizione attuale degli oggetti possono essere grandi.

2.2.2 Query language

Abbiamo visto che un trajectory database è un sistema in grado di immagazzinare le caratteristiche spaziali e temporali degli oggetti del dataset. L'inclusione del tempo, in combinazione con i predicati spaziali, aumenta il numero di tipologie di interrogazioni che possono essere poste al sistema. Le interrogazioni spazio-temporali possono riguardare un istante di tempo o un intervallo di tempo. Alcuni esempi sono:

- Range query: utilizzate per ritrovare tutti gli oggetti che intersecano un'area in un dato intervallo (o istante) di tempo;
- K-nearest neighbor query: per ritrovare i k elementi più vicini a un dato oggetto in movimento in un dato istante/intervallo di tempo;
- Query basate su traiettorie: per ritrovare la traiettoria di un dato oggetto dato un intervallo temporale;
- K-closest objects: per ritrovare i K oggetti più vicini a un dato punto in un determinato istante;
- Similarity query: utilizzati per ottenere le traiettorie simili ad una data traiettoria.

Le query spazio-temporali, coinvolgono una parte o l'intera traiettoria dell'oggetto ritrovato, e spesso non possono essere decomposte come una combinazione di query perchè l'esecuzione necessita di conoscere l'evoluzione dell'oggetto nel tempo [7] (es.: ingresso in un'area in un determinato intervallo temporale). La figura 2.2 mostra alcuni esempi di un'altra classe di query

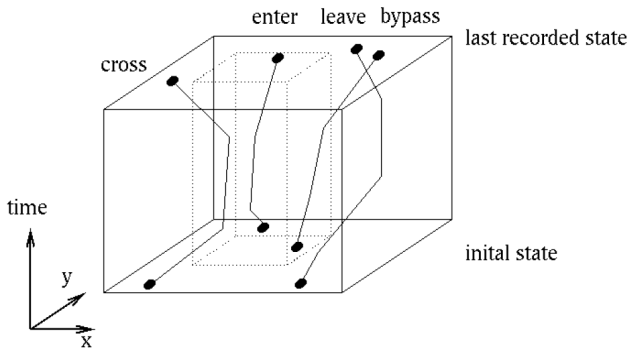


Figura 2.2: Query spazio-temporali topologiche [7]

definite in letteratura come Topological Query. L'obiettivo di queste interrogazione è quello di investigare l'evoluzione delle relazioni topologiche di oggetti nello spazio al variare del tempo. Alcuni esempi di predicati sono:

- cross: utilizzato per ottenere l'insieme di traiettorie che attraversano una determinata area dello spazio in un dato intervallo di tempo;
- enter: utilizzato per ottenere l'insieme di traiettorie che entrano in una determinata area dello spazio in un dato intervallo di tempo;
- leave: utilizzato per ottenere l'insieme di traiettorie che lasciano una determinata area dello spazio in un dato intervallo di tempo.

Le interrogazioni qui esposte sono stati utilizzate per analisi di data mining nel dominio dei dati spazio-temporali.

2.3 Trajectory Indexing

Come nei tradizionali database, l'esecuzione di query può essere un'operazione molto costosa. Gli indici tradizionali come i B-Tree non sono utili con i dati multidimensionali. Un gran numero di strutture dati sono state proposte per i dati spazio-temporali, molte delle quali varianti o estensioni dello R-Tree. Nel dominio dei database spaziali, lo R-Tree proposto da Gutmann è il più utilizzato (es.: GIS, CAD).
Lo R-tree è una struttura dati gerarchica e dinamica, ideata da Gutmann in [4], per gestire e ritrovare in modo efficiente un insieme di dati geometrici multidimensionali. Può essere considerato l'estensione n-dimensionale del B-Tree, da cui eredita molte caratteristiche. L'indice R-Tree, é un albero bilanciato in altezza, che memorizza nei nodi foglia i riferimenti al record

indicizzato. Il numero di elementi N, contenuti in ciascuna foglia, varia da m a M, dove: M è il numero massimo di elementi, e m è un parametro che è solitamente settato a $M/2$ per garantire che lo spazio utilizzato sia almeno del 50% ([6])

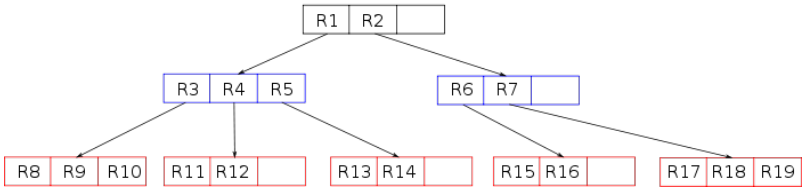


Figura 2.3: Esempio di R-tree su due dimensioni: struttura dell'indice

Lo R-Tree è una struttara dati dinamica in quanto, le operazioni d'inserzione e di eliminazione possono essere eseguite in qualsiasi momento e senza la necessitá di effettuare ri-organizzazioni periodiche.
Ogni nodo foglia contiene entrate della forma: (R, ID), dove ID è l'identificativo univoco dell'oggetto indicizzato, e R è il minimo boundy box (MBR) che lo contiene.
I nodi intermedi contengono entrate della seguente forma: $(R, child_i)$ dove $child_i$ è il riferimento ad un nodo figlio, e R è il minimo boundy box che spazialmente comprende tutti gli MBR dei nodi del livello inferiore. Diversamente dal B-Tree, i boundy box dei nodi che stanno sullo stesso livello, possono sovrapporsi (vedi figura 2.3).

In figura 2.4 è mostrata un esempio di suddivisione dello spazio di indicizzazione in un indice R-Tree. Ogni rettangolo dell'immagine rappresenta il minumo boundy box di un nodo dell'albero in figura 2.3. Da questa immagine, emerge quanto detto prima: l'envelope di ciascun nodo può sovrapporsi all'envolope di nodi allo stesso livello.

Per l'indicizzazione di traiettorie, descritte da oggetti che si muovono nello spazio, si è dimostrata una buona scelta quella dello R-Tree a 3 dimensioni: 2 spaziali, e una temporale. Infatti, nel corso degli anni, sono state proposte tantissime varianti dell'originale R-Tree, per l'indexing di dati spazio-temporali. Di seguito ne vedremo alcune: FNR-Tree, 3D R-Tree, Packed 3D R-Tree.

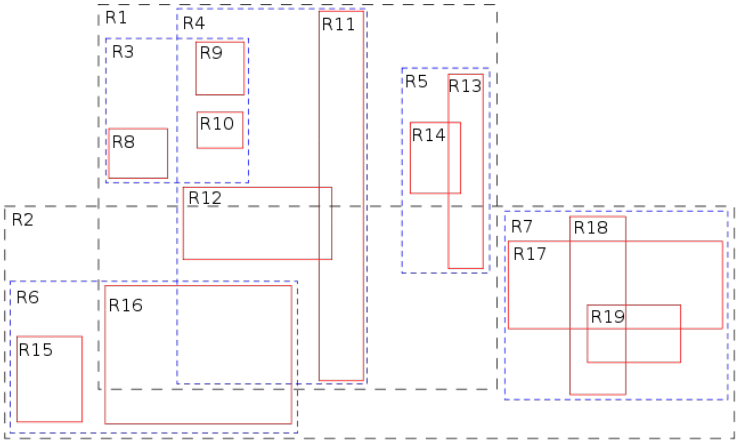


Figura 2.4: Esempio di R-tree su due dimensioni: MBR dei nodi

2.3.1 3D R-tree

IL 3D R-Tree (Figura 2.5) è un'estensione dello R-tree descritto nella sezione precedente. Infatti, il 3D-R-Tree considera oltre alle dimensioni spaziali dell'oggetto, anche la componente temporale, trattandola esattamente come le altre 2 dimensioni. I principali vantaggi nell'utilizzo di questo schema di indicizzazione sono:

- una sola struttura dati da gestire, come vedremo alcuni approcci consistono nell'utilizzo di più indici in quanto gestiscono la parte spaziale separatamente da quella temporale;
- supporto efficiente di operatori spazio-temporali.

Questa struttura dati, e le varianti descritte nei paragrafi successivi, sono state pensate per indicizzare segmenti di traiettorie. Come visto nel paragrafo 2.2.1, un segmento T_i è definito da una coppia di punti con timestamp successivi: $T_i = [(x_i, y_i, t_i), (x_{i+1}, y_{i+1}, t_{i+1})]$.
Analogalmente all'indice R-Tree, ogni nodo foglia conterrà un numero variabile di entry della forma: (R_{T_i}, ID_T), dove R_{T_i} è l'envelope spazio-temporale del segmento T_i e ID_T è l'identificativo della traiettoria T a cui il segmento appartiene.

2.3.2 Packed 3D R-tree

Questa variante differisce dalla precedente esclusivamente nel metodo di inserzione (vedi Figura 2.6). Può essere applicata soltanto nel caso di dataset

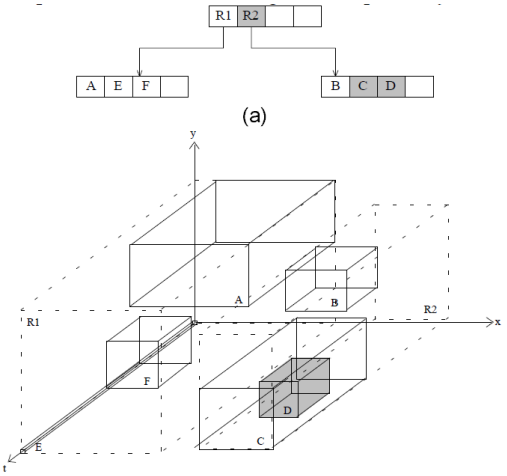


Figura 2.5: Esempio di 3D R-tree: spazio-temporal indexing

Algorithm HilbertPacking(TypeSetOfRectangles $\mathcal{RS}$)
/* Generates a packed R-tree from a set of data rectangles $\mathcal{RS}$ */

1. **foreach** data rectangle $r \in \mathcal{R}$
2. Calculate the Hilbert value of the centroid of r
3. **endfor**
4. Sort the data rectangles based on the Hilbert values
5. **while** there are still unassigned rectangles
6. Create a new leaf node
7. Assign the next c (f) rectangles to the new leaf (internal) node
8. **endwhile**
9. **while** there exist more than one nodes at level l
10. Sort nodes in ascending creation time and goto step 5
11. **endwhile**

Figura 2.6: Pseudo codice dell'algoritmo di creazione del Packed R-Tree

statici, dove per statici si intende che tutte le traiettorie sono note a priori. Questo perchè, prima di effettuare l'inserzione, i centroidi degli envelope dei segmenti di tutte le traiettorie, verranno ordinati in base al valore di Hilbert [9]. A questo punto, la creazione dell'albero avviene con un approccio bottom-up. Inizialmente vengono create le foglie dell'indice, inserendo i segmenti nell'ordine trovato al passo precedente. Successivamente si iniziano a creare i livelli sovrastanti, assegnando ai nuovi nodi le foglie, nel primo passo, e i nodi intermedi in quelli successivi. L'algoritmo d'inserzione termina quando viene raggiunto il livello in cui è presente un solo nodo. Ad ogni passo, i nodi vengono sempre scelti in ordine di creazione. Quest'approccio ha come vantaggio quello di ottenere una percentuale di utilizzo quasi del 100% [7]. Infatti, quasi tutti i nodi di ciascun livello dell'albero sono saturi. Questo in alcuni casi porta ad un miglioramento delle prestazioni.

2.3.3 FNR-Tree

Un approccio alternativo, è quello di indicizzare traiettorie in una rete fissa. La 'Fixed Network R-Tree' (FNR-Tree) è stata pensata per indicizzare oggetti che si muovono su un grafo fisso bidimensionale, utilizza un R-Tree bidimensionale per indicizzare i dati spaziali della rete; ciascuna foglia di quest'albero punta ad un suo R-Tree monodimensionale. Ciascuno di questi R-tree indicizza gli intervalli di tempo relativi al movimento dell'oggetto all'interno del collegamento del grafo associato.

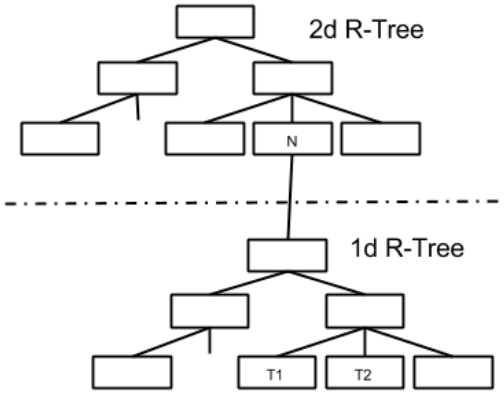


Figura 2.7: Esempio di FNR-tree

Nella figura 2.7 è mostrato un esempio della struttura di un FNR-Tree. Dall'immagine emerge il concetto di doppia indicizzazione: nella parte superiore il primo R-Tree si occupa delle dimensioni spaziali, mentre l'indice

nella parte inferiore dell'immagine gestisce la parte temporale.

2.4 Data Mining e KDD

Il processo di Knowledge Discovery in Database è un processo iterativo, che permette di convertire dati grezzi in conoscenza. Questo processo è costituito da una serie di passi di trasformazione che mostriamo nella Figura 2.9. I più importanti sono:

- preprocessing, in questo step i dati sorgenti vengono trasformati in modo da poter essere utilizzati nei successivi passi
- data mining, consiste nel trasformare i dati in modelli o pattern: classificazione, clustering, ...
- post processing, consiste nell'analisi/interpretazione dei modelli ottenuti al passo precedente.

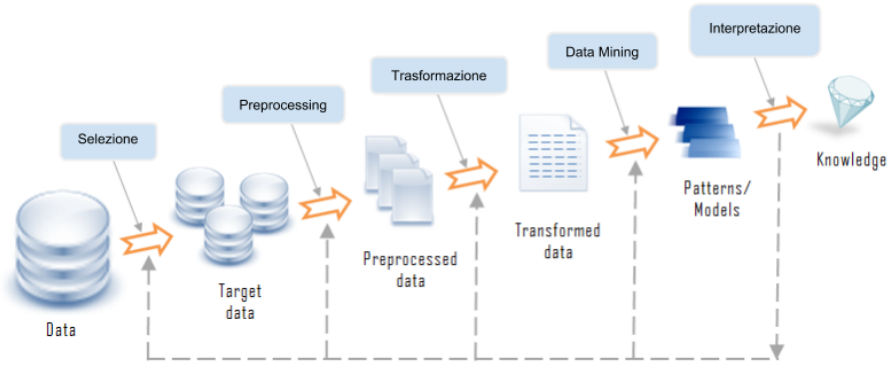


Figura 2.8: KDD process

Il nucleo del processo di Knowledge Discovery in Database è rappresentato dalle tecniche di data mining per l'estrazione di pattern dai dati. L'estrazione di dati (data mining) è il processo che automaticamente ritrova informazioni utili in grosse banche dati. Spesso, gli strumenti tradizionali di analisi non possono essere utilizzati a causa della grossa mole di dati (alcuni esempi possono essere i dati satellitari o del genoma umano). Una prima distinzione tra le diverse tecniche di data mining avviene tra approcci diretti (Directed Knowledge Discovery) e approcci indiretti (Undirected Knowledge Discovery).

L'approccio diretto è utilizzato nel caso in cui si intenda costruire un modello predittivo per un determinato sottoinsieme di variabili del dataset. I metodi utilizzati per questo approccio consistono nella selezione di alcuni attributi che si intende predire in base ai restanti, e nell'applicazione di algoritmi al fine di predire o classificare nuove istanze. Gli algoritmi utilizzati per questo scopo sfruttano principalmente dei metodi basati su alberi decisionali e regole associative.

Nell'approccio indiretto non abbiamo variabili target, ma vogliamo trovare relazioni inaspettate nei dati che potrebbero risultare interessanti. A tale scopo vengono utilizzate regole associative e algoritmi di clustering. Gli algoritmi di clustering hanno l'obiettivo di partizionare le istanze del dataset in diversi insiemi (cluster) cercando di massimizzare le similitudini tra le istanze appartenenti ad uno stesso cluster, e minimizzare quelle tra cluster diversi. Questi algoritmi sono quindi basati su una funzione che indica il grado di similitudine, ed è diversa a seconda degli algoritmi e del tipo di dato (numerico, alfanumerico ...). Un'esempio di queste funzione è la distanza Euclidea utilizzata per gli attributi numerici. Nel paragrafo 2.4.2 vedremo una rivisitazione di quest'approccio pensata per eseguire clustering di traiettorie basato su densità.

2.4.1 Geographic Knowledge Discovery

Il processo appena descritto, può anche essere applicato ai mobility data, in questo caso parleremo di Geographic Knowledge Discovery (GKDD). Definito in [6] come il processo che, da dati grezzi di mobilità, estrae conoscenza utilizzabile in processi decisionali. In questo contesto, i passi descritti precedentemente diventano:

- *ricostruzione delle traiettorie:* fase di trasformazione dei data mobility raw al fine di ottenere un database di traiettorie. I problemi da affrontare in questa fase riguardano la qualità del dato. Come già detto, la posizione è spesso un'approssimazione che introduce un errore di misurazione dipendente dal device utilizzato. Per esempio, con un GPS, viene introdotto un errore di misurazione di qualche metro, mentre l'imprecisione introdotta con la rete GSM/UMTS è la dimensione di una cella e questa varia da meno di cento metri in ambienti urbani a qualche chilometro in ambienti rurali. Il sampling rate, l'intervallo tra due misurazioni consecutive è la seconda sorgente di errore.

- *estrazione della conoscenza:* fase di estrazione di pattern utili dai data mobility attraverso tecniche di data-mining spazio temporali. Vengono riportati alcuni esempi di applicazioni:

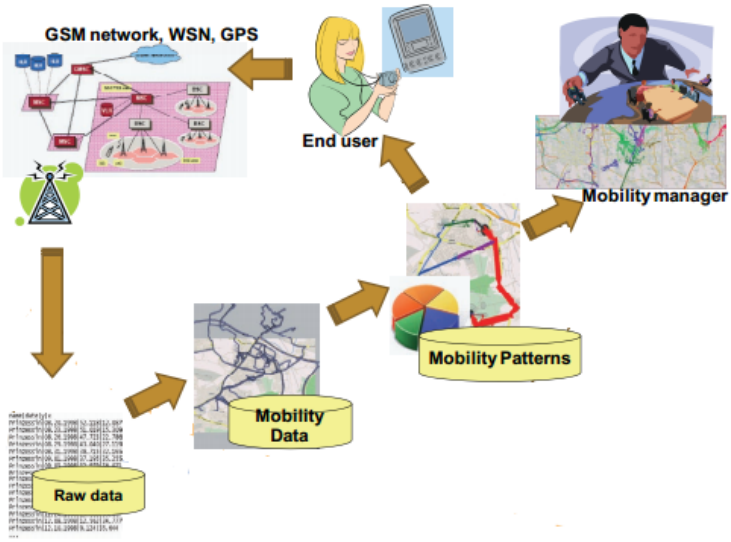


Figura 2.9: Geographic Knowledge Discovery process

– Clustering: la scoperta di gruppi di traiettorie simili (Figura 2.10). Conoscere quali siano le strade maggiormente trafficate può rappresentare un'informazione utile per l'analisi di mobilità

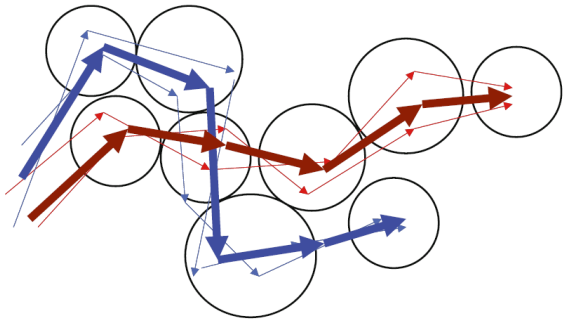


Figura 2.10: Trajectory clustering [6].

– Pattern frequenti: la conoscenza dei pattern frequenti può essere un'informazione utile nella pianificazione urbana (Figura 2.11).
– Classificazione: la conoscenza di regole nel comportamento permette di capire il comportamento degli utenti attuali e di predire quello di utenti futuri (Figura 2.12).

- *presentazione della conoscenza:* i pattern ottenuti al passo precedente, non sempre sono di facile comprensione; in questi casi è necessario

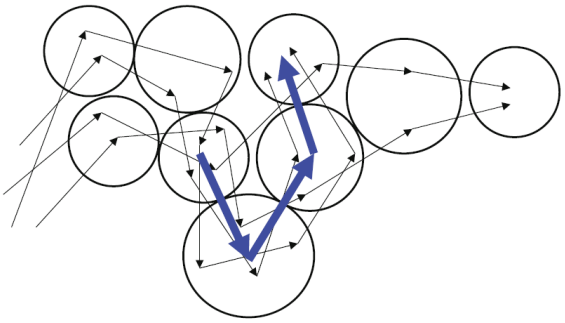


Figura 2.11: Trajectory pattern [6].

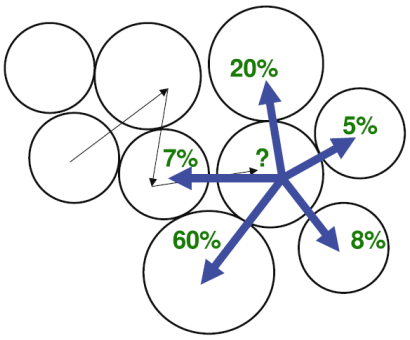


Figura 2.12: Trajectory prediction [6].

un'attenta valutazione da parte di esperti del dominio e successivamente preparare un modo appropriato per presentarla.

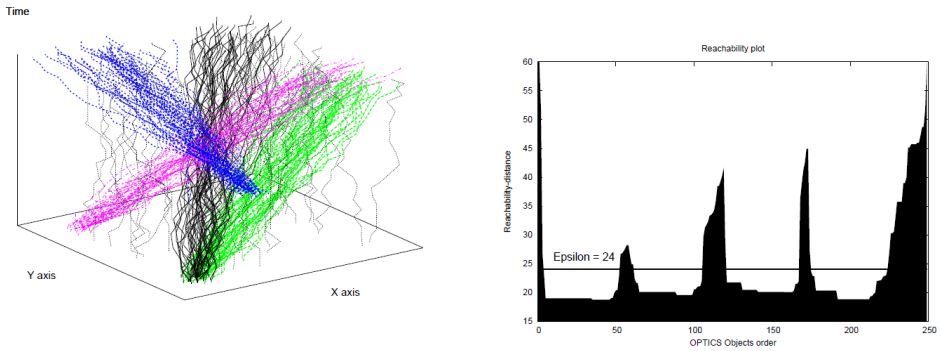
2.4.2 Density based Clustering

L'idea di base degli algoritmi di clustering basati sulla densità è che ogni oggetto per appartenere ad un cluster deve avere, nel suo vicinato di raggio ϵ, almeno n_{pts} oggetti. Visto il vincolo sul minimo numero di vicini, questi algoritmi si sono rivelati molto robusti a possibili errori nei dati, caratteristica spesso presente nei dataset reali.
Alcuni algoritmi di clustering come quelli gerarchici e il k-means, si sono rivelati inadeguati a causa della forma sferica dei cluster ritrovati. Infatti questa caratteristica esclude la possibilità di ritrovare veicoli che si muovono lungo una strada, in quanto formerebbero dei cluster non sferici. Gli algoritmi density-based della famiglia di DBSCAN sono dei buon candidati, e possono essere applicati a dati complessi come i dataset di traiettorie. In [1], viene presentata un'evoluzione del DBSCAN: Trajectory-OPTICS. L'approccio di

quest'algoritmo consiste nel trattare le traiettorie come un unico, indivisibile elemento, e nel raggruppare tra loro tutti gli oggetti che si muovono globalmente in maniera simile.
In pratica, le traiettorie vengono confrontate e clusterizzate basandosi sulla distanza spazio-temporale definita come segue:

$$D_T(\tau_1, \tau_2) = \frac{\int_T d(\tau_1(t), \tau_2(t))dt}{\Delta T} \tag{2.3}$$

dove d è la distaza euclidea, T è l'intervallo temporale in cui le traiettorie τ_1 e τ_2 sono definite, e $\tau_{1/2}$ rappresentano le posizioni dei rispettivi oggetti all'istante t. L'algoritmo si appoggia su un M-Tree [13], un indice generico per metriche spaziali pensato per supportare efficentemente range query e non appositamente orientato alle traiettorie. Nella figura 2.13 mostriamo



(a) Input dataset (b) OPTICS Reach. plot

Figura 2.13: Un dataset sintetico (a) e un reachability plot (b) [1].

un esempio di dataset sintetico di traiettorie (a), e sulla destra (b) l'output dell'algoritmo Trajectory-OPTICS ad esso applicato. Il reachability plot mostra la struttura dei cluster in modo indipende dai dati in input, e può anche essere utilizzato per assegnare gli oggetti ai cluster a cui sono stati assegnati. Il reachability plot mostra sull'ascissa gli oggetti del dataset nell'ordine in cui sono stati visitati (cluster-reordering); sull'ordinata la rechability distance, che rappresenta la minima distanza tra l'oggetto e i suoi predecessori p_j con $0 \leq j < i$. In particolare, in figura 2.13 (b) abbiamo 4 zone ad alta densita' (bassa reachability distance) interrotte da zone a bassa densita' (picchi di alta reachability distance). Inoltre affrontano e propongono una soluzione per la ricerca di un intervallo di tempo ottimale, con una variante del precedente chiamato Time Focused-Optics. Le motivazioni di questa scelta sono una conseguenza del fatto che la struttura dei cluster dei moving object è spesso più significativa in un sottoinsieme dell'intervallo temporale rispetto

che considerare l'intera traiettoria. Per l'individuazione di questi intervalli di tempo è stato utilizzato un algoritmo di ricerca locale con un approccio hill climbing: ad ogni iterazione la finestra temporale è modificata nel tentativo di migliorare una data funzione che misura la qualità.
Nella figura 2.14 mostriamo: sulla sinistra l'intervallo temporale determinato $[30, 72]$ da TF-OPTICS; sulla destra in evidenza le parti di traiettorie che riguardano l'intervallo temporale individuato. L'intervallo temporale viene determinato con l'ausilio di una funzione di qualità (Q_2 nella figura 2.14) che a parità di score predilige intervalli temporali più ampi.
Le performace dell'algoritmo sono legate al costo computazionale del calcolo

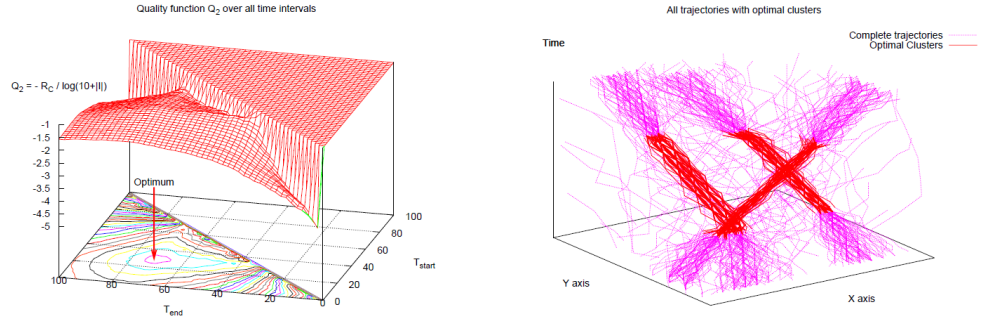


Figura 2.14: Risultato dell'algoritmo Time Focused-OPTICS e il dataset in input. [1].

della distanze tra traiettorie; nel capitolo successivo affronteremo questo problema e cercheremo di migliorare le performance dell'algoritmo TF-OPTICS.

Capitolo 3

Un algoritmo di ricerca per range query su traiettorie

In questo capitolo proponiamo un nuovo metodo index-based per range query sui dati di traiettoria, pensato appositamente per l'algoritmo Trajectory-Optics, un algoritmo di clustering basato su densità. Nel paragrafo 3.1 verrano esposte le motivazione e gli obiettivi. Successivamente (paragrafo 3.2) verranno analizzate le metriche utilizzate dall'algoritmo nella visita dell'albero e in particolare per eseguire il pruning dello spazio di ricerca. Chiuderemo il capitolo con una descrizione dettagliata dell'algoritmo di ricerca (paragrafo 3.3) e con qualche dettaglio implementativo (3.4).

3.1 Obiettivi

L'obiettivo di questa tesi è quello di affrontare il problema di supportare efficientemente l'esecuzione di range query per l'algoritmo di clustering basato su densità Trajectory-OPTICS (Paragrafo 2.4.2).
Trajectory-OPTICS è un'evoluzione spazio-temporale dell'algorimo di clustering DBSCAN. L'algoritmo considera le traiettorie come un unico elemento indivisibile, e raggruppa tutti gli oggetti che si muovono globalmente in maniera simile [1]. Inoltre, invece di concentrarsi su l'intero intervallo temporale, esegue una ricerca greedy dell'intervallo ottimale. Questo processo risulta essere molto costoso a causa del gran numero di distanze calcolate, e le performance complessive ne risentono.
L'obiettivo del mio lavoro è quello di proporre una struttura dati alternativa, pensata ad hoc per dati di traiettorie, e un algoritmo di ricerca più efficiente al fine di provare a migliorare le performance globali dell'algoritmo. Per il primo problema ho individuato una variante spazio-temporale dell'indice

R-Tree: il 3D R-Tree e il Packed 3DR-Tree. Nelle pagine successive verrano descritte le metriche utilizzate e successivamente l'algoritmo di ricerca SimilarityRangeSearch.

3.2 Metriche utilizzate

Come già anticipato, l'algoritmo di ricerca si basa sull'indice 3D R-Tree e sul Packed 3D R-Tree descritti rispettivamente in 2.3.1 e in 2.3.2. In questo paragrafo verranno descritti in dettaglio le metriche utilizzate per ridurre lo spazio di ricerca e minimizzare i calcoli.
In particolare vedremo:

- $DISSIM$: esprime il concetto di diversità (dissimilitudine) tra due traiettorie (paragrafo 3.2.1);
- $MINDIST$: metrica utilizzata per calcolare la minima distanza tra una traiettoria e un nodo (il suo MBR) dell'indice (paragrafo 3.2.2);
- LDD: Linearly Depended Dissimilarity tra due oggetti in movimento, una stima di variazione della distanza basata sulla velocità degli oggetti (paragrafo 3.2.3);
- $OPTDISSIM$: rappresenta una stima ottimistica di $DISSIM$ (paragrafo 3.2.4);
- $PESSDISSIM$: rappresenta una stima pessimistica di $DISSIM$ (paragrafo 3.2.5);
- $OPTDISSIM_{INC}$: $DISSIM$ ottimistica tra una una query trajectory Q e una traiettoria T durante un intervallo specificato (paragrafo 3.2.6);
- $MINDISSIM_{INC}$: minima $DISSIM$ tra una traiettoria T e un nodo dell'indice (paragrafo 3.2.7).

3.2.1 $DISSIM$

DISSIM[5] é una stima della diversitá o dissimilitudine spazio-temporale tra due traiettorie definite da due diversi oggetti in movimento.
Date due traiettorie T e Q entrambe valide nell'intervallo temporale $[t_1, t_n]$, si definisce dissimilitudine tra T e Q l'integrale definito tra t_1 e t_n della distanza punto a punto($D_{Q,T}$) tra T e Q al variare di t:

$$DISSIM(Q,T) = \int_{t_1}^{t_n} D_{Q,T}(t)dt \tag{3.1}$$

dove $D_{Q,T}$ é la funzione distanza tra due punti. Visto che le traiettorie vengono rappresentate come una sequenza di punti discreti, si puó riscrivere la 3.1 come segue:

$$DISSIM(Q,T) = \sum_{k=1}^{n-1} \int_{t_k}^{t_{k+1}} D_{Q,T}(t)dt \tag{3.2}$$

dove t_k e t_{k+1} rappresentano istanti di tempo in cui T e Q hanno posizione nota.
Sfruttando la regola del trapezio, possiamo approssimare la dissimilitudine tra due traiettorie T e Q con la seguente:

$$DISSIM(Q,T) \approx \frac{1}{2} \sum_{k=1}^{n-1} [(D_{Q,T}(t_k) + D_{Q,T}(t_{k+1}))(t_{k+1} - t_k)] \tag{3.3}$$

con un errore di approssimazione:

$$E_{Q,T} \leq \begin{cases} \frac{(t_{k+1}-t_k)^3}{12} |D^2_{Q,T}(-\frac{b}{2a})|, \text{ if } t_k \leq -\frac{b}{2a} \leq t_{k+1} \\ \frac{(t_{k+1}-t_k)^3}{12} |D^2_{Q,T}(t_{k+1})|, \text{ if } t_k < t_{k+1} < -\frac{b}{2a} \\ \frac{(t_{k+1}-t_k)^3}{12} |D^2_{Q,T}(t_k)|, \text{ if } -\frac{b}{2a} < t_k < t_{k+1} \end{cases} \tag{3.4}$$

Nella figura 3.1 viene mostrato un esempio di calcolo della DISSIM (appros-

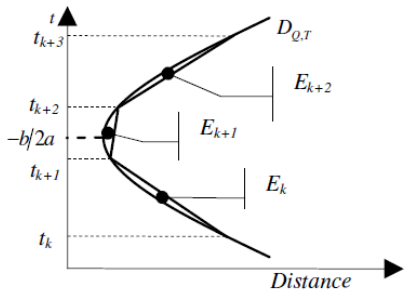
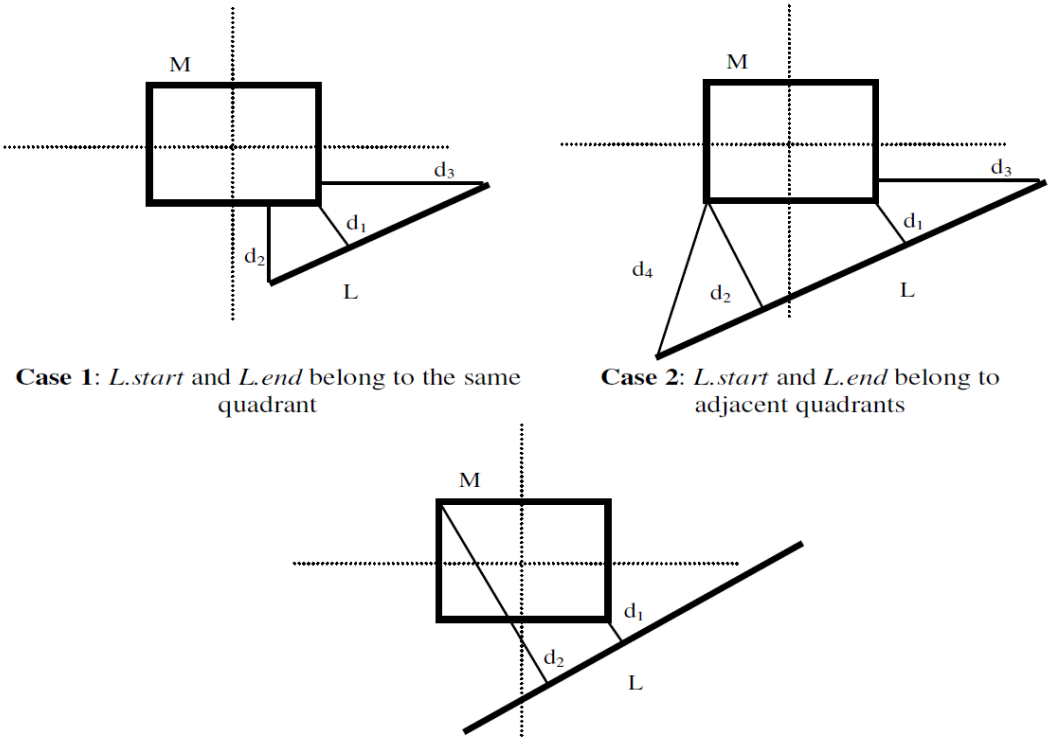


Figura 3.1: Errore di approssimazione nel calcolo della DISSIM

simata), e il relativo errore di approssimazione.

3.2.2 *MINDIST*

Questa metrica é utilizzata per permettere un corretto ordine di visita, e per effettuare il pruning di sottoalberi.
Dati una traiettoria Q, e il bounding box R_N di un nodo N dell'indice,



Case 1: *L.start* and *L.end* belong to the same quadrant

Case 2: *L.start* and *L.end* belong to adjacent quadrants

Case 3: *L.start* and *L.end* belong to non adjacent quadrants

Figura 3.2: Calcolo MINDIST [2]

entrambi validi nell'intervallo temporale I, definiamo $MINDIST$ come la minima distanza tra Q e il rettangolo R_N.

Per il suo calcolo é stato utilizzata una tecnica descritta in [2], allo scopo di minimizzare il numero di distanze da calcolare (vedi figura 3.2).
Inizialmente verificheremo se i due oggetti si intersecano; se é cosí allora:

$$MINDIST(Q, R_N) = 0$$

In caso contrario, lo spazio viene suddiviso in 4 quadranti con il centro in corrispondeza del centro di R_N. Determiniamo i quadranti in cui il segmento inizia (q_{start}) e finisce (q_{end}). Computiamo MINDIST in modo diverso al variare di q_{start} e q_{end}:

- **appartenenti allo stesso quadrante** ($q_{start} = q_{end}$)
  $MINDIST$ corrisponde allla distanza minima tra:
  - l'angolo di $R_N \in q_{start}$ e il segmento $Q_k \in I$ della traiettoria Q
  - tra $Q_{k_{start}}$ e R_N
  - tra $Q_{k_{end}}$ e R_N
- $Q_{k_{start}}$ **e** $Q_{k_{end}}$ **appartenenti a quadranti adiacenti**
  $MINDIST$ corrisponde alla minima distanza tra:
  - l'angolo di q_{start} e il segmento Q_k;
  - l'angolo di q_{end} e il segmento Q_k;
  - $Q_{k_{start}}$ e R_N;
  - $Q_{k_{end}}$ e R_N;
- $Q_{k_{start}}$ **e** $Q_{k_{end}}$ **appartenenti a quadranti non adiacenti**
  $MINDIST$ corrisponde alla minima distanza tra i due angoli di $R_N \notin q_{start}$ o q_{end}

3.2.3 *LinearlyDependedDissimilarity*

La Linearly Depended Dissimilarity (LDD) tra due oggetti, inizialmente a distanza D, che si muovono collinearmente a una velocità V durante l'intervallo temporale $\Delta t = [t_1, t_n]$ é data dalla seguente espressione:

$$LDD(D, V, \Delta t) = \begin{cases} \Delta t(D + V\Delta t/2), \text{ if } D + V\Delta t \geq 0 \\ D^2/(2|V|), \text{ altrimenti.} \end{cases} \tag{3.5}$$

dove V è la velocità relativa con cui i due oggetti si muovono. V è un numero positivo quando la distanza cresce, e negativo quando la distanza diminuisce.

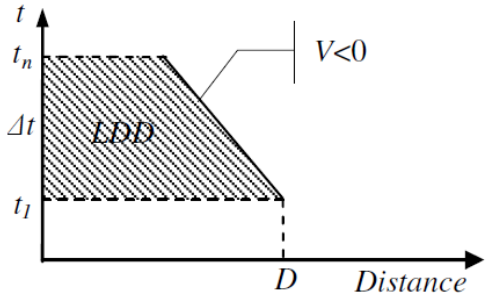


Figura 3.3: Linerly Depended Dissimilarity

3.2.4 *OPTDISSIM*

Dato un intervallo di tempo $[t_1, t_n]$ indichiamo con OPTDISSIM la piú ottimistica dissimilitudine tra una query trajectory Q e una traiettoria indicizzata T.

$$OPTDISSIM = \sum_{k=1}^{n-1} \begin{cases} DISSIM(Q_k, T_k), \text{ if } T_k \in S_r; \\ LDD(D_{Q,T}(t_{k+1}), -V_{max}, (t_{k+1} - t_k)), \text{ if } T_k \notin S_r, k = 1; \\ LDD(D_{Q,T}(t_k), -V_{max}, (t_{k+1} - t_k)), \text{ if } T_k \notin S_r, k = n-1; \\ LDD(D_{Q,T}(t_k), -V_{max}, (t_k^0 - t_k)) + \\ +LDD(D_{Q,T}(t_{k+1}), V_{max}, (t_{k+1} - t_k^0)), \text{ otherwise} \end{cases} \quad (3.6)$$

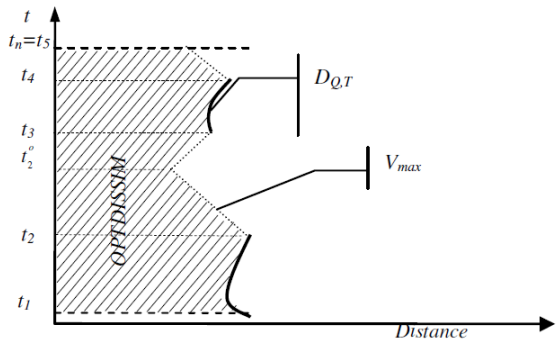


Figura 3.4: OPTDISSIM

dove $D_{Q,T}$ é la funzione della distanza tra le traiettorie Q e T, S_R é l'insieme di segmenti giá ritrovati dall'indice, V_{max} é la somma delle massime

velocitá delle traiettorie indicizzate e della query trajectory, e t_k^0 é dato dalla seguente espressione:

$$t_k^0 = t_k + t_{k+1} + (D_{Q,T}(t_{k+1}) - D_{Q,T}(t_k))/2 \tag{3.7}$$

Dalla 3.6 si evince che la $OPTDISSIM$, tra una query trajectory Q e una traiettoria indicizzata T, corrisponde alla somma delle $DISSIM$ dei segmenti già ritrovati ($T_k \in S_r$) e della stima ottimistica (basata sulla velocità V) dei segmenti non ancora visitati.

3.2.5 $PESSDISSIM$

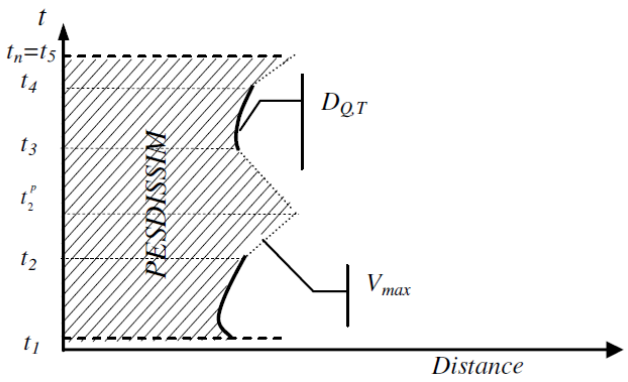


Figura 3.5: PESSDISSIM

Analogalmente a $OPTDISSIM$, $PESSDISSIM$ rappresenta una stima superiore di $DISSIM$ ed è utilizzata per stimarla nel caso di traiettorie parzialmente ritrovate:

$$PESSDISSIM(Q,T,t_1,t_n) = \sum_{k=1}^{n-1} \begin{cases} DISSIM(Q_k,T_k), \text{ if } T_k \in S_r; \\ LDD(D_{Q,T}(t_{k+1}),V_{max},(t_{k+1}-t_k)), \text{ if } T_k \notin S_r, k=1; \\ LDD(D_{Q,T}(t_k),V_{max},(t_{k+1}-t_k)), \text{ if } T_k \notin S_r, k=n-1; \\ LDD(D_{Q,T}(t_k),V_{max},(t_k^p-t_k))+ \\ +LDD(D_{Q,T}(t_{k+1}),-V_{max},(t_{k+1}-t_k^p)), \text{ otherwise} \end{cases} \tag{3.8}$$

Diversamente da quello che viene fatto in 3.6, in 3.8, nel caso di segmenti non ancora ritrovati, si suppone che i due oggetti divergano ad una velocità corrispondente alla velocità massima V_{max} delle traiettorie.

Nelle figure 3.5 e 3.4 è possibile vedere le due metriche a confronto. L'area tratteggiata rappresenta il valore di $OPTDISSIM$ (Fig. 3.4) e di $PESSDISSIM$ (Fig. 3.5). In evidenza il valore della distanza e della $DISSIM$ dei segmenti della traiettoria già ritrovati. Inoltre, per i restanti segmenti, viene mostrata la stima superiore e inferiore di $DISSIM$ che rispecchia i casi 2 - 4 di 3.8 e 3.6.

3.2.6 $OPTDISSIM_{INC}$

Assumendo di visitare i nodi dell'albero in ordine crescente della $MINDIST$ dalla query trajectory Q, definiamo la più ottimistica $DISSIM$ tra Q e una traiettoria T, con un segmento in N, in un intervallo di tempo $[t_1, t_n]$ come:

$$OPTDISSIM_{INC}(Q, T, N, t_1, t_n) = \min \begin{cases} DISSIM(Q_k, T_k), \text{ if } T_k \in S_r; \\ MINDIST(N, T)(t_{k+1} - t_k), \text{ otherwise} \end{cases} \tag{3.9}$$

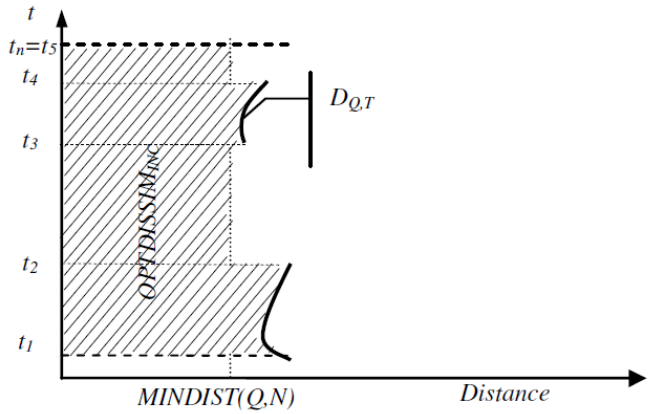


Figura 3.6: $OPTDISSIM_{INC}$

3.2.7 $MINDISSIM_{INC}$

Assumendo di visitare i nodi dell'albero in ordine crescente della $MINDIST$ dalla query trajectory Q, definiamo la minima $DISSIM$ tra una traiettoria indicizzata T, con un segmento nel nodo N, e la query trajectory Q

nell'intervallo $[t_1, t_n]$ come:

$$MINDISSIM_{INC}(Q, N, t_1, t_n) = \min \begin{cases} MINDIST(Q, N)(t_k - t_1) \\ OPTDISSIM_{INC}(Q, T, N, t_1, t_n), \forall T \in S_C \end{cases} \quad (3.10)$$

Questo perchè, visitando i nodi in ordine crescente della loro $MINDIST$ da Q, una traiettoria indicizzata non può avere una $DISSIM$ da Q inferiore a $MINDISSIM_{inc}$.

3.3 Descrizione algoritmo

L'algoritmo SimilarityRangeSearch esegue una visita non completa dell'indice, al fine di ritrovare e restituire tutte le traiettorie che hanno una dissimilitudine (Paragrafo 3.2.1), con la query trajectory, minore o uguale a sRange in un determinato intervallo temporale I. sRange ed I fanno parte dell'input dell'algoritmo, così come QT e orderedList. In dettaglio:

- QT: la query trajectory,
- sRange: rappresenta il range massimo di DISSIM che una traiettoria deve avere per far parte dell'output,
- I: é l'intervallo temporale in cui eseguiamo la ricerca, tutti i segmenti o i nodi interni presenti nell'albero e non appartenenti a quest'intervallo saranno scartati;
- orderedList: é una lista di nodi da visitare, ordinata rispetto al valore di MINDIST computato tra i nodi e la query trajectory. Alla prima iterazione conterrà la radice dell'albero.

Il corpo dell'algoritmo é costituito da un ciclo (linee 2-13) in cui ad ogni sua iterazione, estraiamo il nodo con il valore di MINDIST piú piccolo. Una volta ottenuto tale nodo, verifichiamo la condizione di uscita (linea 4):

$$MINDISSIM_{INC}(QT, currNode) \geq sRange$$

Quando questa condizione è verificata, allora sarà vera per tutti i nodi non ancora visitati, e l'algoritmo può terminare.

Nel caso in cui ci sono ancora nodi da visitare possiamo distingure due differenti casi: il nodo corrente é un nodo intermedio; il nodo corrente é una foglia. Nel primo caso (9-10) l'algoritmo esegue, per ogni figlio che ha una

sovrapposizione temporale con la query trajectory (riga 9), il calcolo di MINDIST e lo inserisce nella lista orderedList (riga 10).
Nel secondo caso(riga 6), l'algoritmo scorrerrà tutti i segmenti contenuti nel nodo foglia, calcolando per ciascun segmento ritrovato, avente una sovrapposizione temporale con QT, le metriche sopra descritte (vedi PROCESSLEAF).

SIMILARITYRANGESEARCH($sRange, QT, orderedList$)

```
1  exit = FALSE;
2  while !exit do
3    currNode = orderedList.Top();
4    if MINDISSIM_INC(QT, currNode) ≤ sRange
5       if currNode.isLeaf() then
6            processLeaf(currNode, QT);
7       else
8            foreach node N into currNode
9                 if N TIME OVERLAPS QT then
10                     orderedList.add( MINDIST(QT,N) );
11   else
12      exit = TRUE;
13
14 return( Completed ⋃ Approved);
```

Come giá detto in precedenza, ogni entry conterrá un segmento di una traiettoria. Per ognuno di esse, verificheremo se la traiettoria associata appartiene all'insieme Valid. Tale insieme contiene tutte le traiettorie parzialmente ritrovate, e che non sono state ancora rifiutate (Rejected), approvate (Approved), o completate (Completed).
Se il segmento appartiene ad una traiettoria valida, dovrá essere aggiunto alla SimilarityList (SL nel codice) della traiettoria che lo contiene. L'oggetto SimilarityList contiene tutti i segmenti ritrovati di una determinata traiettoria, assieme ai valori correnti di OPTDISSIM, PESSDISSIM, ERR, e DISSIM. Inoltre fornisce dei metodi per computare tali metriche in maniera incrementale ed efficiente (vedi paragrafo 3.4.2). Una volta inserito il segmento, andremo ad aggiornare i valori delle metriche di SL invocando il metodo computeMetrics. A questo punto l'algoritmo aggiornerá lo stato della traiettoria. Tale aggiornamento viene eseguito nel seguente modo:

- se abbiamo ritrovato tutti i segmenti di una traiettoria($SL.isCompleted = true$) allora viene etichettata come Completed;
- se il valore di OPTDISSIM è maggiore di sRange, viene etichettata come Rejected;

- se il il valore di PESSDISSIM é minore di sRange, viene etichettata come Approved.

Il significato di questi aggiornamenti é fortemente legato al significato delle due metriche OPTDISSIM e PESSDISSIM. Ricordiamo che OPTDISSIM e PESSDISSIM sono delle stime, rispettivamente, ottimistiche e pessimistiche di DISSIM. Questo significa che la seguente affermazione é sempre vera:

$$OPTDISSIM \leq DISSIM \leq PESSDISSIM$$

Di conseguenza quando una traiettoria parzialmente ritrovata ha un valore di OPTDISSIM maggiore di sRange, non potrá mai avere DISSIM minore di sRange, e di conseguenza possiamo inserirla nell'insieme Rejected. Dall'altro lato, una traiettoria parzialmente ritrovata con valore di PESSDISSIM minore di sRange sará inserita in Approved.

processLeaf($currNode, QT$)

```
1  foreach EntryLeaf E into currNode do
2   if isValid(E.trajectory()) then
3          SL = Valid.getSimList(E.trajectory());
4          Find first QuerySegment QS that t. overlaps E
           do
5                 SL.add(QS,E);
6                 SL.computeMetrics();
7                 if SL.isCompleted() then
8                    Completed.add(Valid.extracts( E.trajectory() );
9                 else if SL.OPTDISSIM ≥ sRange then
10                   Rejected.add(Valid.extracts( E.trajectory() );
11                else if SL.PESSDISSIM ≤ sRange then
12                   Approved.add(Valid.extracts( E.trajectory() );
13                QS.getNextSegment();
14         while OVERLAPS(QS,E);
```

3.4 Dettagli implementativi

In questo paragrafo espongo brevemente alcuni dettagli implementativi riguardante la parte di sviluppo di questa tesi. Inizieremo parlando dell'indice su cui si appoggia l'algoritmo: R-Tree e i derivati ottenuti (3D R-Tree, Packed R-Tree). In seguito verrà descritta un'altra struttura dati, la SimilarityList, utilizzata dall'algoritmo di ricerca per introdurre alcune ottimizzazioni nel calcolo incrementale delle metriche. Infine chiuderemo il capitolo parlando dell'algoritmo di ricerca.

3.4.1 Indici: R-Tree, 3D R-Tree, Packed R-Tree

L'algoritmo di ricerca SimilarityRangeSearch, è stato implementato in C++ e sviluppato in ambiente Visual Studio 2010 (unmanaged). Come già anticipato, l'algoritmo utilizza alcune variente dello R-Tree di Gutmann, in particolare si appoggia sul 3D R-Tree, e sul Packed 3D R-Tree.
In particolare ho utilizzato una libreria open source che espone, sotto forma di una classe generica (C++ template class), l'indice R-Tree ideato da Gutmann. A tempo di esecuzione, i parametri della classe determinano la struttura dell'albero, in particolare essi sono:

- DATATYPE: rappresenta il tipo delle entry contenute nelle foglie. Nel nostro caso, la entry è rappresentata da due informazioni: identificativo univoco di traiettorie e identificativo di segmento della traiettoria
- ELEMENTYPE: rappresenta il tipo delle coordinate. Nel nostro caso float.
- NUMDIMS: il numero di dimensioni da gestire. Come già detto l'indice gestirà 3 dimensioni, due spaziali e una temporale.
- TMAXNODES: indica il numero massimo di figli che un nodo dell'albero può avere.
- TMINNODES: indica il numero minimo di figli che un nodo può avere. L'indice manterrà sempre un occupazione uguale o superiore al 50%.

L'algoritmo di ricerca descritto nel paragrafo 3.3 mantiene, in memoria, la SimilarityList relativa a ciascuna traiettoria parzialmente ritrovata.

3.4.2 SimilarityList

La SimiliratyList conterrà alcune informazioni generali come l'identificativo della traiettoria e il suo stato: Approved, Completed, Rejected e Valid. Inoltre ogni elemento della lista, mantiene le informazioni relative al segmento ritrovato che si sovrappone temporalmente alla query trajectory. Tra queste ricordiamo:

- dStart: distanza iniziale tra il segmento della traiettoria e il segmento della query trajectory che si sovrappone temporalmente ad esso,
- dEnd: distanza finale tra il segmento della traiettoria e il segmento della query trajectory che si sovrappone temporalmente ad esso,

- DISSIM: il valore della DISSIM tra il segmento della traiettoria e il segmento della query trajectory.

Gli oggetti di tipo SimilarityList espongono un metodo per calcolare le metriche descritte nel capitolo precedente. In particolare la $DISSIM$ tra la traiettoria e la query trajectory, e i valori di $OPTDISSIM$ (vedi fig. 3.4) e $PESSDISSIM$. (vedi fig. 3.5) Diverse tecniche sono stati applicate per ottimizzare al meglio la gestione di questa struttura. Tra tutte ricordiamo l'algoritmo di inserzione, in cui si esegue una merge dei segmenti temporalmente vicini, al fine di ridurre la dimensione della lista e di conseguenza i tempi di calcolo delle varie metriche.

3.4.3 SimilarityRangeSearch

L'algoritmo SimilarityRangeSearch è un algoritmo iterativo che è esposto come un metodo della classe D3R-Tree. Come già descritto in precedenza, esegue una visita ordinata dell'albero basata sulla minima distanza tra la query trajectory Q e i nodi dell'albero ritrovati. Questo significa che, ad ogni passo l'algoritmo sceglie, tra i nodi dell'indice già visitati, quello con la minima distanza da Q. Quest'assunzione permette di applicare tecniche di pruning dello spazio di ricerca. Infatti, se la $MINDISSIM_{INC}$ tra la query trajectory Q e il nodo con valore di $MINDIST$ minore è inferiore del range in input, non possiamo più ritrovare traiettorie con $DISSIM \leq sRange$ (vedi paragrafo 3.2.7) e quindi l'algoritmo può terminare. Oltre alla SimilarityList, l'algoritmo utilizza una lista ordinata in base a $MINDISSIM$ (nodo - query trajectory), di nodi già visitati. Ad ogni iterazione viene estratto il nodo in testa alla lista.

Capitolo 4

Test e risultati

Nei capitoli precedenti, abbiamo analizzato lo stato dell'arte nel dominio dei dati di mobilità, successivamente abbiamo introdotto alcune metriche e presentato un algoritmo di ricerca in grado di supportare query basate su traiettoria e sul concetto di distanza Euclidea. In questo capitolo verranno riassunti i test eseguiti al fine di analizzare le performance dell'algoritmo implementato e i tempi medi di inizializzazione dell'indice (lettura da file system delle traiettorie + tempo di caricamento nell'indice). Inizieremo descrivendo i parametri di input dell'indice e dell'algoritmo SimilarityRangeSearch. In 4.2 verranno descritti i test eseguiti al variare di alcuni dei parametri descritti in 4.1. Infine verranno mostrati e analizzati i risultati più rilevanti.

4.1 Descrizione dei parametri dell'indice e dell'algoritmo

I parametri dell'algoritmo possono essere suddivisi in: parametri dell'indice, parametri dell'algoritmo e parametri del dataset.
Il primo gruppo di parametri determinano la struttura dell'indice e la disposizione dei dati:

- $MODE$: questo parametro determina il tipo di struttura: 3D R-Tree, Packed 3D R-Tree;
- $MAXNODES$ rappresenta il numero massimo di elementi che un nodo puó avere;
- $MINNODES$: rappresenta il numero minimo di elementi che una foglia puó contenere;

Per parametri del dataset intendiamo i parametri che sono stati utilizzati per la generazione dei dataset sintetici:

- nTrajectories: il numero di traiettorie del dataset;
- minPoints: il numero minimo di segmenti che una traiettoria del dataset puó avere;
- maxPoints: il numero massimo di punti che una traiettoria del datast potrá avere;
- nCluster: il numero di traiettorie da utilizzare come core trajectory. Le restanti traiettorie saranno definite a partire dai core trajectory.

Ed infine abbiamo i parametri dell'algoritmo di ricerca:

- dataset: il nome del file contenente il dataset;
- nTrajectories: il numero di traiettorie del dataset in input;
- T_{start}: rappresenta il timestamp iniziale dell'intervallo temporale di ricerca;
- T_{end}: assieme a T_{start} definisce l'intervallo temporale $I = [T_{start}, T_{end}]$
- Range: scalare che identifica un intorno della query trajectory in cui una traiettoria é definita simile.

4.2 Descrizione Test

I test sono stati eseguiti con lo scopo di analizzare il tempo medio di esecuzione di una range query, al variare di alcuni parametri dell'algoritmo e dell'indice. I test sono stati eseguiti su una macchina Windows dotata di CPU Intel(R) Core(TM)2 Quad CPU a 2.50Ghz e 4Gb di Ram.

Per i test ho utilizzato una serie di dataset sintentici con un numero variabile di traiettorie che va da 5000 a 80000. Ciascuna traiettoria del dataset è stata generata con un numero di punti (sample point) variabile da 70 a 100. Il generatore di dati sintetici è un algoritmo iterativo che: inizialmente identifica un numero di core trajectory imposto dall'utente (nCluster) e successivamente ad ogni iterazione genera una traiettoria a partire da una core trajectory scelta in modo casuale.

Per l'analisi delle prestazioni dell'algoritmo ho considerato tre diversi aspetti:

- i tempi di caricamento e indicizzazione delle traiettorie,
- i tempi di esecuzione di una range search,
- confronto dei tempi di esecuzione di Time Focused-OPTICS con M-Tree, 3D R-Tree, Packed 3D R-Tree.

Il terzo e ultimo aspetto, riguarda il confronto delle performance dell'algoritmo Trajectory-OPTICS (variante time focused) con le performance delle sue varianti che utilizzano gli indici da me proposti. In questo caso ho messo a confronto i tempi complessivi di esecuzione dell'algoritmo, utilizzando i dataset con un numero di traiettorie che vanno da 5000 a 40000.

4.2.1 Test 1: analisi performance al variare del numero di traiettorie in input

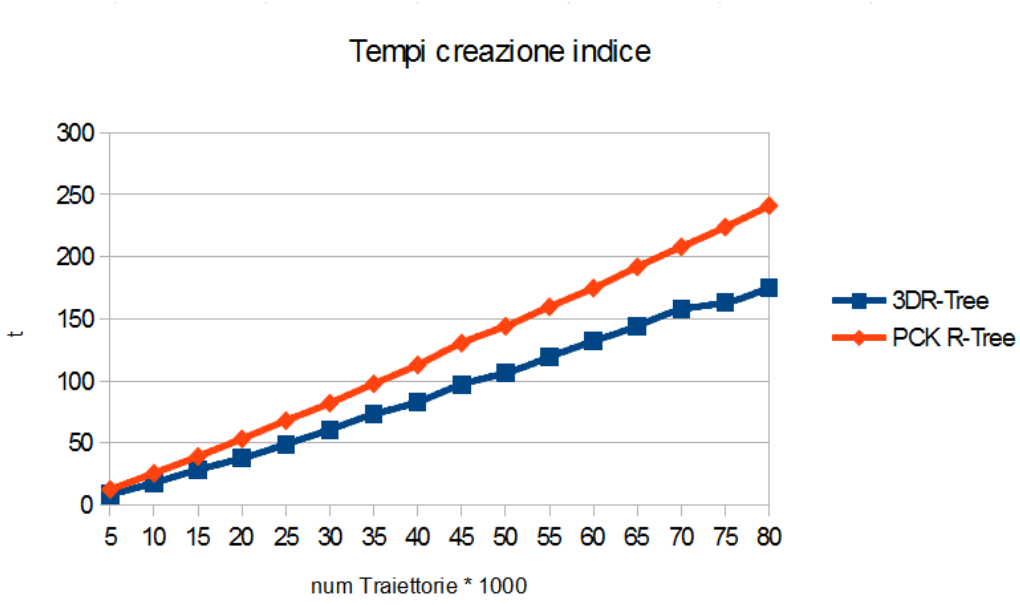


Figura 4.1: Tempi medi di creazione dell'indice al variare del numero di traiettorie del dataset in input.

L'obiettivo di questo test è quello di analizzare il comportamento dell'indice e dell'algoritmo di ricerca al variare del numero di traiettorie del dataset in input.
A questo scopo sono stati eseguiti una serie di test mantenendo costante i

parametri relativi all'indice $MAXNODES$ e $MINNODES$ impostati rispettivamente a 512 e a 256. Inoltre ho imposto ad ogni iterazione un valore di ϵ pari al 1% della diagonale dell'estensione geografica del dataset in input. Per questo test il numero di traiettorie è variato da 5000 a 80000. Le range query sono state eseguite sull'intero intervallo temporale.

Il grafico in figura 4.1 mostra il confronto tra i tempi medi di caricamento e indexing al variare del numero di traiettorie nel dataset. Sull'asse delle ordinate abbiamo il tempo espresso in secondi, mentre sulle ascisse il numero di traiettorie del dataset. Il tempo impiegato in fase di caricamento (I/O) e indexing delle traiettorie (creazione indice) cresce linearmente con il numero di traiettorie. Possiamo anche notare che i tempi di caricamento del Packed 3DR-Tree sono mediamente più alti rispetto a quelli del 3DR-Tree. In figura

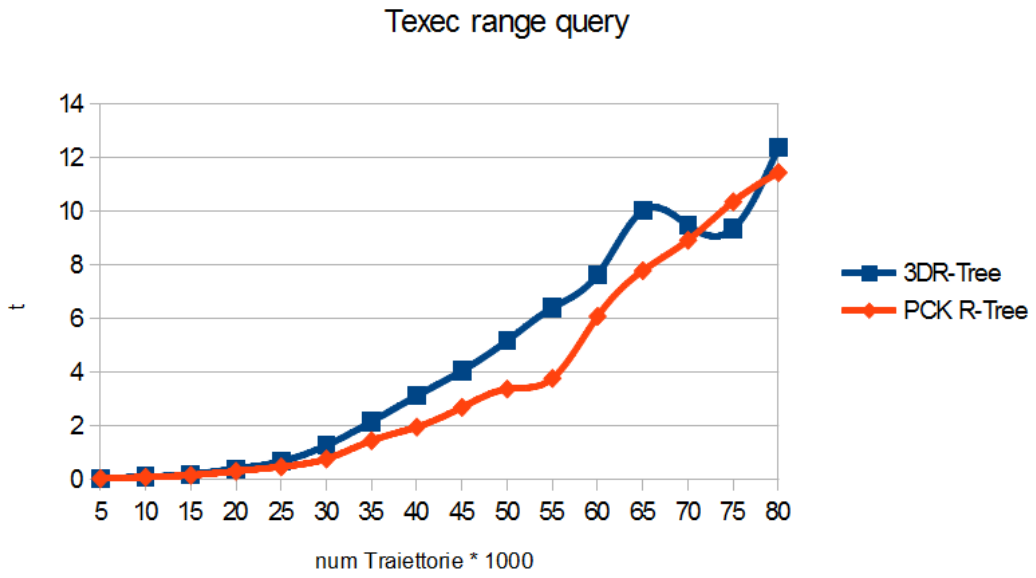


Figura 4.2: Tempi medi di esecuzione di una range query al variare del numero di traiettorie indicizzate.

4.2 il grafico mostra i tempi medi di esecuzione di una range query. Anche in questo caso i tempi si alzano all'aumentare del numero di traiettorie indicizzate. In media i tempi di esecuzione di una range query, eseguita utizzando il Packed 3DR-Tree, sono migliori di quelli ottenuti su 3DR-Tree. Molto probabilmente, la causa di questo piccolo miglioramento è il risultato di una migliore compressione del primo dei due indici.

4.2.2 Test 2: analisi performance al variare di ϵ

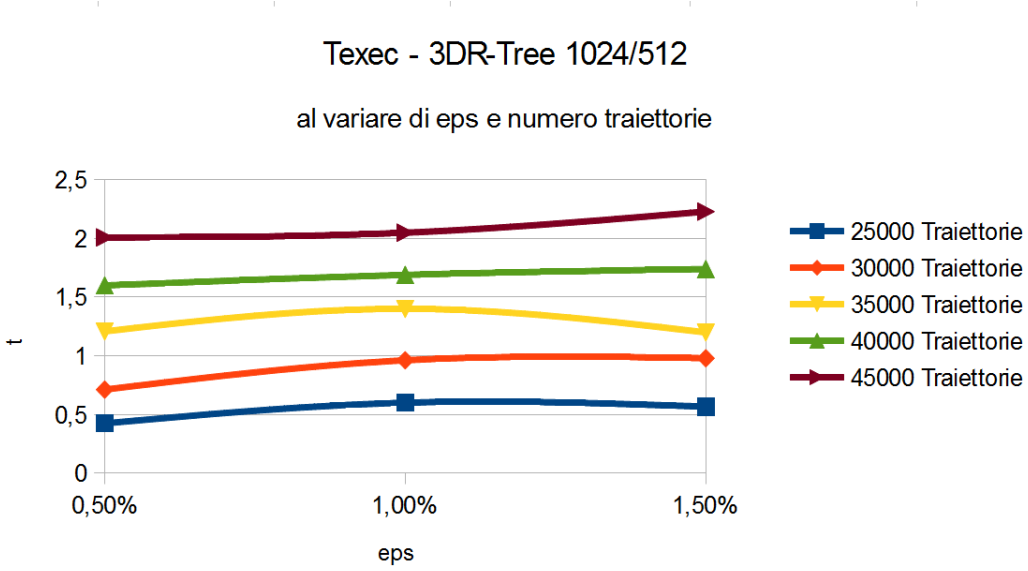


Figura 4.3: Tempi medi di esecuzione di una range query al variare della soglia ϵ.

L'obiettivo di questo test è quello di analizzare il comportamento dell'indice e dell'algoritmo di ricerca al variare della soglia ϵ.
Questi test sono stati eseguiti su dataset con un numero di traiettorie che va da 25000 a 50000. In tutti i test considero sempre l'intero intervallo temporale. Per quanto riguarda i parametri dell'indice, in questo caso sono stati impostati a 1024($MAXNODES$) e a 512 ($MINNODES$). In figura 4.3 e 4.4 vediamo il comportamento dell'algoritmo (tempi esecuzione su 3DR-Tree e Packed 3DR-Tree) al variare del valore di ϵ e del numero di traiettorie del dataset. In entrambi i casi i tempi di esecuzione tendono ad aumentare all'aumentare della soglia ϵ. Questo comportamento potrebbe essere una conseguenza delle tecniche di pruning, che generalmente sono più efficaci per valori piccoli di ϵ.

4.2.3 Test 3: analisi performance al variare dell'intervallo temporale

L'obiettivo di questo test è quello di analizzare il comportamento dell'algoritmo al variare dell'intervallo temporale della range query.
In particolare ho considerato finestre temporali di diverse dimensioni: 10%,

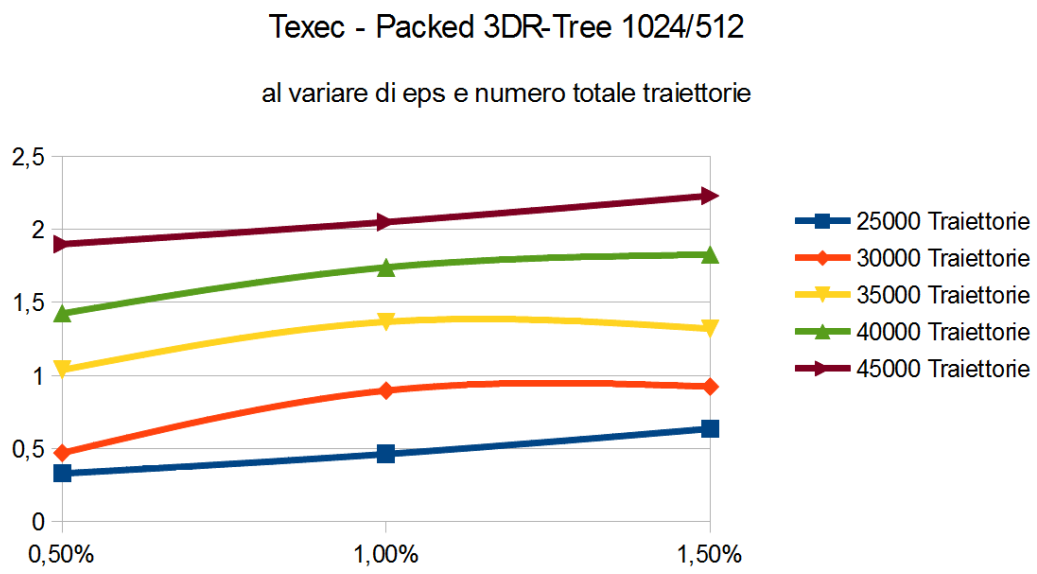


Figura 4.4: Tempi medi di esecuzione di una range query al variare della soglia ϵ.

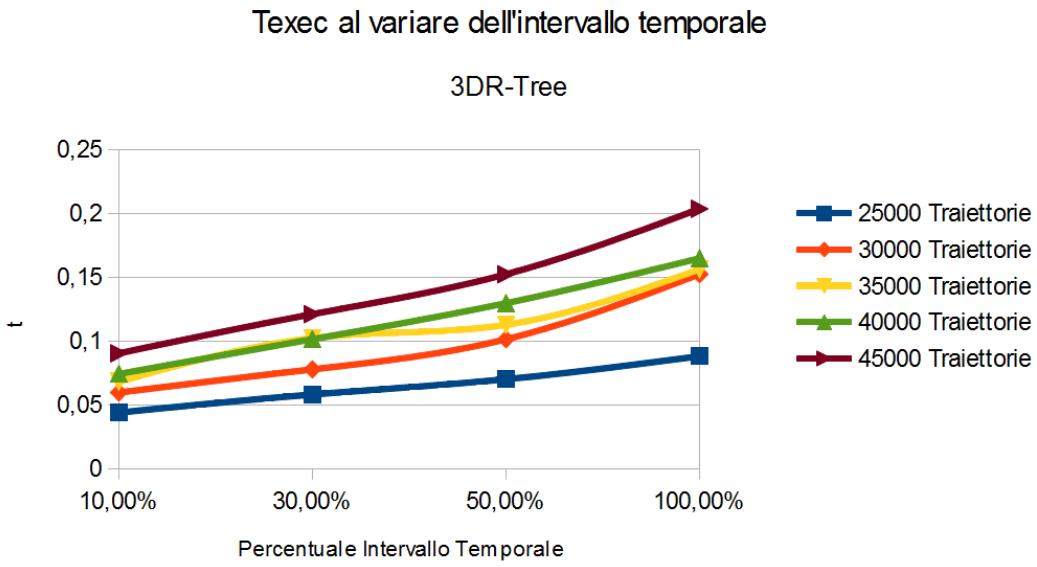


Figura 4.5: 3DR-Tree: Tempi medi di esecuzione di una range query al variare della finestra temporale.

30%, 50% e 100% dell'intervallo temporale. I restanti parametri dell'algoritmo sono stati impostati come segue: dataset di 40000 traiettorie, ϵ pari al 1% della diagonale del dataset. I parametri dell'indice, $MAXNODES$ e $MINNODES$ sono stati impostati rispettivamente a 512 ed a 256. Nelle

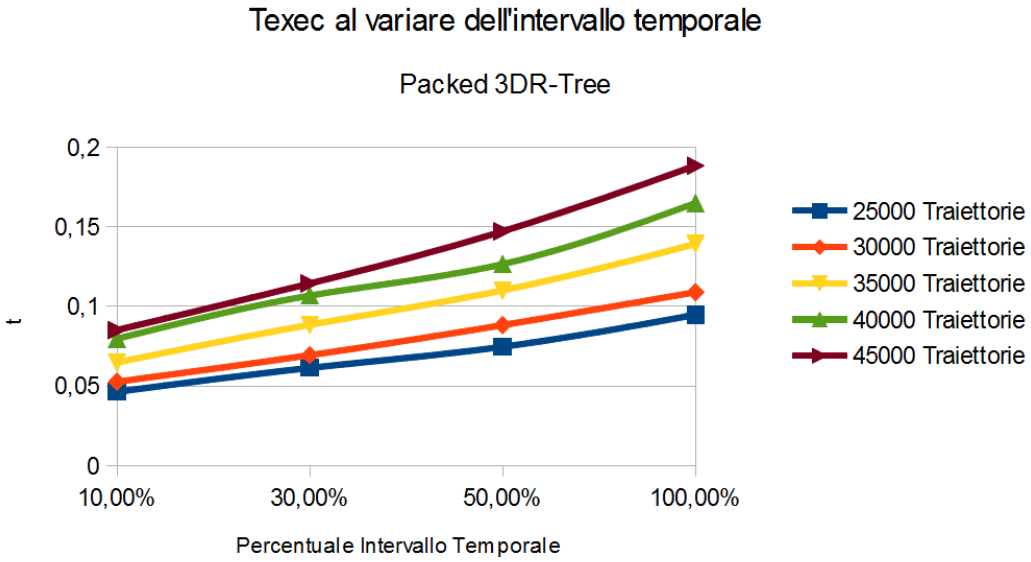


Figura 4.6: Packed 3DR-Tree: Tempi medi di esecuzione di una range query al variare della finestra temporale.

figure 4.5 e 4.6, è possibile vedere l'impatto che ha sulle performance la riduzione della finestra temporale di una range query. In entrambi i casi, possiamo notare come, ad una riduzione dell'intervallo temporale specificato nell'interrogazione, corrisponda una diminuizione dei tempi di esecuzione della query. Anche in questo caso, la spiegazione di questi risultati è legata al pruning dello spazio di ricerca. Infatti molti nodi dell'albero non verranno visitati perchè fuori dalla finestra temporale definita.

4.2.4 Test 4: confronto performance Trajectory-OPTICS

L'obiettivo di questo test è quello di analizzare le differenze di performance dell'algoritmo Trajectory-OPTICS (variante time focused) al variare della struttura dati utilizzata: M-Tree, 3D R-Tree e Packed 3DR-Tree. I parametri utilizzati sono: ϵ pari all'1% della diagonale del dataset, $MAXNODES$ e $MINNODES$ impostati rispettivamente a 1024 e 512. I test sono stati eseguiti su tre dataset diversi, ciascuno con 10000, 20000 e 30000 traiettorie.

Il grafico nella figura 4.7 mostra i tempi di esecuzione dell'algoritmo Trajectory-Optics nelle varianti sopra elencate. Possiamo notare che i tempi di esecuzione dell'algoritmo migliorano con l'utilizzo delle strutture dati introdotte in questa tesi. In particolar modo l'utilizzo del 3DR-Tree dimezza i tempi di esecuzione dell'algoritmo di clustering. La motivazione di questo risultato è che nel 3DR-Tree e nella sua variante Packed, lo spostamento della finestra temporale non ha nessun impatto sulle performance. Inoltre, a una range query eseguita non sull'intero intervallo temporale, generalmente comporterà un minor numero di nodi visitati (vedi paragrafo 4.2.3). Mentre quando si utilizza l'indice M-Tree, ad ogni spostamento della finestra temporale da analizzare, è necessario ricostruire l'indice. Ricordiamo che nella variante time focused l'algoritmo ad ogni iterazione modifica la finestra temporale con l'obiettivo di ritrovare l'intervallo di tempo ottimale. Questo ovviamente impatta negativamente sulle prestazioni.

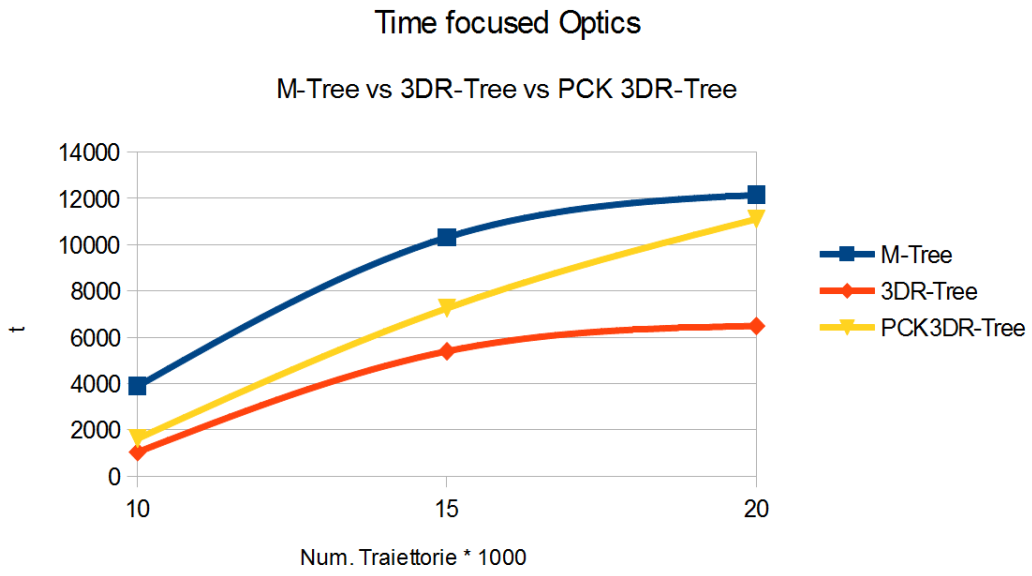


Figura 4.7: Tempi medi di esecuzione TF-Optics nelle varianti: M-Tree, 3DR-Tree e PCK 3DR-Tree.

Capitolo 5

Conclusione e sviluppi futuri

Il problema del clustering di traiettorie è un task importante del processo di Geographic Knowledge Discovery. L'obiettivo di questa tecnica di data mining è quello raggruppare in cluster gli elementi appartenenti alle stesse zone ad alta densità. In questa tesi abbiamo esposto il problema della gestione dei dati di movimento, focalizzandoci principalmente sul problema dell'indexing e del supporto alle interrogazione spazio-temporali. Abbiamo presentato due diversi approcci di indicizzazione per range query su dati di traiettorie. L'obiettivo era quello di migliorare la scalabilità di un particolare algoritmo di clustering basato su densità. In particolare è stato sviluppato l'algoritmo di ricerca introdotto nel capitolo 3 che si appoggia su due varianti dell'indice multidimensionale R-Tree: il 3DR-Tree e il Packed 3DR-Tree. Nel capitolo 4 sono stati eseguiti due tipologie di test: la prima con l'obiettivo di studiare il comportamento dell'algoritmo al variare degli input; la seconda, per analizzare la scalabilità dell'algorimto Trajectory-OPTICS (variante time focused) al variare dell'indice utilizzato: M-Tree, 3DR-Tree e Packed 3DR-Tree. I risultati ottenuti mostrano che, l'utilizzo delle tecniche presentate in questa tesi, portano sempre al miglioramento delle prestazioni dell'algoritmo di clustering Trajectory-OPTICS, soprattuto nella variante time focused (vedi paragrafo 4.2.4).
L'algoritmo ha degli ottimi comportamenti anche dal punto di vista della gestione della memoria, infatti nel caso di utilizzo di dataset contenenti 80000 traiettorie (almeno 5 milioni di sample point in tutto il dataset), l'utilizzo della memoria si stabilizza sui 700MB. Gli sviluppi futuri in questo settore, dovrebbero riguardare l'utilizzo di diverse nozioni di distanza per traiettorie, con l'obiettivo di valutare, oltre alle performance, anche la qualità dei cluster ottenuti.

Bibliografia

[1] Mirco Nanni e Dino Pedreschi. Time-focused clustering of trajectories of moving objects. 2006.

[2] E. Frentzos, K. Gratsias, N. Pelekis, Y. Theodoridis. Algorithms for Nearest Neighbor Search on Moving Object Trajectories.

[3] Y. Theodoridis, M. Vazirgiannis, T. Sellis. Spatio-Temporal Indexing for Large Multimedia Applications.

[4] Antonin Gutmann. R-Trees. A dynamic index structure for spatial searching.

[5] Elias Frentzos and Kostas Gratsias and Yannis Theodoridis and Elias Frentzos and Kostas Gratsias and Yannis Theodoridis. Index-based Most Similar Trajectory Search. 2006.

[6] Giannotti, Fosca and Pedreschi, Dino. Mobility, Data Mining and Privacy: Geographic Knowledge Discovery. 2008

[7] Y. Manolopoulos, A. Nanopoulos, A. Papadopoulos, Y. Theodoridis. R-Trees: Theory and Applications, Springer, 2005

[8] C. Renso, S. Puntoni, E. Frentzos, A. Mazzoni, B. Moelans, and N. Pelekis and F. Pini. Wireless Network Data Sources: Tracking and Synthesizing Trajectories

[9] I. Kamel and C. Faloutsos. On Packing R-trees, Proceedings 2nd ACM International Conference on Information and Knowledge Management (CIKM'93), pp.490-499. Washington, DC, 1993.

[10] Macedo, J. and Vangenot, C. and Othman, W. and Pelekis, N. and Frentzos, E. and Kuijpers, B. and Ntoutsi, I. and Spaccapietra, S. and Theodoridis, Y. Trajectory Data Models. 2008.

[11] Wachowicz, M, Ligtenberg A, Renso C, Gürses SF. Characterising the Next Generation of Mobile Applications Through a Privacy-Aware Geographic Knowledge Discovery Process. 2008.

[12] I. Kamel and C. Faloutsos: "On Packing R-trees", Proceedings 2nd ACM International Conference on Information and Knowledge Management (CIKM'93), pp.490-499, Washington, DC, 1993

[13] P. Ciaccia, M. Patella, P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spasec. 1997.

Elenco delle figure

Indice