Generatori software di traiettorie di oggetti in movimento per valutare le prestazioni di operatori spazio-temporali

Matteo Orsini, Paolino Di Felice (*)

(*) Dipartimento di Ingegneria Elettrica e dell'Informazione
Università dell'Aquila

Riassunto
Con il termine *trasporto* si indica il movimento di persone e merci sulla superficie terrestre. Ad un elevato livello di astrazione, gli uni e gli altri sono modellabili con punti la cui posizione mutua nel tempo con continuità (nel seguito, brevemente, *m-points*). Qualunque applicazione software evoluta nell'ambito dei trasporti si voglia sviluppare, essa deve opportunamente interagire con SGBD "irrobustiti" con un adeguato repertorio di operatori performanti per m-points. Prima del loro rilascio (come User Defined Function) è necessario validarne sperimentalmente il funzionamento e le prestazioni "sul campo", utilizzando dati sintetici significativi. La ricerca riassunta in questo articolo concerne la definizione di due algoritmi per la generazione di dati sintetici di traiettorie e la loro codifica. Nel seguito si darà ampio spazio alla descrizione degli elementi peculiari delle traiettorie generate con i predetti due algoritmi, insieme agli ambiti per un loro utilizzo ottimale.

Abstract
The term *transportation* means the movement of people and goods on the Earth's surface. At an high level of abstraction, one and each other are modeled with points whose position changes continuously over time (in the follow, briefly, *m-points*). Any software application developed in the field of transport you want to develop, it must interact appropriately with DBMS "enhanced" with an adequate repertoire of performed operator for m-points. Before their release (as User Defined Function) it is necessary to experimentally validate their functioning and their performances, using significant synthetic data. The research resumed in this article concern the definition of two algorithm for the generation of synthetic data for trajectories. In the following we will give ample space to the description of particular elements of the trajectories generated with the above two algorithms, along with areas for their optimal use.

Introduzione

La diffusione delle nuove tecnologie di posizionamento fondate sul Global Positioning System (GPS) ha notevolmente accresciuto l'interesse nello sviluppo di applicazioni mobili basate sui *m-points*, la cui modellazione attraverso soluzioni software rappresenta, al giorno d'oggi, un'esigenza concreta per tutte quelle organizzazioni aziendali che vogliono condurre investigazioni spazio-temporali sui dati territoriali in loro possesso. Se si pensa a un ente che gestisce i trasporti pubblici, ad esempio, risulta immediato intuire l'interesse che può avere nel monitorare le corse compiute dai propri mezzi, al fine di ottimizzare la copertura del territorio servito.
L'idea di arricchire i SGBD esistenti con operatori performanti per i *m-points* rappresenta una delle migliori strade perseguibili per l'integrazione semplice e immediata degli strumenti software atti al trattamento dei dati spazio-temporali all'interno delle infrastrutture informatiche utilizzate dalle aziende, le quali, nella maggior parte dei casi, fanno già riferimento a basi di dati gestionali classiche o abilitate spazialmente. A questo tipo di approccio nella gestione degli oggetti di movimento si sono dedicati negli ultimi anni diversi gruppi di ricerca accademici. Testimonianza ne

sono il progetto in corso HERMES (Pelekis et Al., 2010) e i risultati raggiunti dagli stessi autori di questa memoria e pubblicati in (Di Felice, Orsini, 2011), trattazione in cui vengono proposti due algoritmi utili al calcolo delle intersezioni fra coppie di *traiettorie* in presenza di incertezza (dove per *traiettoria* si intende l'insieme di punti dello spazio corrispondenti alle posizioni di un corpo in moto in istanti di tempo successivi). Il fatto che i nuovi operatori possano essere aggiunti come UDFs al gruppo di funzioni "*built-in"* puramente spaziali già presenti nei principali SGBD, rende gli stessi facilmente accessibili a utenti o applicazioni esterne che si collegano al database, i quali possono farne uso per mezzo di query definite con i comuni costrutti SQL.
In questo scenario, la disponibilità di un ambiente di analisi comparativa per testare le prestazioni delle strutture dati e degli operatori che agiscono su m-points è un'esigenza fortemente sentita da tutti coloro che si imbattono nella progettazione ex novo di soluzioni software per la loro gestione e interrogazione. Il bisogno di datasets sintetici di traiettorie con caratteristiche quanto più vicine alla realtà, ha indubbiamente sollecitato il mondo della ricerca e nel corso dell'ultimo decennio diversi gruppi di lavoro si sono concentrati nello studio di tecniche e algoritmi per la loro costruzione e generazione automatica. Risultati interessanti al riguardo sono stati proposti in (Theodoridis et al., 1999) e (Brinkhoff, 2002), articoli nei quali vengono presentati due approcci molto vantaggiosi per testare l'efficienza delle tecniche di memorizzazione e la velocità delle funzioni disponibili per accedere agli oggetti di movimento, ma pressoché di alcuna utilità quando si vuole valutare come gli stessi si relazionano gli uni con gli altri.
A seguire saranno proposti due nuovi algoritmi per la generazione di traiettorie sintetiche per i m-points, il primo dei quali dà luogo a evoluzioni spazio-temporali completamente casuali confinate in un'area di ampiezza predefinita, mentre il secondo origina andamenti più controllati, nei quali gli spostamenti dell'oggetto da una posizione a quella successiva, sono regolati da considerazioni riguardo la velocità massima che esso può assumere e la durata complessiva dell'osservazione. Nel rappresentare la traiettoria all'interno della base di dati, si è scelto di modellarne l'andamento geometrico attraverso un oggetto di tipo *linestring* e di memorizzare gli istanti di campionamento all'interno di un array di *timestamps*. Nel dettaglio, quindi, le traiettorie possono essere mantenute all'interno di una tabella del seguente tipo:

```
Trajectory (
      Pkey:             integer,
      Shape:            linestring,
      TimeValues:       timestamp with time zone ARRAY
)
```

A livello sperimentale, i due algoritmi che seguiranno sono stati implementati come UDFs in linguaggio procedurale PL/pgSQL ed eseguiti sul SGBD PostgreSQL, versione 8.4, abilitato con l'estensione spaziale PostGIS . La visualizzazione delle traiettorie seguite dai m-points verrà ottenuta attraverso l'uso del Sistema Informativo Territoriale open source Quantum GIS, applicativo equipaggiato con le funzionalità necessarie per collegarsi a database mantenuti in SGBD abilitati spazialmente e recuperarne il contenuto.

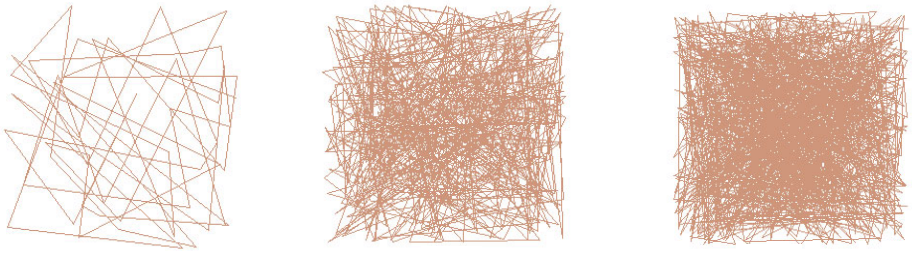
Generazione di traiettorie casuali
Il modo più banale ed intuitivo per dare origine a una traiettoria è quello di generare in maniera casuale i punti che ne determinano l'andamento spaziale e associare a ciascuno di essi un *timestamp* progressivamente incrementato di un valore *delta* costante. La cardinalità dei punti può essere vista come un parametro "*n*" variabile da specificare al momento dell'esecuzione. L'algoritmo proposto a seguire descrive i passi per la progettazione dell'operatore `generate_random_trajectory()` che genera m-points adottando la filosofia appena presentata. Esso riceve in input "*n"* ed al termine dell'elaborazione aggiunge un nuovo record alla tabella `Trajectory` impostando il campo `Shape` a una linestring di dimensione "*n*" e il campo `TimeValues` a un array di altrettanti valori.

Algoritmo: generate_random_trajectory (*integer* n)

Input: l'intero *n* che indica il numero di punti che costituiscono la traiettoria.
Output: Viene aggiunta alla tabella `Trajectory` nel database una nuova traiettoria specificata dalla `linestring` che ne modella l'andamento spaziale e dall'array di *timestamps* che ne descrive l'evoluzione temporale.
Metodo:

1. Siano *Points[]* e *Times[]* due array di oggetti rispettivamente di tipo `Point` e `timestamp with time zone`, inizialmente vuoti.
2. *t* = `now()`
3. *delta* = *"00:10:00"*
4. *i* = 1
5. **WHILE(** *i* ≤ n **) LOOP**
6. *x* = `random()`
7. *y* = `random()`
8. *p* = `st_makepoint(`*x,y*`)`
9. *Points*[*i*] = *p*
10. *Times*[*i*] = *t*
11. *t* = *t*+*delta*;
12. **END LOOP**;
13. *line* = `st_makeline(`*Points[]*`)`
14. INSERT INTO *Trajectory(shape,timevalues) VALUES (line,Times[])*
15. **END**

In riga 1 vengono recuperati per mezzo dell'operatore `now()` la data e l'ora correnti al momento dell'invocazione di `generate_random_trajectory()`, mentre in riga 2 viene impostato a 10 minuti l'intervallo *delta* di acquisizione dei dati. All'interno del ciclo WHILE vengono generate in maniera totalmente casuale le coordinate dell'*i-esimo* punto della traiettoria (righe 6 e 7), le quali vengono utilizzate come argomento dell'operatore OGC `st_makepoint(float, float)` per produrre una geometria di tipo `point` (riga 8). Quest'ultima viene aggiunta ordinatamente all'array dei punti (riga 9), mentre in quello dei *timestamps* vengono inseriti, di iterazione in iterazione, i valori di tempo *t* incrementati di *delta* (riga 10). Quando tutti gli *n* punti sono stati generati (la condizione di uscita dal ciclo WHILE è proprio *i*>*n*), la linestring che descrive l'andamento spaziale viene originata per mezzo dell'operatore OGC `st_makeline(array)`, che collega, interpolandole linearmente, tutte le posizioni <*x,y*> presenti all'interno dell'array *Points[]* (riga 13). Infine, la nuova traiettoria così definita, viene aggiunta all'interno della tabella `Trajectory` (riga 14). Per la generazione delle traiettorie utilizzate negli esperimenti di seguito riportati, abbiamo imposto che i valori delle coordinate *x* e *y* (riga 6 e 7) appartenessero entrambi all'insieme [-10.000, 10.000].
L'esecuzione, come UDF, di `generate_random_trajectory()` in ambiente PostgreSQL porta alla generazione delle traiettorie riportate in Figura 1. Dalla loro mera visione è evidente che quanto più è alto il numero "*n*" di punti che le costituiscono, tanto più "nervosi", "fitti" e "disordinati" risultano i rispettivi andamenti spaziali.



(a) **(b)** **(c)**

Figura 1 – Traiettorie generate con `generate_random_trajectory()` *all'aumentare della cardinalità dei punti per n=50 (a), n=500 (b) e n=1000(c).*

L'osservazione della realtà ci permette di affermare che tali traiettorie sono appropriate a simulare il movimento di m-point all'interno di spazi confinati (ad esempio, riserve per animali, aree chiuse sotto sorveglianza, quali gli aeroporti o aree urbane nel caso di osservazione di veicoli). Si osservi che passare da *n=50* a *n=1000* significa anche coprire un periodo di osservazione del m-point che va da 4 h 20' a 6 giorni 22h 40' (in accordo con l'assunzione che l'intervallo di acquisizione sia fissato in 10 minuti). Ciò che drammaticamente differisce è la distanza coperta dal m-point a seconda che esso sia una persona, un animale o una macchina. Essendo corretto affermare che quando il numero dei punti che costituiscono le traiettorie aumenta, le rispettive evoluzioni spaziali diventano più confuse e disordinate e tendono a riempire l'intero spazio di rappresentazione (come testimoniato da Figura 1), è facilmente immaginabile che tali traiettorie nel relazionarsi tra loro vadano ad incontrarsi geometricamente in un elevatissimo numero di punti. Quest'ultima circostanza non trova riscontro nella modellazione di scenari di effettivo interesse pratico.

Generazione di traiettorie semi-casuali

Con la volontà di migliorare le caratteristiche delle traiettorie da utilizzare nelle campagne di esperimenti e renderle più affini a quelle reali, presentiamo un nuovo algoritmo per la loro generazione, chiamato `generate_actual_trajectory()` il quale riceve in input tre parametri che modellano altrettante proprietà fondamentali del m-point che deve essere creato, nell'ordine:

- *v_max*: indica la massima velocità che esso può raggiungere;
- *duration*: indica la durata (in minuti) dell'osservazione;
- *initial_timestamp*: indica l'istante in cui viene acquisito il primo punto che lo costituisce.

Sebbene non presente tra i parametri, il numero "*n*" di punti di campionamento, elemento importante nella valutazione delle prestazioni, viene determinato in base a una serie di considerazioni sulla velocità massima che il m-point può assumere nel corso della sua osservazione e alla durata di quest'ultima. A tal proposito, con riferimento al parametro *v_max*, esso può essere determinato in analogia a quanto fatto da Brinkhoff in (Brinkhoff, 2002), sia considerando la velocità massima raggiungibile dall'oggetto che si intende modellare (un veicolo può anche superare i 130 Km/h; mentre un essere umano è difficile che possa mantenere una velocità superiore ai 10 Km/h, se non correndo; una formica avanza lentissima mentre una gazzella può raggiungere anche gli 80 Km/h), che valutando le caratteristiche dell'ambiente in cui avviene il movimento (in un centro urbano molto trafficato si cammina a passo d'uomo, mentre in una strada ad elevata comunicazione si possono raggiungere anche i 130 Km/h). Intuitivamente, se un oggetto può spostarsi poco (*v_max* bassa) è inutile campionarne i punti in maniera troppo ravvicinata (ad esempio, un autobus nel bel mezzo di un ingorgo stradale potrebbe rimanere fermo anche per molti

minuti nella stessa posizione), mentre se esso è in grado di muoversi rapidamente può anche raggiungere punti molto lontani tra loro in breve tempo (ad esempio una macchina che percorre un'autostrada non trafficata). In sostanza nella modellazione di una traiettoria, non solo è importante definire i segmenti di linea che la costituiscono, ma anche contestualizzarli allo spazio in cui ci si sta muovendo. Per quanto detto, in `generate_actual_trajectory()` il *passo di campionamento "delta"* viene stabilito in base al valore di *v_max* ed è tanto minore quanto più la traiettoria è veloce. Nel dettaglio riteniamo plausibili le seguenti associazioni:

- *v_max* < 20 Km/h → *delta* = 120 sec;
- 20 Km/h ≤ *v_max* < 40 Km/h → *delta* = 100 sec;
- 40 Km/h ≤ *v_max* < 50 Km/h → *delta* = 80 sec;
- 50 Km/h ≤ *v_max* < 60 Km/h → *delta* = 50 sec;
- 60 Km/h ≤ *v_max* < 70 Km/h → *delta* = 40 sec;
- 70 Km/h ≤ *v_max* < 80 Km/h → *delta* = 30 sec;
- 80 Km/h ≤ *v_max* < 90 Km/h → *delta* = 20 sec;
- *v_max* ≥ 90 Km/h → *delta* = 10 sec.

Fissato *delta* e nota la durata (*duration*) dell'osservazione, il numero "*n*" dei punti che costituiscono la traiettoria rimane univocamente stabilito dal rapporto:

$$n = \left| \frac{duration}{\Delta} \right|$$

Analogamente a quanto fatto in `generate_random_trajectory()` i *timestamps* di acquisizione dei punti possono essere determinati aggiungendo iterativamente la quantità *delta* all'ultimo valore calcolato. La scelta di specificare il *timestamp* iniziale e di non recuperarlo attraverso l'operatore `now()` è dovuta alla considerazione che definendo a priori il momento di inizio del campionamento e la sua durata complessiva è possibile avere il pieno controllo dell'estensione temporale del m-point e del suo posizionamento sull'asse dei tempi.

Particolarmente interessante è il meccanismo adottato per la generazione *semi-casuale* dei punti che determinano l'evoluzione spaziale dell'oggetto modellato. Si parte dal presupposto che nota la velocità estrema raggiungibile dal m-point, è possibile determinare la distanza massima (*d_max*) che esso può coprire nell'arco di tempo *delta*, sfruttando la ben nota formula *spazio = velocità * tempo* da cui segue immediatamente che:

$$d\_max = v\_max * delta$$

Il generico punto P_{i+1} può, di conseguenza, essere generato in maniera più oculata scegliendone le coordinate all'interno di una regione circolare di raggio *d_max* e centro nel punto P_i (da qui la necessità di tenere memoria della posizione campionata nell'iterazione precedente). In Figura 2 viene graficamente sintetizzato il ragionamento appena esposto: in (a) si mostra la costruzione di un buffer "Θ_1" di raggio *d_max* intorno al primo punto P_1; in (b) si sceglie il secondo punto P_2 all'interno di Θ_1 e si costruisce intorno ad esso il nuovo buffer Θ_2 sempre di raggio *d_max*; in (c) il successivo punto P_3 viene scelto sul confine di Θ_2 e con centro in esso viene tracciato il nuovo buffer Θ_3. L'adozione della tecnica appena descritta permette di contemplare tutte le situazioni in cui il m-point si muove a velocità variabile da 0 a *v_max* durante l'intervallo di campionamento. Solo quando il punto appartiene alla linea di confine del buffer Θ (vedi Figura 2-c) si può assumere che esso abbia mantenuto velocità costante pari a quella massima durante il periodo *delta*.

L'algoritmo proposto a seguire sintetizza la sequenza di operazioni necessarie alla realizzazione di `generate_actual_trajectory()`.

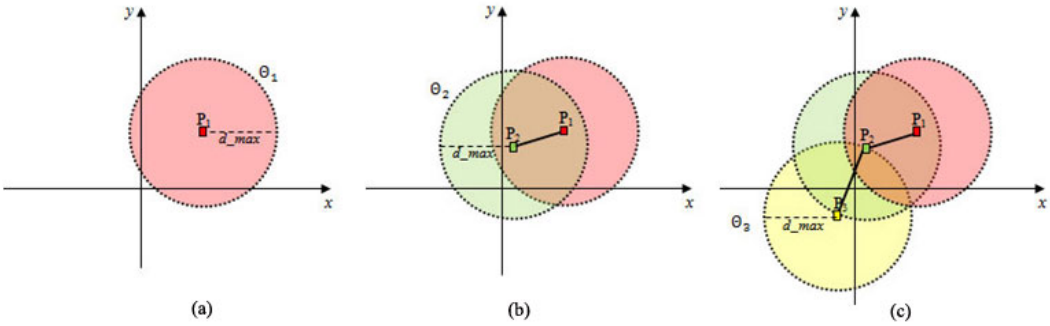


Figura 2 – Visualizzazione del metodo di generazione degli andamenti spaziali dei m-points in `generate_actual_trajectory()`.

Algoritmo: generate_actual_trajectory (*integer* v_max, *integer* duration, *timestamp* t_ini)

Input: i valori *v_max, duration* e *t_ini* che indicano rispettivamente la velocità massima del m-point espressa in Km/h, la durata dell'osservazione espressa in minuti e il *timestamp* di inizio del campionamento.

Output: Viene aggiunta alla tabella `Trajectory` del database una nuova traiettoria specificata per mezzo della `linestring` che ne modella l'andamento spaziale e l'array di timestamps che ne descrive l'evoluzione temporale.

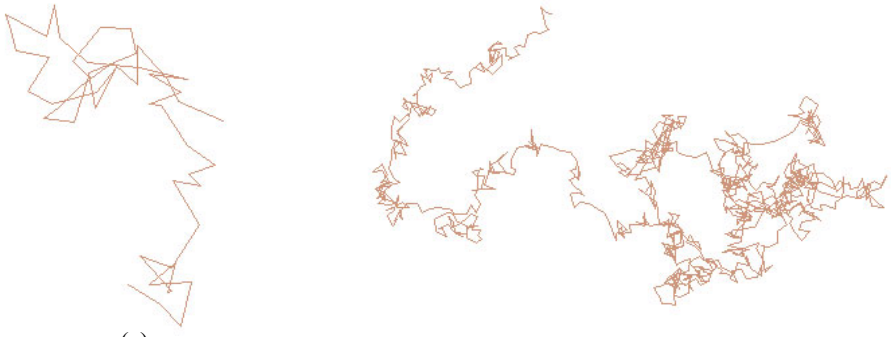
Metodo:

1. Siano *Points[]* e *Times[]* due array di oggetti rispettivamente di tipo `point` *e* `timevalues with time zone`, inizialmente vuoti.
2. Determina il passo di campionamento *delta* in base al valore *v_max*;
3. *n* = |*duration*60/delta*|;
4. *d_max* = *(v_max*delta*1000)/3600*;
5. Genera in maniera casuale il primo punto *p* ;
6. *Points*[1] = *p*;
7. *Times*[1] = *t_ini*;
8. *i=2;*
9. **WHILE(** *i* ≤ n **) LOOP**
10. Costruisci il buffer Θ di raggio *d_max* e centro *p*
11. Genera una coppia casuale di coordinate <*x,y*> appartenente a Θ
12. *p* = `st_makepoint`(*x,y*)
13. *Points*[*i*] = *p*
14. *Times*[*i*] = *t_ini+delta;*
12. **END LOOP**;
13. *line* = `st_makeline`(*Points[]*)
14. INSERT INTO *Trajectory(shape,timevalues) VALUES (line,Times[])*
15. **END**

La generazione casuale del primo punto (riga 5) può essere ottenuta in maniera del tutto analoga a quanto previsto per `generate_random_trajectory(int n)` alle righe 6, 7 e 8. A supporto della costruzione del buffer territoriale (riga 10), può essere utilizzato l'operatore OGC `st_buffer(geometry,float)`, mentre per dare origine a una coppia casuale <*x,y*> appartenente a ⊖ basta generare due valori random compresi nell'intervallo [-*d_max*, *d_max*] e sommarli alle coordinate *x* e *y* del punto precedentemente acquisito. Solo successivamente, per mezzo dell'operatore OGC `st_contains(geometry,geometry)`, si può procedere a verificare se quanto ottenuto sia contenuto all'interno di ⊖. La Figura 3 mostra alcuni esempi di traiettorie ottenute utilizzando l'algoritmo appena presentato. Nel dettaglio per ciascuna di esse sono stati impostati i seguenti parametri:

a) *v_max* = 60 Km/h, *duration* = 34 min → Δ = 40 sec, *n*=51;
b) *v_max* = 60 Km/h, *duration* = 667 min (più di 11 ore) → Δ = 40 sec, *n*=1000;

Come suggeriscono le illustrazioni stesse, gli andamenti spaziali risultanti sono paragonabili agli spostamenti di un veicolo all'interno di una rete stradale più o meno estesa. D'altra parte, quantificare in 60 Km/h la velocità massima raggiungibile dall'oggetto modellato equivale a contestualizzarlo all'interno di una rete stradale urbana al più periferica (basti considerare che in una arteria autostradale sono ammessi valori di velocità molto più elevati).



(a) (b)

Figura 3 – Traiettorie generate con `generate_actual_trajectory()` *all'aumentare della cardinalità dei punti per n=51 (a) e n=1000(b).*

Il fatto che le traiettorie generate per mezzo di `generate_actual_trajectory()` seguano un andamento spaziale meno nervoso rispetto a quelle ottenute con `generate_random_trajectory()` (vedi Figura 1 e Figura 3 per un confronto diretto a parità di "*n*") comporta anche che nel relazionarsi tra loro i m-points si incontrino in meno punti (come testimoniato da Figura 4), situazione auspicabile in un ambiente di analisi comparativa "*semi-realistico*" costruito ad hoc per testare le prestazioni di operatori spazio-temporali.

Conclusioni

Gli algoritmi proposti rappresentano una valida alternativa alle complesse procedure ad oggi note per la generazione delle traiettorie utilizzate nell'ambito dell'analisi sperimentale di operatori che agiscono su punti di movimento. In particolar modo, `generate_actual_trajectory()` consente di modellare in maniera più che accettabile i percorsi seguiti da autovetture o altri mezzi di trasporto, entità comunemente soggette a valutazioni di carattere spazio-temporale.

D'altra parte, avere la possibilità di studiare preliminarmente le prestazioni di operatori che agiscono su oggetti la cui posizione cambia nel tempo, permette di scegliere le soluzioni più efficienti per la loro gestione e facilita l'adozione di eventuali interventi correttivi volti a eliminare comportamenti indesiderati. In definitiva quanto proposto rappresenta un strumento software di semplice implementazione e utilizzo, di ausilio nello studio e nella definizione di operatori spazio-temporali, anche complessi.

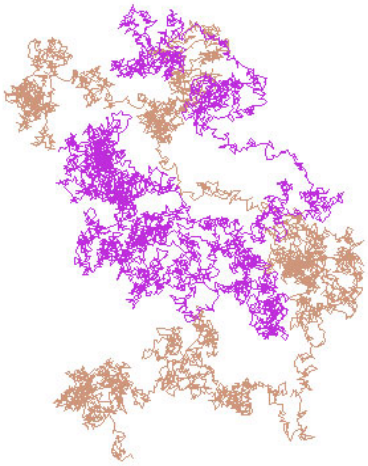


Figura 4 – Due m-points di 5000 punti ciascuno generati con `generate_actual_trajectory()`.

Bibliografia

Brinkhoff, T.: A Framework for Generating Network-Based Moving Object. In GeoInformatica Vol. 6, No. 2: 153-189, 2002

Di Felice, P. and Orsini, M.: Spatio-temporal intersection of trajectories under uncertainties. IXX Simposio sui Sistemi Evoluti per Basi di Dati, 26 Giugno 2011, Maratea.

Pelekis, N., Frentzos, E., Giatrakos, N, Theodoridis, Y.: Supporting Movement in ORDBMS – the 'HERMES' MOD Engine. Technical Report Series, Departement of Informatics, University of Piraeus, UNIPI-INFOLAB-TR-2010-01, July 2010.

Theodoridis, Y., Silva, J.R.O., Nascimento, M.A.: On the Generation of Spatiotemporal Datasets. In *Proceedings 6th International Symposium on Large Spatial Databases*, Hong Kong, China; Lecture Notes in *Computer Science* Vol. 1651: 147-164, 1999.