IDENTIFICACIÓN DE PATRONES Y ALGORITMOS DE CONSOLIDACIÓN EN BASES DE DATOS DE POSICIONAMIENTO

Pilar Barbero Iriarte

10 de diciembre de 2015

Universidad de Zaragoza

TABLE OF CONTENTS

INTRODUCCIÓN

Contexto:

- Empresa Zaragozana de telecomunicaciones.
- Almacenamiento de posiciones GPS de sujetos.
- Capacidad de guardado de posiciones limitada.

Problemas:

- Exceso de éstas.
- No existe preprocesado antes de la inserción.
- No existe postprocesado después de la inserción.
- No todas aportan información.

INTRODUCCIÓN

Objetivo:

- Eliminar posiciones repetidas.
- Eliminar posiciones que no aporten información.

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea
- **Orientación**: entero que representa la orientación respecto al norte en grados

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea
- **Orientación**: entero que representa la orientación respecto al norte en grados
- **Cobertura**: booleano que indica si tiene cobertura (n. satélites)

ANÁLISIS DE LOS DATOS

- **Id**: Identificador numérico
- **IdServidor**: Identificador numérico del servidor que realiza la inserción
- **Recurso**: identificador del sujeto que transfiere la posición
- **Latitud**: real que representa la latitud GPS
- **Longitud**: real que representa la longitud GPS
- **Velocidad**: entero que representa la velocidad instantánea
- **Orientación**: entero que representa la orientación respecto al norte en grados
- **Cobertura**: booleano que indica si tiene cobertura (n. satélites)
- **Error**: error en la toma de posición

¿Cómo abordar el problema?

- Desarrollo de algoritmos de consolidación a través de nociones de distancia y tiempo.
- Uso de algoritmos de *clustering* con el fin de identificar varias posiciones con su centro del clúster y consolidarlas en ésta.

POSICIÓN

Se define la clase `Position` en Python de la siguiente manera:

```
class Position:
    def __init__(self, id, resource, lat, lon, speed, track, date):
        self.id = id
        self.resource = resource
        self.lat = lat
        self.lon = lon
        self.speed = speed
        self.track = track
        self.date = date
```

CLÚSTER

Se define la clase de `Clúster` en Python de la siguiente manera:

```
class Cluster:
"Cluster of points"
def __init__(self, center, points):
self.center = center
self.points = points
```

NOCIONES DE VECINDARIO

Distintas nociones de vecindario para los algoritmos de consolidación simple,

- Vecindario utilizando la distancia euclídea
- Vecindario involucrando velocidad
- Vecindad t_0-alcanzable
- Vecindad involucrando el tiempo

VECINDARIO UTILIZANDO LA DISTANCIA EUCLÍDEA

Utilizando la distancia euclídea, definimos un vecindario de la siguiente manera:

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < \varepsilon$$

donde p es un punto con latitud lat_p y longitud $long_p$.

```
def IsInNeighEUSimple(self, q, eps):
        return self.distance_eu(q) < eps
```

VECINDARIO INVOLUCRANDO VELOCIDAD

En el momento que se toma la posición p_0, aparte de la latidud y su longitud, se toma la velocidad instantánea del sujeto. A mayor velocidad, puntos más alejados de lo que consideraríamos en el primer caso (fuera de nuestro vecindario simple), podrían estar dentro de nuestro nuevo radio, que dependería de la velocidad instantánea.

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < \varepsilon \cdot vel_{p_0}$$

```
def IsInNeighSpeedRelative(self, q, eps):
if self.speed != 0:
return self.distance_eu(q) < eps * self.speed
else:
return False
```

VECINDAD t_0—ALCANZABLE

Fijando un intervalo de tiempo t_0, se define una vecindad t_0-alcanzable como aquellos puntos que nuestro sujeto puede alcanzar en un tiempo t_0. Un sujeto que se desplace a velocidad reducida, tendrá una vecindad t_0-alcanzable más reducida que otro que se desplace a una velocidad superior. Redefiniremos el radio de nuestro vecindario a través de la velocidad instantánea que lleve nuesto sujeto, es decir, $vel_{p_0} \cdot t_0$.

$$d_E(p_0, p) = \sqrt{(lat_p - lat_{p_0})^2 + (long_p - long_{p_0})^2} < vel_{p_0} \cdot t_0$$

Éste es un caso concreto del vecindario involucrando la velocidad.

```
def IsInNeighT0Reachable(self, q, t0):
        return self.distance_eu(q) < t0 * self.speed
```

VECINDAD INVOLUCRANDO EL TIEMPO

Las posiciones de nuestros sujetos vienen muestreadas además con el instante en el que fueron tomadas. Podemos considerar que el tiempo entre tomas también es una distancia y definir un vecindario. Definimos esta distancia temporal como la resta de ambos instantes, y el vecindario como:

$$d_T(p_0, p) = time_p - time_{p_0} < \delta$$

```
def is_neighboorhoudByTime(self, q, lapse):
time1 = time.mktime(self.date.timetuple())
time2 = time.mktime(q.date.timetuple())
return abs(time1 − time2) < lapse
```

CONSOLIDACIÓN POR DISTANCIA

Utilizando los tres tipos de vecindarios que hemos definido, definimos el siguiente método que realizará la consolidación del tipo que le indiquemos:

```
1: function ConsolidationByDistance(positions, typeOfDistance, eps, t0)
2:     for each pos in positions do
3:         if pos.IsInNeighBorhood(typeOfDistance, next(pos), eps) then
4:             Remove position in DB
5:         else
6:             Maintain position in DB
7:         end if
8:     end for
9: end function
```

CONSOLIDACIÓN POR TIEMPO

Se fija un lapso de tiempo que se debe cumplir entre posición y posición, y se eliminan todas aquellas que estén cuya distancia temporal con su siguiente esté por debajo de este lapso fijado.

```
1: function ConsolidationByTime(positions, lapse)
2:     for each pos in positions do
3:         nextpos = pos++
4:         if IsInNeighboorhodByTime(nextpos, pos, lapse) then
5:             Remove pos
6:         end if
7:     end for
8: end function
```

CONSOLIDACIÓN POR ADELGAZAMIENTO

Se puede recurrir a un tipo de consolidación en la cual dada una lista de posiciones normalmente antiguas, se elimine un subconjunto de estas, por ejemplo, 3 de cada 5.

```
1: function ConsolidationByThinning(positions, j, k)          ▷ j < k
2:     for each pos in positions do
3:         if position.Index %k == 0 then
4:             for i = 0; i < k; i++ do
5:                 Remove position with index == position.Index
6:             end for
7:         end if
8:     end for
9: end function
```

K-means es un étodo eficiente de *clustering* que tiene como objetivo la partición de un conjunto de n elementos en k grupos distintos. Datao un conjunto de n elementos, se construye dicha partición $S = \{S_1, S_2, \ldots, S_k\}$ con el fin de minimizar el término del error cuadrático:

$$\sum_{i=1}^{n} \sum_{x \in S_i} \mathrm{d}(x, m_i)$$

donde m_i es el centro de cada clúster S_i y $\mathrm{d}(x, m_i)$ es la distancia definida entre el punto x y m_i.

K-MEANS

Se procede:

1. Se prefija un número de clústers.
2. Se asigna cada punto a su clúster de manera aleatoria.
3. Se itera sobre cada punto, encuentra el centro de clúster más cercano y se lo asigna a dicho clúster.
4. Se calcula el error y se reitera hasta que este error se minimiza o estabiliza.

Inconvenientes:

- Número de clústers prefijado.
- K-means no determinístico.
- No hay puntos considerados ruidos, sólo clústers unipuntuales.

DBSCAN

DJ-CLÚSTER

COMPARATIVA TÉCNICAS CLUSTERING

Método	Tiempo	N.	Iteraciones
K-means	0.69 secs	500	9
DBSCAN	2 min 30 secs	9	111
DJ-Cluster	0.37 secs	22	11

- DBSCAN más lento de todos.
- DBSCAN consolidación mayor.
- K-means se queda en 500 clústers.
- DJ-Clúster baja de los 500.
- DJ-Clúster menor tiempo de ejecución.

COMPARATIVA TÉCNICAS SIMPLES

Método	Tiempo	N.
Cons. por adelgazamiento	<0.01 sec	800
Cons. por distancia simple	<0.01 sec	507
Cons. por distancia t_0−alcanzable	<0.01 sec	21
Cons. por tiempo	<0.01 sec	1786

- Alternar distintos tipos de consolidación en función del espacio crítico en ese momento.

- Algoritmos de consolidación simples son *simples*, pero eficaces.

- Algoritmos de consolidación simples son *simples*, pero eficaces.
- Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.

- Algoritmos de consolidación simples son *simples*, pero eficaces.
- Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.
- Algoritmos de *clustering* más avanzados, pero más complejos a la hora de implementar.

- Algoritmos de consolidación simples son *simples*, pero eficaces.
- Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.
- Algoritmos de *clustering* más avanzados, pero más complejos a la hora de implementar.
- Importante un procesado previo.

- Algoritmos de consolidación simples son *simples*, pero eficaces.
- Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.
- Algoritmos de *clustering* más avanzados, pero más complejos a la hora de implementar.
- Importante un procesado previo.
- A la hora de recuperar una traza con los datos borrados, mejor *DJ-Clúster*

- Algoritmos de consolidación simples son *simples*, pero eficaces.
- Noción de vecindario distinto al euclídeo implementada en algoritmos de consolidación simple.
- Algoritmos de *clustering* más avanzados, pero más complejos a la hora de implementar.
- Importante un procesado previo.
- A la hora de recuperar una traza con los datos borrados, mejor *DJ-Clúster*
- Noción de ruido de DBSCAN importante, tanto DJ-Clúster como K-means sólo encuentra clústers de tamaño 1.

DEMO

Demostración

PREGUNTAS

Preguntas

CÓDIGO

http://github.com/pbarbero/TFM