



Universidad
Zaragoza

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

PILAR BARBERO IRIARTE

Director: *Tomás Alcalá Nalváiz*

13 de diciembre de 2015

Universidad de Zaragoza

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.

Objetivo:

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

- **Id:** Identificador numérico

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

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

- **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

- **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

- **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

- **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

- **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)

- **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.

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
```

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
```

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

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
```

En el momento que se toma la posición p_0 , aparte de la latitud 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
```

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 nuestro 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
```

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_neighborhoodByTime(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**

EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR DISTANCIA

Se fija un $\varepsilon = 0,0001$ y se realiza una consolidación utilizando la distancia euclídea:

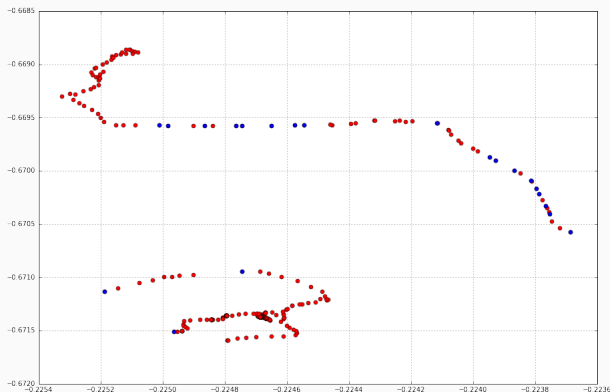


Figura 1: Resultado consolidación por distancia Simple

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 IsInNeighborhoodByTime(nextpos, pos, lapse) then  
5:       Remove pos  
6:     end if  
7:   end for  
8: end function
```


EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR TIEMPO

Se realiza una consolidación por tiempo con un lapso de 20 segundos:

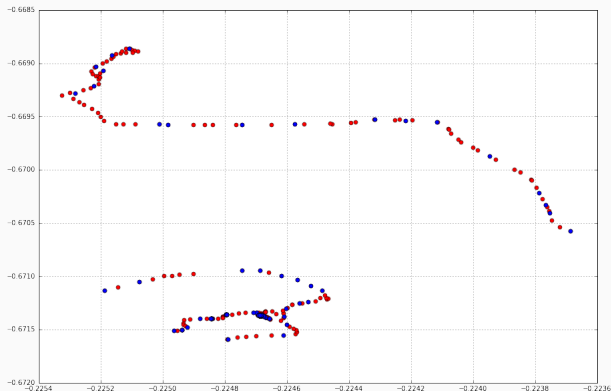


Figura 2: Resultado consolidación por tiempo

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
```

EXPERIMENTO UTILIZANDO CONSOLIDACIÓN POR ADELGAZAMIENTO

Se realiza una consolidación por adelgazamiento, se mantienen 2 posiciones de cada 5:

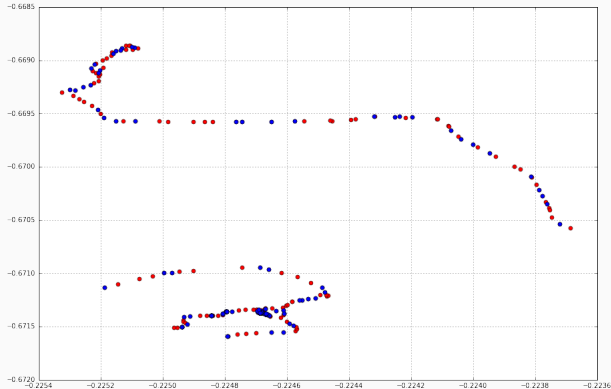


Figura 3: Resultado consolidación por adelgazamiento

Algoritmos de consolidación asociados a métodos de clustering

Un análisis clúster es un conjunto de técnicas multivariantes utilizadas para clasificar a un conjunto de individuos en grupos homogéneos. Hemos elegido una serie de técnicas de aprendizaje no supervisado ya que éste parte de que no hay un conocimiento a priori y es útil en técnicas de compresión de datos.

- K-means
- DBSCAN
- DJ-Clúster

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

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

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

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.

EXPERIMENTO CON K-MEANS

Se utiliza el software **Weka** sobre un sujeto con 2000 posiciones para una consolidación al 25 %, es decir, a 500.

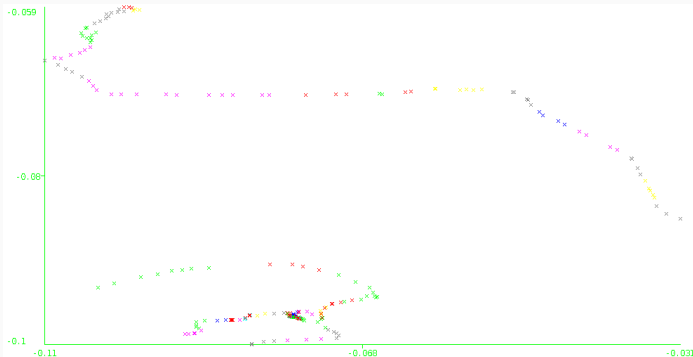


Figura 4: Resultado de K-means

Resultados:

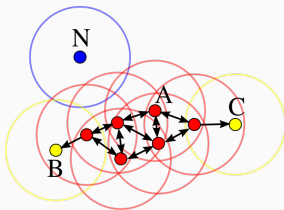
- 10 iteraciones
- Error cuadrático: 0,117363
- número de posiciones que ha agrupado por clúster entre 1 y 9

Inconvenientes:

- Número de clústers prefijado a 500.
- K-means no determinístico.
- No hay puntos considerados ruidos, sólo clústers unipuntuales.

DBSCAN es un algoritmo de clustering basado en la densidad por lo que encuentra el número de clústers comenzando por una estimación de la distribución de densidad de los nodos correspondientes.

- Punto p núcleo: si posee un número mínimo de puntos ($minPts$) en su vecindario sobre ε .
- Punto alcanzable desde p : si existe un camino $p_1 = p, \dots, p_n = q$ donde p_{i+1} es alcanzable por p_i .
- Aislados: puntos no considerados ni núcleo ni alcanzables.



DBSCAN necesita de dos parámetros: ϵ y $MinPts$.

```
1: function DBSCAN(positions, eps, minPts)
2:   C = 0
3:   for each pos in positions do
4:     if pos has been visited then
5:       Continue next position
6:     else
7:       Mark pos as visited
8:       N(pos) = NeighborPts(pos, eps)
9:       if length(N(pos)) < MinPts then
10:        Mark pos as noise
11:       else
12:        C = next Cluster
13:        expandCluster(pos, N(pos), C, eps, MinPts)
14:       end if
15:     end if
16:   end for
```

Mismos datos que para K-means, sujeto con 2000 posiciones.

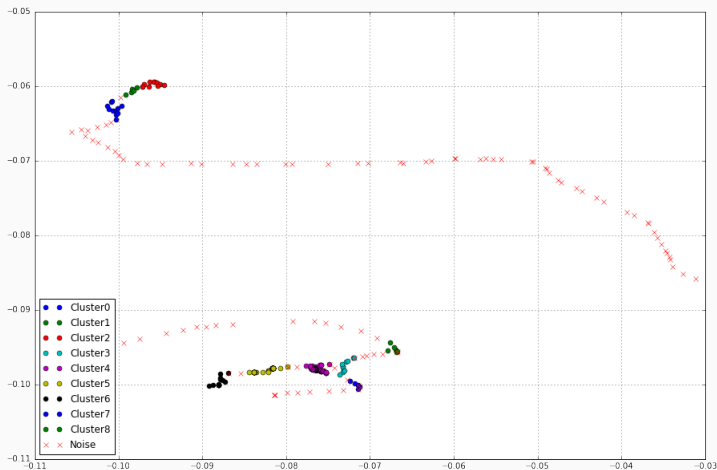


Figura 5: Resultado de DBSCAN

Density-Joinable Clúster es un tipo de algoritmo de clustering basado en densidades de puntos.

```
1: for each  $p$  in set  $S$  do
2:   Compute neighborhood  $N(p)$  for  $\varepsilon$  and  $MinPts$ 
3:   if  $N(p)$  is null ( $|N(p)| < MinPts$  for  $\varepsilon$ ) then
4:     Label  $p$  as noise
5:   else if  $N(p)$  is density-joinable to an existing cluster then
6:     Merge  $N(p)$  with the cluster which is density-joinable
7:   else
8:     Create a new cluster  $C$  based on  $N(p)$ 
9:   end if
10: end for
```

EXPERIMENTO CON DJ-CLUSTER

Utilizando Weka dado un $\varepsilon = 0,0001$ y $minPts = 2$:

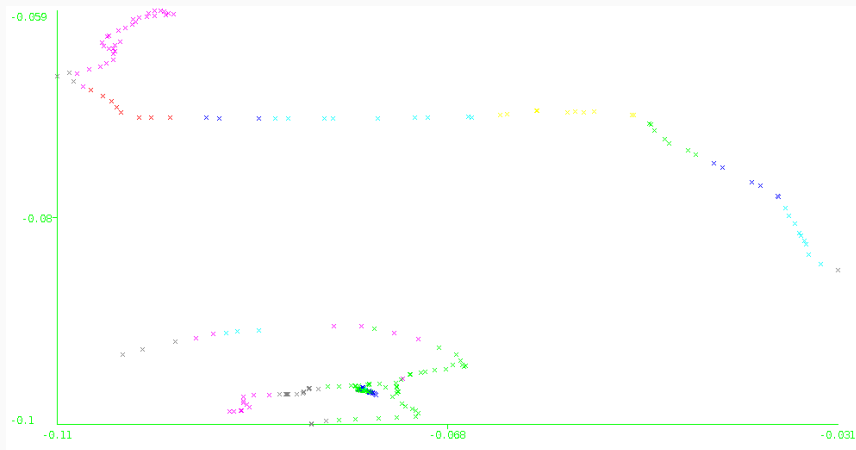


Figura 6: Resultado de DJ-Cluster

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.

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.

- 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 procesamiento 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.

Demostración

http://github.com/pbarbero/TFM

pbarbero / TFM

Watch 1 Star 0 Fork 0

Code Issues Pull requests Wiki Pulse Graphs Settings

Trabajo Fin de Máster Modelización e Investigación Matemática, Estadística y Computación — Edit

88 commits 2 branches 0 releases 1 contributor

Branch: master New pull request New file Find file HTTPS https://github.com/pbarbe Download ZIP

pbarbero	frame.comparatiya	Latest commit deebfea 5 hours ago
consolidationExpert	changes in checkpoints	7 days ago
consolidationSimple	changes in checkpoints	7 days ago
data	reorder files	7 days ago
dataAnalysis	reorder files	7 days ago
defensa	frame comparativa	5 hours ago
papeles	reorder files	7 days ago
report	lasts changes to report	7 days ago
reuniones	add own algorithms to comparative	9 days ago
.gitignore	update gitignore	7 days ago
License.md	Create License.md	9 days ago
README.md	change readme	a day ago

README.md

Identificación de patrones y algoritmos de consolidación en bases de datos de posicionamiento

Preguntas