**软著-基于GeoHash编码的时空轨迹相似度查询系统**

**设计说明书**

**苏州大学 丁庆祝**

**2016.06.18**

**基于GeoHash编码的**

**时空轨迹相似度查询系统**

**设计说明书**

## 一 项目背景

近年来,随着物联网技术及传感器技术的不断推广，伴着大数据分析的浪潮，各种基于时空信息的轨迹数据的分析的技术开始逐渐走入人们的生活当中。比如各种智能交通应用、对轨迹进行兴趣点的分析和推荐，或者公安民警办理涉车案件、对犯罪嫌疑车辆轨迹分析、城市道路规划等等。拓展开来说，时空轨迹相似度分析更可以看作一种行为分析，他可以根据各个事件发生的时间点和坐标点来对行为进行一种索引，是一种对大量数据进行高效统计与分析的方式。因此，时空轨迹相似度分析在生产生活当中有着重要的意义、非常现实的应用价值、以及广阔的应用前景。

时空轨迹相似度分析主要分为空间轨迹相似度度量标准定义、时间对轨迹相似度的权重分析以及大数据集中的相似轨迹查询三部分。实际应用过程中，具体的时间因素主要是在空间轨迹上做的拓展，因此最重要的部分就是空间轨迹相似度的定义计算以及在大数据集中对目标轨迹的相似轨迹进行高效查询。

计算空间轨迹相似度比较常见的度量标准有动态时间规整(DTW)、最长公共前缀(LCSS)、编辑距离(EDR)等，但是DTW算法对噪声比较敏感、对“降采样”和“超采样”的轨迹相似度的判断效果不尽如人意；LCSS和EDR虽然对噪声有抵抗性，但是他们特别依赖于阈值参数的选择，而这就导致了他难以实现通用化。相比之下单向距离(OWD)和归并距离(MD)等方法，不依赖于阈值参数，对于噪声的敏感度较低，比较适合作为通用的轨迹相似度度量标准。

对于在大数据集中进行相似轨迹查找，遍历全集的做法显然是不可取的。通常的做法是通过对轨迹点进行索引，然后根据每个轨迹点附近的相关点来确定一个较小的候选轨迹的集合，再在这当中进行相似轨迹的最终确认。这个建立索引的过程通常是使用DBA中的Spatial扩展，具体的实现方式是R树家族的数据结构。但是，这些功能过度依赖于数据库，移植性并不好，况且除了数据库的性能消耗外，程序本身也要进行计算。而且由于R树自身也存在不足，当数据量增大时，不仅是插入数据的性能会降低，而且查找的性能也不乐观。考虑到这些不足，本系统采用的是GeoHash编码技术对所有的轨迹点进行编码，然后在目标轨迹点的临近范围上由小到大的进行搜索。这样一来就能避免过度依赖DBA，适用性更好，而且效率也。

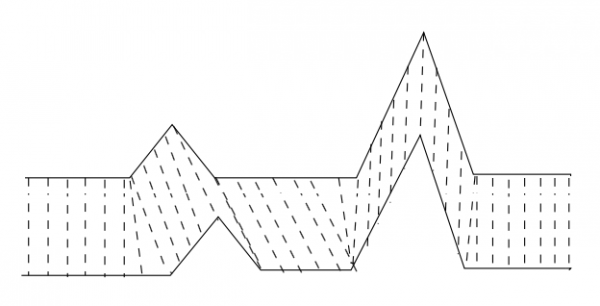
## 二 理论基础

### 2.1 动态时间规整(Dynamic Time Warping)

DTW算法又叫动态时间规整（ Dynamic Time Warping），是一个比较简单的dp算法。常用于不等长的离散的路径点的匹配问题，在孤立词语音识别、手势识别、数据挖掘和信息检索等领域有着很不错的表现。

DTW解决的问题是，给定两个序列 （A1,A2,A3,A4…An） 和 (B1,B2,B3,B4…Bm) ，其中每一个元素可以都是一个二维坐标点或者是更高维度的坐标。我们现在需要求出一个“距离”，使得他能够表示这两个路径的相似度。

很明显，如果m等于n，那么我们可以很方便的用对应节点（下标相等）之间的欧氏距离（也可以是其他类型的距离）之和来表示这个"距离“，这看上去还是能让人信服的。但是当m和n不等的时候，我们就发现这种办法就不适用了。但是我们完全可以仍然采用这个思想，只不过与之前每个节点都是一一对应不同，我们可以令其中的某些节点是一对多的对应关系，如下图所示：



这样一来，我们就可以通过计算新的对应点之间的距离之和来表示这两个路径之间的距离了。而具体的实现过程，其实就是一个动态规划了。

令dtw[i][j]表示A序列的前i个元素与B序列的前j个元素匹配后得到的最小距离（下标从1开始），dis[i][j]表示Ai与Bj的距离。显然，这时候Ai必然和Bj匹配。那么我们很容易得到下面的递推关系式（考虑边界条件）：

i=j=0:

dtw[i][j]=0

i=0 or j=0:

dtw[i][j]=∞

for 1≤i≤n, 1≤j≤m:

dtw[i][j]=dis[i][j]+min

最后dtw[n][m]就是我们所求的距离，复杂度O(n\*m)。

### 2.2 单向距离(One Way Distance)

OWD算法也是一种描述两个路径之间相似度的方法，最早大概提出于2006年左右。基本思想是把路径之间的距离转化为点到路径的距离再加以处理。实现算法如下：

在定义路径间的距离Dowd之前，我们先定义点到路径的距离Dpoint:对于点p和一个由多个点组成的路径T，定义他们之间的距离为

其中表示p.q之间的欧氏距离。

然后，我们定义路径T1到路径T2的单向距离为：

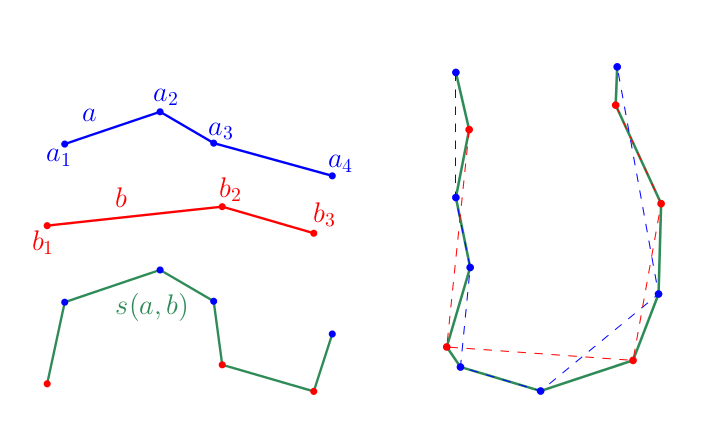
(对于非离散的路径，我们可以把他看成是一个积分过程)

很容易看出来，这个单向距离不具有对称性，即与不一定相等。那么为了得到一个对称的结果，我们定义一个新的度量标准:

这就是OWD距离中最终用来判定路径相似度的标准。

### 2.3 归并距离(Merge Distance)

MD算法解决的问题是，给定两个序列（A1,A2,A3,A4…An）和(B1,B2,B3,B4,…Bm)，其中每一个元素可以都可以是一个二维坐标点或者是更高维度的坐标。现在我们需要找到一条路径，使他经过这两个序列的所有点，且保证若 i<j 那么 Aj 一定在 Ai 后面出现（对于 B亦然）。显然这样我们可以有多种结果，那么我们只要取最短的那个路径来作为最后能够代表他们相似度的路径。具体要求如下图所示：



对于左边的图，下面的那个路径就是对上面的路径进行归并后的最短路径；对于右边的图，绿色的实线路径就是对蓝色和红色的虚线路径归并后的最短路径。

Merge Distance算法最后得出的相似度度量值就是这个最短的路径长度。

具体过程如下：

对于序列A[1:n]，B[1:m]，令MDa[i][j]表示以A[i]为终点的序列A[1:i]和B[1:j]所形成的最短路径的长度；令MDb[i][j]表示以B[j]为终点的序列A[1:i]和B[1:j]所形成的最短路径的长度；dis(X,Y)表示X和Y之间的距离，这样我们构造如下的递推关系式：

i=1:

j=1:

for 1<i≤n,1≤j≤m:

For 1≤i≤n,1<j≤m:

这样我们就可以很容易的求出Mda[n][m]和MDb[n][m]，然后取其中的较小值作为最终的长度，复杂度O(m\*n)。

### 2.4 GeoHash编码

GeoHash是由Gustavo Niemeyer（大概于2013年）提出的，目的原本是为地球上的每一个点（根据经纬度）确定一条短的URL作为唯一标识。只是后来被广泛的应用到空间检索方面、尤其是之前提到的POI查询中。这个服务一直在[http://geohash.org](http://geohash.org/)上，上面还有一些具体的介绍。

GeoHash所做的事就是把一个坐标点映射到一个字符串上，每一个字符串代表的就是一个以经纬度划分的矩形区域。比方下面的图就展现了北京地区所在的九个区域，分别是WX4ER、WX4G2等等。



于此同时，每一个区域又可以继续划分为许多个小区域，如下图所示，WX4G0就包含了WX4G09、WX4G0C、WX4G08等等。而且每一个子区域的GeoHash值都是在父区域后面拓展一个字符。  


这样就形成了一个层次分明的结构，越高级的区域GeoHash值越短，表示的区域最大；越小的区域GeoHash值越长，表示的区域越小。

根据上面的意义我们很容易想到他用的是类似四叉树的方法来寻找一个点；换句话说就是不停的在经度和纬度上进行二分类，最终确定到想要的精度，划分的过程下图所示。

第1次划分

|  |  |
| --- | --- |
| 0 | 1 |

第2次划分

|  |  |
| --- | --- |
| 01 | 11 |
| 00 | 10 |

第3次划分

|  |  |  |  |
| --- | --- | --- | --- |
| 010 | 011 | 110 | 111 |
| 000 | 001 | 100 | 101 |

第4次划分

|  |  |  |  |
| --- | --- | --- | --- |
| 0101 | 0111 | 1101 | 1111 |
| 0100 | 0110 | 1100 | 1110 |
| 0001 | 0011 | 1001 | 1011 |
| 0000 | 0010 | 1000 | 1010 |

交替在两个方向进行分割，一个区域计为0，一个区域记为1，并将结果追加在父区域GeoHash值的后面。不断的进行划分直到得到了想要的深度。对于每一个区域最后会得到一个二进制的字符串，然后每5位为一组，用0-9 b-z（去掉a, i, l, o）进行Base32编码即可得到该区域经过编码后的GeoHash值。当然，编码这个步骤只是为了让结果看上去变短而已，实际应用中可以把它变成一个二进制数（由于存在前导0，所以要在最前面加一个1），而且直接用字符串的形式也未尝不可。

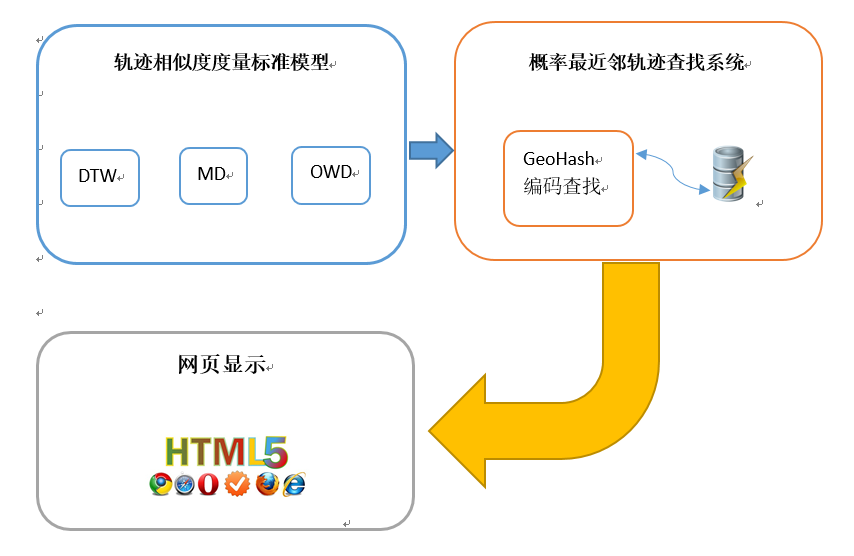
### 2.5 概率最近邻查找模型(Approximate Nearest Neighbor)

如果想要查找某个点附近的Top m个临近点，我们显然不会直接扫描地图中所有的点（这样效率会极其低下），而是用上面的坐标点的定位算法（比如GeoHash编码），将该点定位到一个比较小的区域里（这个区域里有n个临近点，且n>m），然后再扫描这个较小区域里的所有点，用自己定义的权重公式，取出最符合的点。这个方法一般叫做“Filter and Refine”方法，即“先过滤，后提纯”。这样做虽然不能够保证结果完全正确，但是可以通过调节过滤集的大小使得结果的误差在可接受的范围之内。概率最近邻查找模型的优点就是在对结果准确率要求不苛刻的情况下，对于目标坐标点能够十分高效的查找到概率最高的临近点。

## 三 系统框架

**总体框架：**

如下图所示，整个系统主要分成两个部分，第一个部分是轨迹相似度度量标准的设计，作为相似轨迹查找系统的查询依据；第二个部分是概率最近邻轨迹查找系统，将数据经过GeoHash编码后保存到数据库，然后再利用数据库、结合GeoHash进行相似点和相似轨迹的查找。最后将两个系统整合起来在Web端进行显示。整体用Java语言编写，Web部分用Servlet框架，GIS显示部分调用Baidu map Api接口。



**数据库设计：**

在概率最近邻轨迹查找系统中，我们使用mysql数据库设计了如下的表(trajectory)：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Field | Type | Null | Key | Default | Extra |
| id | int(11) | NO | PRI | NULL | auto\_increment |
| longitude | int(11) | NO |  | NULL |  |
| latitude | int(11) | NO |  | NULL |  |
| timestamp | int(11) | NO |  | NULL |  |
| geoHash | varchar(50) | NO |  | NULL |  |
| pathId | int(11) | NO |  | NULL |  |

用这个表来保存每一个轨迹点的信息，包括唯一的标识、经度、纬度、时间戳、GeoHash编码值和所对应的轨迹Id。并且在id、geoHash和pathId字段上加了B树索引，用以加快查询速度。

**系统环境：**整个系统都部署在ubuntu 16.04上，内存4G，CPU i5,硬盘 1TB。Web端使用Tomcat服务器搭建。数据库使用Mysql5.7。

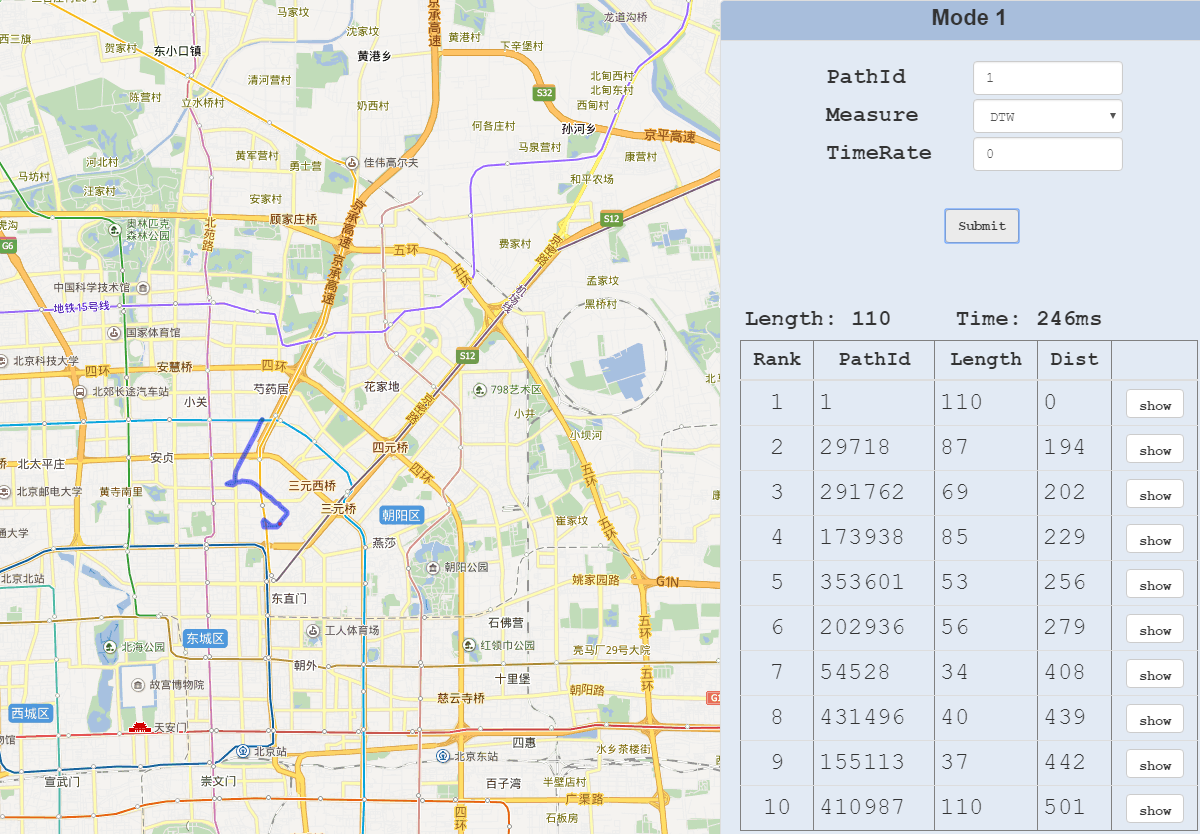
**编程语言：**Java (版本：Jdk8)

## 四 系统演示

本系统使用Web端作为演示，其功能可以方便移植到其他平台，如andriod、ios和windows应用等。Web端分三个模块进行演示。演示所用的数据库中包含约44万个轨迹、1.7亿个坐标点。

**第一模块：**

给定数据库中某条轨迹，查询数据库中和这条轨迹相似度最高的前10条轨迹。输入目标轨迹的Id、采用的度量标准和时间比重；在左边显示出这条轨迹，并且在下方的表格中显示这10条轨迹的具体信息和查询所用的时间。



并且可以进一步在地图中显示查询到的轨迹：



**第二模块：**

输入标准的轨迹数据文件，在数据库中查找与这条轨迹相似度最高的至多10条轨迹。数据文件的格式如下：

不带时间信息的轨迹：

经度，纬度

116.30984,39.90944

116.30911,39.91234

……

带时间信息的轨迹：

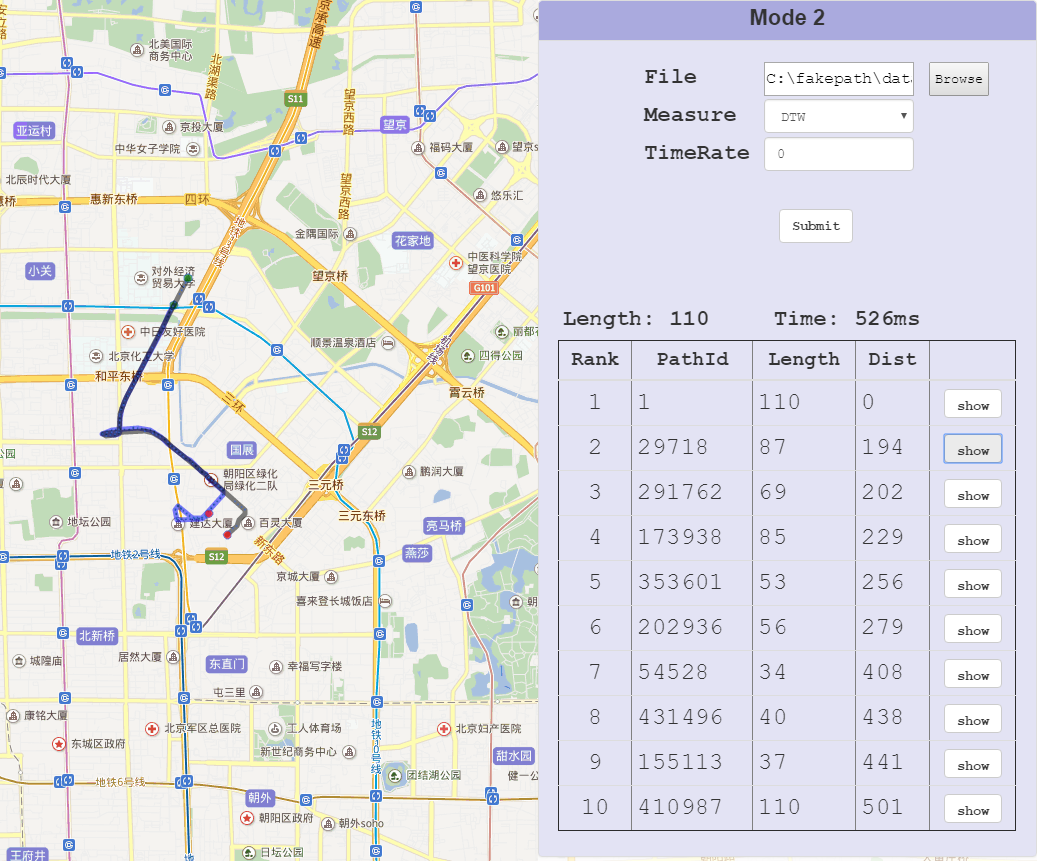
经度，纬度，发生时间

116.30984,39.90944,20150302 11:02:27

116.30911,39.91234,20150302 11:04:03

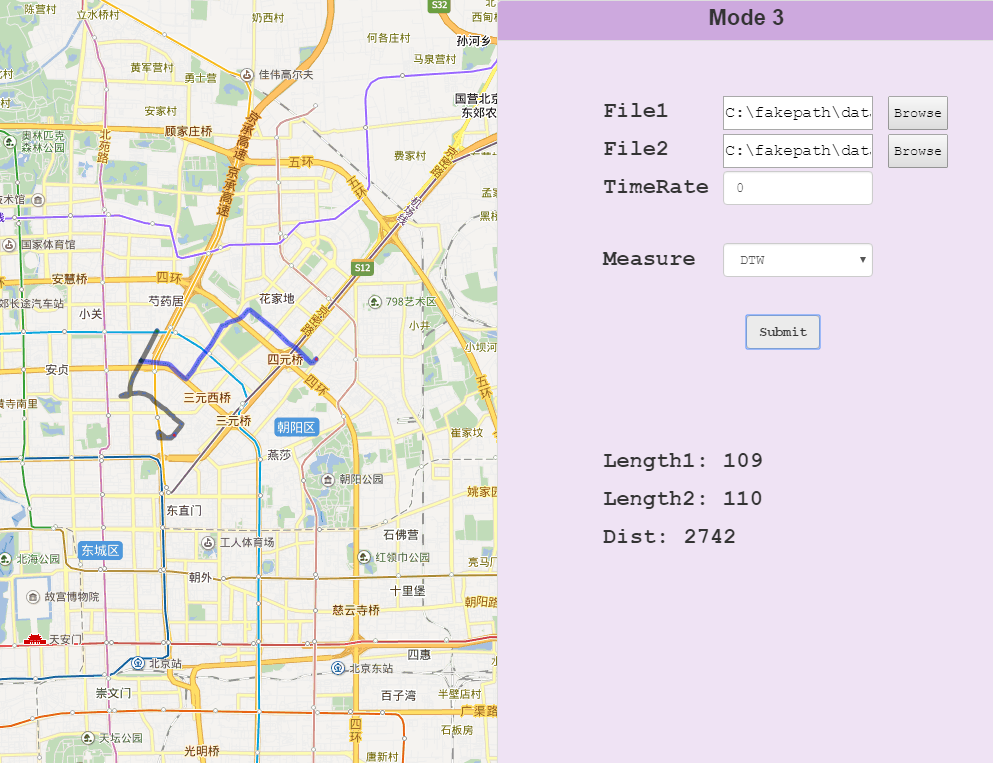
……

运行结果如下：



**第三模块：**

输入两个轨迹文件，计算给定度量标准后两者的相似度 ：



## 五 总结

除了具有高效的查找效率、丰富的评估机制之外，本系统采用的架构还有非常高的可扩展性。一方面，我们可以在轨迹相似度度量模型上进行非常的方便的修改和扩展而不影响系统的其他部分；另一方面，我们也可以在概率最近邻轨迹查找系统中部署上不同类型的数据库，可以很方便的部署上分布式系统。除此之外，他在对时间的处理上也更加灵活，可以根据实际数据的特点来给时间添加不同的权重，非常适合应对不同特点的数据，有着非常广阔的应用前景。