基于GeoHash编码的时空轨迹相似度查询系统功能说明

丁庆祝*

苏州大学 2014级计算机学院

2016年6月

目录

*电子邮件：dqz48548263@qq.com，博客：http://blog.mythsman.com

1 概述

本软件简要实现了针对以经纬度或经纬度带时间定义的不同轨迹之间相似度的计算以及查询。由于现阶段学术界没有提出更加公认有效的轨迹相似度的度量标准，因此相比于提炼出其他相似度度量标准，现阶段本项目主要侧重于提高轨迹相似度的计算和查找性能，即如何在轨迹条目在千万级别以上的数据集中高效的查找到与目标轨迹相似度最高的轨迹。

2 算法设计

2.1 相似度度量算法

本软件集成了常见的三种轨迹间距离的度量算法，包括：“DTW　动态时间规整算法[1]”、“MD　归并路径算法[2]”、“OWD　单向距离算法[3]”。

需要说明的是，这里之所以未采用”LCSS“以及“Edit Distance”家族的算法，是因为这类算法特别依赖于判断“两点近似重合的距离”这一重要的阈值参数。而这个参数通常是凭经验设置，没有一个统一的标准。而且对于规模不同的数据集以及不同形态的轨迹来说，这个值大都不同。因此本项目使用了上述三种非阈值型的相似度度量标准作为参考标准。这三种算法也 各自有其侧重，DTW算法对轨迹点数目不等的轨迹匹配效果好、MD算法对于”降采样”或者”超采样”的抗干扰能力比较好，而OWD则比较适合做归一化。 下面对着三个算法做简要分析。

2.1.1 DTW(动态时间规整)

DTW算法： `http://blog.mythsman.com/?p=2761`

2.1.2 MD(归并距离算法)

MD算法： `http://blog.mythsman.com/?p=2818`

2.1.3 OWD(单向距离算法)

OWD算法: `http://blog.mythsman.com/?p=2852`

[1]Yi, Byoung-Kee, Jagadish, HV and Faloutsos, Christos, Efficient retrieval of similar time sequences under time warping. ICDE 1998

[2]Anas Ismail,Antoine Vigneron.A New Trajectory Similarity Measure for GPS Data, (KAUST) Thuwal 23955-6900, Saudi Arabia

[3]Bin Lin, Jianwen Su, One Way Distance: For Shape Based Similarity Search of Moving Object Trajectories. In Geoinformatica (2008)

2.1.4 未来研究方向

由于上述三种算法各自有不同的侧重点，都从不同的角度反映了轨迹相似的特征。因此相比分别计算三种相似度，我们更期待一种能够糅合这三种相似度度量标准的评价机制，更好的反映出轨迹之间的相互关系。 在接下来的研究过程中，我们将着手设计这样一个综合型更强的评价模型，而不是对着三种算法简单的区别对待。

2.2 轨迹查询算法

轨迹查询、即轨迹的索引，而查询临近的轨迹引通常都会归结到查询临近的坐标点。经典的做法是用R树家族的索引算法，这类算法在当前应用比较广泛，特别是现今的许多数据库系统都推出了Spatial扩展，使得本身较为复杂的操作变得更加易用。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。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变慢，而且查找的性能也不乐观。

因此本软件采用的是先对轨迹点进行GeoHash编码再进行临近点查找的索引算法。利用普通数据库系统，无需采用繁琐的Spatial扩展，就可以实现对于大量轨迹数据的快速插入与查找。相比R树类索引相对高昂的维护代价，利用GeoHash编码可以做到更加轻松的插入和查找。

2.2.1 GeoHash编码

GeoHash 算法：http://blog.mythsman.com/?p=2874

2.2.2 算法设计

给定一条包含n个轨迹点的轨迹T，在N条轨迹中找出相似度最高的M条轨迹。

1、对于待查轨迹T,取出轨迹中的n个轨迹点，对每个点并利用GeoHash编码在数据集中找到距离最近的k个点，总共会得到S个点($S \leq n * k$)。注意到这里选择出来的点并不绝对是最近的点，而是在差错允许的范围内最有可能的点。k的取值越大，结果的准确度越高，反之效率越高，但是k的值过小会导致搜索到的结果不足M。

2、对于找到的S个点，在数据库中查找包含这S个点中任何一个点的轨迹，将这些轨迹作为$Candidates$($Candidates$的个数$\leq S$)。

3、对于所有的$Candidates$，计算他与目标轨迹的相似度的距离，并取相似度最高的M条轨迹。

时间复杂度分析 首先对于轨迹t,利用GeoHash编码查找最近的k个点需要$O(C*k*n)$，

其中C决定于GeoHash编码的精度，C通常小于$log(N)$。 接着对这k个点依次计算相似度并排序，相似度计算的复杂度是$O(n^2)$,排序的复杂度为$O(k*log(k))$。 因此，总体的复杂度约为$O(log(N)k^2log(k)n^3)$。我们可以通过修改参数k的值来达到效率和准确率的平衡。

2.2.3 数据库设计

表结构设计 数据库名叫trajectory，包含一个point表，保存从轨迹中提取出来的坐标点。具体设计如下:

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	

表 1: trajectory.point

point表中的第一个字段表示坐标点的唯一Id，然后是经纬度和时间戳，第四个四段是用字符串表示的GeoHash编码，第五个字段表示该点所在的轨迹的Id。 point表在geoHash字段和pathId字段上各有一个额外的B树索引。

存储空间分析 利用这里的存储方法，每一个点需要存储一次。考虑到地图中的点可以看作是稀疏的，因此不考虑用重合点来压缩存储数据。

2.2.4 未来研究方向

本算法对于常见的运动类轨迹计算效果较好，但是对于那些低速运动或者静止的轨迹计算效果不好。因为在对点进行临近点查找的过程中，那些“相对静止”的轨迹会查找到大量属于本身的轨迹点，从而无法查找到其他的轨迹。这个问题将在今后的研究中加以解决。

对于数据库而言，一方面，我们可以用字符串形式存储的GeoHash再次编码为整型变量，节约存储空间并且提高查找效率；另一方面，由于采用了最为简单的数据库存储方式，因此我们也可以结合NoSQL数据库和Hadoop实现分布式存储计算。

对于相似度最高的轨迹段和轨迹点的处理而言，由于暂时没有更好的办法，最相似的轨迹点就是两个轨迹中距离最近的点($O(n^2)$)；最相似的轨迹段就是穷举两个轨迹中

所有的子轨迹段，并计算归一化后的相似度($O(n^4)$)，效率不高，而且不同相似度的可比性不强。因此未来需要优化算法并构造更加合理的匹配方案来进行处理。

2.3 其他细节说明

2.3.1 对于时间的处理

考虑到数据集的不确定性，我这里对时间的处理采用的是把时间融入轨迹点距离的计算当中，即对于带时间的轨迹点$P_1(x_1, y_1, t_1), P_2(x_2, y_2, t_2), distance(P_1, P_2) = (1 - timeRate) * \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} + timeRate * |t_1 - t_2|$。其中timeRate表示时间在距离中的比重，timeRate为0时就是忽略时间因素，timeRate为1时表示忽略位置因素，具体的值可以根据需求自行决定。 这样我们就把带时间的轨迹与不带时间的轨迹做了统一。

2.3.2 多线程加速计算

对轨迹中的每一个坐标查找临近坐标点的过程是整个算法中最耗时的过程，但是这也适合做多线程查找或分布式计算。分布式计算可以结合数据库进行优化，而多线程处理即使在单机中也会给效率带来极大的提高。实验中，在四个线程的普通商务机中引入多线程运算就能将执行时间缩短到了加速前的30%左右。

3 软件说明

演示软件采用百度地图做GIS系统，使用Servlet开发。

3.1 功能说明

第一模块

第一模块是给定数据库中的一条轨迹的Id，显示与之相似的前10条轨迹（可能不足十条），并在地图上显示。

第二模块

第二个模块是给定一个数据文件(兼容带时间或者不带时间)，显示数据库中与之相似的10条轨迹，并在地图上显示由于题目给出的时间格式与样例文件中的时间格式不统一（说明中要求是YYYYMMDD HH:MM:SS，而数据文件却是yyyy/m/d h:mm的形式），因此我重新规定一下时间格式为YYYYMMDD hh:mm:ss）。

带时间的轨迹:

经度,纬度,发生时间

116.30984,39.90944,20150401 11:02:27

. . . . . .

不带时间的轨迹:

经度,纬度

116.30984,39.90944

. . . . . .

第三模块

第三个模块是给定两个数据文件，计算轨迹相似度。

每个模块都要指定timeRate(即时间在距离中的比重，取0时表示不考虑时间),和measure(三个相似度度量标准)，具体含义见上文。

演示说明

1、演示环境Tomcat8,JDK 8 ,Mysql5.7,win10 xbox录屏软件,电脑配置：i5-6500cpu, 3.20GHz,RAM 4.00GB ,x64处理器。

2、演示用的数据集为北京地区轨迹数据集，共有2240362条轨迹，包含178732637个坐标点。

3、演示前已重启了mysql服务保证不存在数据库缓存，能够反映真实的执行效率。（而且演示中两次查询同一个轨迹可以看出带缓存和不带缓存的执行效率的差别）。

4、演示中所用的k值(对于每个坐标点所考虑的临近点的个数)取的是2，traj为目标轨迹，定义在Display.java中的第61行。事实上我们可以集合数据适当调整这个值，使得效率和精度达到权衡点。

有待提升的功能

更方便数据导入导出和项目部署、更完善的异常处理、更加优美的界面以及更加丰富的展现方式，

3.2 本地搭建

本地搭建步骤

1. 按照上述要求构建数据库， sql命令参考:

   (a) Create database trajectory;

   (b) Use trajectory;

   (c) create table point(id int primary key not null auto_increment,longitude int not null,latitude int not null,timeStamp int not null,geoHash varchar(50) not null,pathId int not null);

   (d) create index geoHashIndex on point(geoHash);(在插入数据后执行)

   (e) create index pathIdIndex on point(pathId);(在插入数据后执行)

2. 编辑Jdbc.java中32-34行，填写用户名密码连接到数据库。

3. 导入数据

   (a) 建议在导入数据之后再创建索引，否则时间消耗特别大。

   (b) 将整理好的所有轨迹数据放在一个文件夹下(不要有子文件夹)，执行Util.java中的“genPointFromDir(String dir,String outputFile)”函数，一次性生成数据文件(样例文件见data.txt)。

   (c) 再在mysql中用load data local infile ”D:/data.txt” into table point;将数据整体导入到point表中。

   由于数据量比较大，一个一个插入相当费时，因此先将所有的数据写入一个数据文件中再一次性倒入表中将会节约很多时间；同时，由于索引会极大的拖慢插入数据的速度，因此建议先导入数据，后建立索引。

4. 在Tomcat服务器中运行，打开http://http://localhost:8080/Cnsoftbei/Display进入系统。