**初版：**

初版的方案大体是这样的：

首先从fp\_daily 中获取北京市20160825这天的所有gps信息，保存列信息是：tm,id,x,y

保存在gps\_data\_simple中

对时间每个10分钟分成一个时间片，总共144个片；

对空间分块，把北京看成是一个100km\*100km的正方形，分成400\*400的block，每个格子250m\*250m,按照gps数据+时间信息，计算gps\_data\_simple表中每条记录的时间和空间属于哪个blockid，公式如下：

Blockid = cur\_time\_index \* num\_rows \* num\_cols + cur\_gps\_x\_index \* num\_cols + cur\_gps\_y\_index

其中cur\_time\_index = 当前时间除以block的时间间隔，也即计算属于第几个时间片。

同时过滤gps\_data\_simple表中超出北京方形区域的记录。

具体方形区域的边界x和y如下：

min\_x = 115.83;

max\_x = 117;

min\_y = 39.55;

max\_y = 40.46;

1. 对gps\_data\_simple表中每条记录了blockid之后，将信息追加在表的最后一列
2. 接着按照blockid进行group ，将在同一个blockid中的user用字符串的方式保存起来，刚开始使用：WM\_CONCAT(",",id) 这个函数串联，后来学会使用UDAF来进行这样的操作。UDAF可以进行更多操作，比WM\_CONCAT更灵活。结果数据保存在start\_block\_users\_set\_table这个表中。
3. 对start\_block\_users\_set\_table表进行UDTF，将user两两配对组合，即可找到起点的pair对，起点相同时间相同位置的user pair数据保存在start\_point\_pairs这个表中，表的格式是u1,u2,blockid.
4. 当时准备使用相同方法对结束点也这么做，但是马上发现一个很严重问题是，结束的blockid不包括时间，即结束点可以不要求是同一个时间到达。那么这时候对结束点的blockid进行group的时候，经过某个blockid的user的个数太多，导致UDAF失败，odps提示如下：

FAILED: ODPS-0010000:System internal error - String append failed, string length (9219723) is large than maximum (8388608) in column (0)

接着想到另一个办法就是将分块划分更小些，这样每个分块通过的车辆更少，那么会不会就不会出现user个数太多，导致的这个问题了呢？

实验之后发现：将时间间隔修改成5分钟，100km\*100km的北京区域分成2000\*2000个block，这样不会出现UDAF失败，但是后来讨论，这样的情况，每个block是50m\*50m，区域太小，不太符合要求，放弃修改block大小的方案

**改版：**

更改策略的主要方向是数据。

接着，在研究某一个用户产生的gps数据个数 以及 在凌晨0点这个时候时间片上user个数，均存在异常情况，譬如出现过某一个用户一天产生了2w多个gps点，这个是不符合常规的，假如一个用户按照最快每隔5s产生一个gps点（没有更快了），那么按照每天用户开车时间最大值 = 3600\*24小时计算，最大值 会产生17280个gps点，发现这个问题比较严重，数据存在错误，咨询了数据挖掘组同事，原来fp\_daily 中的id并不是userid，这里有一个映射关系，并且可能会出现多对一的情况，不过这个情况很少见。

为了验证是不是这个问题，我将该特殊用户的gps轨迹打印出来，绘制在工具上，发现几分钟内，该用户跑遍了大半个北京，可以验证数据挖掘组说明的情况确实存在。

接下来进行了简单的数据清洗工作，将条件设置严格了些，按照每个用户每天最多开车12个小时，按照每5s产生一个gps点，最多只会有8640个点，因此按照最大值10000个gps点 来对user进行了过滤，将数据保存在filtered\_gps\_data这个表中，过滤之后数据记录个数为：685816969

1. 接下来重新进行上一个的策略，10分钟间隔，400\*400的block大小。
2. 接下来做了另一个操作，对blockid进行group，filter\_same\_uid\_appear\_in\_blockid\_UDAF将同一个block内相同的user去掉，只保留第一个相同的user（filter\_same\_uid\_appear\_in\_blockid\_UDAF），保存在start\_block\_users\_set\_table这个表中。
3. 这时候调用gen\_start\_pairs\_UDTF，这时候就不需要判断是否存在重复的user了，但是生成的user pair需要进行排序。
4. 生成用户走过的block集合。gen\_start\_users\_set\_UDAF这个代码有一处问题，如果目的是要生成用户的轨迹序列的话。虽然为了防止block出现重复，使用了set进行了去重，但是由于set在去重之后会丢失顺序，所以，应该按照block排序，但是对block排序还是存在问题，虽然block的计算公式中，时间上可以排序，但是相同时间时，位置信息并不是有序的，这样会存在10分钟内，user走过了几个block，这几个block的经过排序之后，就会按照经纬度排序了，会出现位置跳动的问题。（从下一步意图来看，貌似不会存在这个问题，所以这一步生成的不算是序列，算是user走过的所有block集合）
5. 根据第3步生成的pair对，user1和user2分别join第4步生成的block集合，找到u1和u2各自的经过的block集合，然后调用gen\_start\_end\_pair\_UDTF这个方法，将u1和u2的block集合求交集，得到一个start\_blockid多个end\_blockid这样的记录，其中多个endblockid是tm,x,y得到的计算结果，是无序的,结果保存在start\_end\_pair\_table这个表中。

6．接着曾经试图过滤掉起点相同的情况，filter\_repeat\_start\_point\_UDAF🡪split\_filtered\_start\_block\_UDTF，譬如在start\_end\_pair\_table这个表中，group end\_blockid，将相同用户对，存在相同终点的所有起点聚集在一起，选取两个用户起点在同一条路上行驶了很久，选取start\_block最大的那个（这样做有错误，因为start\_block虽然是带时间的，但是如果处在的时间片相同，那么最大的那个只是代表空间上经纬度更大，而不是时间上更大），这样的数据对最终的pair并没有什么意义。这样的数据保存在filtered\_startblock\_start\_end\_pair\_table这个表中。

7.接着由于还有很多信息没有在表中体现出来，接着重新对数据进行了筛选，将数据中的详细信息做了一定筛选后保留下来，借助gen\_user\_block\_detail\_info\_UDAF这个函数对filtered\_gps\_data这个表，按照userid进行group，也就是生成的是用户走过的所有gps的详细信息（tm,x,y），生成了userid\_block\_list\_detail\_info这个表。

8.这时候准备对start\_end\_pair\_table这个表进行扩充，根据u1和u2，从userid\_block\_list\_detail\_info这个表中查询到用户走过轨迹的详细信息，追加到表中，然后调用gen\_track\_UDTF这个函数，想要从详细信息轨迹中根据tm,x,y函数生成block，对比start\_end\_pair\_table表中的start\_block，找到详细信息轨迹中的起点位置，再从详细信息轨迹中找出终点的位置，保留两者之间的轨迹信息输出，试验后发现仅运行就100条可以，数据量打了之后ODPS运行不起来，太耗内存和CPU

**再改版：**

再改版主要针对上一版本odps运行不起来进行的分析，由于两次join非常耗费内存和CPU导致的，有没有办法在不需要join呢？也就是原始数据就能够带上有效轨迹就非常棒。那么接下来从头分析，发现在上一版本的第3步操作的时候，重新生成原始数据。

第1和2与上一方案一致

3.调用gen\_one\_block\_no\_repeat\_user\_filter\_raw\_data\_UDTF这个函数生成同一个block内没有重复user的原始数据，数据格式为：blockid，timestamp，user，虽然blockid带有时间，但是同一个时间片内无序的问题这样就解决了。

4.调用gen\_user\_block\_track\_UDAF生成user的block轨迹，这时候的block肯定是顺序的

再将该blockid去除时间信息，就可以当终点的blockid了。该信息保存在user\_order\_via\_all\_block这个表中。

5.将该表和第3步生成的原始数据进行join，即可得到原始数据，最后一列追加block的轨迹。one\_block\_no\_repeat\_user\_filter\_add\_user\_track

1. 对上一步生成的原始数据表，按照blockid 进行group，将经过同一个时间同一个地点的blockid的user集合起来，同时找出有效的轨迹段 ，利用函数gen\_combine\_all\_user\_track\_in\_block\_UDAF产生combine\_all\_user\_track\_in\_block这个表。
2. 接着，使用gen\_pair\_with\_track\_UDTF这个函数做最后一步工作：这一步工作有点长，内容如下：

首先将有效轨迹段变成没有时间戳的blockid，求交集，找出共同的终点，对终点按照更长的那个轨迹顺序排序。找出顺序的共同不带时间戳的blockid之后，需要对终点共同blockid进行去重。去重的规则很简单：如果如果在u1的轨迹上，前一个共同的不带时间戳的blockid出现的位置（index） 与 当前这个不带时间戳的blockid出现的位置 相差距离小10，那么认为这两个是重复的。举个例子：

U1的轨迹是：1,2,3,4,5,6,7,8

U2的轨迹是：1,3,4, 8,7

共同的终点blockid顺序是：1,3,4,7,8

1出现在U1的index = 0的位置，3出现在U1的index = 2的位置，如果设置阈值为1的话，那么认为1在U1不算是重复，但是1在U2中算重复。

最后还要保证公共终点blockid不能太接近于起点，中间要求大于10个block。

**优化：**接下来仔细思考，会发现，上述第3步其实有点不完整，只添加timestamp还不如将所有信息都加上，包括x和y。因此另一个表生成了：user\_track\_order\_on\_x\_y，这个表中生成了user经过的顺序轨迹，包括完整信息，而且每个block内仅有一个相同用户。

这时候再根据user\_track\_order\_on\_x\_y利用函数gen\_new\_raw\_data\_UDTF生成一个新表，可惜这时候的新表只包含两个信息，一个是带时间片信息的blockid，另一个是userid；接着准备将user\_track\_order\_on\_x\_y这个数据join在最后一列。到此，同上5的步骤结果。还是没有跳出更加优化的方案，这时候已经到中秋节期间，又想出了更进一步优化。

**接着优化**：gen\_new\_raw\_data\_with\_track\_UDTF函数非常好的优化了前面gen\_new\_raw\_data\_UDTF这个函数，这个表一步到位，将产生新数据和join的工作一次完成，

在产生新数据的时候，已经知道轨迹了，每产生一个新数据的时候，我们是知道有效轨迹段的，因此刚好用一个UDTF可以将此输出，非常有效。

到此为止，这一步骤完成了，在验证数据的时候发现另一个**严重问题**：**gps轨迹不连续！**

原因：上面一直讨论的gps轨迹，认为用户一天只跑了一段轨迹，其实这是无意义的，很多用户一天会产生多段导航段！

**继续优化：**接下来重新对user\_track\_order\_on\_x\_y这个表进行优化，产生了user\_navi\_section\_track\_order\_on\_x\_y，这个表根据轨迹上每个gps点的时间，两个gps之间间隔超过一定阈值（根据赵鹏要求，设置为最大1分钟间隔）就切开，认为是两段轨迹。

接着，在产生带轨迹的原始数据的时候，也做了一些限制，譬如轨迹总长度要大于5个block，有效轨迹长度也要大于5个block。

这样一直做下去，**终于得到轨迹pair对。验证有效。保存在表：navi\_section\_gen\_u1\_u2\_track**

接下来的工作需要将gps有效轨迹转为link轨迹。

**开始研究link轨迹**

使用collect\_all\_link\_of\_user\_UDAF这个函数对road\_profile\_daily这个表按照userid分组，得到一个用户某天经过的所有link的集合，保存在表：collect\_all\_link\_of\_user

合理的思路是：对**navi\_section\_gen\_u1\_u2\_track**使用gen\_u1\_u2\_track\_add\_linktrack这个函数join collect\_all\_link\_of\_user这个表，期望能够根据有效gps轨迹段保存的时间信息来过滤link表中的link轨迹，得到有效link轨迹pair。

但是odps又不工作了！我发现两次join的工作在odps上都是非常耗费内存和CPU的工作。小数据量可以运行起来，大数据量就不动了，可能是**navi\_section\_gen\_u1\_u2\_track** 这个表太大了导致的。

**更进一步优化不能生成link pair问题：**

新方案：