摘要

在全球一体化的今天，对于国际会议的需求越来越多。然而，受到时差的影响，或者由于两地之间气候的不同的原因，这些会议的组织与举行并非易事：许许多多的因素会对会议效率产生显著的影响。本文的目的就在于通过建立数学模型来根据某一特定会议时间以及与会人员及其出发地来选择参会地点，使得会议可以在一定预算之下取得最大的会议效率。

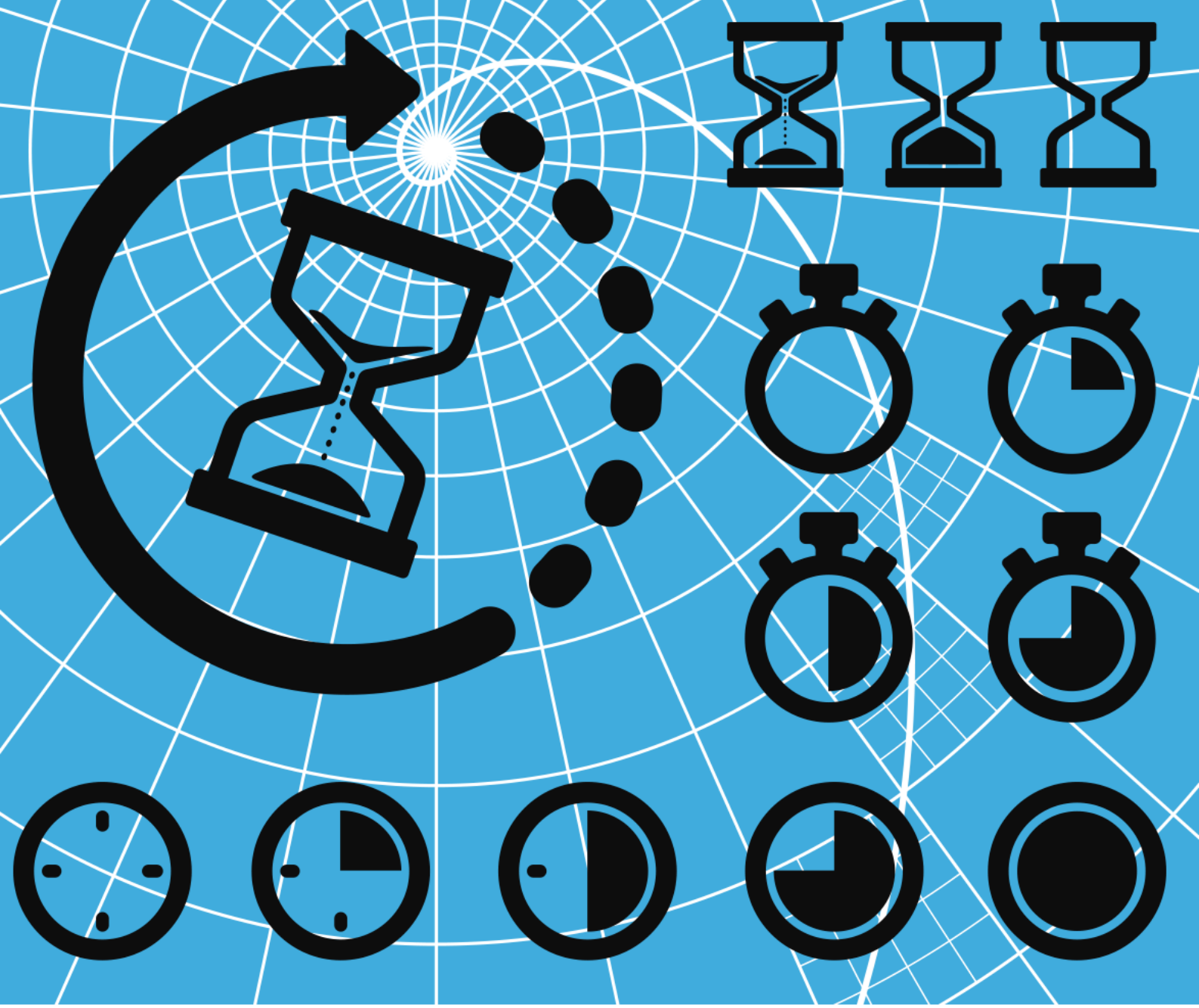
我们首先标记出了所有的机场，并得到地球上已知经纬度的两点之间的球面距离和时差的公式。并且，我们根据数据分析并拟合得到了机票关于距离的函数、在不同等级的城市的住宿费用。由于会议受到预算的控制，我们通过一定的算法计算出了不同地点开会所需要的机票与住宿费用之和，即为会议总成本，并将超出成本的地点以及位于海洋等无法开会的地形的地点筛除。我们通过定义影响开会效率的基础工作效率以及判断某一地点是否适合召开会议的标准“优秀值”，便可以得出满足要求的地点。

在此基础上，我们又建立了高级模型。我们遍历的开会地点以及时间，选择在该开会时间时原时区仍为一天中工作效率较高的时间（如：非深夜时段）。这样，这些专家可以不需要倒时差，也就节省了住宿的成本并且提升了会议效率。我们再将不得不倒时差的专家根据上一模型进行分析，便可以对整个方案进行评估，得到会议效率最高的方案。

建立了两个模型后，我们首先对于一大一小两次会议进行了具体分析，得到了一些适合开会的地点供组委会进行选择。在这之后，我们对于 2014 年的 APEC 会议进行了模型测试。我们发现通过模型所计算出来的地点与实际开会地点虽不相符缺相差不大。我们考虑到实际上会议的召开要收到政治、经济等因素的影响，而举行地点与得到的结论恰在附近说明了模型的可行性以及我们算法的精密性。

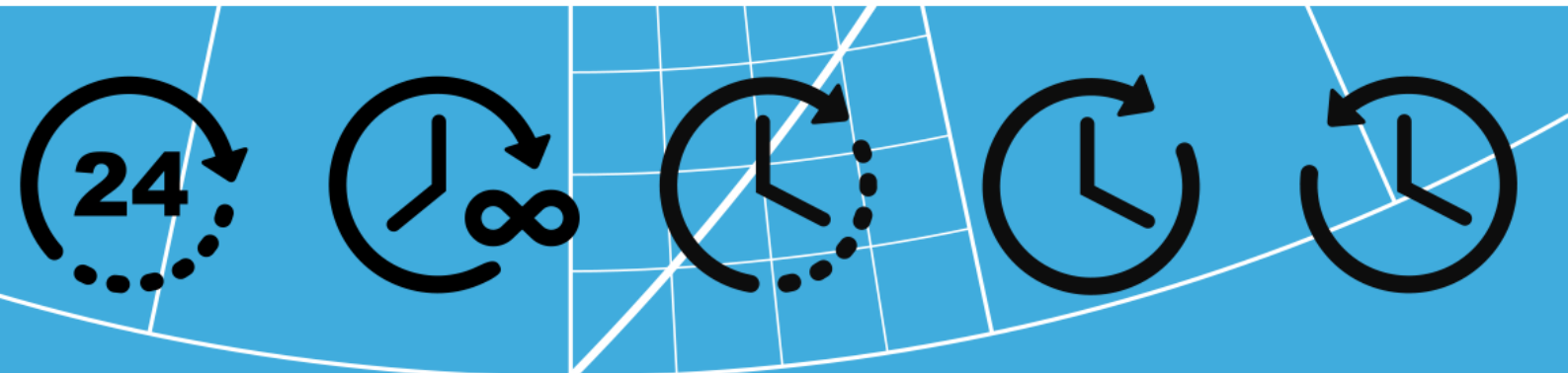
我们最后分析了一定时差下，基础工作效率对于模型灵敏度的影响结果的影响。总的来看，此模型的适用性极强，可以根据不同的会议进行具体的更改，且本模型可以大致囊括重要的影响因素。然而，假设每个人的恢复能力相同，没有考虑每个人对于倒时差能力的不同所带来的脑子工作效率恢复速度的影响，并带入我们根据统计分析的出的函数当中，这在一定程度上影响了精确度。

关键词：会议地点；时差；会议效率



IMMC TEAM
17060464

TIK
TOK



目录

1 问题重述

在全球的经济与贸易联系日加频繁的同时，各国之间的联系也日益密切。在这个时代背景下，人们为了商讨发展性的重大问题，时常需要奔波各国之间开会，而这也就引发了我们关于合适会议地点的选取的思考。在保证会议的效率的同时，减少参会者的时间消耗与酒店、飞机等金钱消耗，也就是说，要让参会者在经过了合理的休息后，达到参会的最佳状态，这也就是我们传统意义上的倒时差。

本题探讨了在不同的季节下，来自不同地区不同数目的参会者的最佳会议地点的选取且最佳会议安排问题。我们需要将时间作为第一考虑因素，尽量使得所有人在到达参会地点的 7 日之内能够调整好时差，恢复会议效率，从而可以开始开展活动，并且在此要求之下，尽量减少参会者的酒店以及飞机等金钱开销。

2 问题分析

在会议安排问题的解答过程中，因考虑到评估分析某一目的地是否适合于开会取决于会议效率以及成本两方面因素，且时差和温差为影响会议效率的主要因素，住宿开销与机票开销则为经济成本的组成部分，我们把体力、大脑的灵敏性以及决策力等对会议进展起到重要作用的条件总体定义为基础工作效率。通过数据分析可以得到在时差或温差一定的条件下，休息时间与工作效率的相关性的函数。同时将其与会人员休息的时间定义为会议投入时间。将与工作效率相关的会议结果与会议投入时间进行比较，便可以得到会议的效率。

由于开会进行的地点不能够在海上或是其他一些有着极端地理环境的地点，我们把可供选择的参会地点选取在各个机场附近。随后，考虑到会议组委会的经费有限，我们决定首先通过对于成本的限制来排除一些地点，而对于剩下的城市，我们定义会议效率与经济成本的加权平均值为优秀值，并选取优秀值最高的一个地点为我们的答案。因不同城市的等级不同而引起的住宿开销不同，以及机票价格与距离相关，我们便可以得出会议成本的相对值。由于地球上城市机场过多，我们扫描一遍地球，依照优秀值选取前 10%的点。如此，便可以精确地得到适合开会的地点。

随着对问题思考的深入，我们发现并不所有人都需要将生物钟调整至目的地时间，参会者也可以采用出发地时间来进行会议。如此便要考虑到有些国家所采用的冬、夏令时的制度。我们将来自相对时差在半天之内的国家的专家统计起来，假设他们并不去调

整自己的时差；对于来自与大部分国家时差较大的国家的参会者，我们仍然适用第一种方法中对于会议效率的计算方法。运用此方法，时差对于会议效率的影响大大减小，同时，因为恢复体力而产生的住宿费也被解约了下来，因此花费也得到减少，使得会议可以得到优化，同时与实际情况更加贴近。

3 模型的自定义量、变量表与假设

3.1 自定义量

- **经济限度：**委员会的金钱成本的限值
- **个人工作效率（k）：**我们把一个人的体力、大脑思维活跃及准确性以及决策力等对会议进展起到重要作用的条件总体定义为个人工作效率
- **会议投入时间（t）：**与会人员休息的时间以及往返乘坐飞机所用时间定义为会议投入时间
- **会议效率（v）：**参会者每人的个人工作效率对会议的影响
- **优秀值（P）：**会议效率与经济成本的加权平均值为优秀值

3.2 变量表

符号	符号说明
优秀值	P
个人基础工作效率	k
个人受时差影响后实际工作效率	K_t
个人受温差影响后实际工作效率	K_T

实际工作效率	K
平均工作效率	$\overline{K}$
机票价格	$M_{机票}$
球心角	a
纬度	La
经度	Lo
路程	s
住宿价格	$M_{住宿}$
每人金钱成本	M
总金钱成本	$M_{总}$
温差	ΔT
飞机飞行时间	t_f
飞机落地后的在宾馆内休息时间	t_h
个人休息时间	t'
会议总投入时间	t
时差	Δt
参会人数	n
会议效率	v

会议投入时间 t

3.3 假设

- **假设 1**：不考虑个人基础工作效率的差异，且参会者的倒时差与身体自动调节温度的能力相同

理由：题目中并未提及关于人们的能力不同，只给予了他们所在地点的信息， 所以可基本认为他们的基础会议效率和基础个人工作效率为相等的，对时 间、温度的调整效率一致。

- **假设 2**：不考虑因地球自转而引起的自西向东与自东向西飞行的路程差距，假定飞机可沿直线飞行，从出发地直达目的地。

理由：不同航班的飞行线路不同且中途可能有转机，我们并没有给予相关的 数据，且若计算，则本题过于复杂化。

- **假设 3**：在计算机票价格时，不考虑其因竞争选手票价、购票日期等特殊因素而引起的价格变化。

理由：我们无法从已有渠道得到此类信息且不同航空公司的票价不尽相同， 所以我们只得忽略此类影响情况。

- **假设 4**：假定酒店就在机场附近，且从机场赶到酒店不需要额外的时间与费用。

理由：机场为重要的交通枢纽且国际间往来时多乘坐飞机，所以机场为酒店 选取的最佳地点。我们可以假设酒店就在机场旁边或机场附近。

4 最优点选择模型

4.1 基础模型

4.1.1 基础模型建立

根据题意，我们将会议安排的影响因素拆分为两个大块，分别为时间成本以及金钱成本。因组委会的投资数有一定限制，所以我们把金钱成本作为限制因素，而时间成本这是我们考虑分析最优的评判标准。在我们分析金钱成本的影响因素是，我们发现其主要由机票价格与住宿费用组成。我们借助旅游网站[1]http://jipiao.kuxun.cn/jichang/ 寻找到了所有的 1219 个机场，随后通过在百度地图网站［2］http://www.gpsspg.com/maps.htm 上输入，筛选出地图内有标记并且有经纬度信息的 168 个机场。这样避免了参会最优地点在海中或者沙漠等不适宜地点的情况。为了计算出任意两个机场之间的球面距离（s），我们在网络上查询并在之后经过自己推导得到了公式（4-1-1）［3］http://www.cnblogs.com/android-zcq/p/5996002.html 我们便可将纬度、经度转化为两点之间的球面距离（s）：

$$s = 2\sin^{-1}\sqrt{sin^2\frac{Lo_1-Lo_2}{2} + \cos La_1 * \cos La_2 * sin^2\frac{La_1-La_2}{2}} * 6378.137 \quad （4\text{-}1\text{-}1）$$

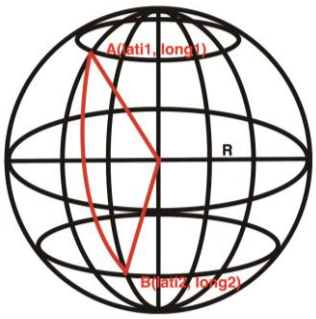


图 4-1-1 球面距离计算图示

利用经纬度，我们可以得到参会者从出发地到参会地所跨时区，随后可以根据公式（4-1-2）得到参会者所需要倒的小时数，也就是时差（Δt）：

$$\Delta t = \frac{a}{15^\circ} \quad （4\text{-}1\text{-}2）$$

之后，我们通过查找资料（《民用航空法》、《价格法》和《中央定价目录》）以及分析网络上已有的数据并进行拟合，得到了机票价格（$M_{机票}$）关于距离（s）的函数:

$$M_{机票} = 0.94 * s \quad （4\text{-}1\text{-}3）$$

因此我们得到了机票的价格，随后又跟据城市的分级得到了不同等级城市的大概每日住宿费用（$M_{住宿}$）以便后续的计算。通过这两项，我们可以得到每人所需要的全部金钱成本（M），通过加和每个人的我们可以得到金钱成本总值（$M_{总}$）：

$$\mathrm{M} = \mathrm{M}_{机票} + \mathrm{M}_{住宿} \quad (4\text{-}1\text{-}4)$$

$$M_{总} = \sum_{i=1}^{n} M_i \quad (4\text{-}1\text{-}5)$$

我们设定了一个限度，若是金钱成本总值（$\mathrm{M}_{总}$）在此限度之内，则可被考虑为会议地点，如此便可以排除超出预算的地点，且继续寻找保留下来的地点中更优越的地方。

接着，我们就引发了关于如何继续筛选最佳的会议地点的思考，我们开始考虑将时间成本作为主要的考虑因素。因此，我们就定义个人基础工作效率（k）由体力、大脑反应速度以及决策力等因素共同组成，且其直接影响会议的进展。根据题干，个人工作效率主要受到目的地与出发地时间的时差与温差影响。而同时，个人休息时间（t'）则包括飞机飞行时间（t_f）与飞机落地后的在宾馆内休息时间（t_h），也就是飞机落地时刻到会议开始时刻内的所有时间。通过拟合一些飞行长度和飞行时间的关系数据，我们可以得到简化函数关系式：

$$t_f = s * \frac{4}{3} \quad (4\text{-}1\text{-}6)$$

但由于题目限制，我们需要将飞机落地后的在宾馆内休息时间（t_h）控制在 168h 也就是 7 天之内。则为我们的结果增添了一条限制。随后，我们通过查找资料以及对于数据的分析，拟合出在出发地与目的地的时差一定的条件下，个人休息时间（t'）与个人受时差影响后实际工作效率（K_t）的函数关系为：

$$K_t = -\frac{k}{t\prime + k} + 1 \quad (4\text{-}1\text{-}7)$$

相似的，我们找出了出发地与目的地的温差一定的条件下，休息时间（t'）与个人受时差影响后实际工作效率（K_T）的函数关系为：

$$K_T = -\frac{k}{t\prime + k} + 1 \quad (4\text{-}1\text{-}8)$$

由图 4-1-2 我们分析可得，若 1 为个人工作效率的最高限额，当参会者休息约为一天时，其个人工作效率随时间的增长速率变缓。所以，必定有一个最适时间，也就是一个转折点，为状态恢复速率变缓以及时间和温度适应度皆处于合适值内。

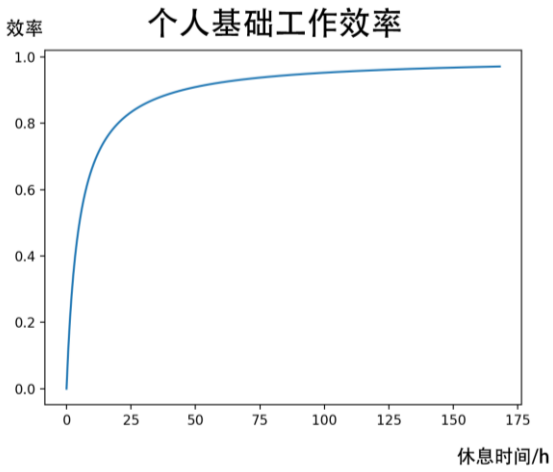


图 4-1-2 个人基础工作效率图示

因此，我们便可以得知一个人的实际工作效率（K）为：

$$K = \frac{K_t + K_T}{2} \tag{4-1-9}$$

而会议中的与会人员的实际工作效率的平均数（$\overline{K}$）则为：

$$\overline{K} = \frac{\sum_{i=1}^{n} K}{n} \tag{4-1-10}$$

我们定义与会人员休息的时间总和为会议投入时间，并且规定会议效率（v）为实际平均工作效率（$\overline{K}$）相关的会议结果与会议投入时间（t）的比值，而会议投入时间（t）则为个人休息时间（t'）的总和减去所重叠部分，也就等于第一个人出发的时刻和所有人的平均工作效率达到合适值的时刻之间的差值：

$$v = \frac{\overline{K}}{t} \tag{4-1-11}$$

为了更好的衡量与抉择最佳的会议地点和会议安排，我们把会议效率与经济成本的加权平均值定义为优秀值（P），而这也将成为我们选择的主要参考因素。

http://baike.baidu.com/link?url=UII6FsQEx_arXtx3-F644Jswi0cP4473aDl4dGCy868Vx2ytgweatrIhD1GTWeIeJl_gyxaFTKcM0S95MD0oQpvxYJpFAgsiTL5ZbllhTI9lw418LpXm530fin-kKIeL［4］在如上网址中我们找到了 Sigmoid 函数，其常被用作阈值函数，且其将变量映射到 0～1 之间很符合我们所定义的优秀值的含义，其形如：

$$S(x) = \frac{1}{1+e^{-x}} \tag{4-1-12}$$

所以我们便可根据公式（4-1-12）推得优秀值的计算公式：

$$P = \frac{v+S(\frac{10000}{M_{总}})}{2} \quad (4\text{-}1\text{-}12)$$

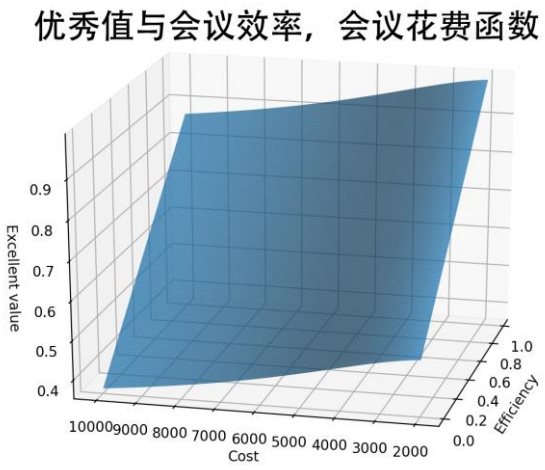


图 4-1-2 优秀值与会议效率和金钱成本的函数

然而，由于可供选择的宾馆数目还是过多，我们决定将所有情况的优秀值进行排序，选取其中前 10%的点的所在地作为备选点，以便组委会进行选择。因题目中提及不考虑政治因素等不可控因素，多一些选择方案可以更加符合实际情况，满足他们的实际需求。

4.1.2 基础模型小型会议求解

在准备工作中，我们查询到6位参会者所在区域的最大的机场及其经纬度分别为：

Monterey CA, USA 36.588, -121.848

Zutphen, Netherlands 52.344, 4.956

Melbourne, Australia -37.671, 144.849

Shanghai, China 31.196, 121.340

Hong Kong (SAR), China 22.304, 113.924

Moscow, Russia 55.410, 37.902

在得知了会议时间约为6月中旬后，我们将开会时间设定为6月15日，从而倒推其出发时间。

随后，我们根据如4-1-1所建立的数学模型进行了简单的程序运算。

步骤	程序
1	定义6位与会者的所在地点为起始点 测算每一个点与所有选取的机场所在点的球面距离（s） 根据4.1.1的一系列公式得到每种会议安排的总金钱成本（$M_{总}$） 若其超过最高金钱限度，则删去此种情况；反之，进入步骤2
2	计算每种情况的优秀值 取其中前10%作为结果并进入步骤3
3	输出其每个点所对应的经纬度和地区名称 输出每个人的出发时间以及结束时间

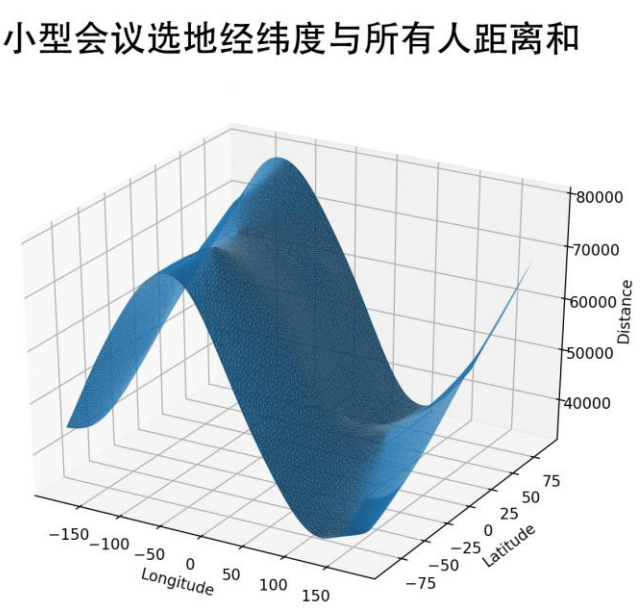


图 4-1-2 小型会议选地经纬度与所有人距离和

如图 4-1-2，我们不难发现当处于纬度 40 经度 160 左右的位置时，所有人的距离和最小。在综合考虑各种影响因素之后，在编程的帮助下，我们寻找到了一些最优结果：

42.895332，129.454833 延吉朝阳川机场

45.628829，126.249602 哈尔滨太平国际机场

49.279245，120.042882 呼伦贝尔海拉尔机场

41.114521，121.078611 锦州小岭子机场

36.272427，120.388394 青岛流亭国际机场

44.54397，129.59619 牡丹江海浪机场

39.79004，116.367509;北京南苑机场

为了使我们的结果更加直观，我们把它放在地图中，发现大致的选址位置集中在中国日本附近。由此观之，会议选取在此附近为最优，既可以保证参会者的时间成本与委员会的金钱成本降到最低，避免了不必要的消费。

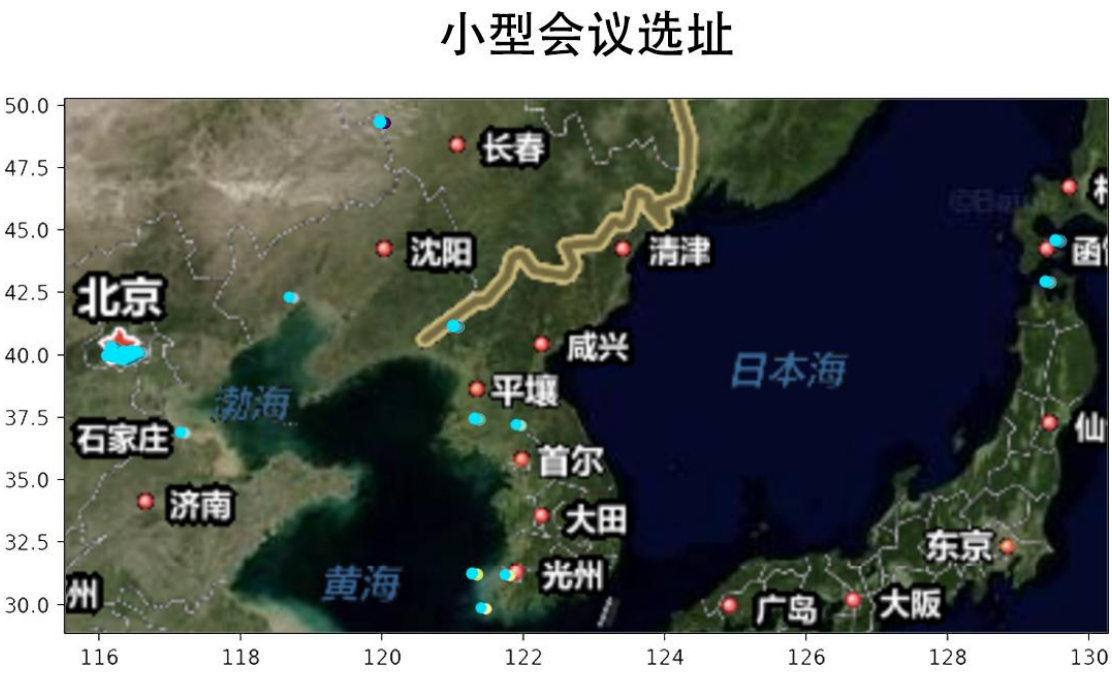


图 4-1-3 小型会议选址

4.1.3 基础模型大型会议求解

与 4.1.2 相类似，我们运用类似的方法进行了第二问的求解，因为此问有同属一个出发地的人，我们便特殊考虑了一番。但想到若是分开计算，则毫无影响，便利用类似的方法首先选取每地的著名机场，作为始发地。

Boston MA,USA	42.366，-71.010
Singapore	1.364，103.992
Beijing, China	40.799，116.603

Hong Kong(SAR), China 22.308，113.918

Moscow,Russia 55.410，37.902

Utrecht, Netherlands 52.312，4.768

Warsaw, Poland 52.167，20.968

Copenhagen, Denmark 55.618，12.651

Melbourne, Australia -37.671，144.849

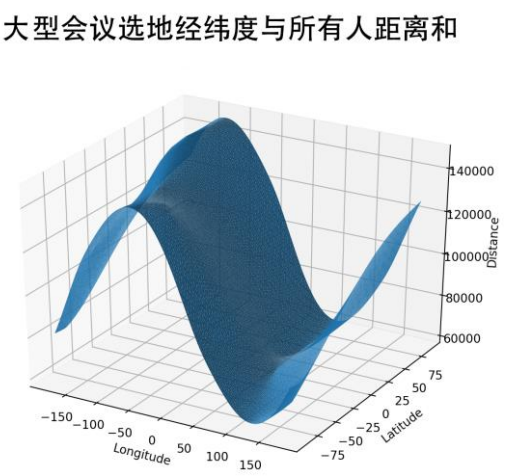


图 4-1-2 大型会议选地经纬度与所有人距离和

在得到大型会议选地经纬度与所有人距离和的图表之后，我们发现它与小型会议选地经纬度与所有人距离和的图标大致相同，所以我们便猜测最终的结果可能也大致分布在小型会议的结果附近。

经过计算我们得出，大型会议的结果与小型会议结果相类似，在中国东部的地方开会为最佳开会地点，既可以满足组委会的需求，又能够为参会者着想，减少他们的调整所需要时间，又可以保证会议的成功召开与快是推进。从这两问的模型求解来看，我们发现中国东部非常适宜安排会议，这与其独特的地理位置与适宜的气候和价格适宜的酒店密不可分。

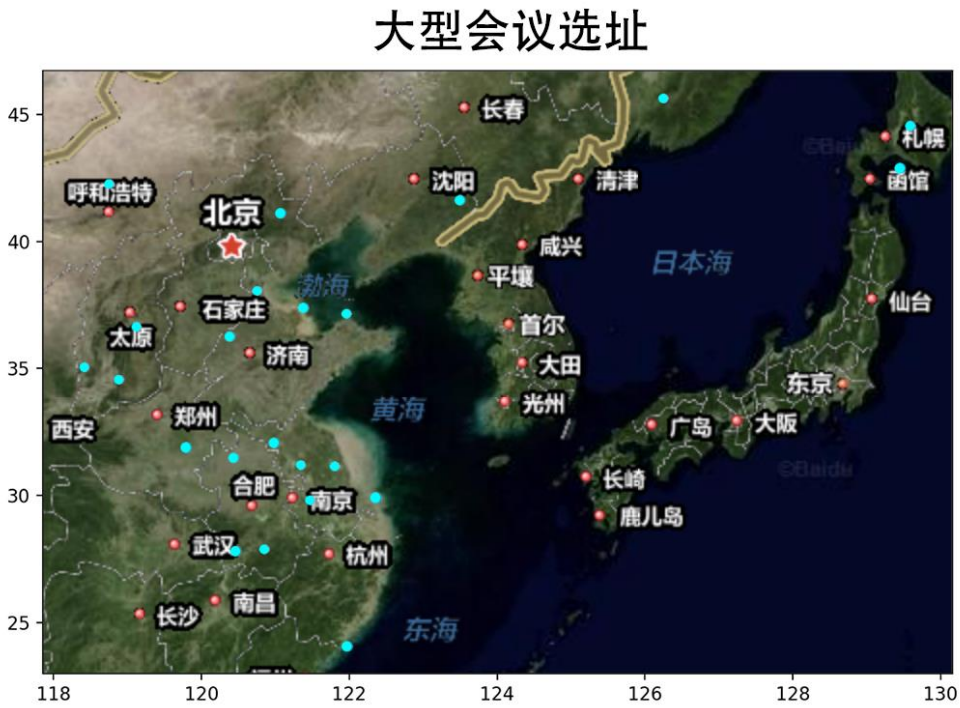


图 4-2-3 大型会议选址

4.2 高级模型

4.2.1 高级模型建立

在考虑问题的过程中，我们发现会议进行时并不需要所有人将生物钟均向目的地时间调整，他们可以仍采用出发地的生物钟来开会。这样，会议可能在某一地点的半夜进行，然而与会人员仍采用出发地的时间，使得工作效率并没有因为时间在深夜或是因为倒时差的原因而降低。如此，住宿方面的花费将大大减小。但是，根据具体会议设计考虑，也许并不能找到某一段时间使每一位参会的专家都可以以出发点时间、以最大的工作效率进行会议，所以我们利用类似于第一种方法解题的步骤可以得到花费最少、会议效率最高的方案。

图 4-2-1 高级模型思维导图

如4.2.1前面的准备步骤一样，我们先筛选出总金钱成本（$M_{总}$）在限度值以下的会议地点，并根据其相应的经纬度得到相应的时区。我们利用遍历的方法得到会议开始的时间，并根据每个人相对于会议地点的时差，遍历每个人原来所在时区的会议开始的时间。根据专家在原来所在时区正处于的工作效率的时间段，我们遍可以把参会者分为两个类别，分别为处于较高时段的不需要倒时差的参会者与较低时段的需要倒时差的参会者。

对于不需要倒时差的专家，根据该专家所在城市机场的经纬度和会议地点的经纬度坐标利用公式（4-1-6）来推算出飞机所需需要的飞行时间。并依照会议开始时间可以得出他所乘坐的飞机应该的起飞、降落时间可以保证这两个时间均为其原所在时区的白天，使得他的实际工作效率并不会受到影响。其中需要注意的是，降落时间应在会议开始时间之前。而对于需要倒时差的参会者，我们利用基础模型中的公式（4-1-12）计算出其分别的优秀值。

最终，我们选取不需要倒时差的人数最多且对于需要倒时差的人使用基础模型计算得出的优秀值（P）最高的情况中所对应的会议地点坐在经纬度以及会议开始时间即为高级模型的最优解。

4.2.2 高级模型小型会议求解

经纬度数据基本如 4.1.2 一样，这可以确保两个模型的条件基本相同。

步骤	程序
1	定义6位与会者的所在地点为起始点,测算每一个点与所有选取的机场所在点的球面距离（s）。根据4.1.1的一系列公式得到每种会议安排的总金钱成本（$M_{总}$）。 若其超过最高金钱限度，则删去此种情况；反之，进入步骤2
2	根据相应的机场经纬度，得出相应的时区。遍历会议开始的时间的同时，根据每个人相对于会议地点的时差，遍历每个人原来所在时区的会议开始的时间并分别分析某一参会专家是否在原来所在时区正处于工作效率较高的时间段之中。 如果否，则使用基础模型来评估并计算该专家行程的优秀值；如果是，进

	入步骤3.
3	根据其所在城市机场的经纬度和会议地点的经纬度坐标推算出飞机所需需要的飞行时间并计算得出其所乘坐的飞机应该的起飞、降落时间可以保证这两个时间均为其原所在时区的白天，使得他的到脑灵敏度并不会受到影响。 完成后进入到步骤4
4	选取不需要倒时差的人数最多且对于需要倒时差的人使用基础模型计算得出的优秀值最高的情况中所对应的会议地点坐在经纬度以及会议开始时间即为高级模型的最优解 输出高级模型的最优解

在编程的帮助下，我们得到了高级模型的小型会议的选址情况。

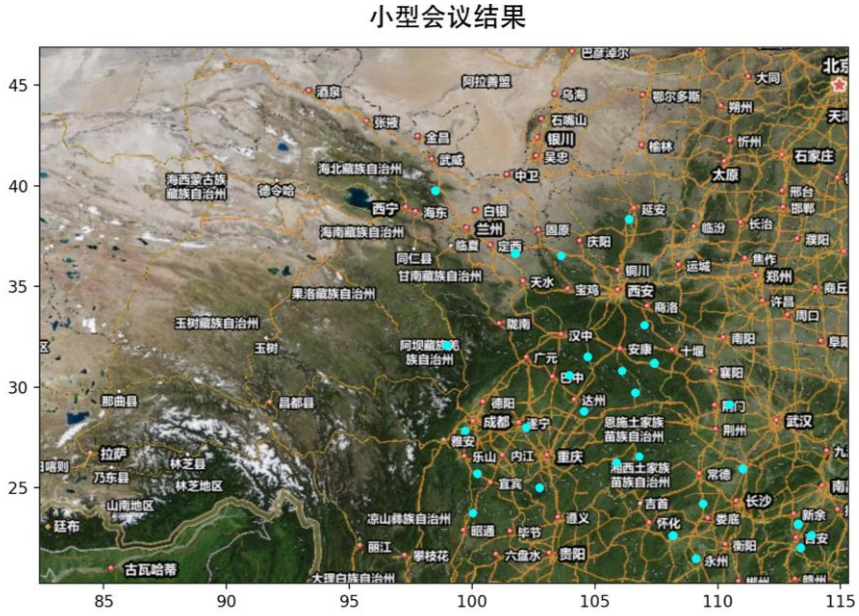


图 4-2-3 高级模型小型会议选址

根据编程模型，我们可以得到在如下的两个地方，为最佳的会议地点，而图 4-2-3 中的蓝点位置则为相较之下的优点：

27.831029, 99.713682 迪庆香格里拉机场

31.182047，107.424315 四川省达州市达州机场

不难发现，高级模型的小型会议选址比基础模型的小型会议选址更加往西北，但是却仍在附近。由此可见，高级模型比基础模型更加精准且偏差不大。两个模型都有可靠性。

4.2.3 高级模型大型会议求解

利用 4.1.3 的经纬度数据，并仿照 4.2.2 的步骤，我们可以得到高级模型的大型会议的最优选址。

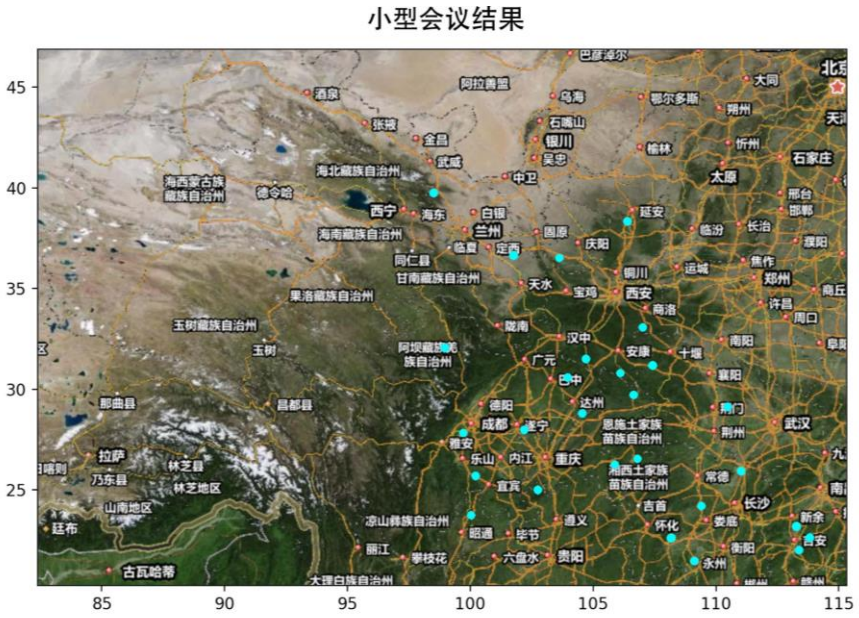


图 4-2-4 高级模型大型会议选址

高级模型的大型会议选址结果如下：

30.800965，106.105554 四川省南充市高坪机场

43.903344，87.491727 新疆维吾尔自治区乌鲁木齐市地窝堡国际机场

4.3 模型测试

我们利用现实世界中的数据进行了模型测试，今而验证我们模型的可靠性与可行性。我们依靠百度百科里面的关于 2014 年 APEC 大会的与会人员的安排与地点数据来进行分析。其中，我们查询结果的网址为

http://baike.baidu.com/link?url=yIPPAsCq3YTHrFmVKrguYCff0b_QLyBGVuz7U58iTTMBANqN9QRUfnBfthCntscdjBQlj9DR1vwqbKRZm3ad7C5544l13oQzOglHKEiAbjVbhHKBMPDiUEAm722ceM46

我们发现其主要的参会人员分布在如下区域：

美国	40.641，-73.778
韩国	37.469，126.441
日本	35.772，140.393
加拿大	49.200，-123.182
俄罗斯	55.512，38.037
台北	25.068，121.553
香港	22.308，113.918
墨西哥	19.436，-99.072
巴基斯坦	25.233，62.329

经过计算，我们发现我们所得到的结果与实际开会地点（北京，雁栖湖）并不相符，但却在附近。我们思考后发现这关乎 APEC 大会的意义与一些政治上的因素，导致了我们的模型结果与实际结果的不符。但在附近恰恰说明了，我们模型的可行性以及我们算法的精密性。

4.4 灵敏度分析

在我们得到了如上结果之后，我们开始思考会议安排的影响因素。在我们所有自由设定的值中，人的基础工作效率为一个颇为重要的因素。所以我们决定探讨人的基础工作效率对我们模型的影响大小。在原本的模型中，我们将人的基础工作效率设定为 10000，而改变后的结果如图（4-4-1）所示。

根据图 4-4-1 我们可以得知，随着基础工作效率（k）的设定增大，金钱成本对参会地点的选择的影响越来越大。由此可见，人的适应力的增强会导致金钱成本的因素在

评价模型中的比例越来越重。所以，适当的基础工作效率（k）的设定对于模型及其结果而言十分重要。

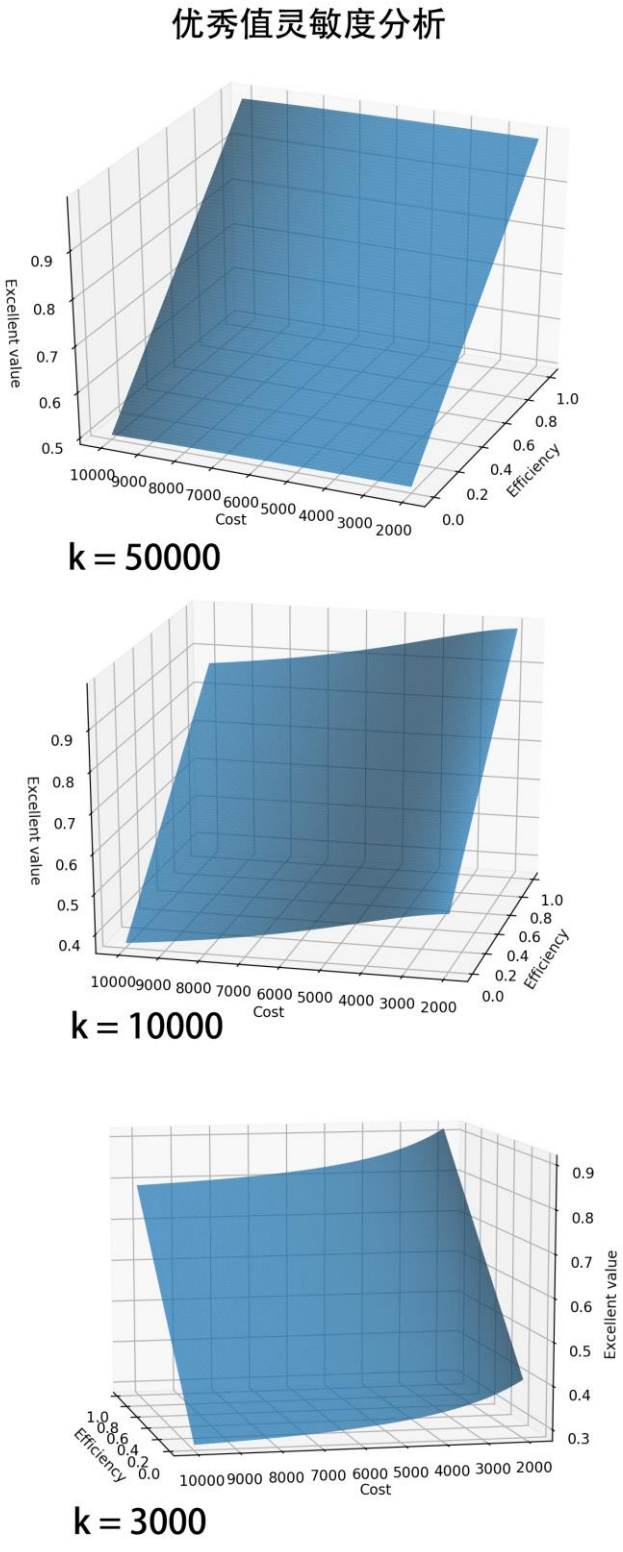


图 4-4-1 基础工作效率分析

4.5 模型评价

本文模型的试用性极强，因模型有许多自定义值，可以根据实际情况的不同而进行更改。且本文关于模型进行了多方面的考虑，可以大致囊括重要的影响因素，且本模型推测了可变量对模型整体结果的影响，并选取其中之一的工作效率进行实际的分析与计算，这为模型的实用性与多用性进行了测试。而模型测试中，对人类世界中的数据的测试，可以体现其严谨性与真实性、可靠性。

但其缺点仍存在，本模型中没有考虑每个人对于倒时差能力的不同所带来的脑子工作效率恢复速度的影响，而是假设每个人的恢复能力相同，并带入我们根据统计分析的出的函数当中，这在一定程度上影响了精确度。我们在模型中假设机票价格只与飞行距离相关，这也对成本的分析的精确性产生了一定的影响。

4.6 模型推广

我们在模型中预留了经济效率和时间效率的平衡权重，使用者可以根据其时间成本的高低来调整这个权重，以此获得更符合其个性化需求的结果。

本模型中没有考虑每个人对于倒时差能力的不同所带来的脑子工作效率恢复速度的影响，而是假设每个人的恢复能力相同，并带入我们根据统计分析的出的函数当中；在未来的推广当中，可以进行更大范围的调查得出更符合每个人的恢复能力的函数以获得更准确地结果。

我们在模型中假设机票价格只与飞行距离相关，在未来推广当中，我们可以连接机票网站的数据，获取更加准确地机票信息，因而推出更准确地价格和优秀值得预测，获得更准确的结果。

5 参考文献

［1］旅游网站，机场信息查询：http://jipiao.kuxun.cn/jichang/

［2］将机场信息转化为经纬度信息：http://www.gpsspg.com/maps.htm

［3］球面距离公式来源：http://www.cnblogs.com/android-zcq/p/5996002.html

［4］Sigmoid 函数来源：

http://baike.baidu.com/link?url=UII6FsQEx_arXtx3-F644Jswi0cP4473aDl4dGCy868Vx2ytgweatrIhD1GTWeIeJl_gyxaFTKcM0S95MD0oQpvxYJpFAgsiTL5ZbllhTI9lw418LpXm530fin-kKIeL

6 附录

附录 1 主程序（进化+神经网络）（小）

```
# -*- coding:gb2312 -*-

from __future__ import division

import math

import random

import datetime


k = 0.1


personone = [36.588, -121.848]

persontwo = [52.344, 4.956]

personthree = [-37.671, 144.849]

personfour = [31.196, 121.340]

personfive = [22.304, 113.924]

personsix = [55.410, 37.902]


placeone = [36.588, -121.848]

placetwo = [52.344, 4.956]

placethree = [-37.671, 144.849]

placefour = [31.196, 121.340]

placefive = [22.304, 113.924]

placesix = [55.410, 37.902]


earth_r = 6378.137
```

pi = 3.1415926535

flydiscasearray = [

[0.2547],

[0.4855],

[0.3556],

[0.7500],

[0.8225],

[0.8514],

[0.6182],

[0.9753],

[0.8127],

[1.1902],

[0.8620],

[0.9079]

]

flydislabelarray = [

[0.180],

[0.370],

[0.265],

[0.585],

[0.600],

[0.630],

[0.460],

[0.740],

[0.680],

[0.810],

[0.630],

[0.595]

]

all = [

[36.588, -121.848, 0],

[52.344, 4.956, 1],

[-37.671, 144.849, 2],

[31.196, 121.340, 3],

[22.304, 113.924, 4],

[55.410, 37.902, 5]

]

hotelcasearray = [

[0.013756331, 0.100501765],

[0.039904211, 0.116407395],

[0.05252000, 0.013404954],

[-0.034603684, -0.058381559],

[-0.033922673, 0.018426454],

[0.025204849, 0.055270783],

[0.003139003, 0.101686855],

[0.05150929, -0.00125334],

[0.019435879, -0.099129961],

[0.040416775, -0.00370379],

[0.025764726, -0.080189831],

[0.055755826, 0.0376173],

[0.019075984, 0.072877656],

[0.040712784, -0.074005941],

[0.048856614, 0.002352222],

[-0.022906847, -0.043172896],

[0.041902783, 0.012496366],

[0.037566535, 0.126977969],

[0.031230416, 0.121473701],

[0.001352083, 0.103819836],

[-0.03386882, 0.151209296],

[0.035689487, 0.139691706],

[0.043653226, -0.079383184],

[0.038907192, -0.077036871]

]

hotellabelarray = [[0.406],

[0.504],

[0.602],

[0.644],

[0.742],

[0.868],

[0.497],

[1.155],

[0.546],

[0.630],

[0.847],

[0.875],

[0.574],

[0.1442],

[0.973],

[1.176],

[0.728],

[0.791],

[0.490],

[0.1050],

[1.141],

[0.945],

[0.784],

[1.190]]

```
def shenjingwangluooneinput(casearray, labelarray, testarray):

    random.seed(0)

    def logn(a, n):
```

```
    return math.log(n) / math.log(a)



def rand(a, b):
    return (b - a) * random.random() + a



def make_matrix(m, n, fill=0.0):
    mat = []
    for i in range(m):
        mat.append([fill] * n)
    return mat



def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))



def sigmod_derivate(x):
    return x * (1 - x)



class BPNeuralNetwork:
```

```
def __init__(self):
    self.input_n = 0
    self.hidden_n = 0
    self.output_n = 0
    self.input_cells = []
    self.hidden_cells = []
    self.output_cells = []
    self.input_weights = []
    self.output_weights = []
    self.input_correction = []
    self.output_correction = []


def setup(self, ni, nh, no):
    self.input_n = ni + 1
    self.hidden_n = nh
    self.output_n = no
    # init cells
    self.input_cells = [1.0] * self.input_n
    self.hidden_cells = [1.0] * self.hidden_n
    self.output_cells = [1.0] * self.output_n
    # init weights
    self.input_weights = make_matrix(self.input_n, self.hidden_n)
    self.output_weights = make_matrix(self.hidden_n, self.output_n)
    # random activate
```

```
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                self.input_weights[i][h] = rand(-0.2, 0.2)
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                self.output_weights[h][o] = rand(-2.0, 2.0)
        # init correction matrix
        self.input_correction = make_matrix(self.input_n, self.hidden_n)
        self.output_correction = make_matrix(self.hidden_n, self.output_n)


def predict(self, inputs):
        # activate input layer
        for i in range(self.input_n - 1):
            self.input_cells[i] = inputs[i]
        # activate hidden layer
        for j in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n):
                total += self.input_cells[i] * self.input_weights[i][j]
            self.hidden_cells[j] = sigmoid(total)
        # activate output layer
        for k in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
```

```
                total += self.hidden_cells[j] * self.output_weights[j][k]
            self.output_cells[k] = sigmoid(total)
        return self.output_cells[:]


    def back_propagate(self, case, label, learn, correct):
        # feed forward
        self.predict(case)
        # get output layer error
        output_deltas = [0.0] * self.output_n
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            output_deltas[o] = sigmod_derivate(self.output_cells[o]) * error
        # get hidden layer error
        hidden_deltas = [0.0] * self.hidden_n
        for h in range(self.hidden_n):
            error = 0.0
            for o in range(self.output_n):
                error += output_deltas[o] * self.output_weights[h][o]
            hidden_deltas[h] = sigmod_derivate(self.hidden_cells[h]) * error
        # update output weights
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                change = output_deltas[o] * self.hidden_cells[h]
                self.output_weights[h][o] += learn * change + correct *
self.output_correction[h][o]
```

```
                    self.output_correction[h][o] = change
            # update input weights
            for i in range(self.input_n):
                for h in range(self.hidden_n):
                    change = hidden_deltas[h] * self.input_cells[i]
                    self.input_weights[i][h] += learn * change + correct *
self.input_correction[i][h]
                    self.input_correction[i][h] = change
            # get global error
            error = 0.0
            for o in range(len(label)):
                error += 0.5 * (label[o] - self.output_cells[o]) ** 2
            return error


        def train(self, cases, labels, limit=10000, learn=0.05, correct=0.1):
            for i in range(limit):
                error = 0.0
                for i in range(len(cases)):
                    label = labels[i]
                    case = cases[i]
                    error += self.back_propagate(case, label, learn, correct)

        def test(self, casearray, labelarray, testarray):
            global answerarray
            cases = casearray
```

```
            labels = labelarray
            test_cases = testarray

            self.setup(1, 5, 1)
            self.train(cases, labels, 10000, 0.05, 0.1)
            for case in test_cases:
                return self.predict(case)

    if __name__ == '__main__':
        nn = BPNeuralNetwork()
        return nn.test(casearray, labelarray, testarray)


def shenjingwangluotwoinput(casearray, labelarray, testarray):

    random.seed(0)

    def logn(a, n):
        return math.log(n) / math.log(a)

    def rand(a, b):
        return (b - a) * random.random() + a

    def make_matrix(m, n, fill=0.0):
        mat = []
```

```
    for i in range(m):
        mat.append([fill] * n)
    return mat


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))


def sigmod_derivate(x):
    return x * (1 - x)


class BPNeuralNetwork:


    def __init__(self):
        self.input_n = 0
        self.hidden_n = 0
        self.output_n = 0
        self.input_cells = []
        self.hidden_cells = []
        self.output_cells = []
        self.input_weights = []
        self.output_weights = []
        self.input_correction = []
        self.output_correction = []


    def setup(self, ni, nh, no):
```

```
        self.input_n = ni + 1
        self.hidden_n = nh
        self.output_n = no
        # init cells
        self.input_cells = [1.0] * self.input_n
        self.hidden_cells = [1.0] * self.hidden_n
        self.output_cells = [1.0] * self.output_n
        # init weights
        self.input_weights = make_matrix(self.input_n, self.hidden_n)
        self.output_weights = make_matrix(self.hidden_n, self.output_n)
        # random activate
        for i in range(self.input_n):
            for h in range(self.hidden_n):
                self.input_weights[i][h] = rand(-0.2, 0.2)
        for h in range(self.hidden_n):
            for o in range(self.output_n):
                self.output_weights[h][o] = rand(-2.0, 2.0)
        # init correction matrix
        self.input_correction = make_matrix(self.input_n, self.hidden_n)
        self.output_correction = make_matrix(self.hidden_n, self.output_n)

    def predict(self, inputs):
        # activate input layer
        for i in range(self.input_n - 1):
            self.input_cells[i] = inputs[i]
```

```
        # activate hidden layer
        for j in range(self.hidden_n):
            total = 0.0
            for i in range(self.input_n):
                total += self.input_cells[i] * self.input_weights[i][j]
            self.hidden_cells[j] = sigmoid(total)
        # activate output layer
        for k in range(self.output_n):
            total = 0.0
            for j in range(self.hidden_n):
                total += self.hidden_cells[j] * self.output_weights[j][k]
            self.output_cells[k] = sigmoid(total)
        return self.output_cells[:]


    def back_propagate(self, case, label, learn, correct):
        # feed forward
        self.predict(case)
        # get output layer error
        output_deltas = [0.0] * self.output_n
        for o in range(self.output_n):
            error = label[o] - self.output_cells[o]
            output_deltas[o] = sigmod_derivate(self.output_cells[o]) * error
        # get hidden layer error
        hidden_deltas = [0.0] * self.hidden_n
        for h in range(self.hidden_n):
```

```
                error = 0.0
                for o in range(self.output_n):
                    error += output_deltas[o] * self.output_weights[h][o]
                hidden_deltas[h] = sigmod_derivate(self.hidden_cells[h]) * error
            # update output weights
            for h in range(self.hidden_n):
                for o in range(self.output_n):
                    change = output_deltas[o] * self.hidden_cells[h]
                    self.output_weights[h][o] += learn * change + correct *
self.output_correction[h][o]
                    self.output_correction[h][o] = change
            # update input weights
            for i in range(self.input_n):
                for h in range(self.hidden_n):
                    change = hidden_deltas[h] * self.input_cells[i]
                    self.input_weights[i][h] += learn * change + correct *
self.input_correction[i][h]
                    self.input_correction[i][h] = change
            # get global error
            error = 0.0
            for o in range(len(label)):
                error += 0.5 * (label[o] - self.output_cells[o]) ** 2
            return error

        def train(self, cases, labels, limit=10000, learn=0.05, correct=0.1):
            for i in range(limit):
```

```
                error = 0.0
                for i in range(len(cases)):
                    label = labels[i]
                    case = cases[i]
                    error += self.back_propagate(case, label, learn, correct)

        def test(self, casearray, labelarray, testarray):
            global answerarray
            cases = casearray
            labels = labelarray
            test_cases = testarray

            self.setup(2, 5, 1)
            self.train(cases, labels, 10000, 0.05, 0.1)
            for case in test_cases:
                return self.predict(case)

    if __name__ == '__main__':
        nn = BPNeuralNetwork()
        return nn.test(casearray, labelarray, testarray)


def personaltime(person, starttime, meettime, meetplace):
    flightdis = (person[0] - meetplace[0])**2 - (person[1] - meetplace[1])**2
    flighttime = flightdis * 4 / 3
```

```
    flighttime = int(flighttime)
    resttime = meettime - starttime - flighttime * datetime.timedelta(minutes=1)
    # print resttime
    resttime = countminutes(resttime)
    # print resttime
    jetlag = countjetlag(person[0], meetplace[0])
    eff = personalbraineff(resttime, flighttime, jetlag)
    cost = flighttime + resttime
    return [eff, cost]


def allperson(all, starttimearray, meettime, meetplace):
    alleff = []
    allcost = []
    for i in all:
        alleff.append(personaltime([i[0], i[1]], starttimearray[i[2]], meettime,
meetplace)[0])
        allcost.append(personaltime([i[0], i[1]], starttimearray[i[2]], meettime,
meetplace)[1])
    avereff = average(alleff)
    avercost = average(allcost)
    meeteff = avereff / avercost
    return meeteff


def hotelprise(meetplace):
```

```
    # prise = shenjingwangluotwoinput(hotelcasearray, hotellabelarray, [meetplace[0]
* 0.001, meetplace[1] * 0.001])
    prise = 1000
    return prise



def hotelcost(starttime, meetingtime, meetplace):
    resttime = meetingtime - starttime
    resttime = countminutes(resttime)
    if resttime > 180:
        resttime /= 60
        cost = (resttime // 24 + 1) * hotelprise(meetplace)
        return cost
    else:
        cost = 0
        return cost



def rad(d):
    return d * pi / 180



def round(d):
    return math.floor(d + 0.5)
```

```
def distance(lat1, lng1, lat2, lng2):
    radlat1 = rad(lat1)
    radlat2 = rad(lat2)
    a = radlat1 - radlat2
    b = rad(lng1) - rad(lng2)
    s = 2 * math.asin(math.sqrt(math.pow(math.sin(a / 2), 2) + math.cos(radlat1) *
math.cos(radlat2) *
                                        math.pow(math.sin(b / 2), 2)))
    s *= earth_r
    s = round(s * 10000) / 10000
    return s


def flightcost(x, y):
    all = 0
    all += distance(x, y, placeone[0], placeone[1])
    all += distance(x, y, placetwo[0], placetwo[1])
    all += distance(x, y, placethree[0], placethree[1])
    all += distance(x, y, placefour[0], placefour[1])
    all += distance(x, y, placefive[0], placefive[1])
    all += distance(x, y, placesix[0], placesix[1])
    all *= 0.94
    return all


def countminutes(deltatime):
```

```
    i = 0
    j = 0
    while i < 100000:
        if datetime.timedelta(minutes=i) == deltatime:
            return i - 1
        i += 1
    while i > -100000:
        if datetime.timedelta(minutes=i) == deltatime:
            return i + 1
        i -= 1



def countjetlag(lati1, lati2):
    shiqu1 = lati1 // 15
    shiqu2 = lati2 // 15
    timetochange = abs(shiqu1 - shiqu2)
    return timetochange


def personalbraineff(resttime, flighttime, jetlag):
    k = 5
    if resttime + flighttime < 0:
        eff = 0
        return eff
    else:
        eff = 1 - jetlag / 12 * (k / (resttime + flighttime + k))
```

```
        return eff



def average(allarray):
    sum = 0
    for i in allarray:
        sum += i
    return sum



def adaption(meetplace, starttimearray):
    alleffiency = allperson(all, starttimearray, meettime, meetplace)
    allhotelcost = 0
    for starttime in starttimearray:
        cost = hotelcost(starttime, meettime, meetplace)
        allhotelcost += cost
    flycost = flightcost(meetplace[0], meetplace[1])
    allcost = allhotelcost + flycost


    def sigmoid(x):
        return 1.0 / (1.0 + math.exp(-x))


    k = 10000
    adptn = (alleffiency + sigmoid(k / allcost)) / 2
```

```
    return adptn



meet_time = '2017-06-15 10:00:00'

earlist_time = '2017-06-8 10:00:00'



meettime = datetime.datetime.strptime(meet_time,'%Y-%m-%d %H:%M:%S')

earlisttime = datetime.datetime.strptime(earlist_time,'%Y-%m-%d %H:%M:%S')

delta = meettime - earlisttime

datetime.timedelta(minutes=1)

print delta

print countminutes(delta)

possibleairport = [

[ 116.478157,39.905498 ],

[ 116.621214,40.062281 ],

[ 105.882967,26.25988 ],

[ 117.05906,30.588582 ],

[ 116.410206,39.946267 ],

[ 116.367509,39.79004 ],
```

[116.444556,39.924644],

[116.481089,39.942221],

[116.481089,39.942221],

[90.123401,31.218112],

[109.846239,40.647119],

[109.122628,21.472718],

[116.603409,40.088208],

[116.40063,39.79055],

[116.465209,39.938641],

[116.417278,40.085974],

[116.234368,40.268034],

[116.523679,40.107461],

[116.40755,39.833741],

[125.313642,43.898338],

[113.226123,28.198972],

[120.755996,38.077456],

[113.135075,36.252866],

[119.793648,31.918471],

[103.965093,30.58483],

[106.651538,29.722384],

[107.424315,31.182047],

[116.309659,39.939725],

[116.462281,39.954764],

[100.219209,25.693967],

[113.496054,40.06268],

[116.412216,39.791554],

[116.412216,39.791554],

[98.967481,32.059409],

[99.713682,27.831029],

[116.485382,40.013074],

[109.764419,39.805586],

[116.478157,39.905498],

[119.67945,25.934447],

[108.449166,21.662036],

[116.543582,40.061558],

[116.543582,40.061558],

[116.543582,40.061558],

[113.272872,23.190937],

[106.806675,26.547013],

[110.058387,25.221013],

[126.249602,45.628829],

[120.042882,49.279245],

[107.021411,33.069791],

[120.441409,30.247224],

[117.311591,31.784174],

[111.826212,40.864065],

[116.367509,39.79004],

[116.367509,39.79004],

[118.267355,29.736128],

[115.050683,30.216127],

[117.220266,36.857816],

[116.205131,39.875936],

[116.315294,39.939225],

[117.188351,29.341811],

[121.078611,41.114521],

[98.508415,39.741474],

[116.4511,39.848957],

[116.267404,39.953913],

[102.749725,25.003252],

[116.234368,40.268034],

[116.523679,40.107461],

[116.332912,37.457437],

[91.111891,29.662557],

[116.391732,39.850796],

[116.391732,39.850796],

[103.624668,36.513577],

[100.25485,26.674978],

[116.478157,39.905498],

[100.033121,23.746964],

[118.886693,34.575653],

[118.419563,35.053453],

[109.409607,24.20871],

[105.389595,28.848193],

[116.38025,39.91557],

[116.523679,40.107461],

[116.523679,40.107461],

[112.447525,34.657368],

[116.38025,39.91557],

[116.234368,40.268034],

[116.199543,24.371491],

[116.422888,39.921439],

[116.551824,39.766007],

[116.321861,39.853897],

[116.321861,39.853897],

[116.234368,40.268034],

[104.705519,31.504701],

[116.266333,39.912893],

[129.59619,44.54397],

[116.415448,39.913665],

[115.918117,28.864379],

[106.105554,30.800965],

[108.178307,22.615295],

[118.877696,31.738884],

[120.989153,32.078061],

[121.473447,29.824468],

[112.626393,32.987098],

[116.396768,39.94541],

[116.527691,39.92942],

[120.388394,36.272427],

[118.598548,24.806085],

[111.026435,25.936465],

[120.46834,27.829231],

[121.8105,31.155342],

[116.769877,23.433998],

[121.34514,31.202595],

[123.500123,41.643278],

[113.818645,22.632856],

[116.48489,39.915476],

[116.183929,39.931486],

[114.698938,38.288955],

[116.343317,39.998843],

[116.494982,39.994492],

[118.757106,42.268753],

[117.345264,35.640741],

[116.186367,39.964747],

[112.637907,37.761473],

[121.973871,24.086957],

[121.973871,24.086957],

[117.210813,39.14393],

[116.417278,40.085974],

[121.968291,37.166083],

[116.40459,39.975691],

[116.460318,39.960018],

[116.460318,39.960018],

[116.36088,39.929828],

[119.126592,36.64727],

[116.309345,39.918423],

[120.856993,27.917074],

[87.491727,43.903344],

[120.43941,31.508847],

[118.011525,27.710607],

[114.220217,30.781008],

[102.197595,27.982227],

[116.287063,39.880667],

[116.287063,39.880667],

[101.767921,36.640739],

[100.927318,22.111413],

[116.577541,40.058249],

[83.895485,45.656986],

[118.143501,24.54742],

[115.910511,40.040598],

[116.435299,39.852194],

[117.566296,34.062469],

[121.380693,37.406298],

[109.477197,36.526644],

[129.454833,42.895332],

[116.444943,39.891227],

[116.562727,39.9259],

[114.478087,33.793632],

[111.492255,30.554257],

```
[ 104.562954,28.801335 ],

[ 106.397489,38.331163 ],

[ 110.48162,29.124889 ],

[ 113.852605,34.531462 ],

[ 122.365915,29.938168 ],

[ 116.364563,39.914667 ],

[ 113.383726,22.015148 ],

]


system = []

n = 100

def init():

    while len(system) < n:


        meetplace = possibleairport[random.randint(0, len(possibleairport) - 1)]

        timearray = []

        for i in range(0, 6):

            time = earlisttime + datetime.timedelta(minutes=60) * random.randint(0,
168)

            timearray.append(time)

        system.append([meetplace, timearray])

    for i in system:

        print i, "/n"


"""

def prechoose():
```

```
    global system
    for answer in system:
        for pertime in answer[1]:
            flightdis = int(distance(personone[0], personone[1], answer[0][0],
answer[0][1]))
            print flightdis
            flighttime = flightdis * 3 / 4
            flighttime = int(flighttime)
            if pertime + datetime.timedelta(minutes=1) * flighttime > meettime:
                del answer
    print system
"""

def countadaption(meetplace, starttimearray):
    return(adaption(meetplace, starttimearray))



def averadpt():
    global system
    global adpt
    sum = 0
    for i in adpt:
        sum += i
    sum /= len(adpt)
    j = 0
    print "####################################"
```

```
    print sum
    print "####################################"
    todelete = []
    for i in adpt:
        if i < sum:
            todelete.append(j)
        j += 1
    print todelete
    todelete.reverse()
    print todelete
    for i in todelete:
        del system[i]
    print len(system)



def change():
    global system, n, adpt
    sum = 0
    for i in adpt:
        sum += i
    sum /= len(adpt)
    while len(system) < n:
        p1 = random.randint(0, len(system) - 1)
        p2 = random.randint(0, len(system) - 1)
```

```
            f = [system[p1][1][0], system[p2][1][1], system[p1][1][2], system[p2][1][3],
system[p1][1][4], system[p2][1][5]]
            if countadaption(possibleairport[random.randint(0, len(possibleairport) - 1)],
f) > sum:
                system.append([possibleairport[random.randint(0, len(possibleairport) -
1)], f])
                print "#"
        """
        while len(system) < n:
            meetplace = possibleairport[random.randint(0, len(possibleairport) - 1)]
            timearray = []
            for i in range(0, 6):
                time = earlisttime + datetime.timedelta(minutes=60) * random.randint(0,
240)
                timearray.append(time)
                system.append([meetplace, timearray])
                print "$"
        """



init()
print"#######"
adpt = []
for i in system:
        print countadaption(i[0], i[1])
        adpt.append(countadaption(i[0], i[1]))
print adpt
```

```
averadpt()


change()


averadpt()


m = 0
while m < 10:
    change()
    adpt = []
    for i in system:
        print countadaption(i[0], i[1])
        adpt.append(countadaption(i[0], i[1]))
    averadpt()
    m += 1


print system
```

主程序（进化+神经网络）（大）与小会议主程序基本相同

附录 2 优秀值函数绘制

```
# -*- coding:gb2312 -*-
from __future__ import division
import math
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


k = 10000


def sigmoid(x):
    return 1.0 / (1.0 + math.exp(-x))



def adaption(alleff, allcost):
    adptn = (alleff + sigmoid(k / allcost)) / 2
    return adptn



def floatrange(start,stop,steps):
    return [start + float(i) * (stop - start) / (float(steps) - 1) for i in range(steps)]


hax = floatrange(0, 1, 100)
hay = range(2000, 10000, 10)


x = []
y = []
z = []


for hx in hax:
```

```
    for hy in hay:
        x.append(hx)
        y.append(hy)
        z.append(adaption(hx, hy))

print x
print y
print z

fig = plt.figure()
ax = Axes3D(fig)
ax.plot_trisurf(x, y, z)
ax.set_zlabel('Excellent value')
ax.set_ylabel('Cost')
ax.set_xlabel('Efficiency')
plt.show()
```

附录 3 大脑灵敏度函数图象绘制

```
from __future__ import division
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
```

```
def personalbraineff(resttime_and_flighttime, jetlag):
    k = 5
    if resttime_and_flighttime < 0:
        eff = 0
        return eff
    else:
        eff = 1 - jetlag / 12 * (k / (resttime_and_flighttime + k))
        return eff



flighttime_and_reattime = range(0, 168, 1)
jetlag = range(0, 12, 1)

disx = []
disy = []
disz = []

for x in flighttime_and_reattime:
    for y in jetlag:
        disx.append(x)
        disy.append(y)
        disz.append(personalbraineff(x, y))

print disx
```

```
print disy

print disz


fig = plt.figure()

ax = Axes3D(fig)

ax.plot_trisurf(disx, disy, disz)

ax.set_zlabel('Brain_Efficiency')

ax.set_ylabel('Absolute_jet lag')

ax.set_xlabel('Time from flight to meeting')

plt.show()
```

附录 4 计算经纬度到所有人距离和（小）

```
import math

import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D


earth_r = 6378.137

pi = 3.1415926535


placeone = [36.588, -121.848]

placetwo = [52.344, 4.956]

placethree = [-37.671, 144.849]

placefour = [31.196, 121.340]

placefive = [22.304, 113.924]

placesix = [55.410, 37.902]
```

```
def rad(d):
    return d * pi / 180


def round(d):
    return math.floor(d + 0.5)


def distance(lat1, lng1, lat2, lng2):
    radlat1 = rad(lat1)
    radlat2 = rad(lat2)
    a = radlat1 - radlat2
    b = rad(lng1) - rad(lng2)
    s = 2 * math.asin(math.sqrt(math.pow(math.sin(a / 2), 2) + math.cos(radlat1) *
math.cos(radlat2) *
                                math.pow(math.sin(b / 2), 2)))
    s *= earth_r
    s = round(s * 10000) / 10000
    return s


def all_distance(x, y):
    all = 0
    all += distance(x, y, placeone[0], placeone[1])
```

```
    all += distance(x, y, placetwo[0], placetwo[1])
    all += distance(x, y, placethree[0], placethree[1])
    all += distance(x, y, placefour[0], placefour[1])
    all += distance(x, y, placefive[0], placefive[1])
    all += distance(x, y, placesix[0], placesix[1])

    all *= 0.94
    return all

lati = range(-90, 91, 2)
long = range(-180, 181, 2)

disx = []
disy = []
disz = []

for x in long:
    for y in lati:
        disx.append(x)
        disy.append(y)
        disz.append(all_distance(x, y))

fig = plt.figure()
```

```
ax = Axes3D(fig)

ax.plot_trisurf(disx, disy, disz)

ax.set_zlabel('Distance')

ax.set_ylabel('Latitude')

ax.set_xlabel('Longitude')

plt.show()
```

计算经纬度到所有人距离和（大）与小会议算法基本相同

附录 5 经纬爬虫

```
<html><head>

    <meta charset="UTF-8">

    <meta name="keywords" content="IMMC 17460464  地点-经纬转换查询">

    <link rel="stylesheet" type="text/css" href="css/basic.css" media="all">

    v=2.0&amp;ak=CG8eakl6UTlEb1OakeWYvofh"></script><script
type="text/javascript"
src="http://api.map.baidu.com/getscript?v=2.0&amp;ak=CG8eakl6UTlEb1OakeWYvofh
&amp;services=&amp;t=20170324173232"></script>

    <script type="text/javascript"
src="http://api.map.baidu.com/library/TextIconOverlay/1.2/src/TextIconOverlay_min.
js"></script>

    <script type="text/javascript"
src="http://api.map.baidu.com/library/MarkerClusterer/1.2/src/MarkerClusterer_min.
js"></script>

    <script type="text/javascript">

        window.onload = function () {
```

```
        // 百度地图 API 功能
        var map = new BMap.Map("map_canvas");
        map.enableDragging();
        map.enableScrollWheelZoom();
        var point = new BMap.Point(114.05786799999998, 22.543099);
        map.centerAndZoom(point, 12);

        var myGeo = new BMap.Geocoder();

        var markerClusterer = new BMapLib.MarkerClusterer(map, {markers:
[]});

        $(function () {
            $('#toLatLngBtn').live('click', function (e) {
                $('#showResults').html("").fadeIn();
                map.clearOverlays();
                markerClusterer.clearMarkers();
                var addrStr = $('#addrs').val();
                var addrs = addrStr.split('\n');
                for (var i in addrs) {
                    var addr = addrs[i];
                    geoSearch(addr);
                }
                e.stopImmediatePropagation();
            });
```

```
        $('#toAddressBtn').live('click', function (e) {
            $('#showResults').html("").fadeIn();
            map.clearOverlays();
            markerClusterer.clearMarkers();
            makers = [];
            var addrStr = $('#latLng').val();
            var addrs = addrStr.split('\n');
            var inProgressing = false;
            for (var i in addrs) {
                var addr = addrs[i];
                geoParse(addr);
            }

            //最简单的用法，生成一个 marker 数组，然后调用
markerClusterer 类即可。
            e.stopImmediatePropagation();
        });

        function geoSearch(addr) {
            myGeo.getPoint(addr, function (point) {
                if (point) {
                    var str = addr + ":" + point.lng + "," + point.lat + "<br>";
                    var po = new BMap.Point(point.lng, point.lat);
                    map.centerAndZoom(po, 12);

                    var _marker = new BMap.Marker(po);
```

```
                    _marker.addEventListener("click", function (e) {
                        this.openInfoWindow(new
BMap.InfoWindow(str));
                    });

                    _marker.addEventListener("mouseover", function (e) {
                        this.setTitle("位于: " + point.lng + "," + point.lat);
                    });

                    markerClusterer.addMarker(_marker);
                    map.addOverlay(_marker);                    // 将标注
添加到地图中
                    $("#showResults").append(str);
                }
            });
        }


        function geoParse(str) {
            str = str.toString();
            //去除中间所有空格，将中文'，'号替换成英文','并按','分割
            str = str.replace(/[(^\s+)(\s+$)]/g, "").replace('，', ',').split(',');
            //第一个值为纬度并转化为 float 类型
            var lat = parseFloat(str[1]);
            //第二个值为经度并转化为 float 类型
```

```
                var lng = parseFloat(str[0]);
                if (lat == 0 || lng == 0 || isNaN(lat) || isNaN(lng)) return false;
                var po = new BMap.Point(lng, lat);
                console.log(po);
                myGeo.getLocation(po, function (rs) {
                    if (rs) {
                        var str = lng + "," + lat + "：" + rs.address + '<br>';
                        var po = new BMap.Point(lng, lat);
                        var _marker = new BMap.Marker(po);

                        _marker.addEventListener("click", function (e) {
                            this.openInfoWindow(new
BMap.InfoWindow(str));
                        });

                        _marker.addEventListener("mouseover", function (e) {
                            this.setTitle("位于: " + point.lng + "," + point.lat);
                        });

                        markerClusterer.addMarker(_marker);
                        map.centerAndZoom(po, 12);
                        map.addOverlay(_marker);                    // 将标注
添加到地图中

                        $('#showResults').append(str);
                    }
```

```
                });
            }



            $('.clear').live('click', function () {
                $('#showResults').html("");
            });
            $('.hide').live('click', function () {
                $('#showResults').fadeOut();
            });
            $('.expand').live('click', function () {
                $('#showResults').fadeIn();
            });
        });
    }
</script>
<style type="text/css">
    #main {
        border: 1px solid #f0f0f0;
        padding: 10px;
    }
    #main #left {
        float: left;
        width: 400px;
        margin: 0 10px 0;
```

```
}
#main #right {
    float: right;
    margin: 0 10px 0 0;
    width: 520px;
}
h2 {
    font-size: 16px;
    line-height: 42px;
    height: 42px;
    color: #333;
    font-family: '微软雅黑';
    border-bottom: 1px solid #f0f0f0;
    text-shadow: 1px 1px 3px rgba(0, 0, 0, .3);
    margin: 8px 0;
}
h2 button {
    float: right;
}
h2 span {
    font-weight: normal;
    font-size: 12px;
    color: #666;
}
h3 a, h2 a {
```

```
        color: #06f;
    }
    h3 a:hover, h2 a:hover {
        color: #03f;
        text-shadow: 1px 1px 3px rgba(0, 0, 0, .3);
    }
    h3 {
        font-size: 16px;
        line-height: 30px;
        height: 30px;
        color: #333;
        font-family: '微软雅黑';
        border-bottom: 1px solid #f0f0f0;
        text-shadow: 1px 1px 3px rgba(0, 0, 0, .3);
        margin: 8px 0;
    }
    h3 em {
        font-weight: normal;
        font-size: 12px;
        color: #666;
        font-style: normal;
    }
    h3 span {
        font-weight: normal;
        font-size: 12px;
```

```
        color: #666;
        float: right;
    }
    #addrs {
        width: 388px;
        height: 160px;
        border: 1px solid #e0e0e0;
        border-radius: 5px;
        padding: 8px 5px;
        font-size: 12px;
        color: #666;
    }
    #latLng {
        width: 298px;
        height: 100px;
        border: 1px solid #e0e0e0;
        border-radius: 5px;
        padding: 8px 5px;
        font-size: 12px;
        color: #666;
    }
    #toAddressDiv {
        float: left;
        width: 310px;
    }
```

```
#info {
    margin: 12px 6px 0 0;
    float: right;
    width: 180px;
}
#info p {
    border: 1px solid #f0f0f0;
    color: #666;
    border-radius: 5px;
    padding: 10px 20px;
    line-height: 24px;
}
#showResults {
    border: 1px solid #e0e0e0;
    height: 318px;
    padding: 8px;
    overflow-y: auto;
    margin: 0 0 8px 0;
}
#map_canvas {
    background: url(images/load_s.gif) no-repeat center center;
}
#hm_t_30690 {
    display: none;
}
```

```
    </style>


</div>
<div class="wrap">


    <div id="main">
        <h1 class="title">经纬-地点查询</h1>


        <div id="content">
            <div id="left">
                <h2>
                    <button id="toLatLngBtn" class="button">解析地名</button>
                    输入地名<span>(每个地名一行) <a href="javascript:;"
id="clearAddress">清空输入</a></span>
                </h2>
                <textarea name="" id="addrs" class="input">IMMC</textarea>


                <h3><span><a href="">重新载入</a></span>地图展示<em>(基于
百度地图)</em></h3>


                <div id="map_canvas" style="width: 398px; height: 298px; border:
1px solid rgb(224, 224, 224); overflow: hidden; position: relative; z-index: 0;
background-color: rgb(243, 241, 236); color: rgb(0, 0, 0); text-align: left;"><div
style="overflow: visible; position: absolute; z-index: 0; left: 0px; top: 0px; cursor:
url(http://api0.map.bdimg.com/images/openhand.cur) 8 8, default;"><div
class="BMap_mask" style="position: absolute; left: 0px; top: 0px; z-index: 9; overflow:
hidden; -webkit-user-select: none; width: 398px; height: 298px;"></div><div
style="position: absolute; height: 0px; width: 0px; left: 0px; top: 0px; z-index:
```

200;"> <label class="BMapLabel" unselectable="on" style="position: absolute; display: none; cursor: inherit; background-color: rgb(190, 190, 190); border: 1px solid rgb(190, 190, 190); padding: 1px; white-space: nowrap; font-style: normal; font-variant-caps: normal; font-weight: normal; font-size: 12px; line-height: normal; font-family: arial, sans-serif; z-index: -20000; color: rgb(190, 190, 190);">shadow</label> <img src="http://online0.map.bdimg.com/tile/?qt=tile&x=774&y=156&z=12&styles=pl&scaler=2&udt=20170301" style="position: absolute; border: none; width: 256px; height: 256px; left: -246px; top: -186px; max-width: none; opacity: 1;"><img src="http://online1.map.bdimg.com/tile/?qt=tile&x=775&y=156&z=12&styles=pl&scaler=2&udt=20170301" style="position: absolute; border: none; width: 256px; height: 256px; left: 10px; top: -186px; max-width: none; opacity: 1;"><img src="http://online4.map.bdimg.com/tile/?qt=tile&x=774&y=155&z=12&styles=pl&scaler=2&udt=20170301" style="position: absolute; border: none; width: 256px; height: 256px; left: -246px; top: 70px; max-width: none; opacity: 1;"><img src="http://online0.map.bdimg.com/tile/?qt=tile&x=775&y=155&z=12&styles=pl&scaler=2&udt=20170301" style="position: absolute; border: none; width: 256px; height: 256px; left: 10px; top: 70px; max-width: none; opacity: 1;"> <div class="pano_close"

title="退出全景" style="z-index: 1201; display: none;"> 出口 <img style="border:none;width:77px;height:32px" src="http://api0.map.bdimg.com/images/copyright_logo.png"> <div unselectable="on" class=" BMap_cpyCtrl BMap_noprint anchorBL" style="cursor: default; white-space: nowrap; color: black; background-image: none; font-style: normal; font-variant-caps: normal; font-weight: normal; font-size: 11px; line-height: 15px; font-family: arial, sans-serif; bottom: 2px; right: auto; top: auto; left: 2px; position: absolute; z-index: 10; background-position: initial initial; background-repeat: initial initial;"><span _cid="1" style="display: inline;">© 2017 Baidu - GS(2015)2650 号 - Data © 长地万方

<h2>

<button id="toAddressBtn" class="button">解析经纬度</button>

输入经纬度 清空输入

</h2>

```
                    <p style="line-height:24px;color:#999;">每个经纬度换一行，格式：经度,纬度</p>
                    <textarea name="" id="latLng" class="input">17460464</textarea>
                </div>



                </div>
                <div class="clear"></div>
                <h3>
                    <span><a href="javascript:;" id="clearResult">清空结果</a></span>解析结果
                </h3>

                <div id="showResults">
                    <p style="color: #999;">等待解析</p>
                </div>

            </div>


            <div class="clear"></div>

        </div>
    </div>
</div>
```

<script type="text/javascript">

var _bdhmProtocol = (("https:" == document.location.protocol) ? " https://" : " http://");

document.write(unescape("%3Cscript src='" + _bdhmProtocol + "hm.baidu.com/h.js%3F2a1063cadb2ff3dbe7a08094b86840e7' type='text/javascript'%3E%3C/script%3E"));

</script><script src=" http://hm.baidu.com/h.js?2a1063cadb2ff3dbe7a08094b86840e7" type="text/javascript"></script>

</body></html>

附录 5 结果分析（小）（与大会议算法基本相同）

```
import collections
import datetime
from matplotlib import pyplot as plt
import numpy as np

system = [[[100.219209, 25.693967], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),
                                    datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),
                                    datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],
          [[83.895485, 45.656986], [datetime.datetime(2017, 6, 14, 20, 0),
datetime.datetime(2017, 6, 15, 2, 0),
```

datetime.datetime(2017, 6, 14, 7, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 2, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[83.895485, 45.656986], [datetime.datetime(2017, 6, 14, 11, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 14, 17, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 12, 12, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[87.491727, 43.903344], [datetime.datetime(2017, 6, 14, 20, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 14, 7, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 2, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[99.713682, 27.831029], [datetime.datetime(2017, 6, 14, 20, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 14, 7, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 2, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[107.424315, 31.182047], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[103.965093, 30.58483], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[104.705519, 31.504701], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[106.105554, 30.800965], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[98.967481, 32.059409], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[101.767921, 36.640739], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[104.705519, 31.504701], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[98.508415, 39.741474], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[113.818645, 22.632856], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[100.219209, 25.693967], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[106.806675, 26.547013], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[111.026435, 25.936465], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[106.651538, 29.722384], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[98.967481, 32.059409], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[109.409607, 24.20871], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[113.383726, 22.015148], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[102.197595, 27.982227], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[108.178307, 22.615295], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[101.767921, 36.640739], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[109.122628, 21.472718], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[109.122628, 21.472718], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[100.033121, 23.746964], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[104.562954, 28.801335], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[108.178307, 22.615295], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[107.021411, 33.069791], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[106.397489, 38.331163], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[104.562954, 28.801335], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[113.818645, 22.632856], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[101.767921, 36.640739], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[91.111891, 29.662557], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[103.965093, 30.58483], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[106.651538, 29.722384], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[91.111891, 29.662557], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[110.48162, 29.124889], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[103.624668, 36.513577], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[111.026435, 25.936465], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[113.272872, 23.190937], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[102.197595, 27.982227], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[102.749725, 25.003252], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[113.272872, 23.190937], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[105.882967, 26.25988], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[90.123401, 31.218112], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[107.424315, 31.182047], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[108.178307, 22.615295], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[87.491727, 43.903344], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[113.272872, 23.190937], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[98.508415, 39.741474], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[107.424315, 31.182047], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[106.397489, 38.331163], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[102.749725, 25.003252], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]],

[[110.48162, 29.124889], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 12, 14, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 14, 19, 0),

datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 15, 1, 0)]],

[[108.178307, 22.615295], [datetime.datetime(2017, 6, 15, 6, 0),
datetime.datetime(2017, 6, 15, 2, 0),

datetime.datetime(2017, 6, 15, 5, 0),
datetime.datetime(2017, 6, 12, 12, 0),
datetime.datetime(2017, 6, 15, 1, 0),
datetime.datetime(2017, 6, 14, 16, 0)]]
]

possibleairport = [
[116.478157,39.905498],
[116.621214,40.062281],
[105.882967,26.25988],
[117.05906,30.588582],
[116.410206,39.946267],
[116.367509,39.79004],
[116.444556,39.924644],
[116.481089,39.942221],
[116.481089,39.942221],
[90.123401,31.218112],
[109.846239,40.647119],
[109.122628,21.472718],
[116.603409,40.088208],
[116.40063,39.79055],
[116.465209,39.938641],
[116.417278,40.085974],
[116.234368,40.268034],
[116.523679,40.107461],

[116.40755,39.833741],

[125.313642,43.898338],

[113.226123,28.198972],

[120.755996,38.077456],

[113.135075,36.252866],

[119.793648,31.918471],

[103.965093,30.58483],

[106.651538,29.722384],

[107.424315,31.182047],

[116.309659,39.939725],

[116.462281,39.954764],

[100.219209,25.693967],

[113.496054,40.06268],

[116.412216,39.791554],

[116.412216,39.791554],

[98.967481,32.059409],

[99.713682,27.831029],

[116.485382,40.013074],

[109.764419,39.805586],

[116.478157,39.905498],

[119.67945,25.934447],

[108.449166,21.662036],

[116.543582,40.061558],

[116.543582,40.061558],

[116.543582,40.061558],

[113.272872,23.190937],

[106.806675,26.547013],

[110.058387,25.221013],

[126.249602,45.628829],

[120.042882,49.279245],

[107.021411,33.069791],

[120.441409,30.247224],

[117.311591,31.784174],

[111.826212,40.864065],

[116.367509,39.79004],

[116.367509,39.79004],

[118.267355,29.736128],

[115.050683,30.216127],

[117.220266,36.857816],

[116.205131,39.875936],

[116.315294,39.939225],

[117.188351,29.341811],

[121.078611,41.114521],

[98.508415,39.741474],

[116.4511,39.848957],

[116.267404,39.953913],

[102.749725,25.003252],

[116.234368,40.268034],

[116.523679,40.107461],

[116.332912,37.457437],

[91.111891,29.662557],

[116.391732,39.850796],

[116.391732,39.850796],

[103.624668,36.513577],

[100.25485,26.674978],

[116.478157,39.905498],

[100.033121,23.746964],

[118.886693,34.575653],

[118.419563,35.053453],

[109.409607,24.20871],

[105.389595,28.848193],

[116.38025,39.91557],

[116.523679,40.107461],

[116.523679,40.107461],

[112.447525,34.657368],

[116.38025,39.91557],

[116.234368,40.268034],

[116.199543,24.371491],

[116.422888,39.921439],

[116.551824,39.766007],

[116.321861,39.853897],

[116.321861,39.853897],

[116.234368,40.268034],

[104.705519,31.504701],

[116.266333,39.912893],

[129.59619,44.54397],

[116.415448,39.913665],

[115.918117,28.864379],

[106.105554,30.800965],

[108.178307,22.615295],

[118.877696,31.738884],

[120.989153,32.078061],

[121.473447,29.824468],

[112.626393,32.987098],

[116.396768,39.94541],

[116.527691,39.92942],

[120.388394,36.272427],

[118.598548,24.806085],

[111.026435,25.936465],

[120.46834,27.829231],

[121.8105,31.155342],

[116.769877,23.433998],

[121.34514,31.202595],

[123.500123,41.643278],

[113.818645,22.632856],

[116.48489,39.915476],

[116.183929,39.931486],

[114.698938,38.288955],

[116.343317,39.998843],

[116.494982,39.994492],

[118.757106,42.268753],

[117.345264,35.640741],

[116.186367,39.964747],

[112.637907,37.761473],

[121.973871,24.086957],

[121.973871,24.086957],

[117.210813,39.14393],

[116.417278,40.085974],

[121.968291,37.166083],

[116.40459,39.975691],

[116.460318,39.960018],

[116.460318,39.960018],

[116.36088,39.929828],

[119.126592,36.64727],

[116.309345,39.918423],

[120.856993,27.917074],

[87.491727,43.903344],

[120.43941,31.508847],

[118.011525,27.710607],

[114.220217,30.781008],

[102.197595,27.982227],

[116.287063,39.880667],

[116.287063,39.880667],

[101.767921,36.640739],

[100.927318,22.111413],

[116.577541,40.058249],

[83.895485,45.656986],

[118.143501,24.54742],

[115.910511,40.040598],

[116.435299,39.852194],

[117.566296,34.062469],

[121.380693,37.406298],

[109.477197,36.526644],

[129.454833,42.895332],

[116.444943,39.891227],

[116.562727,39.9259],

[114.478087,33.793632],

[111.492255,30.554257],

[104.562954,28.801335],

[106.397489,38.331163],

[110.48162,29.124889],

[113.852605,34.531462],

[122.365915,29.938168],

[116.364563,39.914667],

[113.383726,22.015148],

]

airportstime = []

airportstime1 = []

```
for i in system:
    airportstime.append(i[0])


print airportstime




plt.figure(figsize=(9,6))
x = []
y = []
for i in airportstime:
    x.append(i[0])
    y.append(i[1])
T = np.arctan2(x, y)
plt.scatter(x, y, c=T, s=25, alpha=0.4, marker='o')
plt.show()
```

附录 6 机场经纬

阿布杜拉齐兹国王国际机场:116.478157,39.905498

阿迪苏特吉普托机场:116.621214,40.062281

安顺黄果树机场:105.882967,26.25988

安庆天柱山机场:117.05906,30.588582

奥斯汀博格斯托姆国际机场:116.410206,39.946267

巴尔第摩华盛顿国际机场:116.367509,39.79004

巴库国际机场:116.444556,39.924644

巴黎奥利机场:116.481089,39.942221

巴黎戴高乐国际机场:116.481089,39.942221

班戈国际机场:90.123401,31.218112

包头机场:109.846239,40.647119

北海机场:109.122628,21.472718

北京首都国际机场:116.603409,40.088208

北京南苑机场:116.40063,39.79055

波音菲尔德国际机场:116.465209,39.938641

布拉格鲁兹耶内机场:116.417278,40.085974

布雷斯将军机场:116.234368,40.268034

布罗马机场:116.523679,40.107461

查德隆机场:116.40755,39.833741

长春龙嘉机场:125.313642,43.898338

长沙黄花机场:113.226123,28.198972

长岛麦克阿瑟机场:120.755996,38.077456

长治机场:113.135075,36.252866

常州奔牛机场:119.793648,31.918471

成都双流国际机场:103.965093,30.58483

重庆江北国际机场:106.651538,29.722384

达县机场:107.424315,31.182047

大草原城机场:116.309659,39.939725

大都会地区机场:116.462281,39.954764

大理市机场:100.219209,25.693967

大同机场:113.496054,40.06268

德尔罗萨里奥港机场:116.412216,39.791554

德尔内机场:116.412216,39.791554

德格兰－卡纳里亚机场:98.967481,32.059409

迪庆机场:99.713682,27.831029

东京成田国际机场:116.485382,40.013074

鄂尔多斯东胜机场:109.764419,39.805586

法赫德国王国际机场:116.478157,39.905498

福州长乐国际机场:119.67945,25.934447

港口区-王子机场:108.449166,21.662036

格兰特郡机场:116.543582,40.061558

格兰特郡机场:116.543582,40.061558

格兰特雷－亚当斯国际机场:116.543582,40.061558

广州白云机场:113.272872,23.190937

贵阳龙洞堡机场:106.806675,26.547013

桂林两江机场:110.058387,25.221013

哈尔滨太平国际机场:126.249602,45.628829

海拉尔机场:120.042882,49.279245

汉中机场:107.021411,33.069791

杭州萧山机场:120.441409,30.247224

合肥骆岗机场:117.311591,31.784174

呼和浩特白塔机场:111.826212,40.864065

华盛顿国内机场:116.367509,39.79004

华盛顿杜勒斯国际机场:116.367509,39.79004

黄山屯溪机场:118.267355,29.736128

黄石地区机场:115.050683,30.216127

济南遥墙机场:117.220266,36.857816

加德满都机场:116.205131,39.875936

金边机场:116.315294,39.939225

坦佩雷－博卡拉机场:117.188351,29.341811

锦州机场:121.078611,41.114521

酒泉嘉峪关机场:98.508415,39.741474

开普吉拉多机场:116.4511,39.848957

凯捷维克机场:116.267404,39.953913

昆明巫家坝机场:102.749725,25.003252

拉法尔布埃尔纳将军机场:116.234368,40.268034

拉罗马纳机场:116.523679,40.107461

拉皮德城地区机场:116.332912,37.457437

拉萨贡嘎机场:91.111891,29.662557

莱克查尔斯地方机场:116.391732,39.850796

莱克哈瓦苏城地方机场:116.391732,39.850796

兰州中川机场:103.624668,36.513577

丽江三义机场:100.25485,26.674978

利雅得卡雷德国王国际机场:116.478157,39.905498

临沧机场:100.033121,23.746964

连云港白塔埠机场:118.886693,34.575653

临沂机场:118.419563,35.053453

柳州白莲机场:109.409607,24.20871

泸州机场:105.389595,28.848193

路易斯－威尔逊机场:116.38025,39.91557

罗马费尤米西诺国际机场:116.523679,40.107461

罗马西阿姆皮诺国际机场:116.523679,40.107461

洛阳机场:112.447525,34.657368

落基山城－威尔逊机场:116.38025,39.91557

马里阿诺－埃斯科贝多将军机场:116.234368,40.268034

梅县机场:116.199543,24.371491

米盖尔－希达尔机场:116.422888,39.921439

米拉马雷机场:116.551824,39.766007

米兰里纳特国际机场:116.321861,39.853897

米兰马尔潘萨国际机场:116.321861,39.853897

密尔沃基米歇尔将军机场:116.234368,40.268034

绵阳机场:104.705519,31.504701

名古屋国际机场:116.266333,39.912893

牡丹江机场:129.59619,44.54397

那不勒斯机场:116.415448,39.913665

南昌昌北机场:115.918117,28.864379

南充机场:106.105554,30.800965

南宁吴圩机场:108.178307,22.615295

南京禄口国际机场:118.877696,31.738884

南通兴东机场:120.989153,32.078061

宁波栎社机场:121.473447,29.824468

南阳机场:112.626393,32.987098

纽瓦克自由国际机场:116.396768,39.94541

诺伊拜机场:116.527691,39.92942

青岛流亭机场:120.388394,36.272427

泉州晋江机场:118.598548,24.806085

全州机场:111.026435,25.936465

瑞安机场:120.46834,27.829231

上海浦东机场:121.8105,31.155342

汕头外砂机场:116.769877,23.433998

上海虹桥机场:121.34514,31.202595

沈阳桃仙国际机场:123.500123,41.643278

深圳宝安国际机场:113.818645,22.632856

圣约翰机场:116.48489,39.915476

什里夫波特地区机场:116.183929,39.931486

石家庄正定机场:114.698938,38.288955

首尔仁川国际机场:116.343317,39.998843

首都机场:116.494982,39.994492

松山机场:118.757106,42.268753

泗水机场:117.345264,35.640741

塔林机场:116.186367,39.964747

太原武宿机场:112.637907,37.761473

台湾高雄国际机场:121.973871,24.086957

台湾台中清泉岗机场:121.973871,24.086957

天津机场:117.210813,39.14393

图卢兹布拉格纳克机场:116.417278,40.085974

威海文登国际机场:121.968291,37.166083

威尼斯马可波罗机场:116.40459,39.975691

维多利亚瀑布城机场:116.460318,39.960018

维多利亚国际机场:116.460318,39.960018

维尔罗杰斯机场:116.36088,39.929828

潍坊机场:119.126592,36.64727

温莎机场:116.309345,39.918423

温州永强机场:120.856993,27.917074

乌鲁木齐地窝铺国际机场:87.491727,43.903344

无锡硕放机场:120.43941,31.508847

武夷山机场:118.011525,27.710607

武汉天河国际机场:114.220217,30.781008

西昌青山机场:102.197595,27.982227

西蒙－玻利瓦尔机场:116.287063,39.880667

西蒙－玻利瓦尔机场:116.287063,39.880667

西宁曹家堡机场:101.767921,36.640739

西双版纳景洪机场:100.927318,22.111413

西棕榈滩国际机场:116.577541,40.058249

西塔迪－托里诺机场:83.895485,45.656986

厦门高崎国际机场:118.143501,24.54742

仙台机场:115.910511,40.040598

辛辛那提/北肯德基机场:116.435299,39.852194

徐州观音机场:117.566296,34.062469

烟台莱山机场:121.380693,37.406298

延安二十里铺机场:109.477197,36.526644

延吉机场:129.454833,42.895332

仰光明加拉登机场:116.444943,39.891227

伊利国际机场:116.562727,39.9259

伊斯塔巴/西华达内荷国际机场:114.478087,33.793632

宜昌三峡机场:111.492255,30.554257

宜宾机场:104.562954,28.801335

银川河东机场:106.397489,38.331163

张家界大庸机场:110.48162,29.124889

郑州新郑机场:113.852605,34.531462

舟山普陀山机场:122.365915,29.938168

朱利亚纳王子机场:116.364563,39.914667

珠海金湾机场:113.383726,22.015148

##

向翻到附录最下面的评委致意最由衷的感谢

##