基于prim算法和遗传算法的无线可充电传感器系统充电路线规划

摘要

随着物联网的快速发展，无线传感器网络在生活中的应用越来越广泛，对传感器充电路线的规划的研究对节省能源消耗有着重要意义。本文聚焦无线可充电传感器系统，研究如何规划移动充电器的充电路线使能量消耗最小、传感器电池容量确定问题。

在问题一中，我们首先将各传感器节点绘图表示如图2，计算各节点的距离，然后采用基于询问式充电方式，即不管传感器结点是否需要充电，每次经过都询问是否需要充电，需要则充电。在此基础上，将最小化移动充电器在路上的能量消耗问题转换为最小化移动充电器在路上行使的时间，进而转换成求以数据中心为起始点，经过所有节点一次且仅一次，最终又回到数据中心的最短路线。此问题类似经典的旅行商问题。我们采用改进的最小生成树算法求解能量消耗最小的充电路线，最终求得该路线为：[0, 17, 20, 19, 18, 25, 26, 29, 21, 22, 23, 24, 28, 4, 3, 5, 10, 13, 16, 12, 8, 9, 7, 6, 11, 14, 15, 27, 2, 1, 0]，路线总长度为：12269m。再运用遗传算法进行优化，最终得到的充电路线为：[0, 17, 20, 19, 18, 25, 26, 29, 21, 23, 24, 28, 22, 4, 3, 5, 10, 13, 16, 27, 15, 12, 8, 11, 14, 6, 7, 9, 1, 2, 0]，路径总长度为：11485m。

在问题二中，我们在问题一求得的充电路线基础上，假设移动充电器的能量是足够对所有充电器进行充电。此问求解的是最小电池容量，可转换成临界问题。移动充电器执行完一次充电任务，再次到达一个传感器节点时，该节点刚好达到阈值，基于此建立相关方程组，求解得传感器得电池容量。

在问题三中，为了提供充电效率，移动充电器增至4个，故我们对所有传感器节点运用K-means聚类算法，随机选取聚簇中心，将传感器节点分成4个类，每个类中的节点都集中在某个区域。最终分类如下：[0, 9, 7, 6, 11, 14, 8, 2, 1]、[0, 12, 15, 27, 16, 13, 10, 5]、[0, 21, 22, 23, 24, 28, 4, 3]、[0, 19, 29, 26, 25, 18, 20, 17]。对每个类中的传感器节点分别运用问题一中的算法即改进的最小生成树算法，再用遗传算法进行优化，求解出每个类能量消耗最小的路径。最终求得的四条路径分别为：[0, 2, 8, 11, 14, 6, 7, 9, 1, 0]、[0, 5, 13, 16, 27, 15, 12, 10, 0]、[0, 4, 21, 22, 23, 24, 28, 3, 0]、[0, 20, 18, 25, 26, 29, 19, 17, 0]。在此基础上，运用问题二中所采用的临界分析法，对每个类分别建立相应的方程组并进行求解，求得每个传感器的容量。

关键词：无线可充电器传感器网络、充电规划、最小能量消耗

一、问题重述

1.1 问题的背景

随着物联网的快速发展，无线传感器网络WSN（Wireless Sensor Network）在生活中的应用也越来越广泛。无线传感器网络中包括若干传感器（Sensors）

以及一个数据中心（Data Center）。传感器从环境中收集信息后每隔一段时间将收集到的信息发送到数据中心。数据中心对数据进行分析并回传控制信息。WSN生命周期一个最重要的影响因素是能量。能量提供有两种方式，一种是能量收集，利用太阳能等环境能源让传感器从环境中汲取能量；一种是用电池供电并定期用移动充电器进行。利用能量收集方式给传感器进行供电太过于依赖环境，提供的能量不稳定，故需要采用方式二，定期为传感器进行充电。故需要对传感器进行定期充电。

1.2 问题的重述

无线可充电传感器网络包括三个部分：一个数据中心 DC（Data Center）、若干传感器（Sensors）、一个或多个移动充电器 MC（Mobile Charger）。

数据中心和若干传感器分布在一个二维空间中，如图 1 所示（虚线箭头表示数据中心与传感器之间、传感器与传感器之间均存在一条路径互相连通；实线箭头表示 MC 的充电路线）。

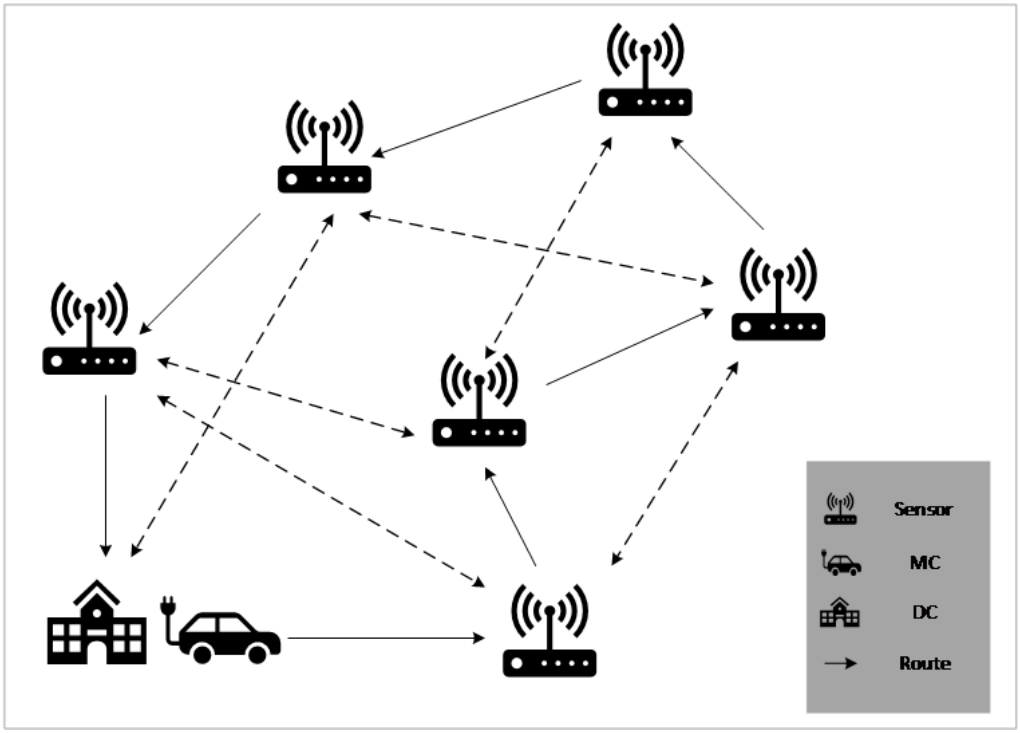

图 1 无线可充电传感器网络

问题一：在考虑只有一个移动充电器情况下，规划最小能量消耗的充电路线。

问题二：已知每个节点的经纬度和能量消耗速率，并假设传感器的电量只有在高于 f(mA)时才能正常工作，移动充电器的移动速度为 v(m/s)、移动充电器的充电速率为 r（mA/s），基于问题一中规划出来的充电路线，求解在保证整个系统一直正常运行情况下，每个传感器的电池的容量的最小值。

问题三：已知每个节点的经纬度和能量消耗速率，并假设传感器的电量只有在高于 f(mA)时才能正常工作，移动充电器的移动速度为 v(m/s)、移动充电器的充电速率为 r（mA/s）。同时派出 4 个移动充电器进行充电，在此情况下规划移动充电器的充电路线，使所有移动充电器在路上的总能量消耗最小，并且求解能保证整个系统一直正常运行的每个传感器电池的最小容量。

二、问题的分析

2.1 问题一的分析

题目中已知各节点的经纬度，将经纬度换算成弧度，然后运用公式：$d = R * arcos[cos(y_1) * cos(y_2) * cos(x_1 - x_2) + sin(y_1) * sin(y_2)]$,计算任意两个节点中的距离。因为各节点之间都是互相连通的，可将所有节点简化成一个全连通图，且各边的权值为其两节点间的距离。将最小化移动充电器在路上的能量消耗的问题转换为最小化移动充电器在路上的能量消耗的时间，进而转换成最短路径问题。考虑到在每一次充电路线中，只对每个传感器节点充电一次，所以每个节点只经过一次。最终此问可转换成经典的旅行商问题。我们采用改进的最小生成树算法 Prim 算法进行求解。移动充电器从数据中心出发，

2.2 问题二的分析

已知每个节点的经纬度和相应传感器的能量消耗速率，传感器能量阈值为 f(mA)，移动充电器的移动速度为 v(m/s)、移动充电器充电速率为 r(mA/s)。问题一中求解得出的充电路线为：[0, 17, 20, 19, 18, 25, 26, 29, 21, 23, 24, 28, 22, 4, 3, 5, 10, 13, 16, 27, 15, 12, 8, 11, 14, 6, 7, 9, 1, 2, 0]。假设移动充电器的电池容量足以走完上述充电路线并且能够给所有传感器充电。为了保证每个传感器的电池容量最小，故移动充电器每到达一个传感器节点时，该传感器刚好处于阈值。记走完一次充电路线所消耗的时间为 t_0秒，移动充电器在 0 节点即数据中心充电所耗时间为 t_c 秒。基于上述分析可建立起相关方程组进行求解。

2.3 问题三的分析

为了提高充电速率，同时派出 4 个移动充电器进行充电。我们将所有节点通过 k-means 算法进行聚簇分类，随机选取 4 个聚簇中心，将传感器节点分成 4 个类，类中的节点集中分布在某个区域。然后根据所得分类，对每个类中的节点分别运用问题一中所用的改进的最小生成树算法求得充电路线，再通过遗传算法进行优化，即可求得 4 类中分别对应的最短路径。再运用问题二中的临界思想，分别对 4 个类建立其方程组，求得各传感器节点对应的最小电池容量。

三、模型假设

1.假设移动充电器能量是无限的。
2.假设移动充电器第一次走完最短路径后，再次到达该路径的首个节点，该节点的能量也恰好处于阈值。
3.假设移动充电器不受到交通、天气等因素的影响。
4.传感器节点容量是恒定不变的，不受电池寿命的影响。

四、符号说明

表 1 符号说明

符号	含义	单位
v	移动充电器移动速度，固定	m / s
r	移动充电器充电速率，固定	mA / s
d_i	最短路径中第 i 节点与上一个节点的距离	m
F_i	第 i 个传感器的电池容量	mA
f	传感器电量阈值	mA
T_i	走完一次最短路径再次到达传感器 i 所耗时间	s
C_i	传感器的消耗速率	mA / s
t_0	执行完一次充电任务所耗时间	s
s	充电路径总长度	m
t_c	移动充电器充电所耗时间	s
E_c	移动充电器对传感器充电消耗的总能量	J
E_m	移动充电器行驶过程的总能量消耗	J

五、模型的建立和求解

5.1 问题一求解

5.1.1 模型建立

已经数据中心和传感器节点的经纬度，以经度为 x，纬度为 y 建立坐标系如图 2 所示，其中，红色 0 节点表示数据中心。将经纬度转换成弧度制，通过公式：

$$d = R * arcos[cos(y_1) * cos(y_2) * cos(x_1 - x_2) + sin(y_1) * sin(y_2)]$$

计算出各节点之间的距离。各个节点之间都可通行，所以可以将所有节点连接成一个带权无向完全图 G，权值为两点间的距离。记两点间的距离为 d_{ij}（i, j 分别为对应顶点），则权值为 d_{ij}。故问题则为求：

$$min\, f(\omega) = \sum_{l=1}^{n} d_{i_{l-1} i_l}$$

$$\omega = (i_0, i_1, i_2, \cdots, i_n, i_{n+1})$$

$$i_{n+1} = i_0$$

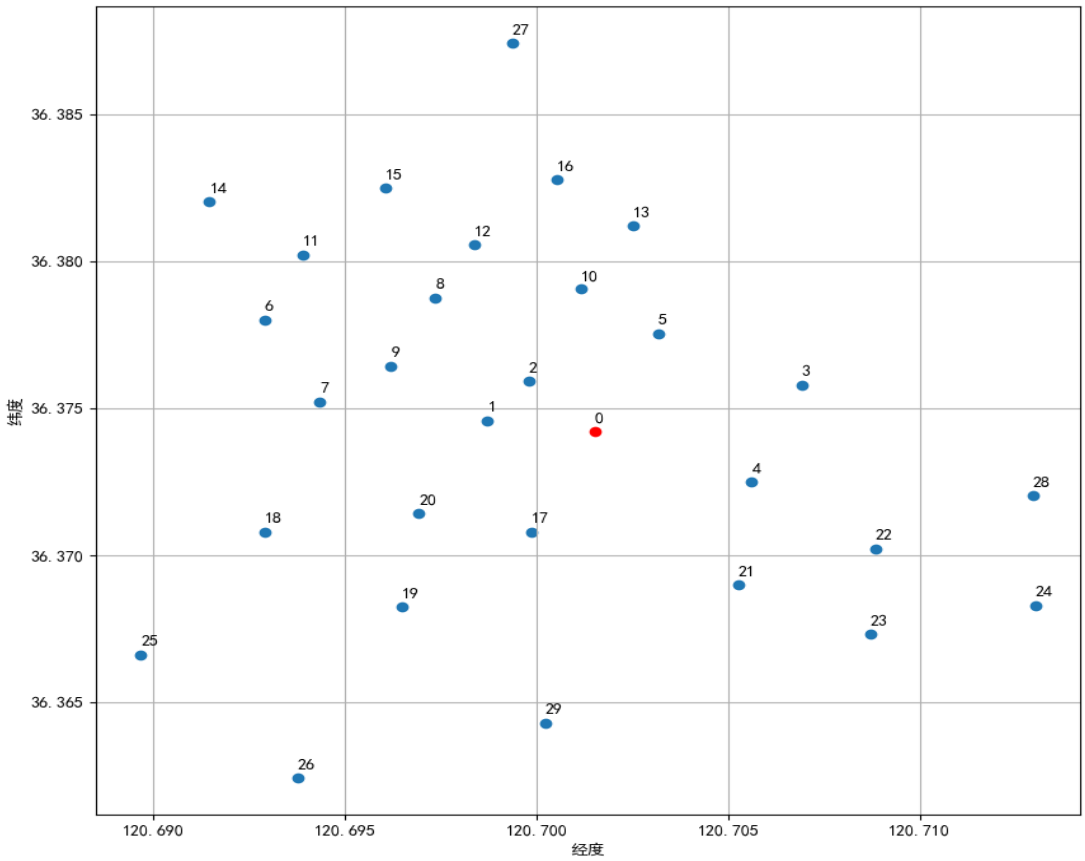

图 2 节点坐标图

5.1.2 算法设计

此类问题为 NP 问题，要求得精确解需对各节点进行求排列且得相应的路径大小，最终通过比较取最小值。这种求解方法计算量巨大，求解相当困难甚至是不可能，故采取其他近似算法。考虑到最小生成树 Prim 算法，可以求得权值最小的生成树，与我们所求的最短回路有点类似。不同之处为我们所求的是回路，而最小生成树中不可能包含回路故需对算法作出改进。Prim 算法如下：

1).输入：一个带权连通图，其中顶点集合为 V，边集合为 E；

2).初始化：V_{new}={x}，其中 x 为集合 V 中的任一节点（起始点），E_{new}={},为空；

3).重复下列操作，直到 V_{new} = V：

a.在集合 E 中选取权值最小的边<u, v>，其中 u 为集合 V_{new} 中的元素，而 v 不在 V_{new} 集合当中，并且 v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b.将 v 加入集合 V_{new} 中，将<u, v>边加入集合 E_{new} 中；

4).输出：使用集合 V_{new} 和 E_{new} 来描述所得到的最小生成树。

在最小生成树有分支点，从一个节点访问到其余节点，会出现重复访问某个节点，而我们所求解的问题中，每个节点只可经过一次。为了解决改问题，防止出现分支点，我们每次选择下一个节点时，改节点应是与上一次选择的顶点相连，不可与 V_{new} 中其他顶点相连。在每次选取下一个节点时，可保证每次选取的路径都是最短的，但选取到最后一个节点时，改路径是将最后一个节点与起始节点直接相连，无法控制改路径最短。所以需要将所有节点分别作为起始点，求出相应的路径长度，最终通过比较取得最小值。更改后的算法如下：

1). 输入：一个带权连通图，其中顶点集合为 V，边集合为 E；
2). 创建顶点集合 V’=V，创建 E_r 用于保存当前所求路径，创建并 s 初始化为+∞，用于保存路径大小。
3). 从 V’中取出一个节点作为起始点 x；
4). 初始化：V_{new}={x}， E_{new}={}，为空；
5). 重复下列操作，直到 V_{new} = V：

a. 在集合 E 中选取权值最小的边<u, v>，其中 u=x，而 v 不在 V_{new} 集合当中，并且 v∈V（如果存在有多条满足前述条件即具有相同权值的边，则可任意选取其中之一）；

b. 将 v 加入集合 V_{new} 中，将<u, v>边加入集合 E_{new} 中；

6). 记最后一个节点为 y，将<y, x>加入集合 E_{new} 中；计算当前集合 E_{new} 中的总路径大小并与 s 进行比较，若小于 s，则更新 s 为当前计算所得值，并将 E_r 更新为 E_{new}；否则，则不更新。
7). 重复步骤 3)、4)、5)、6)，直至 V’中的节点都遍历过。
8). 输出：使用集合 E_{new} 和 s 来描述所得到的最短路径和最短路径大小。

5.1.3 模型求解

用 python 编程求解的最短路径为：[0, 17, 20, 19, 18, 25, 26, 29, 21, 22, 23, 24, 28, 4, 3, 5, 10, 13, 16, 12, 8, 9, 7, 6, 11, 14, 15, 27, 2, 1, 0]。最短充电路线图示如图 3。路径总长度为：12269m。

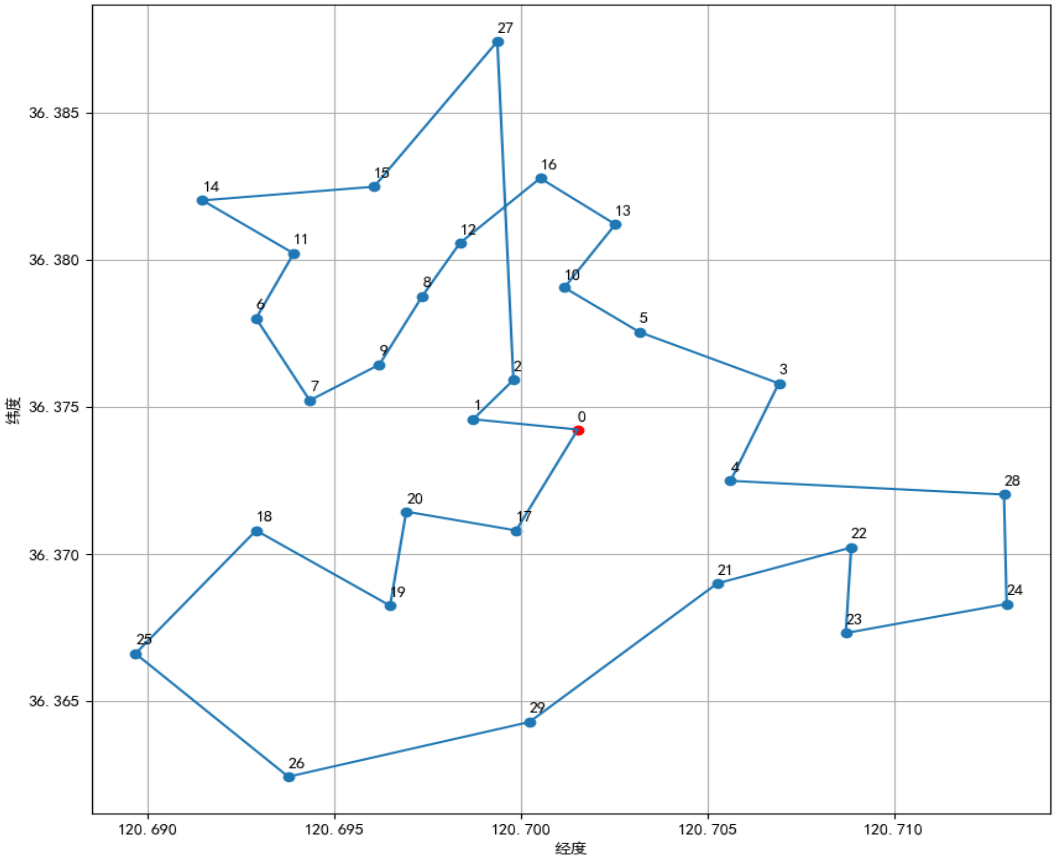

图 3 充电路线

5.1.4 模型优化

使用改进后的 Prim 算法只能求得路径较短、能量消耗较小的近似解。其还有优化的空间。我们使用遗传算法，将所有传感器节点组成的任意序列视为成为一条染色体，序列中的某个值称为基因。记产生下一代的次数为 n，交叉概率为 P_{cr}，突变概率为 P_{mr}。每次产生下一代时，随机生成一个个数为 450 的种群。交叉策略为选取随机种群中的两个个体 p1 和 p2，选取 p2 中的某个基因序列即某段区间，插入新个体中，将 p1 中的基因按顺序插入新个体中，该过程需要注意防止重复插入相同的基因即相同的值。突变策略则随机选取某个个体中的两个基因进行交换。将路径的长度的倒数视为该个体的适应度，路径越长则适应度越小。**我们使用遗传算法，并以 prim 算法输出的路径作为初始路径，加速其的迭代下降速率，并把路径缩短。**设置产生下一代的次数为 10000，即 n=10000，动态调整交叉概率 P_{cr} 和突变概率 P_{mr}，选取相对较优的参数值。

通过 python 编程实现并求解得到优化后的路径为：[0, 17, 20, 19, 18, 25, 26, 29, 21, 23, 24, 28, 22, 4, 3, 5, 10, 13, 16, 27, 15, 12, 8, 11, 14, 6, 7, 9, 1, 2, 0]，坐标图中表示如图 4。路径总长度为：11485m。

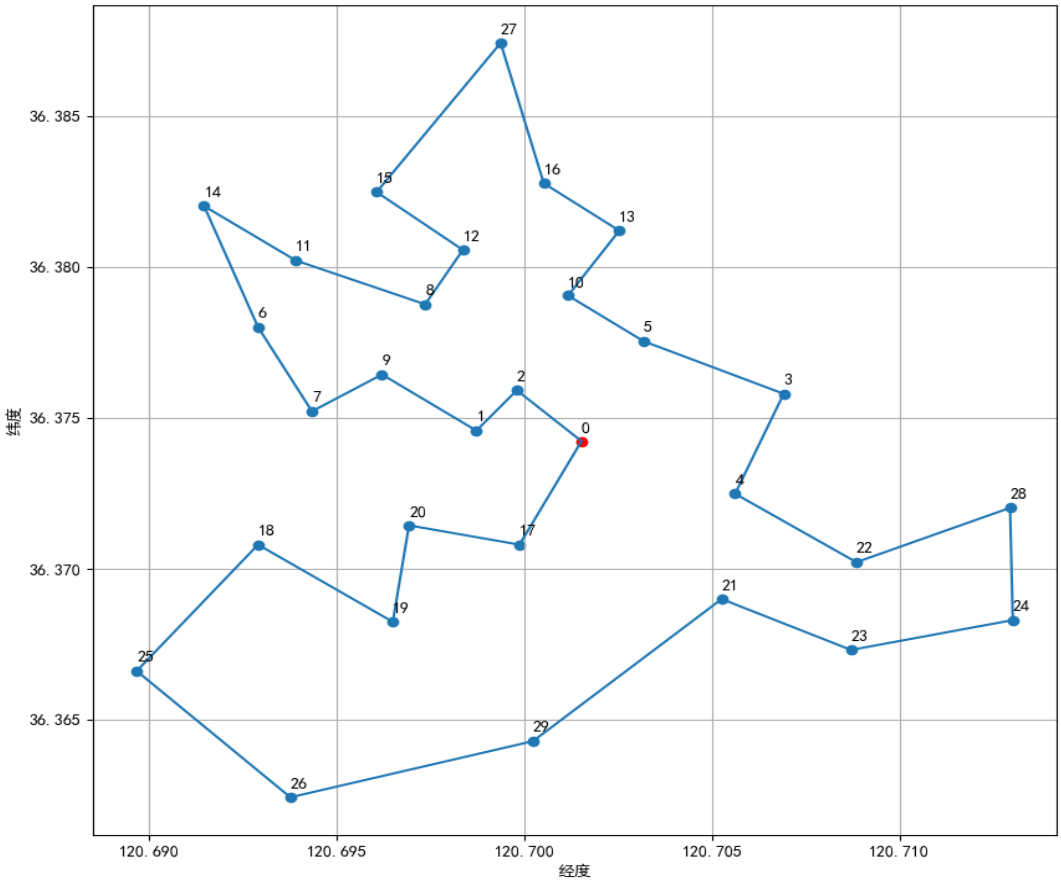

图 4 优化后的充电路线

5.2 问题二求解

5.2.1 模型建立

在无线可充电传感器系统中，已知问题一中求得的充电路径，计算该路径的总长度为 s=11485。移动充电器在数据中心充电所耗时 t_c，移动充电器移动速度

为 v，移动充电器充电速率为 r，第 i 个传感器的能量消耗速率为 C_i，第 i 个传感器的电池能量为 F_i，传感器的阈值为 f，移动充电器行走完一次最短路径再次到达传感器 i 所用时间为 T_i，行走完一次最短路径并且对所有传感器进行充电共花费时间为 t_0。要使得传感器电池容量最小，则有：$\boldsymbol{F_i - C_i T_i = f}$。
建立方程组如下：

$$\boldsymbol{F_i - C_i T_i = f} \quad i \in [0,28]$$

$$T_i - T_{i-1} = \frac{d_i}{v} + \frac{F_{i-1} - f}{r} \quad i \in [1,28]$$

$$T_0 = t_0$$

$$t_0 = t_c + \frac{s}{v} + \frac{\sum_{i=0}^{29}(F_i - f)}{r}$$

求解该方程组即可得到相应传感器电池的最小容量。因为题目未给出相应标量的准确值，带入相应符号求解计算量巨大，不可行。我们查询相关资料，得出各标量比较可靠的值，分别为：f=100，v=11，r=0.2，t_c=36000。将其带入方程，用 python 编程求解。

5.2.2 模型求解

用 python 编程求解并且保留整数，得各传感器节点的最小容量如表 2 所示：

传感器节点	最小电池容量(mA)	传感器节点	最小电池容量(mA)
传感器 1	175	传感器 16	184
传感器 2	162	传感器 17	200
传感器 3	176	传感器 18	170
传感器 4	204	传感器 19	157
传感器 5	177	传感器 20	160
传感器 6	161	传感器 21	217
传感器 7	177	传感器 22	174
传感器 8	151	传感器 23	173
传感器 9	208	传感器 24	173
传感器 10	151	传感器 25	174
传感器 11	194	传感器 26	205
传感器 12	182	传感器 27	175
传感器 13	182	传感器 28	190
传感器 14	168	传感器 29	231
传感器 15	155		

表 2 各传感器节点最小容量

5.3 问题三模型建立与求解

5.3.1 模型建立

将所有传感器节点运用 k-means 算法，分成 4 个类，分别记为 V_1、V_2、V_3、V_4。具体过程如下：随机选取 4 个节点分别作为 4 个类的聚簇中心。计算每个传感器节点与聚簇中心相应的距离，将每个传感器节点归类到距离最短的那个类中，每次有传感器节点的类别改变时，则更新每个类的聚簇中心，循环上述过程直至所有传感器节点的类别不再发生变化。此算法生成的 4 个类，每个类都会集中分布在某个区域，可有效避免移动充电器充电路径的交叉重叠，从而减少能量消耗。注意，每辆车给相应所有传感器节点充电完成后，都需回到数据中心，即都需再次回到 0 节点，所以最后求得的所有类中，都需增加一个 0 节点。

将通过 k-means 算法生成的 4 个类，分别运用问题一中改进的 Prim 算法，并且通过遗传算法进行优化，计算出相应的最短路径。

5.3.2 模型求解

用 python 编程实现 k-means 算法，求解得到四个类分别如表 3 所示：

类别	包含节点
cluster1	9, 7, 6, 11, 14, 8, 2, 1, 0
Cluster2	16, 13, 10, 5, 0, 12, 15, 27
Cluster3	21, 22, 23, 24, 28, 4, 3, 0
Cluster4	18, 20, 17, 0, 19, 29, 26, 25

表 3 分类结果

分别对 4 个类中运用问题一中改进的 Prim 算法，得到相应的最短路径如图 5 所示，四个类路径的总长度如表 4 所示。对该结果再运用问题一中的遗传算法进行优化，由于各类中节点个数较少，故将产生后代的次数 n 设置为 1000。优化后的路径如图 6 所示。四个类路径的总长度如表 4 所示。

类别	优化前路径总长度	优化后路径总长度
cluster1	3101	2848
Cluster2	3369	3346
Cluster3	3716	3489
Cluster4	4161	4015
cluster1+ Cluster2+ Cluster3+ Cluster4	14347	13698

表 4 各类路径总长度

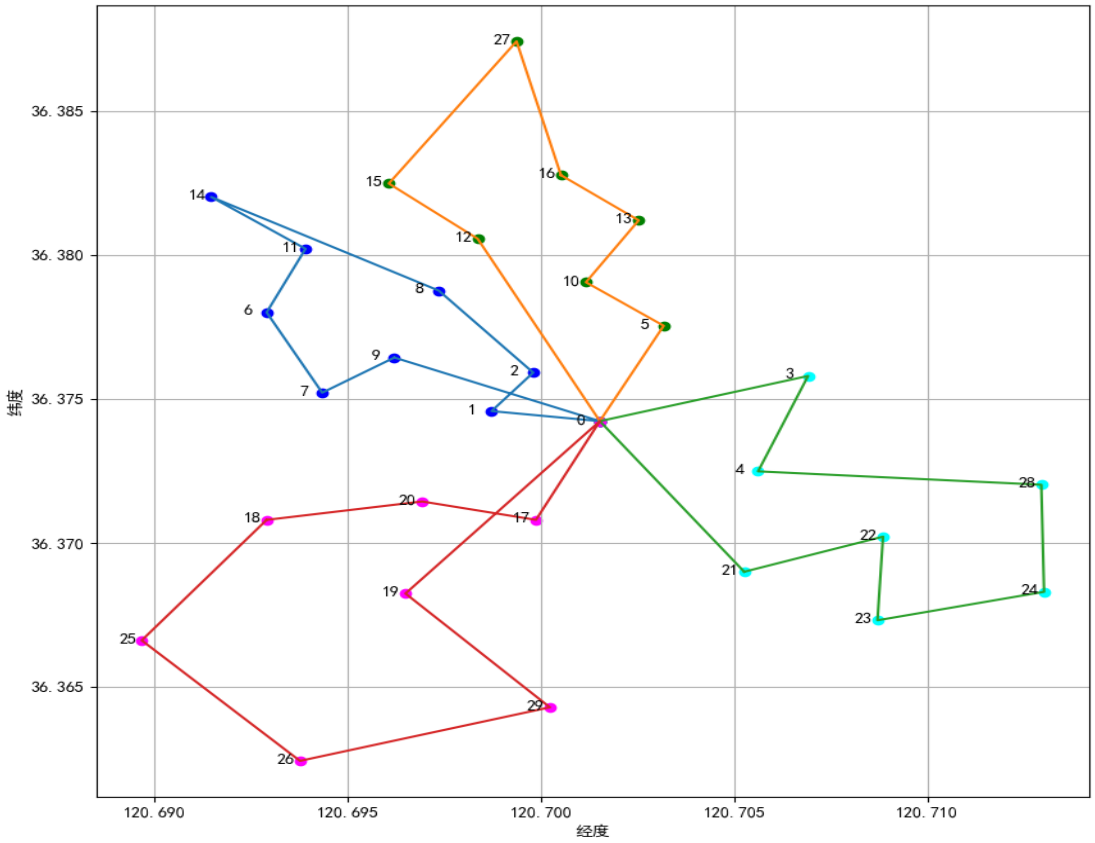

图 5 优化前各类充电路线

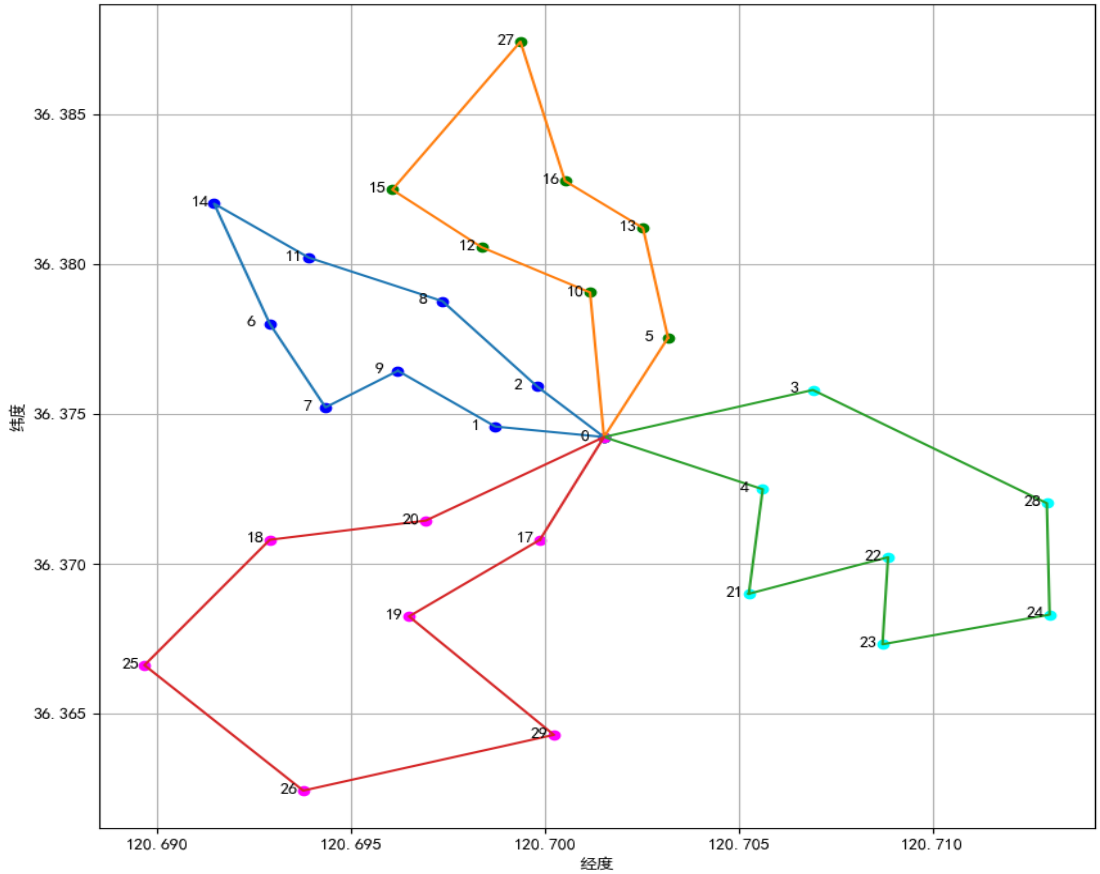

图 6 优化后各类充电路线

5.3.3 电池容量求解

由 4.3.2 求解得到相应的类和各类对应的最短充电路径，因为各路径的充电互不影响，故分别对 4 条路线运用问题二中的方式建立方程并且求解。因为各路线的传感器数目变少，相应调整移动充电器充电时间 t_c=9000，复用问题二中的代码，求解得到各节点的最小电池如表 5 所示：

传感器节点	最小电池容量(mA)	传感器节点	最小电池容量(mA)
传感器 1	116	传感器 16	113
传感器 2	122	传感器 17	117
传感器 3	114	传感器 18	121
传感器 4	116	传感器 19	117
传感器 5	110	传感器 20	113
传感器 6	113	传感器 21	110
传感器 7	119	传感器 22	116
传感器 8	113	传感器 23	122
传感器 9	114	传感器 24	111
传感器 10	116	传感器 25	116
传感器 11	113	传感器 26	113
传感器 12	122	传感器 27	111
传感器 13	118	传感器 28	119
传感器 14	113	传感器 29	116
传感器 15	111		

表 5 各传感器节点最小容量

六、仿真实验与结果分析

6.1 仿真条件假设

假设移动充电器移动速度为 11m/s，小车行使平均消耗功率为 20J/s，每个节点的电池容量均为 300J，阈值为 100J，移动充电器每到达一个节点时，该节点恰好达到阈值。

6.2 性能指标

在一次充电过程中，移动充电器对节点充电的总能量与移动充电器在充电过程中消耗的总能量之比 E 即为能量利用率 Δ。记移动充电器对节点充电的总能量消耗为 Ec，移动充电器行驶过程的总能量消耗为 Em，公式表示如下：

$$\Delta = \frac{E_c}{E_c + E_m}$$

该无线可充电传感器系统中，传感器节点有 29 个，规定移动充电器从数据中心出发，故共有 29! 条路径。通过 python 编程实现，随机选取一条充电路径，计算路径长度，重复选取 20000 次。该过程中充电路径长度均大于 20000m。将 20000m 代入公式计算得出能量利用率为 13.76%，将该利用率称随机情况下的最

优能量利用率，简称随机最优能量利用率。

6.3 仿真结果分析

改进的 prim 算法求得的充电路径长度为：12269m，用遗传算法优化后的充电路径长度为：11485m，分别带入能量利用率公式，结果如表 6 所示。

计算方式	能量利用率	相较于随机最优能量利用率
随机最优能量利用率	13.76%	
改进的 prim 算法	20.64%	提升 6.88%
遗传算法优化后	21.74%	提升 7.98%

表 2 能量利用率

由表 2 分析可知，prim 算法求得的路径对应的能量利用率相较于随机最优能量利用率提升了 6.88%。通过遗传算法优化后，其能量利用率再提升了 1.1%，由此可得，通过改进的 prim 算法求得的路径再经过遗传算法优化后，所得的结果远超过了随机情况下的路径，故该方式求得的路径较优。

七、模型评价与改进

7.1 模型优点

1. 改进的 Prim 算法可以通过较简单有效的方式求得近似的最短路径。
2. 利用遗传算法，通过有限次的产生后代，不断更新当前的最优路径，进一步优化近似的最短路径。
3. 运用 k-means 算法进行聚簇分类，所得的各类中的节点可集中在某个区域，从而有效避免移动充电器经过的路线有交叉或重叠。

7.2 模型缺点

1. 模型只考虑到移动充电器的电量在一次最短路径中能够对所有充电器充电，未考虑移动充电器电量不满足对所有传感器充电的情况。
2. 问题一中运用遗传算法进行优化，其采取的交叉策略和突变策略不能确保产生的后代比前一代好，具有随机性，所得结果取决于产生后代的次数。
3. 问题三种采取 k-means 分类算法，其没有一个标准衡量分类的好坏，具有随机性。

7.3 模型改进

1. 考虑移动充电器不满足对所有传感器充电的情况。
2. 改进分类算法，构造一个衡量分类好坏的标准。

参考文献

[1] 姜启源,谢金星.数学模型[J].北京:高等教育出版社,2011.
[2] 阿斯顿·章,李牧.动手学深度学习.人民邮电出版社,2019.

[3] 百度百科 Prim 算法 https://baike.baidu.com/item/Prim/10242166
[4] TSP 问题—近似算法 https://www.jianshu.com/p/d1e004bdea8d
[5] 俞立春,吕红芳,何力,李俊甫. 无线可充电传感器网路充电路径优化

附录 A 最短路径求解 Python 代码

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from math import radians, cos, sin, asin, sqrt
from util import cal_prim, haversine, kMeans
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文标签
plt.rcParams['axes.unicode_minus']=False #用来正常显示负号

class analysis(object):
    def __init__(self, data_path):
        self.init(data_path)
        #print(df.info)


    def init(self,data_path = 'data/Q1.xlsx'):
        self.df = pd.read_excel(data_path)
        self.t_to_np()
        self.len_map()

    def run_one_car(self):
        self.cal_one_car()

    def run_k_car(self):
        self.cal_k_cluster()
        self.cal_k_mean_prim()

        for num in range(len(self.all_cluster_path[0])):

            all_path = []
            for num_cluster in range(len(self.all_cluster_path)):
                #print(num_cluster)
                all_path.append(self.all_cluster_path[num_cluster][num])
                #print(all_path)
            self.scatter_k_prim(all_path = all_path, num = num)


    @property
    def get_len_map(self):
        return self.len_map
```

```
@property
def get_np_df(self):
    return self.np_df

@property
def get_all_path(self):
    return self.all_path

@property
def get_all_cos(self):
    return self.all_cos


@property
def get_k_cluster(self):
    return self.index_cluster


def cal_one_car(self):
    path,cos = cal_prim(self.len_map)
    for i in range(len(path)):
        print('path_',i,':',path[i])
        self.scatter_xy(path[i], i)

def scatter_xy(self, path = None, num = 0):
    np_df = self.np_df
    plt.figure(figsize=(10,10))
    plt.scatter(np_df[0,0], np_df[0,1], color = 'r',marker ='o')
    plt.scatter(np_df[1:,0], np_df[1:,1],marker ='o')


    for i in range(30):
        t_x, t_y = float(np_df[i,0]) + 0.000001, float(np_df[i, 1]) + 0.0003
        plt.annotate(str(i), xy = (np_df[i,0], np_df[i, 1]), xytext = (t_x, t_y))

    if path:
        plt.plot(np_df[path, 0], np_df[path, 1])
    #plt.colorbar()
    plt.grid(True)
    plt.xlabel('经度')
    plt.ylabel('纬度')
    plt.savefig('img/sensor_site_'+ str(num)+'.png')
    plt.show()
```

```
def t_to_np(self,):
        list_x = [i for i in self.df['传感器经度'].values]
        list_y = [i for i in self.df['传感器纬度'].values]
        np_df = zip(list_x, list_y)
        np_df = np.array([*np_df])
        #print(np_df.shape)
        self.np_df = np_df


def scatter_k_mean(self):
        np_df      = self.np_df
        index_cluster = self.index_cluster



        plt.figure(figsize=(10,10))
        plt.scatter(np_df[0,0], np_df[0,1], color = 'r',marker ='o')
        plt.scatter(np_df[index_cluster[0] , 0], np_df[index_cluster[0] , 1], color = 'blue',marker
='o')
        plt.scatter(np_df[index_cluster[1] , 0], np_df[index_cluster[1] , 1], color = 'green',marker
='o')
        plt.scatter(np_df[index_cluster[2] , 0], np_df[index_cluster[2] , 1], color = 'cyan',marker
='o')
        plt.scatter(np_df[index_cluster[3] , 0], np_df[index_cluster[3] , 1], color =
'magenta',marker ='o')

        for i in range(centroids.shape[0]):
                plt.scatter(centroids[i, 0], centroids[i,1] , color = 'black',marker ='o')

                t_x, t_y = float(centroids[i,0]) + 0.0003, float(centroids[i, 1]) - 0.0003
                plt.annotate('k_' + str(i), xy = (centroids[i,0], centroids[i, 1]), xytext = (t_x, t_y))


        for i in range(30):
                t_x, t_y = float(np_df[i,0]) - 0.0006, float(np_df[i, 1]) - 0.0001
                plt.annotate(str(i), xy = (np_df[i,0], np_df[i, 1]), xytext = (t_x, t_y))

        #plt.colorbar()
        plt.grid(True)
        plt.xlabel('经度')
        plt.ylabel('纬度')
        plt.savefig('img/k_mean_cluster.png')
        plt.show()
```

```
# 计算距离
def len_map(self,):
    len_np = np.zeros((30,30), dtype = np.float)

    for i in range(30):
        for j in range(30):
            len_np[i,j] = haversine(self.np_df[i,0], self.np_df[i,1], self.np_df[j,0],
self.np_df[j,1])

    #len1 = self.haversine(120.70051409,36.38276987, 120.69986731, 36.37079794)
    #print(len_np)
    self.len_map = len_np


def cal_k_cluster(self):
    np_df    = self.np_df
    len_map = self.len_map
    dataSet = np_df[1:]

    while True:
        centroids, clusterAssment = kMeans(dataSet, 4)
        size = np.unique(clusterAssment[:, 0]).shape[0]
        if size == 4:
            break


    index_cluster = []

    for k in range(4):
        k_cluster = [0]
        for i in range(clusterAssment.shape[0]):
            if clusterAssment[i, 0] == k:
                k_cluster.append(i + 1)
        index_cluster.append(k_cluster)

    self.index_cluster = index_cluster
```

```
def cal_k_mean_prim(self,):

    index_cluster = self.index_cluster
    len_map       = self.len_map
    all_cluster_path = []
    all_cluster_cos  = []
    for k in range(len(index_cluster)):
        k_map_len = np.zeros((len(index_cluster[k]), len(index_cluster[k])) ,dtype=np.float)
        index_x = 0
        for i in index_cluster[k]:
            index_y = 0
            for j in index_cluster[k]:
                k_map_len[index_x, index_y] = len_map[i,j]
                index_y = index_y + 1
            index_x = index_x + 1

        print('The calculation of cluster:', k)
        path,cos = cal_prim(k_map_len)
        all_cluster_cos.append(cos)
        all_path = []
        index = 0
        for one_path in path:
            #  把 path 转变为大图的 path
            c_to_path = []
            for i in one_path:
                c_to_path.append(index_cluster[k][i])
            print('path_to_all_map',index,':', c_to_path)
            print('path_cos:', cos[index])
            index += 1
            all_path.append(c_to_path)

        all_cluster_path.append(all_path)

    self.all_cluster_path = all_cluster_path
    self.all_cluster_cos  = all_cluster_cos
```

```
    def scatter_k_prim(self, all_path = None ,num = 0):
            np_df          = self.np_df
            index_cluster = self.index_cluster
            all_path       = all_path

            plt.figure(figsize=(10,10))
            plt.scatter(np_df[0,0], np_df[0,1], color = 'r',marker ='o')
            plt.scatter(np_df[index_cluster[0] , 0], np_df[index_cluster[0] , 1], color =
'blue',marker ='o')
            plt.scatter(np_df[index_cluster[1] , 0], np_df[index_cluster[1] , 1], color =
'green',marker ='o')
            plt.scatter(np_df[index_cluster[2] , 0], np_df[index_cluster[2] , 1], color =
'cyan',marker ='o')
            plt.scatter(np_df[index_cluster[3] , 0], np_df[index_cluster[3] , 1], color =
'magenta',marker ='o')

            '''
            for i in range(centroids.shape[0]):
                plt.scatter(centroids[i, 0], centroids[i,1] , color = 'black',marker ='o')
                t_x, t_y = float(centroids[i,0]) + 0.0003, float(centroids[i, 1]) - 0.0003
                plt.annotate('k_' + str(i), xy = (centroids[i,0], centroids[i, 1]), xytext = (t_x,
t_y))

            '''
            for i in range(30):
                t_x, t_y = float(np_df[i,0]) - 0.0006, float(np_df[i, 1]) - 0.0001
                plt.annotate(str(i), xy = (np_df[i,0], np_df[i, 1]), xytext = (t_x, t_y))


            for i in range(len(all_path)):
                plt.plot(np_df[all_path[i], 0], np_df[all_path[i], 1])

            #plt.colorbar()
            plt.grid(True)
            plt.xlabel('经度')
            plt.ylabel('纬度')
            plt.savefig('img/k_alg_'+ str(num) +'.png')
            plt.show()

if __name__ == '__main__':
    ana = analysis('data/Q1.xlsx')
    ana.run_one_car()
    ana.run_k_car()
```

附录 B 遗传算法 python 代码

```
# -*- encoding: utf-8 -*-
SCORE_NONE = -1
class Life(object):
        """个体类"""
        def __init__(self, aGene = None):
                self.gene = aGene
                self.score = SCORE_NONE

import random
from Life import Life

class GA(object):
        """遗传算法类"""
        def __init__(self, aCrossRate, aMutationRage, aLifeCount, aGeneLenght,aMatchFun =
lambda life : 1,prim_path = None):
                self.croessRate = aCrossRate
                self.mutationRate = aMutationRage
                self.lifeCount = aLifeCount
                self.geneLenght = aGeneLenght
                self.matchFun = aMatchFun                    # 适配函数
                self.lives = []                              # 种群
                self.best = None                             # 保存这一代中最好的个体
                self.generation = 1
                self.crossCount = 0
                self.mutationCount = 0
                self.bounds = 0.0                    # 适配值之和，用于选择是计算概率
                self.prim_path = prim_path
                self.initPopulation()

        def initPopulation(self):
                """初始化种群"""
                self.lives = []
                for i in range(self.lifeCount):
                        if i==0 and self.prim_path:
                                gene = self.prim_path
                                life = Life(gene)
                                self.lives.append(life)
                        else:
                                gene = [ x for x in range(self.geneLenght) ]
                                random.shuffle(gene)
                                life = Life(gene)
                                self.lives.append(life)
```

```
def judge(self):
        """评估，计算每一个个体的适配值"""
        self.bounds = 0.0
        self.best = self.lives[0]
        for life in self.lives:
                life.score = self.matchFun(life)
                self.bounds += life.score
                if self.best.score < life.score:
                        self.best = life

def cross(self, parent1, parent2):
        """交叉"""
        index1 = random.randint(0, self.geneLenght - 1)
        index2 = random.randint(index1, self.geneLenght - 1)
        tempGene = parent2.gene[index1:index2]     # 交叉的基因片段
        newGene = []
        p1len = 0
        for g in parent1.gene:
                if p1len == index1:
                        newGene.extend(tempGene)        # 插入基因片段
                        p1len += 1
                if g not in tempGene:
                        newGene.append(g)
                        p1len += 1
        self.crossCount += 1
        return newGene

def    mutation(self, gene):
        """突变"""
        index1 = random.randint(0, self.geneLenght - 1)
        index2 = random.randint(0, self.geneLenght - 1)
        newGene = gene[:]           # 产生一个新的基因序列，以免变异的时候影响父种群
        newGene[index1], newGene[index2] = newGene[index2], newGene[index1]
        self.mutationCount += 1
        return newGene

def getOne(self):
        """选择一个个体"""
        r = random.uniform(0, self.bounds)
        for life in self.lives:
                r -= life.score
                if r <= 0:
                        return life
        raise Exception("选择错误", self.bounds)
```

```
def newChild(self):
        """产生新后的"""
        parent1 = self.getOne()
        rate = random.random()

        # 按概率交叉
        if rate < self.croessRate:
                # 交叉
                parent2 = self.getOne()
                gene = self.cross(parent1, parent2)
        else:
                gene = parent1.gene

        # 按概率突变
        rate = random.random()
        if rate < self.mutationRate:
                gene = self.mutation(gene)

        return Life(gene)


def next(self):
        """产生下一代"""
        self.judge()
        newLives = []
        newLives.append(self.best)                    #把最好的个体加入下一代
        while len(newLives) < self.lifeCount:
                newLives.append(self.newChild())
        self.lives = newLives
        self.generation += 1
```

```
from GA import GA
class TSP(object):
        def __init__(self,mapLen, aLifeCount = 450, prim_path = None):
                #self.initCitys()
                self.mapLen = mapLen
                print(self.mapLen.shape)
                self.lifeCount = aLifeCount
                if prim_path:
                        self.prim_path = prim_path
                        self.ga = GA(aCrossRate = 0.85,
                                aMutationRage = 0.1,
                                aLifeCount = self.lifeCount,
                                aGeneLenght = self.mapLen.shape[0],
                                aMatchFun = self.matchFun(),
                                prim_path = prim_path)
                else:
                        self.ga = GA(aCrossRate = 0.9,
                                aMutationRage = 0.11,
                                aLifeCount = self.lifeCount,
                                aGeneLenght = self.mapLen.shape[0],
                                aMatchFun = self.matchFun(),
                                )
        def distance(self, order):
                distance = 0.0
                for i in range(-1, self.mapLen.shape[0] - 1):
                        index1, index2 = order[i], order[i + 1]
                        distance += self.mapLen[index2, index1]
                return distance
        def matchFun(self):
                return lambda life: 1.0 / self.distance(life.gene)

        def run(self, n = 0):
                best_path = None
                best_cos    = 0.0
                while n > 0:
                        self.ga.next()
                        distance = self.distance(self.ga.best.gene)
                        n -= 1
                        if n % 500 == 0:
                                print (("%d : %f") % (self.ga.generation, distance))
                best_path = self.ga.best.gene
                best_cos =    self.distance(self.ga.best.gene)
                return best_path, best_cos
```

附录 C 最小电池容量求解 python 代码

```
import numpy as np
import Q1_analysis
import math
# 求解最低电池容量
class SolveMBC():
    """
    f 为电池阈值，v 为移动充电器移动速度，
    r 为充电速率，tc 为移动充电器充电时间
    pathLength 为路径长度数组
    energyCon 为各节点能量消耗数组
    totalLength 为路径总长度
    """
    def __init__(self,f,v,r,tc,pathLength,energyCon,totalLength):
        self.f = f
        self.v = v
        self.r = r
        self.tc = tc
        self.pathLength = pathLength
        self.energyCon = energyCon
        self.totalLength = totalLength

    def genCoe(self,f,v,r,tc,pathLength,energyCon,totalLength):
        """获取方程组系数"""
        coe = []
        nodeNum = len(energyCon)
        for i in range(nodeNum):
            if i==0:
                cur_row = [-energyCon[0]/r for j in range(nodeNum)]
                cur_row[0] += 1
            else:
                cur_row = [-2*energyCon[i]/r for j in range(i)]
                cur_row.extend([-energyCon[i]/r for j in range(nodeNum-i)])
                cur_row[i] += 1
            coe.append(cur_row)
        return coe
```

```
def solve(self):
        A = self.genCoe(f=self.f, v=self.v, r=self.r, tc=self.tc, pathLength=self.pathLength,
energyCon=self.energyCon, totalLength=self.totalLength)
        b = self.genB(f=self.f, v=self.v, r=self.r, tc=self.tc, pathLength=self.pathLength,
energyCon=self.energyCon, totalLength=self.totalLength)
        A = np.array(A)
        # print("**********  系数矩阵  **********")
        # print(A)
        b = np.array(b)
        # print("**********  方程组的值  **********")
        # print(b)
        x = np.linalg.solve(A,b)
        x = [math.ceil(i) for i in x]
        print("********** result **********")
        print(x)
        return x

@staticmethod
def genPathLengthAndEnergyCon(path=[]):
        """根据路径求解各路径长度和能量消耗"""
        ana = Q1_analysis.analysis('data/Q1.xlsx')
        #  题目中所有节点对应的能量消耗
        cost = [0, 5.4, 7.8, 4.5, 5.5, 3.6, 4.5, 6.4, 4.6, 4.5, 5.5, 4.5, 7.4, 6.5, 4.5, 3.8, 4.5, 5.5, 7.5, 5.5,
4.5,3.5, 5.5, 7.5, 3.5, 5.5, 4.3, 3.6, 6.4, 5.4]


        pathLength = []
        for i in range(len(path) - 1):
                pathLength.append(ana.len_map[path[i], path[i + 1]])
        pathLength = [round(i,1) for i in pathLength]

        energyCon = []
        for i in range(len(path)):
                energyCon.append(cost[path[i]])
        energyCon = energyCon[1:-1]
        #将消耗速率的单位转换为：mA/s
        energyCon = [i / 3600 for i in energyCon]
        return pathLength, energyCon
```

```
if __name__ == '__main__':
    #Q2
    print("\nQ2")
    path_2 = [0, 17, 20, 19, 18, 25, 26, 29, 21, 23, 24, 28, 22, 4, 3, 5, 10, 13, 16, 27, 15, 12, 8,
11, 14, 6, 7, 9, 1, 2, 0]
    pathLength_2, energyCon_2 = SolveMBC.genPathLengthAndEnergyCon(path=path_2)
    solveMBC = SolveMBC(f=100,v=11,r=0.2,tc=36000,pathLength=pathLength_2,
                        energyCon=energyCon_2,totalLength=sum(pathLength_2))
    x = solveMBC.solve()
    #Q3_1
    print("\nQ3_1")
    path_3_1 = [0, 2, 8, 11, 14, 6, 7, 9, 1, 0]
    pathLength_3_1, energyCon_3_1=
                        SolveMBC.genPathLengthAndEnergyCon(path=path_3_1)
    solveMBC = SolveMBC(f=100,v=11,r=0.2,tc=36000/4,pathLength=pathLength_3_1,
                        energyCon=energyCon_3_1,totalLength=sum(pathLength_3_1))
    x = solveMBC.solve()

    #Q3_2
    print("\nQ3_2")
    path_3_2 = [0, 5, 13, 16, 27, 15, 12, 10, 0]
    pathLength_3_2, energyCon_3_2 =
    SolveMBC.genPathLengthAndEnergyCon(path=path_3_2)
    solveMBC = SolveMBC(f=100,v=11,r=0.2,tc=36000/4,pathLength=pathLength_3_2,
                        energyCon=energyCon_3_2,totalLength=sum(pathLength_3_2))
    x = solveMBC.solve()

    #Q3_3
    print("\nQ3_3")
    path_3_3 = [0, 4, 21, 22, 23, 24, 28, 3, 0]
    pathLength_3_3, energyCon_3_3 =
    SolveMBC.genPathLengthAndEnergyCon(path=path_3_3)
    solveMBC = SolveMBC(f=100,v=11,r=0.2,tc=36000/4,pathLength=pathLength_3_3,
                        energyCon=energyCon_3_3,totalLength=sum(pathLength_3_3))
    x = solveMBC.solve()

    #Q3_4
    print("\nQ3_4")
    path_3_4 = [0, 20, 18, 25, 26, 29, 19, 17, 0]
    pathLength_3_4, energyCon_3_4 =
    SolveMBC.genPathLengthAndEnergyCon(path=path_3_4)
    solveMBC = SolveMBC(f=100,v=11,r=0.2,tc=36000/4,pathLength=pathLength_3_4,
                        energyCon=energyCon_3_4,totalLength=sum(pathLength_3_4))
    x = solveMBC.solve()
```

附录 C 相关工具类 python 代码

```
import numpy as np
from math import radians, cos, sin, asin, sqrt
from TSP_GA import TSP

# 经度1，纬度1，经度2，纬度2 （十进制度数）
def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # 将十进制度数转化为弧度
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine 公式
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    r = 6371 # 地球平均半径
    # 乘以一千是为了把单位转化为米
    return c * r * 1000

def prim(start_site, map_len):

    def Minmum(closedge):
        min_len = float('inf')
        index = -1
        for i in range(map_len.shape[0]):
                if closedge[i]['lowcost'] < min_len and closedge[i]['lowcost'] != 0:
                    min_len = closedge[i]['lowcost']
                    index = i
        return index
    closedge = dict()
    vextex = [i for i in range(map_len.shape[0])]

    for i in range(map_len.shape[0]):
        point = dict()
        # 保存上一个点
        point['prior'] = start_site
        point['lowcost'] = map_len[start_site, i]
        closedge[i] = point
```

```
    prior = []
    prior.append(start_site)
    total_cos = 0.0
    for i in range(map_len.shape[0] - 1):
        next_point = Minmum(closedge)
        total_cos += map_len[prior[-1],next_point]
        closedge[next_point]['lowcost'] = 0
        #print(vextex[closedge[next_point]['prior']], '--->', vextex[next_point])
        for i in range(map_len.shape[0]):
            if i not in prior:
                closedge[i]['prior'] = next_point
                closedge[i]['lowcost'] = map_len[next_point, i]

        prior.append(next_point)
    total_cos +=    map_len[prior[-1], start_site]
    prior.append(start_site)
    total_cos +=    map_len[prior[-1], start_site]
    return prior, total_cos

def cal_prim(map_len):
    all_path = []
    all_cos    = []
    #------------------------------------------------------------
    #prim
    all_prim_path = []
    all_prim_cos    = []
    for i in range(map_len.shape[0]):
        prior,total_cos = prim(i, map_len)
        all_prim_path.append(prior)
        all_prim_cos.append(total_cos)

    all_prim_path = np.array(all_prim_path, dtype=np.int)
    all_prim_cos    = np.array(all_prim_cos, dtype=np.float)
    sm_start = np.argsort(all_prim_cos)[0]

    prim_path       = [x for x in all_prim_path[sm_start]]
    prim_cos        = all_prim_cos[sm_start]

    all_path.append(prim_path)
    all_cos.append(prim_cos)
```

```
        #----------------------------------------------------------------------
        #GA
        prim_tsp_path = [i for i in prim_path[0:-1]]
        tsp = TSP(map_len,prim_path = prim_tsp_path)

        ga_path, ga_cos = tsp.run(1000)
        ga_path.append(ga_path[0])

        all_path.append(ga_path)
        all_cos.append(ga_cos)
        print('prim_path:',prim_path)
        print('prim_path_cos:', prim_cos)
        print('prim_path->ga_path:',ga_path)
        print('prim_path_cos->ga_cos:', ga_cos)

        return all_path, all_cos

#  构建簇的质心
def randCent(dataSet,k):
            centroids = np.array(
            [[120.69478412 ,36.37505913],
              [120.70522949 , 36.38363639],
              [120.70000861 , 36.37696267],
              [120.69710952 , 36.37223663]],dtype=np.float
        )
        #print(centroids)
        return centroids
```

```
def kMeans(dataSet, k, createCent = randCent):
    '''
        @params：
            dataSet    : (x,y),用于计算距离
            k          ： 聚类的数量
            creatCent : 用于随机初始化聚类的中心
        @returns:
            centroids       ： 聚类的 k 个中心，为(x,y)
            clusterAssment : 每个点属于哪个聚类，以及距离聚类中心的聚类（type, distance）
    '''
    # 获得数据的组数
    dataShape       = np.shape(dataSet)[0]

    # 生成一个 m 行,2 列的矩阵,用来存每个数据到簇的距离,以及对应簇的类型
    # 第一个值是簇的类型，第二个值是距离
    clusterAssment = np.zeros((dataShape,2))

    # 获得随机质心
    centroids       = createCent(dataSet,k)

    # 判断是否继续
    clusterChanged = True
    while clusterChanged:
        clusterChanged = False
        # 将每组数据跟 k 个求距离
        for i in range(dataShape):
            # 先将最小距离设为无穷
            minDist = np.inf
            # 索引是-1
            minIndex = -1
            # 这里对每个数据进行分类
            for j in range(k):
                # 计算距离
                distJI = haversine(centroids[j, 0], centroids[j, 1], dataSet[i, 0], dataSet[i, 1])
                # 判断是不是最小距离
                if distJI < minDist:
                    minDist = distJI
                    # 记住对应簇的类型
                    minIndex = j
            # 如果与某个数据对应的最近距离的点的下标出现了变化,则更改
            if clusterAssment[i, 0] != minIndex:
                clusterChanged = True
            clusterAssment[i, :] = minIndex, minDist
```

```
    #  更新簇
        for cent in range(k):

            # https: // blog.csdn.net / xinjieyuan / article / details / 81477120
            # nonzero 返回使括号中为 True 的下标
            cs = np.nonzero(clusterAssment[:, 0] == cent)[0]

            # print(cs)
            ptsInClust = dataSet[cs]

            #  对每一行进行更新，这里是更新簇
            centroids[cent, :] = np.mean(ptsInClust, axis=0)

        # print(clusterAssment)
    return centroids, clusterAssment
```