中国研究生创新实践系列大赛
“华为杯”第十七届中国研究生数学建模竞赛

题 目　　B 题-无线可充电传感器网络充电路线规划

摘　　要：

现有一无线可充电传感器网络需要进行充电路线规划，如何合理的给移动充电器分配充电路线以及根据路线设计各传感器的电池容量是有现实意义的，不仅能够有效降低成本，还能提高整体充电网络的效率。

在对整体问题进行分析前，本文先对原始数据进行了预处理，由于经纬度绝对值对本题意义不大，故整体减去最小值可以得到一组相对坐标，然后通过坐标映射进行**可视化**绘图，并采用 **Haversine** 公式计算各节点之间的距离矩阵。

对于问题一中规划最优充电路线是典型的旅行商问题（Traveling Salesman Problem, **TSP**），属于 NP 完全问题，在离散组合优化中研究较多，通过不断优化路径，来有效节省成本。一般常用动态规划法、概率论算法、近似解法、人工智能启发式算法及神经网络算法进行计算，但无论使用什么策略进行解题，都只能无限逼近最优解，并不能保证得到通解。

在本题中，不考虑每个传感器的耗电速率及电池容量，只考虑**最短线路规划**，则可以看成**只有权重边的无向图**，只需求出以数据中心为起点的**最短环路**即可。本题采用典型的**启发式算法**进行求解，以**蚁群算法**为主，模拟退火算法、遗传算法及禁忌搜索算法作为对比，根据最优结果出现时间以及最短总路程进行效果评价，蚁群算法由于较快的运行时间以及较优的路线规划从而评价最佳，**最优结果为 11.84km**。

对于问题二，在给定规定路线后，若要求解各传感器的电池容量，则需建立相应的模型进行计算，题目中并未明确给出充电器的行驶速度，充电速率以及能够正常工作的最低电量 $f(mA)$，不妨先将最低电量 $f(mA)$ 设为 0，便于计算，最后将每个传感器的总电量加上 $f(mA)$ 即是实际电池总电量。

本题建立了一个基础的**状态方程**来表示移动充电器的充电过程，即认为移动充电器为最后一个传感器即任务结束，那么对于每个传感器有两个时间，一个是**等待充电时间**，一个是**等待传感器任务结束的时间**，而每个时间本质上是由充电器在路上的时间和给充

电器充电的时间构成的，那么在给定路线、移动速率和充电速率后，根据差分方程求解即可求得每个传感器的电池容量。再求得模型后通过控制变量对移动充电器的移动速率和充电速率绘制了关于传感器电池总量的**灵敏度曲线**。

对于问题三，在前两问的基础上，将 1 个移动充电器换成 4 个移动充电器，那么按照前两问的求解，可以先求解出 4 辆移动充电器的最佳充电路线，再按照问题二建立的电池容量模型进行求解。

对于多车辆问题，直接采用蚁群算法并不容易求解，本题构建了 **k-means 聚类算法+蚁群算法**模型，首先对各节点进行聚类，对聚类后的各分支节点分别采用蚁群算法进行求解，可以极大地简化问题，并求出较优解。

关键词：启发式算法 路径规划 蚁群算法 状态方程 灵敏度曲线 k-means 聚类

目　　录

一、引言

1.1 问题背景

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。

影响 WSN 生命周期最重要的一个因素是能量。想要让 WSN 能够持续不断地运转，就必须持续为 WSN 提供能量。提供能量的方式之一是能量收集（Energy Harvesting），通过利用太阳能或风能等环境能源让传感器自行从环境中汲取能量以维持其运作。然而这种方式提供的能量不但不稳定，而且太过于依赖环境，一旦环境达不到条件，WSN 无法从环境中汲取能量自然也就无法运转。提供能量的另外一种方式是电池供电，并利用**移动充电器**定期为传感器的电池补充能量，从而源源不断地为 WSN 提供稳定的能量使其正常运转。通过这种方式供电的网络也被称为**无线可充电传感器网络** WRSN（wireless Rechargeable Sensor Network）。

1.2 问题提出

在该系统中，传感器从环境中收集信息并将收集到的信息传递给数据中心。当一个传感器的电量低于一个阈值时便无法进行正常的信息采集工作，为了让 WRSN 正常运转，移动充电器需要定期为传感器进行充电以避免其电量低于阈值。移动充电器从数据中心出发，以**固定的速度**依次经过每个传感器，在每个传感器处停留一段时间并以**固定的充电速率**为传感器充电，直到为所有传感器充电完成之后返回数据中心。每个传感器都有**特定的能量消耗速率**，以及**固定的电池容量**。移动充电器的能量消耗主要有两个方面：一是为传感器节点充电所导致的正常的能量消耗；另外一方面则是移动充电器在去为传感器充电的路上的能量消耗。为了减小移动充电器在路上的能量消耗，需要合理地规划移动充电器的充电路线。请考虑以下问题：

1.若给出每个节点的经纬度（见附件 1），请考虑当**只派出一个**移动充电器时，如何规划移动充电器的充电路线才能最小化移动充电器在路上的能量消耗。

2.若给出每个节点的经纬度、每个节点的能量消耗速率（见附件 2），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），在只派出一个移动充电器的情况下，若采用问题 1 规划出来的充电路线，每个传感器的电池的容量应至少是多大才能保证整个系统一直正常运行（即系统中每个传感器的电量都不会低于 f（mA））？

3.若给出每个节点的经纬度、每个节点的**能量消耗速率**（同见附件 2），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），但为了提高充电效率，同时**派出 4 个**移动充电器进行充电，在这种情况下应该如何规划移动充电器的充电路线以最小化所有移动充电器在路上的总的能量消耗？每个传感器的电池的容量应至少是多大才能保证整个系统一直正常运行？

1.3 问题分析

问题一：本问题为典型的旅行商问题（Traveling Salesman Problem, TSP），属于 NP 完全问题，在离散组合优化中研究较多，通过不断优化路径，来有效节省成本。一般常用动态规划法、概率论算法、近似解法、人工智能启发式算法及神经网络算法进行计算，但无论使用什么策略进行解题，都只能无限逼近最优解，并不能保证得到通解。

在本题中，不考虑每个传感器的耗电速率及电池容量，只考虑最短线路规划，则可以看成只有权重边的无向图，只需求出以数据中心为起点的最短环路即可。本题拟采用典型的启发式算法进行求解，例如蚁群算法、模拟退火算法、遗传算法及禁忌搜索算法。

问题二：本问题是要求解各传感器的电池容量，在给定规定路线后，如何确定每个传感器电池总量则需要建立相应的模型进行计算，题目中未给出充电器的行驶速度，充电速率以及能够正常工作的最低电量 $f(mA)$，不妨先将最低电量 $f(mA)$ 设为 0，便于计算，最后将每个传感器的总电量加上 $f(mA)$ 即是实际总电量。

本题先建立基础的模型，认为移动充电器充为最后一个传感器即任务结束，那么对于每个传感器有两个时间，一个是等待充电时间，一个是等待传感器任务结束的时间，而每个时间本质上是有充电器在路上的时间和给充电器充电的时间构成的，那么在给定路线、移动速率和充电速率后，根据差分方程求解即可求得每个传感器的电池容量。

问题三：问题 3 在前两问的基础上，将 1 个移动充电器换成 4 个移动充电器，那么按照前两问的求解，可以先求解出 4 辆移动充电器的最佳充电路线，再按照问题 2 建立的电池容量模型进行求解。

对于多车辆问题，直接采用蚁群算法并不容易求解，本题拟采用聚类算法对各节点进行聚类，对聚类后的各节点分别采用蚁群算法进行求解，可以极大地简化问题。

二、模型假设与符号说明

2.1 模型假设

1、移动充电器从起点到下一个点的路程为直线，不考虑绕路等因素；
2、移动充电器移动速度、充电速率固定，不考虑环境因素；
3、每个节点的能量消耗速率固定，不考虑长时间带来的损耗因素；
4、多个移动充电器的硬件参数相同；

2.2 符号说明

表 1 符号说明

变量名	符号说明	单位
v	移动充电器的移动速率	km/h
r	移动充电器的充电速率	mA/h
$f(mA)$	传感器能够正常工作的最低电量	mA
S_i	传感器i，$i \in [1,29]$	
DC	数据中心	
$L_{i,j}$	路径点i和路径点j之间的距离，$i,j \in [1,29]$	km
C_i	路径点i的耗电速率，$i \in [1,29]$	mA/h
B_i	路径点i的电池容量，$i \in [1,29]$	mA
t_wait_i	充电器移动到路径点i所耗时间	s
t_pass_i	充电器离开路径点i到充电器任务结束所耗费的时间	s

注：列出符号及重复的符号以出现处为准

三、数据分析、预处理及可视化

本题所给的数据由经纬度来表示传感器及数据中心的位置，但涉及到最优路径规划，在计算两点距离时需更具待求距离两端的经纬坐标来得到实际测地距离，再进行最优路劲规划的相关计算。

Haversine 公式是一种使用两个点的经度和纬度来计算球体表面上两个点之间的距离的非常准确的方法。Haversine 公式是余弦球定律的重新公式化，但是以 Haversines 表示的公式对于较小的角度和距离更有用。

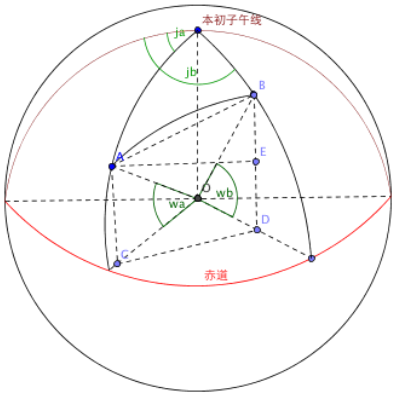


图 1 经纬度距离计算图

先将经纬度转换为弧度：

$$\begin{cases} x = lon \cdot \pi / 180 \\ y = lat \cdot \pi / 180 \end{cases} \tag{1}$$

由图，其中 A、B 两点的经纬度分别为(j_A, w_A)和(j_B, w_B)，确定的测地距离 d 与经纬度转换的模型建立如下：

$$d = 2r \cdot \arcsin^2(\sqrt{\sin^2((j_B - j_A)/2) + \cos\varphi_A \cos\varphi_B \sin^2((w_B - w_A)/2)}) \tag{2}$$

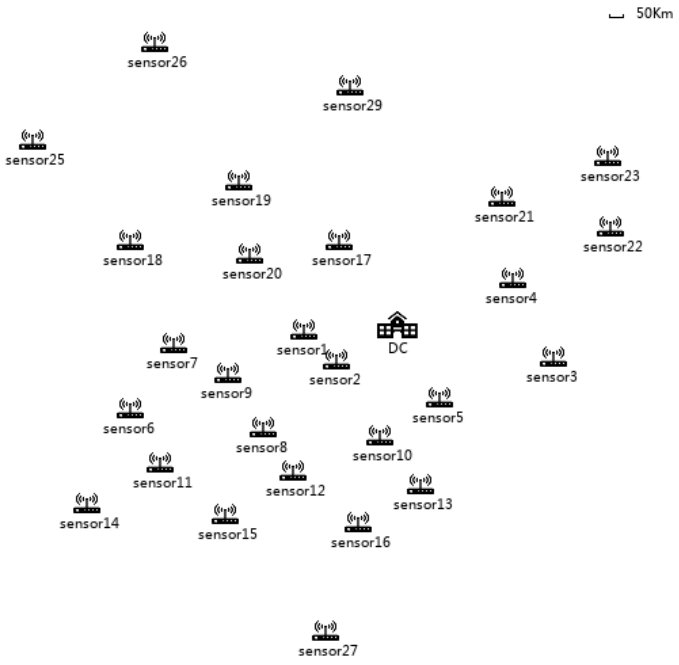


图 2 平面位置图

为了后续结果的可视化，故需要将各个坐标显示在图上，而在编程中图的左上角一般是坐标原点，与我们常用的平面直角坐标系不同，而且各坐标点的实际位置与像素位置不处于一个数量级，需要进行映射才能得到较好的效果图，如图 2 所示。

四、建立最优规划模型

欲求解最短充电线路，可将本题作为典型的 TSP 问题进行求解。由于该问题属于 NP 完全问题，故拟采用典型的启发式算法进行求解。

4.1 蚁群算法求解 TSP

蚁群算法作为一种新型启发式智能算法，于 1996 年由意大利学者 MARCO DORIGO 人等提出，被广泛应用于 TSP 问题(旅行商问题[1])、车辆路由问题[2]、机器人路径规划问题[3]。蚂蚁觅食往往依靠蚁群之间的协作性进行，在觅食过程中，前期路径选择会有比较强的随机性，但由于蚂蚁能够在所经过的路径上分泌信息素，且能够感知信息素强度，并趋向于向信息素浓度高的路径移动，因此能够形成一条趋向于最短的觅食路径[4]。故基于蚁群行为提出的蚁群算法具有群体协作、正反馈、并行计算的三大特点。

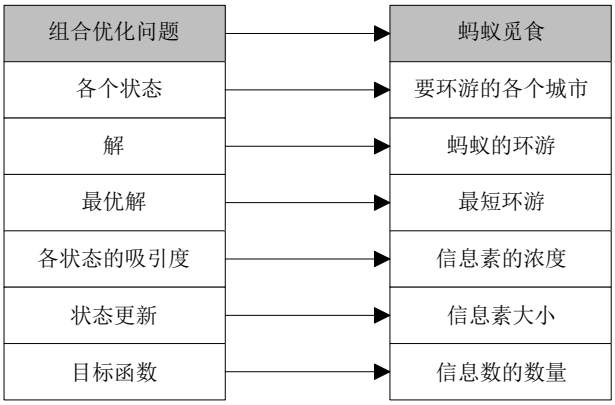


图 3 蚂蚁觅食与 TSP 问题对比

在求解现实 TSP 问题时，需要先生成一定数量的蚁群，让每只蚂蚁建立一个解或者一部分解，每只蚂蚁从初始状态出发，根据“激素”浓度来选择下一个要转移到的状态，直到建立一个完整解，根据每个解的好坏程度在路程上释放成正比例的“激素”，然后不断迭代，直到寻找到最优解。

在本题中，设 N 为路径点的个数； M 为蚁群大小； $L_{i,j}$ 为路径点 i 和路径点 j 之间的距离， $i,j \in [1,29]$； $P_{i,j}$ 为路径点 i 和路径点 j 之间的信息素浓度；

每只蚂蚁在出发前都需要进行初始化，每只蚂蚁都从路径点 0（即数据中心 DC）出发。且第一次出发时将所有路径上的信息素浓度都设定为 1.0，在行走过程中选择下一个路径点（**没有走过的路径点**）的概率则是由信息素浓度和距离共同决定的：

$$select\_route\_prob[i,j]=\frac{P_{i,j}^{\alpha}}{L_{i,j}^{\beta}} \tag{3}$$

该概率与信息素浓度成正比，与距离成反比。然后根据该概率随机选择下一个路径点，显然当下一个路径点路程越短、路程上的信息素浓度越高则越容易吸引蚂蚁。每只蚂蚁如此进行不断行走，直到走到最后一个路径点，此时必须返回初始点 0。此时能够得到蚂蚁 m 的总路程 $Total\_L_m$，以及路程 $route_m$。

每次搜索都先让每只蚂蚁分别进行搜索，而更新信息素浓度则在所有蚂蚁搜索结束后进行更新，每个路径点之间的更新的信息素如下：

$$\Delta P_{i,j}=\sum_{m=0}^{M}\frac{Q}{Total\_L_m} \tag{4}$$

其中 Q 为常数参量。

在得到每个路径点之间的更新信息素浓度后，则需要结合之前的信息素进行叠加。而考虑到**信息素挥发**的因素，之前存在的信息素应该存在一个衰减系数 γ，此时每个路径点之间的信息素浓度为：

$$P_{i,j}^{'}=\gamma P_{i,j}+\Delta P_{i,j} \tag{5}$$

此时不断迭代即可不断优化最短路径，从而得出一条较为合理的路线。

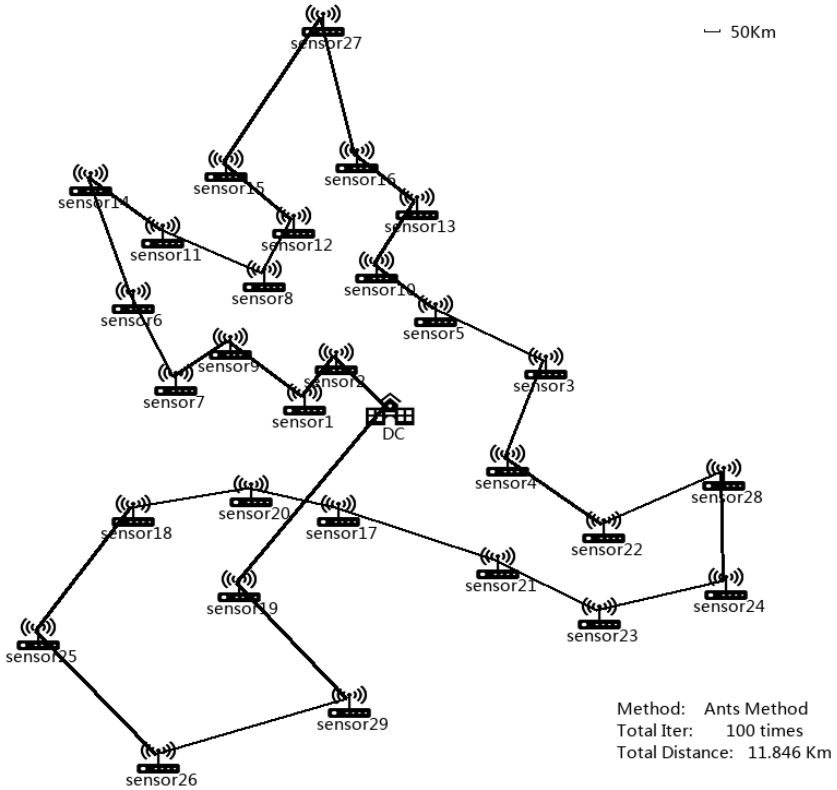


图 4 蚁群算法最优结果规划

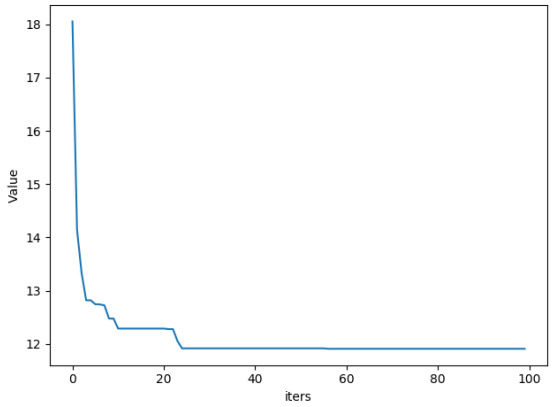
4.2 启发式模型效果对比

常用的启发式算法除了蚁群算法外，还有模拟退火算法、遗传算法和禁忌搜索算法，为了对比各模型的效果，使用 Python 构建了四种模型并封装成函数，以下是参数说明。

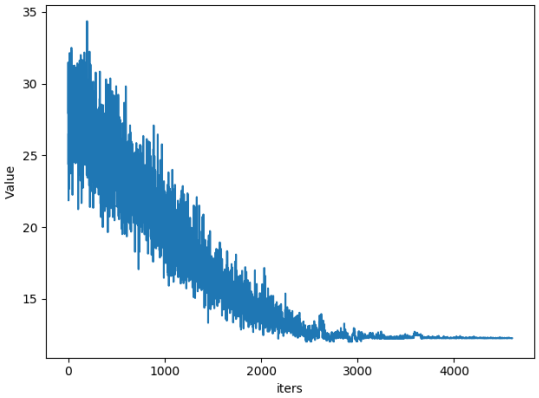
表 2 各算法参数说明

模型	参数描述	值
蚁群算法	信息启发因子	1.0
	距离因子	2.0
	挥发系数	0.5
	信息素浓度更新常数	100
	蚁群数量	30
模拟退火算法	退火系数	0.99
	初始温度	3000
	最低温度	1
遗传算法	种群规模	1000
	最大进化代数	200
	交叉算子	0.7
	变异算子	0.4
禁忌搜索算法	禁忌表长度	40
	置换次数	40
	候选集大小	50

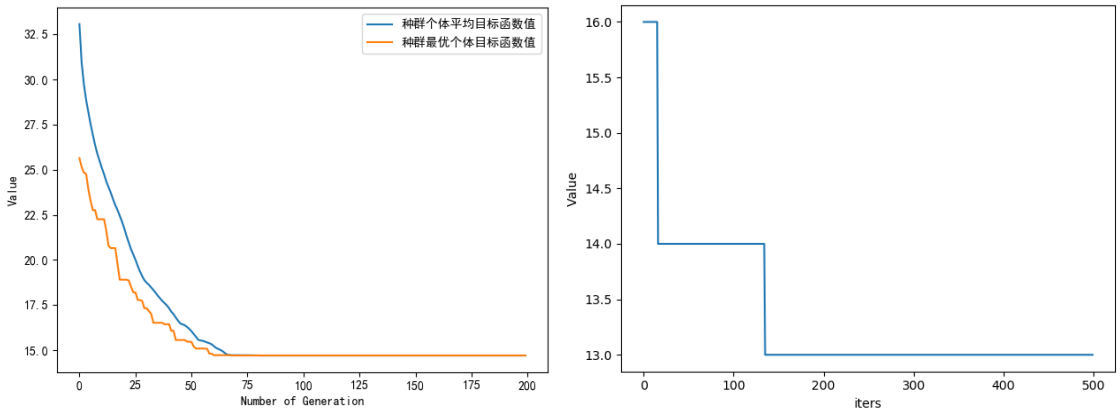
在编写好各模型函数后需要建立评价指标来进行对比，为了更好的对比各模型，先绘制迭代曲线图来寻找最优，根据曲线不断调整参数。



（a）蚁群算法



（b）模拟退火算法



（c）遗传算法　　（d）禁忌搜索算法

图 5 各算法迭代过程最优距离曲线

为了评价整体效果，使用最优结果出现所用时间、最优结果路线距离按照权重 0.2 和 0.8 计算总得分（**得分越小越好**），如下：

$$Score = 0.2t + 0.8dist \tag{6}$$

表 3 各算法指标对比

算法	最优结果出现所用时间(s)	最短路线距离（km）	总得分
蚁群算法	**2.23**	**11.846**	**11.71**
模拟退火算法	19.23	12.02	13.46
禁忌搜索算法	52.31	13.383	63.01
遗传算法	**23.79**	**14.168**	35.13

注：总得分越小，效果越好

从以上结果中可以看出蚁群算法不论在时间上还是最短路线距离上都有着足够的优势，而遗传算法不仅计算时间长，得到的最短路线距离也较大，综合比较，在本题中若单独使用启发式算法，蚁群算法能够以较好的效果来得到最短路线距离。

五、建立电池容量模型

在给定规定路线后，如何确定每个传感器电池总量则需要建立相应的模型进行计算，题目中未给出充电器的行驶速度，以及充电速率和能够正常工作的最低电量 $f(mA)$，不妨先将最低电量 $f(mA)$ 设为 0，便于计算，最后将每个传感器的总电量加上 $f(mA)$ 即是实际总电量。

5.1 构建电池容量模型

考虑构建电池容量模型，首先需要考虑整个充电任务如何才算结束，目前只有一个充电器从 DC 开始按照行程对节点进行充电，最简单的情况是给最后一个传感器充完电

回到 DC 时所有的传感器电量都在最低电量以上，而复杂的情况是考虑到充电器能够持续充电，即能够维持闭环充电。

对于每个节点，在充电器刚刚抵达时的等待时间如下：

$$t\_wait_i = \sum_{k=0}^{i-1} \frac{L_{k,k+1}}{v} + \sum_{k=1}^{i-1} \frac{t\_wait_k C_k}{r - C_k} \tag{7}$$

等待时间主要由两部分构成，一部分是路程的时间，另一部分是传感器的充电时间，需要充电的容量 $t\_wait_k C_k$ 暂定为之前的等待时间与耗电速率的乘积，充电时间即为充电容量除以充电速率与耗电速率的差值。

除了考虑等待时间，还需要计算充电器离开该传感器以后到充电器结束任务的时间。

$$t\_pass_i = t\_wait_{29} + \frac{t\_wait_{29} C_{29}}{r - C_{29}} - t\_wait_i - \frac{t\_wait_i C_i}{r - C_i} \tag{8}$$

该时间主要由两部分构成，一部分是整体任务执行时间，一部分是该传感器之前的等待时间，由于整体任务执行时间包含该传感器的充电时间，故需要再减去该传感器的充电时间，由此可以得到该传感器等待任务结束所需的时间。

对于传感器来说，等待充电时间与等待任务结束时间都是一个耗电过程，而时间最长的那个过程所消耗的电量是最多的，那只需要让该部分电量等于该传感器的电池容量即可。

$$B_i = \begin{cases} t\_wait_i C_i & t\_wait_i > t\_pass_i \\ t\_pass_i C_i & t\_wait_i < t\_pass_i \end{cases} \tag{9}$$

如果再考虑阈值电量，那么实际电池总容量应为 $B_i + f(mA)$ 。

由方程可以看出，该方程为差分方程，如果将移动充电车行驶速度 v，和移动充电车的充电速率 r 设为常数，则可迭代出每个传感器的等待充电时间和等待任务结束时间。

在构建好模型后，通过 python 编程得到基本框架，为了验证模型的有效性，将未知参数设为常数来计算总电量。这里将速度设置为 **50km/h**，充电速率为 **100mA/h**，每个传感器的耗电速率已知，则可以得到如下结果。

表 4 传感器电池容量结果 1

传感器	耗电速率（mA/h）	等待充电时间(h)	等待任务结束时间（h）	总容量（mA）
2	5.4	0.008	6.656	35.942
1	7.8	0.014	6.647	51.847
9	4.5	0.026	6.635	29.857
7	5.5	0.038	6.619	36.404
6	3.6	0.054	6.604	23.774
14	4.5	0.078	6.574	29.583
11	6.4	0.095	6.545	41.888
8	4.6	0.12	6.524	30.01

12	4.5	0.154	6.485	29.182
15	5.5	0.185	6.439	35.415
27	4.5	0.233	6.392	28.764
16	7.4	0.292	6.278	46.457
13	6.5	0.342	6.229	40.488
10	4.5	0.446	6.141	27.634
5	3.8	0.549	6.037	22.941
3	4.5	0.641	5.912	26.604
4	5.5	0.733	5.768	31.724
22	7.5	0.859	5.521	41.408
28	5.5	1.038	5.395	29.672
24	4.5	1.338	5.092	22.914
23	3.5	1.584	4.873	17.056
21	5.5	1.832	4.423	24.326
17	7.5	2.059	3.921	29.408
20	3.5	2.479	3.859	13.506
18	5.5	3.178	2.774	17.479
25	4.3	3.524	2.552	15.153
26	3.6	4.257	1.828	15.325
29	6.4	4.867	0.479	31.149
19	5.4	5.466	0	29.516

5.2 灵敏度分析

由此可以得出电池总容量与耗电速率成反比，与等待时间成正比。除了这两个参数外，由于移动充电器的移动速率和充电速率都是给定的，故这里拟采用控制变量法分析移动速率与充电速率与传感器电池容量的关系。

1）固定移动速率为 30km/h:

将移动速率设为常数后，令充电速率从 20mA/h 逐步增长到 100mA/h，可以得到如下曲线：

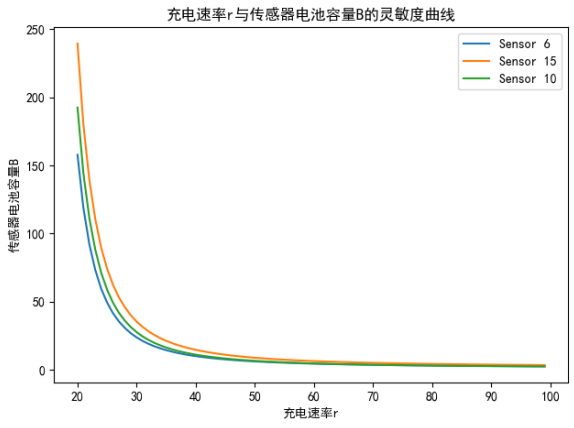


图 6 充电速率 r 与传感器电池容量 B 的灵敏度曲线

从图中可以看出电池容量与充电速率呈指数衰减，在充电速率逐渐增大到 70mA/h 后逐渐收敛。

2）固定充电速率为 30mA/h:

将充电速率设为常数后，令充电速率从 20mA/h 逐步增长到 100mA/h，可以得到如下曲线：

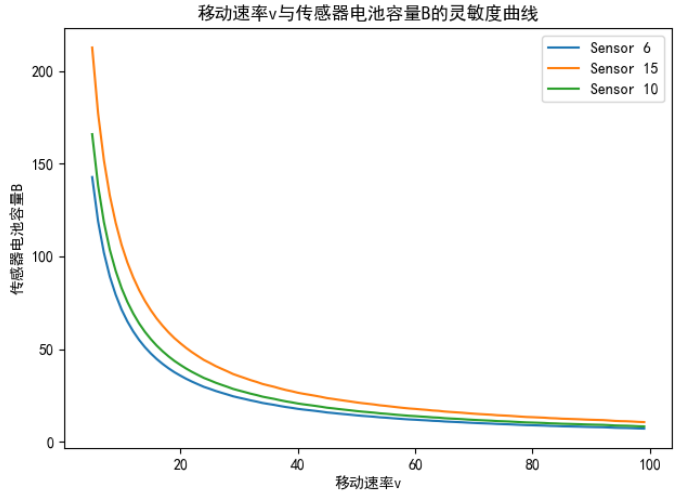


图 7 移动速率 v 与传感器电池容量 B 的灵敏度曲线

从图中可以看出电池容量与移动速率也呈指数衰减，在移动速率逐渐增大到 90km/h 后逐渐收敛。

3）同时改变充电速率与移动速率：

为了更好的研究充电速率与移动速率与传感器电池容量之间的关系，通过调整算法，将移动速率在 5-100km/h 之间步进，充电速率在 30-60mA/h 之间步进，可以得到如下的灵敏度曲线。

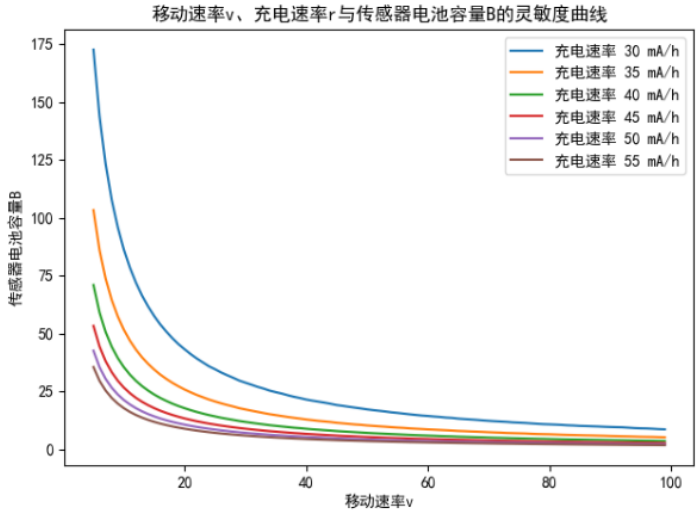


图 8 移动速率 v、充电速率 r 传感器电池容量 B 的灵敏度曲线

从图中可以看出不断增加移动速率，可以指数衰减电池容量，到最后可以逐渐忽视路程对电池容量造成的影响，而逐步增加充电速率可以使得电池容量整体下降，到后面

可以逐渐忽视充电速率对电池容量带来的影响。

六、模型改良

移动充电器的充电路线若只单独考虑行程最短，则会忽略到每个移动充电器充电速率不同的因素，在问题 1 的基础上，需要按照问题 3 所说考虑移动充电器的移动速率、充电速率以及传感器的耗电速率来规划最佳充电线路,而且移动充电车也增加到了 4 辆。那该问题应先分解为两个小问题，先考虑 4 个移动充电车的最佳充电线路，再考虑求解 4 条线路下各传感器的电池容量。

6.1 4 个移动充电车的最佳路线

在问题 1 中采用蚁群算法之间求解 TSP 路径问题，但是当将移动充电器增加至 4 个时就不是简单的 TSP 问题。如果之间在蚁群算法上进行修改，很难控制蚁群分开寻找最优路线，本文拟采用聚类算法+蚁群算法进行求解，通过聚类算法，可以将各节点进行聚类，得到一个先验分类。根据聚类结果，分别采用蚁群算法进行迭代，即可得到较优解。

为使得各个移动充电器行走的距离最小，需将所有传感器按地理位置分为 4 类，将相近的传感器归位一类，这样能够相对减少无线充电车的行驶路程，一定程度上减少了其在路程中的电量损耗。之后对每个分区内的各个传感器的路径进行后续的优化。

在对传感器节点分类是，我们使用 k 均值聚类算法（k-means clustering algorithm），它是一种迭代求解的聚类分析算法，其步骤是：

1）首先，选择 K 个初始质心（通常随机初始化），其中 K 是用户预期的簇个数。

2）每个点指派到最近的质心，而指派到一个质心的点集合为一个簇。这里的距离量度有多种，通常采用欧几里得距离：

$$\text{SSE}=\sum_{i=1}^{K}\sum_{x\in T_i}(t_i-x)^2 \tag{10}$$

SSE 为误差的平方和（Sum of Squared Error, SSE），T_i 为第 i 个簇，x 为 t_i 中的点，t_i 为第 i 个簇的均值。

3）然后，根据指派到簇的点，更新每个簇的质心。

$$t_k=\frac{1}{m_k}\sum_{x\in C_k}x_k \tag{11}$$

4）重复指派和更新步骤，直到簇不发生变化，或等价为质心不发生变化。

K 均值聚类算法在聚类领域具有很广泛的应用，对此问题也同样有效。在本题中，本文将各传感器节点，按照其坐标位置使用 k 均值聚类算法将它们分为四个区域，最终分类结果如下：

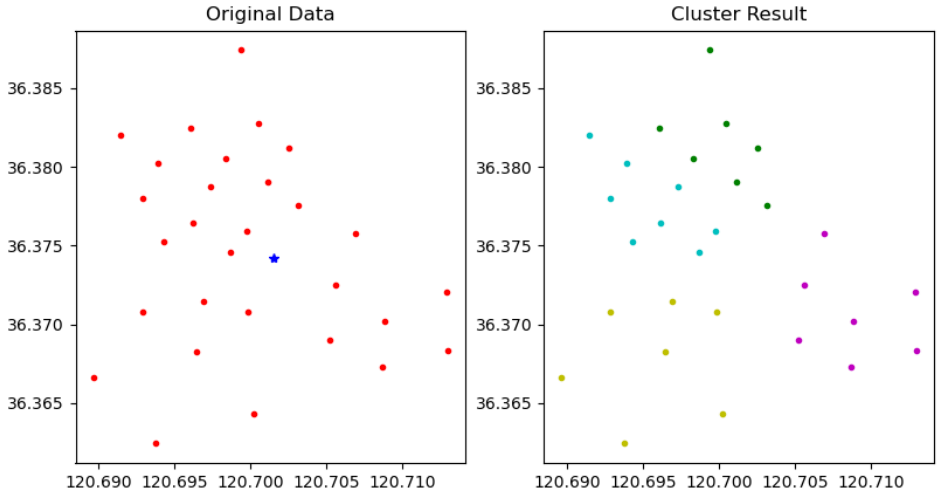


图 9 k 均值聚类算法聚类结果

图 1（左）中红点为各传感器节点位置分布，蓝点为数据中心位置，图 2（右）为数据节点进行 4 类聚类的结果，可以看出相对数据中心位置，较为相近的各个数据节点划分成一块区域，且各区域节点数基本相同。在得到聚类结果以后可以分别进行蚁群算法进行迭代，迭代后的结果如下：

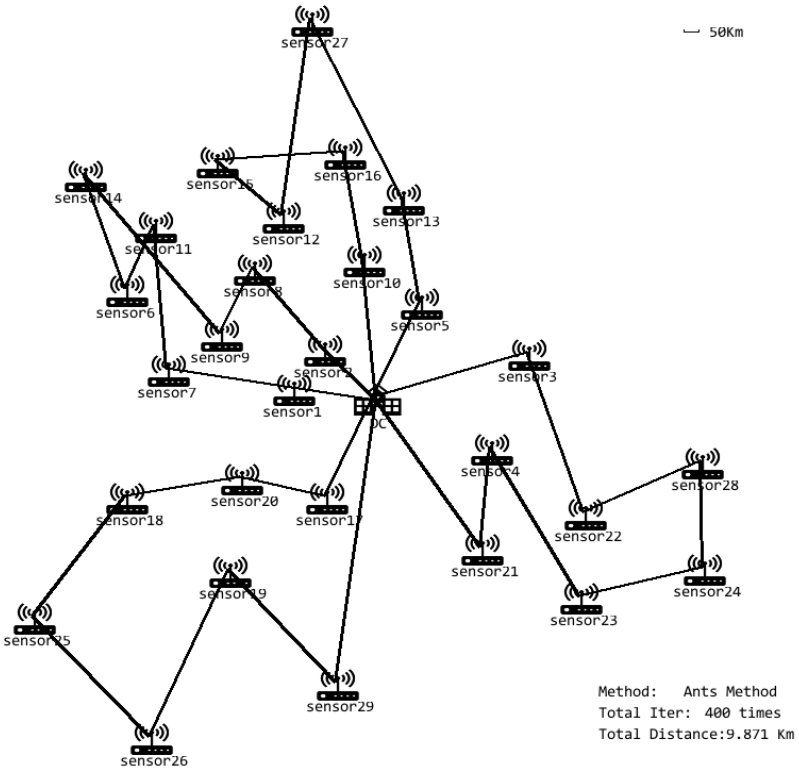


图 10 四个移动充电器的充电路线示意图

6.2 最佳路线下各个传感器的电池容量

在得到最佳路线时，可以将问题分解为 4 个子问题进行计算，即分别计算四条子路线下的电池容量，按照问题 2 构建的模型，可以得到结果如下：

表 5 四条路线下各传感器的总容量

路线类别	传感器	耗电速率（mA/h）	等待充电时间(h)	等待任务结束时间（h）	总容量（mA）
1	1	5.4	0.008	0.196	1.058
1	7	7.8	0.022	0.176	1.373
1	11	4.5	0.042	0.156	0.702
1	6	5.5	0.059	0.133	0.732
1	14	3.6	0.081	0.113	0.407
1	9	4.5	0.12	0.064	0.54
1	8	6.4	0.14	0.028	0.896
1	2	4.6	0.174	0	0.8
2	3	5.4	0.017	0.202	1.091
2	22	7.8	0.039	0.17	1.326
2	28	4.5	0.056	0.157	0.706
2	24	5.5	0.084	0.12	0.66
2	23	3.6	0.107	0.101	0.385
2	4	4.5	0.147	0.05	0.662
2	21	6.4	0.175	0	1.12
3	27	5.4	0.049	0.247	1.334
3	12	7.8	0.075	0.205	1.599
3	15	4.5	0.096	0.193	0.868
3	16	5.5	0.135	0.141	0.776
3	10	3.6	0.166	0.118	0.598
3	5	4.5	0.205	0.065	0.922
3	13	6.4	0.241	0	1.542
4	29	5.4	0.037	0.264	1.426
4	17	7.8	0.061	0.226	1.763
4	19	4.5	0.083	0.211	0.95
4	26	5.5	0.128	0.152	0.836
4	25	3.6	0.162	0.125	0.583
4	18	4.5	0.209	0.063	0.94
4	20	6.4	0.243	0	1.555

注：这里的移动速率未 30km/h，充电速率为 30mA/h

上表即使按照问题 2 的电池容量模型进行计算得到的各个传感器的电池容量，如果再考虑移动充电器回到数据中心，每条路线上各传感器的电池容量只需增加一个固定值即可，这个固定值可由这段路程最后节点到数据中心的行驶时间与各传感器的耗电时间相乘即可。

七、模型评价

7.1 传感器电池容量模型

若考虑实际问题，并不能将问题简单假设为：移动充电器给最后一个传感器充完电就算任务结束，而应能够保持循环充电。故理想状态应该是传感器给最后一个传感器充好电以后回到 DC 给自身充电，然后再次按照既定路线继续充电，这样才是合理的良性循环。

那么该问题会比之前定义的情况 1 更为简化，要求得每个传感器的电池容量，可直接通过计算每个传感器**从移动充电器离开到移动充电器再次过来的时间**与耗电速率的乘积即可得到该传感器的电池容量，即直接从**平衡状态**分析电池容量。

每个传感器的等待时间计算：

$$t\_wait_i = t\_DC + t\_L + \sum_{k=0,k\neq i}^{30} \frac{t\_wait_k c_k}{r - c_k} \tag{12}$$

由此计算得到的应比在前文所建立的模型更加符合情况，但因时间有限，未能顺利解得方程。

7.2 多移动充电器路径规划模型

多移动充电器进行规划时，本文是先进行 k-means 聚类，然后再分别通过蚁群算法求解最佳路线的。但是 k-means 聚类的聚类中心是随机初始化质心，以至于最终聚类中心是在四类节点的位置中心，但移动充电器的出发点是从数据中心开始的，这样会给结果带来不小的误差。

参考文献

[1] 张于贤, 丁修坤, 薛殿春, 王晓婷. 求解旅行商问题的改进蚁群算法研究[J]. 计算机工程与科学, 2017, 39(8): 1576-1580.

[2] Yu, S.P. and Li, Y.P. (2012) An Improved Ant Colony Optimization for VRP with Time Windows. Applied Mechanics and Materials, 263-266, 1609-1613.

[3] Bennet, D.J. and McInnes, C.R. (2010) Dis-tributed Control of Multi-Robot Systems Using Bifurcating Potential Fields. Robotics & Autonomous Systems, 58, 256-264.

[4] 张森,王奔,孙梦亚, 等.改进的蚁群算法在灭火机器人多火源路径规划的应用[J].计算机科学与应用,2020,10(05):851-859.

附录

Main.py(问题一主函数)

```
import copy
from models.ant_model import Ant
from models.geat_model import MyProblem
from models.Tabu_model import tabusearch
from models.sa_model import Simulated_Annealing
import geatpy as ea
import pandas as pd
import numpy as np
from libs.visulize import visulize, plot_iter
from libs.tools import millerToXY, get_dist, getDistance
import time
def ants_method(distance_graph, Total_iter, ALPHA=1.0, BETA=2.0, RHO=0.5, Q=100.0, city_num=30, ant_num=30):
    """================================ 算 法 参 数 设 置 ================================"""
    # 参数
    '''
    ALPHA:信息启发因子，值越大，则蚂蚁选择之前走过的路径可能性就越大
        ，值越小，则蚁群搜索范围就会减少，容易陷入局部最优
    BETA:Beta 值越大，蚁群越就容易选择局部较短路径，这时算法收敛速度会
        加快，但是随机性不高，容易得到局部的相对最优
    '''
    start_time = time.time()
    pheromone_graph = [[1.0 for col in range(city_num)] for raw in range(city_num)]
    """================================== 搜 索 ==================================="""
    ants = [Ant(ID, city_num, ALPHA, BETA, distance_graph, pheromone_graph) for ID in range(ant_num)]  # 初始蚁群
    best_ant = Ant(-1, city_num, ALPHA, BETA, distance_graph, pheromone_graph)  # 初始最优解
    best_ant.total_distance = 1 << 31  # 初始最大距离
    iter = 1  # 初始化迭代次数
    logs = []
    # 开始搜索
    for i in range(Total_iter):
        # 遍历每一只蚂蚁
        for ant in ants:
            # 搜索一条路径

            ant.search_path()
            # 与当前最优蚂蚁比较
            if ant.total_distance < best_ant.total_distance:
                # 更新最优解
                best_ant = copy.deepcopy(ant)
        logs.append(best_ant.total_distance)
        # 更新信息素
        # 获取每只蚂蚁在其路径上留下的信息素
        temp_pheromone = [[0.0 for col in range(city_num)] for raw in range(city_num)]
        for ant in ants:
            for i in range(1, city_num):
                start, end = ant.path[i - 1], ant.path[i]
                # 在路径上的每两个相邻城市间留下信息素，与路径总距离反比
```

```
                    temp_pheromone[start][end] += Q / ant.total_distance
                    temp_pheromone[end][start] = temp_pheromone[start][end]

        # 更新所有城市之间的信息素，旧信息素衰减加上新迭代信息素
        for i in range(city_num):
            for j in range(city_num):
                pheromone_graph[i][j] = pheromone_graph[i][j] * RHO + temp_pheromone[i][j]
        print(u"迭代次数：", iter, u"最佳路径总距离：", round(best_ant.total_distance, 3))
        iter += 1
    end_time = time.time()
    info = ["Ants Method", str(Total_iter) + " times", str(round(best_ant.total_distance, 3)) + " Km"]

    return best_ant.path, info, end_time - start_time, logs

def geat_method(dist, NIND=500, Encoding='P',MAXGEN=300,XOVR=0.2,Pm=0.5):
    '''NIND:种群规模
    Encoding:编码方式
    MAXGEN：最大进化代数
    XOVR:交叉算子
    Pm：变异算子'''
    problem = MyProblem(dist)  # 生成问题对象
    """================================  种  群  设  置  ================================"""
    Field = ea.crtfld(Encoding, problem.varTypes, problem.ranges, problem.borders)  # 创建区域描述器
    population = ea.Population(Encoding, Field, NIND)  # 实例化种群对象（此时种群还没被初始化，仅仅是完成种群对象的实例化）
    """================================  算  法  参  数  设  置  ================================"""
    myAlgorithm = ea.soea_SEGA_templet(problem, population)  # 实例化一个算法模板对象
    myAlgorithm.MAXGEN = MAXGEN  # 最大进化代数
    myAlgorithm.recOper = ea.Xovox(XOVR=XOVR)  # 设置交叉算子
    myAlgorithm.mutOper = ea.Mutinv(Pm=Pm)  # 设置变异算子
    myAlgorithm.drawing = 1  # 设置绘图方式（0：不绘图；1：绘制结果图；2：绘制过程动画）
    """=========================  调  用  算  法  模  板  进  行  种  群  进  化  ========================="""
    [population, obj_trace, var_trace] = myAlgorithm.run()  # 执行算法模板，得到最后一代种群以及进化记录器
    population.save()  # 把最后一代种群的信息保存到文件中
    # 输出结果
    best_gen = np.argmin(obj_trace[:, 1])  # 记录最优种群是在哪一代
    best_ObjV = np.min(obj_trace[:, 1])
    print('最短路程为：%s' % (best_ObjV))
    print('最佳路线为：')
    best_journey, edges = problem.decode(var_trace[best_gen, :])
    print(best_journey)
    for i in range(len(best_journey)):
        print(int(best_journey[i]), end=' ')
    print()
    print('有效进化代数：%s' % (obj_trace.shape[0]))
    print('最优的一代是第 %s 代' % (best_gen + 1))
    print('评价次数：%s' % (myAlgorithm.evalsNum))
    print('时间已过 %s 秒' % (myAlgorithm.passTime))
    info = ["Geat Method", str(MAXGEN) + " times", str(int(best_ObjV)) + " Km"]
    return best_journey, info, myAlgorithm.passTime
```

```
if __name__ == "__main__":
    data = pd.read_excel("data.xlsx")
    lonlat = data.loc[0:30]
    real_data = []
    for i in range(len(lonlat)):
        real_data.append([lonlat.loc[i]["传感器经度"] * 1000, lonlat.loc[i]["传感器纬度"] * 1000, i])
    real_data = np.array(real_data)
    x_min, y_min = np.min(real_data[:, 0]), np.min(real_data[:, 1])
    real_data = real_data - np.array([x_min, y_min, 0])
    # 可视化基本地图
    vi = visulize()
    base_map = vi.base_map(real_data)
    # 求解距离矩阵
    dist = np.zeros((30, 30), np.float)
    for i in range(len(real_data)):
        for j in range(len(real_data)):
            dist[i][j] = getDistance(lonlat.loc[i]["传感器经度"], lonlat.loc[i]["传感器纬度"],
                                     lonlat.loc[j]["传感器经度"], lonlat.loc[j]["传感器纬度"])
    # 蚁群算法
    path, info, pass_time, logs = ants_method(dist, 100, ALPHA=1.0, BETA=1.8, ant_num=80)
    # 遗传算法
    # path, info, pass_time = geat_method(dist, NIND=1000, Encoding='P', MAXGEN=200, XOVR=0.5,
Pm=0.5)
    # 禁忌搜索算法
    # ta = tabusearch(dist, city_nums=30, diedaitimes=150, cacu_time=40, tabu_length=130,
origin_times=500)
    # path, info, pass_time, logs = ta._search_()
    # 模拟退火算法
    # SA = Simulated_Annealing(dist*100, city_nums=30, alpha=0.999, T=100, Tmin=1, iters=1200)
    # path, info, pass_time, logs = SA.run()
    plot_iter(logs)
    print(path)
    print(info)
    print(pass_time)

    vi.arrow_map(base_map, path, info)
```

ant_model.py(蚁群算法模型)

```
import random
import sys
# ----------- 蚂蚁 -----------
class Ant(object):

    # 初始化
    def __init__(self, ID, city_num, ALPHA, BETA, distance_graph, pheromone_graph):

        self.ID = ID    # ID
        self.city_num = city_num
        self.ALPHA = ALPHA
        self.BETA = BETA
        self.pheromone_graph = pheromone_graph
        self.distance_graph = distance_graph
        self.__clean_data()   # 随机初始化出生点

    # 初始数据
    def __clean_data(self):

        self.path = [0]   # 当前蚂蚁的路径
```

```
        self.total_distance = 0.0   # 当前路径的总距离
        self.move_count = 0   # 移动次数
        self.current_city = 0   # 当前停留的城市
        self.open_table_city = [True for _ in range(self.city_num)]   # 探索城市的状态
        '''起点必须是 0，所以不用随机初始化'''
        # city_index = random.randint(0, self.city_num - 1)   # 随机初始出生点，随机一个
        # self.current_city = city_index
        # self.path.append(city_index)
        self.open_table_city[self.current_city] = False
        self.move_count = 1

    # 选择下一个城市
    def __choice_next_city(self):

        next_city = -1
        select_citys_prob = [0.0 for _ in range(self.city_num)]   # 存储去下个城市的概率
        total_prob = 0.0

        # 获取去下一个城市的概率
        for i in range(self.city_num):
            if self.open_table_city[i]:
                try:
                    # 计算概率：与信息素浓度成正比，与距离成反比
                    select_citys_prob[i] = pow(self.pheromone_graph[self.current_city][i],
self.ALPHA) * pow(
                        (1.0 / self.distance_graph[self.current_city][i]), self.BETA)
                    total_prob += select_citys_prob[i]
                except ZeroDivisionError as e:
                    print('Ant ID: {ID}, current city: {current}, target city:
{target}'.format(ID=self.ID,

current=self.current_city,

target=i))
                    sys.exit(1)

        # 轮盘选择城市
        if total_prob > 0.0:
            # 产生一个随机概率,0.0-total_prob
            temp_prob = random.uniform(0.0, total_prob)
            for i in range(self.city_num):
                if self.open_table_city[i]:
                    # 轮次相减
                    temp_prob -= select_citys_prob[i]
                    if temp_prob < 0.0:
                        next_city = i
                        break

        # 未从概率产生，顺序选择一个未访问城市
        # if next_city == -1:
        #     for i in range(city_num):
        #         if self.open_table_city[i]:
        #             next_city = i
        #             break

        if (next_city == -1):
            next_city = random.randint(0, self.city_num - 1)
```

```
            while ((self.open_table_city[next_city]) == False):    # if==False,说明已经遍历过了
                next_city = random.randint(0, self.city_num - 1)

        # 返回下一个城市序号
        return next_city

    # 计算路径总距离
    def __cal_total_distance(self):

        temp_distance = 0.0
        for i in range(len(self.path) - 1):
            start, end = self.path[i], self.path[i + 1]
            temp_distance += self.distance_graph[start][end]
        self.total_distance = temp_distance

    # 移动操作
    def __move(self, next_city):

        self.path.append(next_city)
        self.open_table_city[next_city] = False
        self.total_distance += self.distance_graph[self.current_city][next_city]
        self.current_city = next_city
        self.move_count += 1

    # 搜索路径
    def search_path(self):

        # 初始化数据
        self.__clean_data()

        # 搜素路径，遍历完所有城市为止
        while self.move_count < self.city_num:
            # 移动到下一个城市
            next_city = self.__choice_next_city()
            self.__move(next_city)
        self.__move(0)    # 最后的一个路径必须回到 0
        # 计算路径总长度
        self.__cal_total_distance()
```

problem_2.py(问题 2 主函数)

```
import pandas as pd
import numpy as np
from libs.visulize import visulize, plot_iter
from libs.tools import millerToXY, get_dist, getDistance
import pandas as pd
import matplotlib.pyplot as plt

plt.rcParams['font.sans-serif'] = ['SimHei']  # 步骤一（替换 sans-serif 字体）
plt.rcParams['axes.unicode_minus'] = False  # 步骤二（解决坐标轴负数的负号显示问题）
def get_B(route, dist, v, r, c,):
    def t_wait(t_wait_map, route, dist, v, r, c, i):
        total_L = 0
        # 计算路程时间
        for k in range(i):
            total_L += dist[route[k], route[k+1]]  # i=1 则距离为 0->1，range 默认从 0 开始
        run_time = total_L / v
        # 计算充电时间
        r_time = 0.0
        for k in range(1, i-1):
            r_time += (t_wait_map[k] * c[k]) / (r - c[k])
        return run_time + r_time
    def t_pass(t_wait_map, r, c, i):
        t_pass_total = t_wait_map[len(route) - 1] + (t_wait_map[len(route) - 1] * c[len(route) - 1])/(r -
c[len(route) - 1])
        t_pass_time = t_pass_total - t_wait_map[i] - (t_wait_map[i]*c[i])/(r-c[i])
        return t_pass_time
    # 初始化
    t_wait_map = np.zeros((len(route), 1), dtype=np.float)
    t_pass_map = np.zeros((len(route), 1), dtype=np.float)

    for i in range(0, len(route)):
        t_wait_map[i] = np.round(t_wait(t_wait_map, route, dist, v, r, c, i), 3)
    for i in range(1, len(route) - 1):
        t_pass_map[i] = np.round(t_pass(t_wait_map, r, c, i), 3)
    # 计算电池容量， 时间最长的×耗电速率就是电池容量
    B = np.zeros((len(route) - 1, 1), dtype=np.float)
    logs = []
    for i in range(1, len(route)):
        t = t_wait_map[i] if t_wait_map[i] > t_pass_map[i] else t_pass_map[i]
        B[i - 1] = np.round(t * c[i], 3)
        logs.append([route[i], c[i], float(t_wait_map[i]), float(t_pass_map[i]), float(B[i - 1])])
    return B, logs




if __name__ == "__main__":
    data = pd.read_excel("data.xlsx")
    lonlat = data.loc[0:30]

    # 求解距离矩阵
    dist = np.zeros((30, 30), np.float)
    for i in range(30):
        for j in range(30):
            dist[i][j] = getDistance(lonlat.loc[i]["传感器经度"], lonlat.loc[i]["传感器纬度"],
                                     lonlat.loc[j]["传感器经度"], lonlat.loc[j]["传感器纬度"])
```

```
C = lonlat.loc[:]["能量消耗速率(mA/h)"]
# route = [0, 2, 1, 9, 7, 6, 14, 11, 8, 12, 15, 27, 16, 13, 10,
#                 5, 3, 4, 22, 28, 24, 23, 21, 17, 20, 18, 25, 26, 29, 19, 0]
routes = [[0, 1, 7, 11, 6, 14, 9, 8, 2],
                  [0, 3, 22, 28, 24, 23, 4, 21],
                  [0, 27, 12, 15, 16, 10, 5, 13],
                  [0, 29, 17, 19, 26, 25, 18, 20]]
flag = 0
for route in routes:
      B, logs = get_B(route, dist, v=30, r=30, c=C)
      for log in logs:
            lo = pd.DataFrame(logs)
            lo.to_excel("logs%d.xlsx"%flag)
      flag += 1
# B_ = [[], [], []]
# label = []
# for r in range(20, 100, 1):
#       B, logs = get_B(route, dist, v=30, r=r, c=C)
#       if r == 20:
#             label.append([logs[4][0], logs[9][0], logs[13][0]])
#       B_[0].append(logs[4][-1])
#       B_[1].append(logs[9][-1])
#       B_[2].append(logs[13][-1])
#
# for i in range(3):
#       plt.plot([r for r in range(20, 100, 1)], B_[i], label="Sensor %d" % label[0][i])
# plt.xlabel("充电速率 r")
# plt.legend()
# plt.ylabel("传感器电池容量 B")
# plt.title("充电速率 r 与传感器电池容量 B 的灵敏度曲线")
# plt.show()


# B_ = [[], [], []]
# label = []
# for v in range(5, 100, 1):
#       B, logs = get_B(route, dist, v=v, r=30, c=C)
#       if v == 5:
#             label.append([logs[4][0], logs[9][0], logs[13][0]])
#       B_[0].append(logs[4][-1])
#       B_[1].append(logs[9][-1])
#       B_[2].append(logs[13][-1])
#
# for i in range(3):
#       plt.plot([r for r in range(5, 100, 1)], B_[i], label="Sensor %d" % label[0][i])
# plt.xlabel("移动速率 v")
# plt.legend()
# plt.ylabel("传感器电池容量 B")
# plt.title("移动速率 v 与传感器电池容量 B 的灵敏度曲线")
# plt.show()


# B_ = []
# label = []
# for v in range(5, 100, 1):
#       temp = []
#       for r in range(30, 60, 5):
```

```
#                 B, logs = get_B(route, dist, v=v, r=r, c=C)
#                 temp.append(logs[10][-1])
#                 if v == 5:
#                     label.append(r)
#         B_.append(temp)
# B_ = np.array(B_)
# print(B_)
# for i in range(len(label)):
#         print(B_[:][i])
#         plt.plot([v for v in range(5, 100, 1)], B_[:, i], label="充电速率  %d mA/h" % label[i])
# plt.xlabel("移动速率 v")
# plt.legend()
# plt.ylabel("传感器电池容量 B")
# plt.title("移动速率 v、充电速率 r 与传感器电池容量 B 的灵敏度曲线")
# plt.show()
```

计算距离

```
def getDistance(lng1,lat1,lng2,lat2):
    lng1, lat1, lng2, lat2 = map(radians, [float(lng1), float(lat1), float(lng2), float(lat2)]) #  经纬度转换成弧度
    dlon = lng2-lng1
    dlat = lat2-lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    distance = 2*asin(sqrt(a)) * 6371 * 1000    #  地球平均半径，6371km
    distance = round(distance/1000, 3)
    return distance
```

绘制图像

```
import cv2 as cv
from PIL import Image, ImageDraw, ImageFont
import numpy as np
import matplotlib.pyplot as plt
def plot_iter(L):
    '''绘制迭代过程'''
    plt.plot([i for i in range(len(L))], L)
    plt.xlabel("iters")
    plt.ylabel("Value")
    plt.show()


class visulize():
    def base_map(self, real_data):
        '''可视化基本地图，将原来的点放大两倍，左下角为坐标原点，左->右为 x 轴，右->左为 y 轴
        将原来的点除以 5，放缩一下基本像素点为 5KM
        '''
        img = Image.fromarray(np.zeros((750, 800, 3), dtype=np.uint8) + 255)
        # real_data[:, 0:2] = real_data[:, 0:2]
        self.new_data = []    #  把新的位置重新定义下，  可以根据索引画线
        for i in range(len(real_data)):
            self.new_data.append([int(real_data[i][0] * 27 + 50), int(750 - real_data[i][1] * 27) - 50, i])
        #  两个贴图
        DC = Image.fromarray(cv.resize(cv.imread("icon/DC.png"), (45, 30)))
        sensor = Image.fromarray(cv.resize(cv.imread("icon/sensor.png"), (40, 30)))
        #  将贴图放上去
        for i in range(len(real_data)):
            if i == 0:
                img.paste(DC, (self.new_data[i][0], self.new_data[i][1]))
            else:
```

```
                    img.paste(sensor, (self.new_data[i][0], self.new_data[i][1]))
            # 标签放上去
            self.font = ImageFont.truetype(font='msyh.ttf', size=15)
            draw = ImageDraw.ImageDraw(img)
            for i in range(len(real_data)):
                if i == 0:
                    draw.text((self.new_data[i][0] + 15, self.new_data[i][1] + 27), "DC", "black",
font=self.font)
                else:
                    draw.text((self.new_data[i][0] - 10, self.new_data[i][1] + 25), "sensor%d" % i, "black",
font=self.font)
            draw.line((680, 50, 694, 50), "black")  # 横线
            draw.line((680, 48, 680, 50), "black")  # 竖线 1
            draw.line((694, 48, 694, 50), "black")  # 竖线 2
            draw.text((704, 40), "50Km", "black", font=ImageFont.truetype(font='msyh.ttf', size=15))
            cv.imwrite("baseMap.png", np.array(img))
            return img

        def arrow_map(self, base_map, path, info):
            """TODO:后面增加到 4 辆车，应该会有 4 个路径，格式应该在 path 里面放四个列表"""
            img = base_map.copy()
            draw = ImageDraw.ImageDraw(img)
            # dist = get_dist((current_x, current_y), (next_x, next_y))
            # cv.arrowedLine(img, (current_x, current_y), (next_x, next_y), (0, 255, 0), thickness=3,
tipLength=1 / dist)
            current_x, current_y = self.new_data[path[0]][0] + 20, self.new_data[path[0]][1] + 15
            next_x, next_y = self.new_data[path[1]][0] + 17, self.new_data[path[1]][1] + 12
            draw.line((current_x, current_y, next_x, next_y), "black", width=3)
            for i in range(1, len(path) - 1):
                current_x, current_y = next_x, next_y
                next_x, next_y = self.new_data[path[i + 1]][0] + 17, self.new_data[path[i + 1]][1] + 12
                draw.line((current_x, current_y, next_x, next_y), "black", width=3)

            draw.text((600, 660), "Method:", "black", font=self.font)
            draw.text((600, 680), "Total Iter:", "black", font=self.font)
            draw.text((600, 700), "Total Distance:", "black", font=self.font)
            draw.text((680, 660), info[0], "black", font=self.font)
            draw.text((700, 680), info[1], "black", font=self.font)
            draw.text((720, 700), info[2], "black", font=self.font)
            cv.imshow("Root", np.array(img))
            cv.waitKey()
```