武漢理工大學

数学建模暑期培训论文

第 3 题

基于注意力机制和神经网络的移动充电器路径优化模型

第 026 组

姓名	方向
许鸢飞（组长）	编程
尹可汗	写作
李想	建模

2020 年 8 月 15 日

摘要

本文根据无线可充电传感器网络的基本条件和运行模式，从时间约束和能量约束以及路径均衡的角度，建立基于注意力机制的神经网络移动充电器路径规划问模型。

针对问题一，本文首先对数据进行预处理，利用数据中心和传感器的**经纬度坐标**数据求得各节点之间路径构成的**距离矩阵**，建立了**单一充电车路径规划组合优化模型**. **以最小路径长度为优化目标**，利用**人工免疫算法**求得小车行驶最短路径为 **11.48276km**. 使用 lingo 软件对该算法的结果进行验证，发现求得路径相同，结果一致，且人工免疫算法在运算时间上具有优势.

针对问题二，通过对充电小车和传感器**运行机理**进行分析，发现无线可充电传感器系统在运行时间上受到小车行驶速度，充电速度和传感器可持续使用时间的限制，在能量供应上受到传感器电量阈值与能量消耗速度的限制。本文建立了以**能量**和**时间**作为约束的传感器容量设计模型, 以系统内各传感器最小设计电池容量为优化目标，通过模型推倒得出传感器设计最小容量的表达式. 通过对系统内各设备参数进行估计，求出了**系统运行周期为 5.04 小时，传感器电池的设计大小为 52.4 毫安时.** 对影响电池设计容量的参数进行数值分析，我们发现小车在充电站休息时间 1 小时，电池设计容量会增加 29.2 毫安时. 同时传感器设计电压也会对电池容量产生较大影响，应使传感器在相对低压环境内工作.

针对问题三，我们从多充电器**路径分配均衡**以及能量时间约束等角度建立了基**于注意力机制**的神经网络充电器路径规划模型，该模型借助注意力**编解码器**对节点坐标特征进行编码和映射，实现对传感器网络整体结构的有效提取，并利用**神经网络训练**和学习在给定的约束下，降低损失进行路径选择和调整，给出对应路径选则节点的**概率分布**，从而构建多充电车的最佳路径，并在第二问的容量设计上得出了 **4 个移动充电器的路径分别为 3.06km,4.31km,3.49km,3.83km, 各地区传感器电池容量设置 13.6,15.8,17.3,12.8 毫安时, 运行周期分别为 1.36h,1.58h,1.66h,1.46h 小车休息时间为 1 小时.**

本文的优点：1. 考虑了时间和能量以及小车休息间隔和负载均衡等多项指标进行路径优化。2. 采用基于注意力机制编解码器的神经网络算法能快速获取节点网络特征得到规划结果。

关键词： 路径规划　人工免疫算法　注意力模型　神经网络

目录

一、 问题重述

1.1 问题背景

信息的生成, 获取, 传输是网络连通环节的重要组成部分，随着现代无线充电技术，快速充电技术等的迅速发展，无线传感器网络受到了越来越多的关注. 无线传感器网络是一个复杂的系统，由传感器，处理器，无线通信和能量供应三个模块组成，传感器模块用以接受外界数据信息并将其发送至处理器模块，处理器模块对信息进行处理并储存，由无线通信模块进行信息的收发. 无线可充电传感网络 WRSN(Wireless Rechargeable Sensor Network) 已经被大量应用于天气信息收集，电子信号收发等任务中. 无线可充电网络的示意图如1

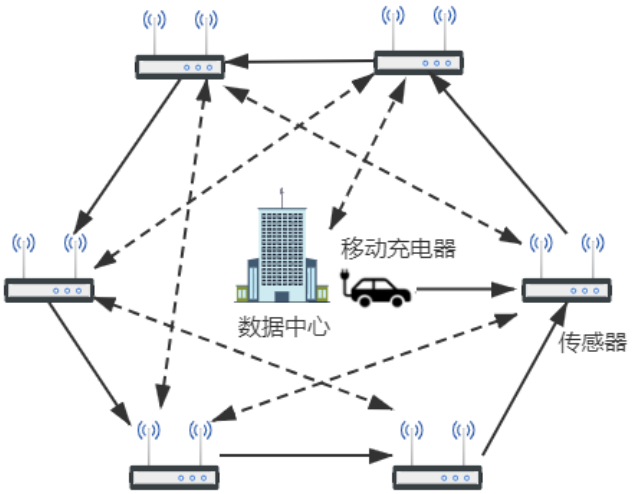

图 **1**　无线可充电网络示意图

1.2 待解决的问题

问题一: 对一般性的路径规划问题进行研究，给出 WRSN 充电规划模型和相应的求解算法. 算出充电器在路上的最小能量消耗以及相应路线.

问题二: 利用表二给出的 3 组参数，通过增加约束条件：充电器的充电，移动以及传感器的能量消耗限制，求出在指派单一充电车的情况下，系统内各传感器需要的最小电池容量。

问题三: 对于多充电车路径优化模型, 给出相应算法求解出系统最小总能量消耗，并求出每个传感器的最小电池容量。

二、 模型假设

1. 假设每次充电时，充电器一定会将传感器电充满再离开，防止出现充电不完全的状态。
2. 假设小车移动过程中呈匀速运动，忽略启动和停车的加速度影响。

三、 符号说明

符号	含义
C	移动充电器集合
O	数据中心的集合
M	表示传感器的集合
$m_0, m_{1..29}$	表示数据中心和传感器的元素
d_{ij}	表示节点 i 和节点 j 之间的距离
f	传感器电量的最低阈值
v	移动充电器移动速率
r	移动充电器的充电速率
T_1	小车在数据中心的休息时长
h_i^n	第 n 层注意层第 i 个网络节点
$p(\pi \mid s)$	第 π 个节点在路径 s 上被访问的概率

注：表中未说明的符号以首次出现处为准

四、 问题一模型的建立与求解

4.1 问题分析

问题给出了 WRSN 系统中传感器的经纬度坐标位置，需要首先进行转化以求出传感器，数据中心间的两两距离, 而在充电车对每个传感器充电是正常且必需的，那么要使得该充电器能量消耗最小，则必须使得充电器在路上的能量消耗最少即就是路径最少，在指派单一充电车的情况下，则该问题可转化为一个简单旅行商问题 (TSP 问题)，所以我们可以采用组合优化的思想，通过组合优化路径规划模型来求解该问题。

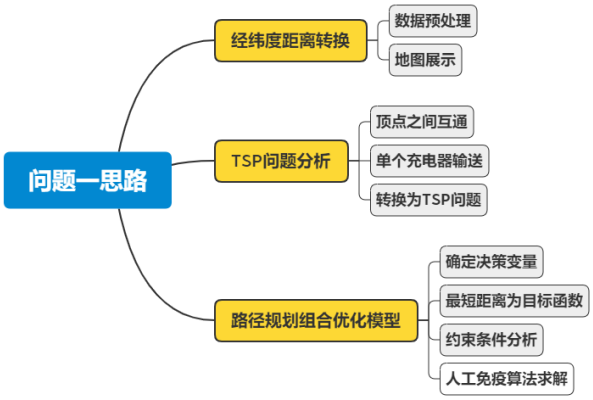

图 **2**　问题一思维导图

4.2 数据预处理

为了得到传感器，数据中心两两之间的路径距离，我们需要对经纬度进行转换.

地球上任意一点的地理坐标都可以用有序数对 (u,v),u,v 分别表示经度和维度。以地心 O 为坐标原点，赤道平面为 xOy 平面，0 度经线圈所在的平面为 xOZ 平面建立三维直角坐标系，则

$$\begin{cases} x = R\cos u\cos v. \\ y = R\sin u\cos v. \\ z = R\sin v. \end{cases} \tag{1}$$

其中，$R = 6370$ 为地球半径. 任意两点 $A(u_A,v_A),B(u_B,v_B)$ 间的实际距离为

$$d = R\arccos(\frac{OA\times OB}{|OA|\times|OB|}). \tag{2}$$

综合 (1)(2) 式得

$$d = R\arccos\left[\cos(u_A-u_B)\cos v_A\cos v_B+\sin v_A\sin v_B\right]. \tag{3}$$

4.3 单一充电器路径规划组合优化模型

在题中要求无线可充电传感器网络 WRSN 需要外界传导能量以维持运行，为此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，由于传感器之间，传感器与数据中心之间都存在一条通路，而且只有一个单一充电器供能，所以该问题可以转换成一个旅行商问题，利用组合优化的方法来解决。

图 3　各节点经纬度所在原始区域地图

4.3.1 决策变量

设数据中心为 O 点集合内包含 $\{m_0\}$, 充电车集合 $C = \{c_1\}$, 接收器集合 $M = \{m_1, m_2, m_3....m_{29}\}$. 这里我们引入 01 变量.

$$x_{ij} = \begin{cases} 0, & m_i \text{ 和 } m_j \text{ 存在路径.} \\ 1, & m_i \text{ 和 } m_j \text{ 不存在路径.} \end{cases} \tag{4}$$

4.3.2 目标函数

充电车从数据中心出发，依照一定顺序经过所有接收点，任意两接收点有一确定的道路连通. 我们通过 (3) 式求出接收器两两之间的距离 d_{ij} 表示接收器 i 与 j 之间的距离,$(0 < i, j \leq n)$

$$d_{ij} = R \arccos\left[\cos(u_i - u_j)\cos v_i \cos v_j + \sin v_i \sin v_j\right]. \tag{5}$$

$$\min Length = \min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}. \tag{6}$$

4.3.3 约束条件

保证每个传感器节点后只有一个节点与其想连通，即节点出度为 1

$$\sum_{j=1}^{n} x_{ij} = 1, i = 1, 2...n. \tag{7}$$

保证每个传感器节点前只有一个节点与其想连通，即节点入度为 1

$$\sum_{i=1}^{n} x_{ij} = 1, i = 1, 2...n. \tag{8}$$

为防止产生多于 1 个的连通回路，且保证每个结点都将连通构成回路，引入该节点边数与传感器之间编号构成的联系。

$$m_i - m_j + nx_{ij} \leq n-1, 1 < i \neq j \leq n. \tag{9}$$

保证出去回到节点的边以后，所构成的路径边数满足树的边数构造即等于顶点数 n-1。

$$\sum_{i,j \in S} x_{ij} \leq |S| - 1, 2 \leq |S| \leq n-1, S \in (1, 2, ...n.). \tag{10}$$

4.3.4 模型汇总

综上所述，我们建立了以最短移动路径 $\min Length$ 为目标函数, 节点出度和入度为 1，回路限定等条件，建立了用于解决单一充电器路径规划旅行商问题的组合优化模型。

$$\min Length = \min \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}. \tag{11}$$

$$s.t. \begin{cases} \sum_{j=1}^{n} x_{ij} = 1, & i = 1, 2...n. \\ \sum_{i=1}^{n} x_{ij} = 1, & i = 1, 2...n. \\ \sum_{i,j \in S} x_{ij} \leq |S| - 1, & 2 \leq |S| \leq n-1, S \in (1, 2, ...n.). \\ m_i - m_j + nx_{ij} \leq n-1, 1 < i \neq j \leq n, & 1 < i \neq j \leq n. \\ x_{ij} = 0 \quad or \quad 1, & i, j = 1, ..n. \\ m_i \in O \cup M, & i = 1, ..n. \end{cases} \tag{12}$$

4.4 模型求解

4.4.1 经纬度距离矩阵计算

我们将该问题的经纬度利用地球半径和球面的计算公式求得了整个区域各顶点之间的距离矩阵 D。

$$D = \begin{bmatrix} d_{00} & \cdots & d_{0n} \\ \vdots & \ddots & \vdots \\ 0 & & d_{nn} \end{bmatrix}_{n \times n} \tag{13}$$

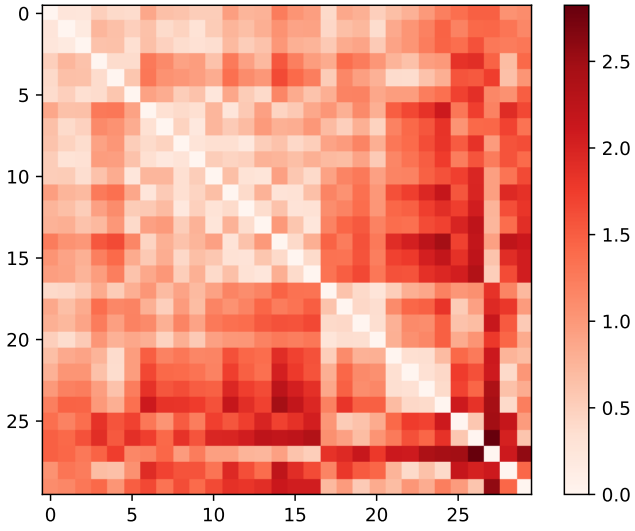

图 **4**　各节点构成的距离矩阵

4.4.2 人工免疫算法

抗体编码: 首先我们将 01 变量中都为 1 的路径按照顺序提取出来成为一组序列，将一个城市看成是一个氨基酸分子，将城市之间的边看作是链接氨基酸分子的肽键。一条遍历 n 个城市的路径则相当于一条包含 n 个不同氨基酸分子的肽链。抗体采用以城市的遍历次序进行编码，$L = \{m_0, m_i, m_j, ...m_k\}$，$m_i$ 的节点的编号，整体表示一条路径。**亲和力计算**在这里我们的亲和力就是我们的路径长度，亲和力越小，则说明该路径的长度越小。

$$A_\alpha = \frac{1}{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij}}. \tag{14}$$

移位逆序变异在第 i 次迭代中，对已有抗体进行采用随机选择片段然后位置进行反转或者交换部分氨基酸序列。

$$L_i = \{m_0^i, m_1^i, m_2^i, m_p^i, m_{p+1}^i ... m_{p+q}^i ... m_k^i\}.$$

反转序列 $\{m_p^i, m_{p+1}^i ... m_{p+q}^i\}$.

$$L_i = \{m_0^i, m_1^i, m_2^i, m_{p+q}^i, m_{p+1}^i .... m_p^i .. m_k^i\}.$$

交换节点 m_p^i, m_{p+q}^i

$$L_i = \{m_0^i, m_1^i, m_2^i, m_{p+q}^i, m_{p+1}^i ... m_p^i ... m_k^i\}.$$

精英库保留策略将所有序列的亲和力进行降序排列，选取其中亲和力较高的前 k 个氨基酸序列，然后将原问题中的库存进行合并与舍弃，然后对剩余的氨基酸序列进行刷新和合并，然后进入下一次的迭代。

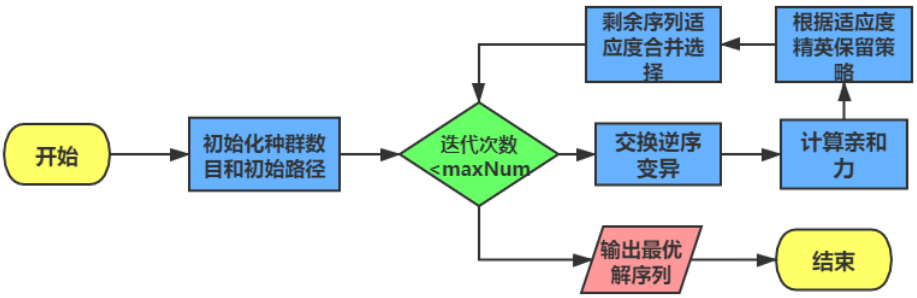

图 **5**　人工免疫算法求解流程

4.5 结果分析

4.5.1 结果展示

通过人工免疫算法的高效计算,我们得到了该问题的最短路径 $\min Length = 11.48276km$，移动充电器在地图上的真实路径规划如7所示。

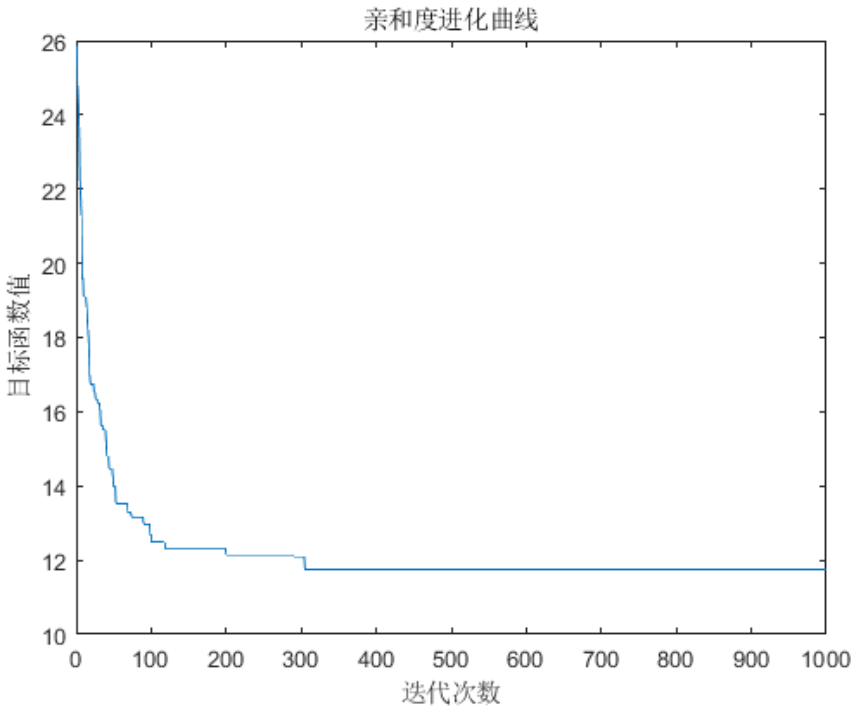

图 **6**　算法迭代过程

图 7　单一充电器路线规划图

表 1　问题一路径详细过程

编号详细路径
m_0(数据中心)$->m_2->m_1->m_9->m_7->m_6->m_{14}->m_{11}->m_8->m_{12}->$
$m_{15}->m_{27}->m_{16}->m_{13}->m_{10}->m_5->m_3->m_4->m_{22}->m_{28}->$
$m_{24}->m_{23}->m_{21}->m_{29}->m_{26}->m_{25}->m_{18}->m_{19}->m_{20}->m_{17}->m_0$

4.5.2 结果验证

由于该问题规模较小可通过 lingo 软件编程对该算法的结果进行验证，可知两者的路径是一样的，且最短路径结果一致。

表 2　结果对比

方法	路径长度	运行时间
人工免疫算法	11.48km	3.5129s
lingo 运算	11.48km	41s

五、 问题二模型的建立与求解

5.1 问题分析

问题二要求我们在第一问求得的最短路径的基础上，算出个传感器携带的电池容量大小. 需考虑到传感器的工作能量损耗 P 和电量阈值 f 限制，充电小车的运行速度 v_c 以及充电速度 r 对整个无线传感器网络工作的影响.

问题二的思维导图如8 所示

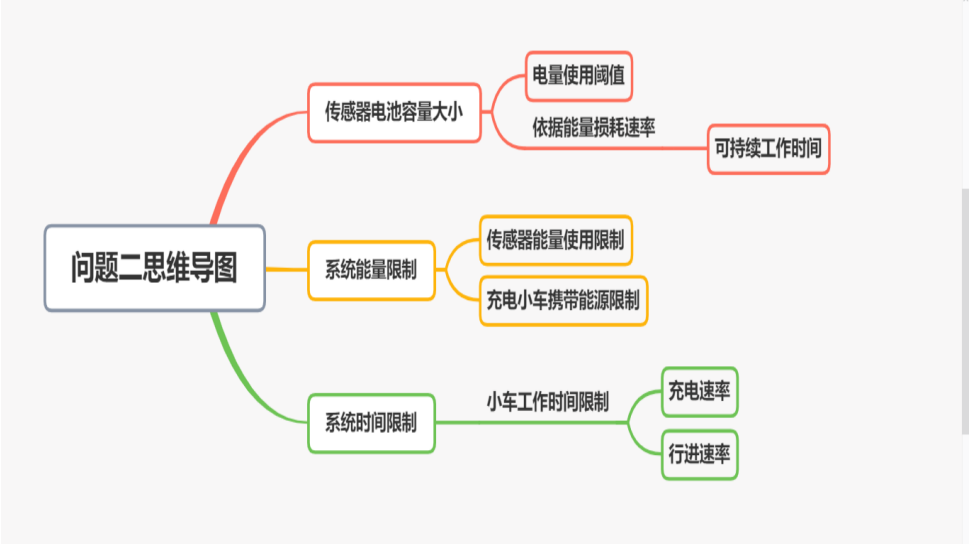

图 **8**　问题二思维导图

5.2 传感器容量设计模型的建立

传感器电池的设计容量大小会受到电量使用阈值和能量损耗速度的限制，单个传感器由于处在一个多节点构成的系统中，为此还要受到整个系统运行周期的限制，为此我们建立了一个多重线性约束模型.

5.2.1 决策变量

充电小车在系统内工作的时间内，可以认为控制其在数据中心的休息时间. 如果进行小车的轮换，则可以保证系统在任意时间内都有一辆小车正在运行. 如果在数据中心内对小车进行充电，则可以调整小车的休息时间.

$$T_1 = \begin{cases} 0, & \text{小车不进行休息.} \\ t, & \text{小车在数据中心修整 t 的时长.} \end{cases} \tag{15}$$

5.2.2 目标函数

根据小车运行一圈的时间与小车和传感器的设定参数，从时间和能量两个方面建立了能量和时间约束的传感器容量设计模型.

$$\min e_{nmax} = f_n + p_n \times T'. \tag{16}$$

e_{nmax} 表示的是第 n 个传感器的最大容量设计，f_n 表示的是第 n 个传感器工作的最低阈值 (mA/h)，p_n 是第 n 个传感器的消耗速率，$T' = T + T_1$ 表示该小车工作一圈的时间和休息的时间之和，代表小车的运行过程中一圈的时间周期。

5.2.3 约束条件

能量约束

无论是充电小车和传感器都会受到能源使用限制，传感器受到的限制来自使用电量的阈值限制和传感器能量消耗限制，小车则受到携带能量大小的限制.

$$\begin{cases} e_{ncu} \geq f_n, \quad n = 1, 2, ...29. \\ f_n = k e_{nmax}. \\ \sum_{i=1}^{n} e_{nmax} - e_{ncu} + E_c \leq E_{max}. \\ E_c = \sum_{i \neq j} (\frac{d_{ij}}{v_c}) \times g_c. \end{cases} \tag{17}$$

式中：e_{ncu} 表示编号为 n 的传感器现有电量值，f_n 为编号为 n 的传感器的能量阈值,k 为一比例系数，表示阈值与设计容量的比例.e_{nmax} 表示编号为 n 的传感器的能量设计容量.E_c 表示充电车在路上的能量消耗, 与充电车行进的时间有关，行进过程中由于动力消耗能量的功率为 g_c.

时间约束

传感器的有效使用时间 t_n 指的是传感器从充满电状态变为能量接近阈值状态所经过的时间，其大小需大于充电周期的时长 T', 而小车在系统内运行一周的时间取决于行进耗费的时间和为传感器充电耗费的时间.

$$\begin{cases} T' = T + T_1. \\ T = \sum_{i \neq j} (\frac{d_{ij}}{v_C}) + \sum_{i=1}^{n} \frac{e_{nmax} - e_{ncu}}{r}. \\ \min t_n \geq T'. \\ t_n = \frac{e_{nmax} - f_n}{p_n}. \end{cases} \tag{18}$$

式中: T_1 为充电车在数据中心充电时间,p_n 为编号为 n 的传感器的能量损耗速度. r 为充电车给传感器的充电速度.

5.2.4 模型汇总

综上所述，我们从时间和空间两个方向的约束, 通过查阅相关资料的参数设定，在问题一的路径规划基础上建立了一个传感器容量设定的模型。

$$\min e_{nmax} = f_n + p_n \times T'. \tag{19}$$

$$\begin{cases} e_{max} = \frac{p_n \times (T+T_1)}{1-K}. \\ T = \frac{\sum_{i=1}^{n} p_n T_1 + r \sum_{i \neq j} (\frac{d_{ij}}{v_c})}{r - \sum_{i=1}^{n} p_n}. \end{cases} \tag{20}$$

5.3 相关参数设置

为了真实反映该模型中移动充电器和传感器的能量和时间变化，我们需要对文中所提到的相关参数进行真实合理的设置，该通过查阅相关文献资料[1][2][3] 我们对系统内相关参数进行设置，具体参数范围如表3.

表 **3**　参数估计表

名称	参数范围
充电速率	3-5J/s
小车运行速度 v	3-5m/s
能量阈值系数 k	20%-30%
传感器能量消耗速度 p_n	3.5-7.8mA/h
传感器电压	10-24V

5.4 结果分析

5.4.1 结果展示

我们首先对所有参数指标取范围的平均值以估算传感器设计电池容量. 在充电速率 r 取 4 焦耳每秒, 小车运行速度 v_c 取 4m/s，能量阈值系数取 25%. 传感器电压取 17 伏特. 带入 (19) 式, 求解得:$T = 4.04$ 小时. 系统一个周期运行的时间 T' 为 5.04 小时. 如果需要统一传感器规格，则根据系统内电池容量要求最大的传感器选择电池容量设计大小，为 52.4 毫安时.

表 **4**　各个小车容量设计

编号	容量	编号	容量	编号	容量	编号	容量	编号	容量
1	36.3	7	43.0	13	43.7	19	37.0	25	50.4
2	52.4	8	30.9	14	30.2	20	30.2	26	28.9
3	30.2	9	30.2	15	25.5	21	37.0	27	24.1
4	36.7	10	37.0	16	30.2	22	30.2	28	43.0
5	24.2	11	30.2	17	25.5	23	23.5	29	36.3
6	30.2	12	49.7	18	30.2	24	37.0	/	/

5.4.2 参数分析

我们发现小车运行速度与传感器充电功率大小与传感器设计最小容量呈负相关，而小车的休息时间长短与传感器电压大小与传感器设计最小容量呈正相关. 从直观的角度分析，小车的前进速率越快，充电效率越高，每一圈运行的时间就越短，传感器电池容量要求就会越小. 而小车休息市场越长则是变相延长了传感器的可持续使用时间，传感器电压越高，释放的能力就越多，对传感器电池容量的要求就越高. 所以我们的结果是符合逻辑的. 我们画出了该系统中传感器电池设计的最小容量随 4 种参数数值变化的图像。

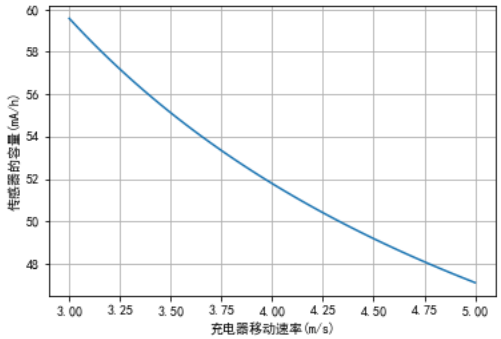

图 **9**　小车运行速度的影响

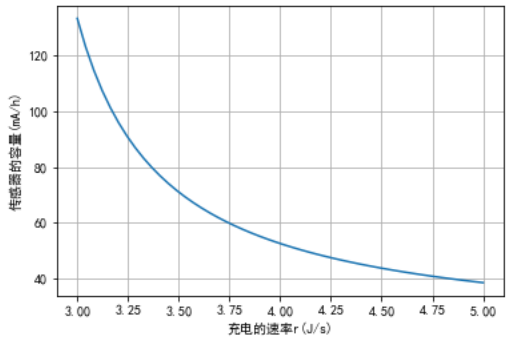

图 **10**　传感器充电功率的影响

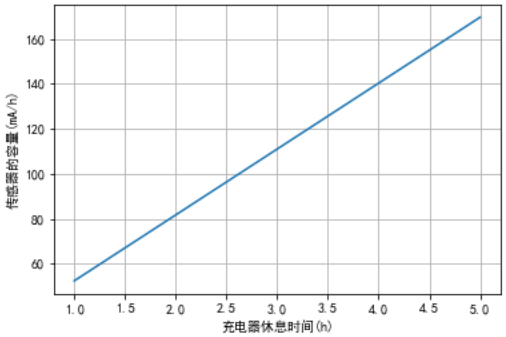

图 **11**　充电小车休息时间的影响

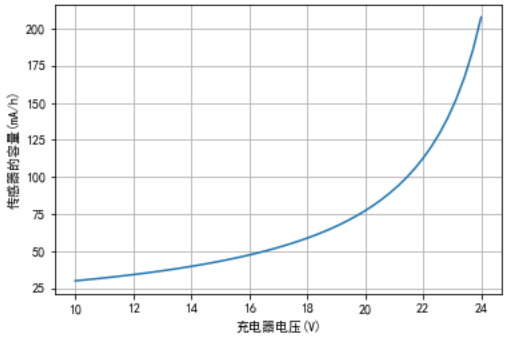

图 **12**　传感器电压的影响

对传感器设计电池容量大小影响最大的因素是小车的休息时间，小车休息时间每多增加一个小时，则会使传感器电池最小容量增加 29.2 毫安时. 相对于其他因素来说，小车运行速度对传感器电池容量大小的影响最小，我们认为这是因为在小车在路径上行驶的时间只占系统周期性运行时间的一小部分，而充电小车给传感器充电的时间则相对更长，图 7 和图 8 也验证了我们的猜想.

图 10 显示传感器的电压对传感器电池容量影响较大，要注意的是，传感器的电压常常是一个确定值，在传感器设计师就已经确定了. 所以为了控制电池容量，应尽可能提高传感器性能，使其在相对低压环境下工作.

六、 问题三模型的建立与求解

6.1 问题分析

问题三要求我们在问题一的单充电车路径规划的基础上, 思考为提高充电效率而派出 4 台移动充电车时，如何规划移动充电器的路径以使得移动充电器在路上的总能量消耗最少，即求出移动充电器形势的最短路径，这实际上是一个单起点的多旅行商问题，同时由于充电器仍需满足问题二中提到的能量约束和时间约束, 我们可以相应的求出各个路径上不同的传感器设计容量大小. 这里我们可以使用注意力机制下的图注意力神经网络对该问题进行求解。

6.2 多充电车旅行商图注意力神经网络

注意力模型是依附在编码器-解码器框架下如图**??**, 能够针对用户的输入，通过编码构造将整个图中节点的坐标输入其中，就可以将整个地区当做一个图形作为输入，然后利用编码器的学习参数，通过多层注意层和权重的计算，投射和压缩嵌入从而得到一组具有能反映真实图像的数据层，并通过解码和反馈训练得到多旅行商的训练结果。

6.2.1 编码器原理

我们使用的编码器类似于 Vaswani[4] 等人在 Transformer 体系结构中使用的编码器，但是我们不使用位置编码，产生的节点嵌入对输入的顺序是不变的. 输入二维特征 x_i, 编码器同过具有参数 W^X 和 b^x 的学习线性投影计算初始二维节点嵌入 $h_i^{(0)}$.

$$h_i^{(0)}. = W^X x_i + b^x. \tag{21}$$

嵌入后使用 N 个注意层进行更新，每个注意层由两个子层组成, 我们用 $h_i^{(\ell)}$ 表示由 ℓ 层产生的节点嵌入，编码器计算输入图的聚合嵌入 $\overline{h}^{(N)}$ 作为最终节点嵌入 $h_i^{(N)}$ 的平均值.

$$\overline{h}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^{(N)}. \tag{22}$$

节点嵌入 $h_i^{(N)}$ 和图嵌入 $\overline{h}^{(N)}$ 都被用作解码器的输入.

Attention layer　遵循 Transformer 架构如图13。每个注意层由两个子层组成: 一种多头注意层 (MHA)，执行节点和节点上完全连接的前馈层 (FF) 之间的消息传递。每个子层增加 skip-connection 和 batch nomrmlization。

$$\hat{\mathbf{h}}_i = \mathrm{BN}^{\ell}\left(\mathbf{h}_i^{(\ell-1)} + \mathrm{MHA}_i^{\ell}\left(\mathbf{h}_1^{(\ell-1)}, \ldots, \mathbf{h}_n^{(\ell-1)}\right)\right). \tag{23}$$

$$\mathbf{h}_i^{(\ell)} = \mathrm{BN}^{\ell}\left(\hat{\mathbf{h}}_i + \mathrm{FF}^{\ell}\left(\hat{\mathbf{h}}_i\right)\right). \tag{24}$$

每层的索引 ℓ 表示图层不共享参数。

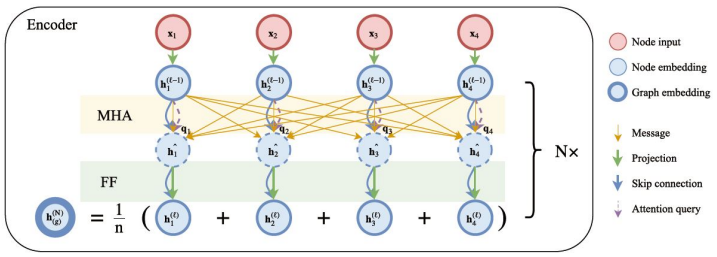

图 **13**　编码器示意图

6.2.2 解码器原理

解码器按照顺序解码，按照时间步长 $t \in 1, \ldots n$, 解码器基于编码器的嵌入输出节点 π_t，在步长 $t' < t$ 时输出 $\pi_{t'}$

内容嵌入　解码器在时间 t 的内容来自编码器和输出的内容. 对于 TSP 问题，它由前一个节点 π_{t-1} 和第一个节点 π_1 的嵌入构成. 对于 $t = 1$ 时，我们使用学习的 d_h 维参

数 v^l 和 V^f 作为输入占位符：

$$\mathbf{h}_{(c)}^{(N)} = \begin{cases} \left[\overline{\mathbf{h}}^{(N)}, \mathbf{h}_{\pi_{t-1}}^{(N)}, \mathbf{h}_{\pi_1}^{(N)}\right], & t > 1. \\ \left[\overline{\mathbf{h}}^{(N)}, \mathbf{v}^1, \mathbf{v}^f\right], & t = 1. \end{cases} \tag{25}$$

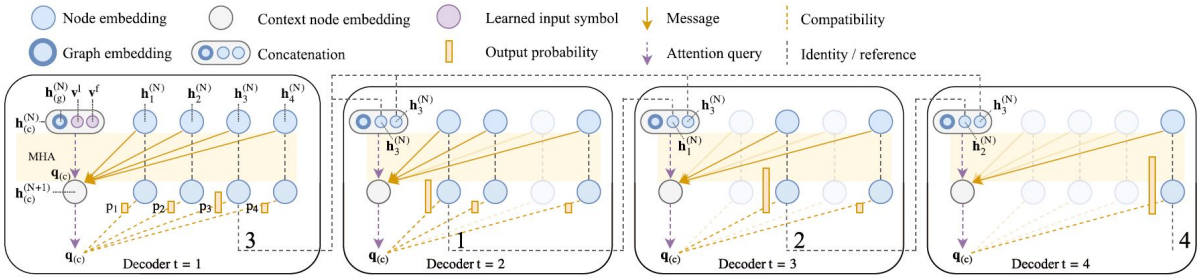

图 **14**　解码器示意图

根据内容嵌入计算访问的各个节点成为要访问的下一节点的的概率分布。

$$q(c) = W^Q h_{(c)}, \quad k_i = W^K h_i, \quad v_i = W^V h_i. \tag{26}$$

对数概率　在时间 t 不能访问的节点，也就是以及访问过的节点，我们通过遮罩赋值为 $-\infty$

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{q_{(c)}^T k_j}{\sqrt{d_k}}\right), & \text{if } j \neq \pi_{t'} \quad \forall t' < t. \\ -\infty, & \text{其他.} \end{cases} \tag{27}$$

对所有节点的的 $u_{(c)j}$ 通过 softmax 层，如图15，变换为访问的概率值。

$$p_i = p_\theta\left(\pi_t = i \mid s, \pi_{1:t-1}\right) = \frac{e^{u_{(c)t}}}{\sum_j e^{u_{(c)j}}}. \tag{28}$$

网络的搭建到此结束，我们通过编码器编码场上节点的位置信息，解码器考虑以访问

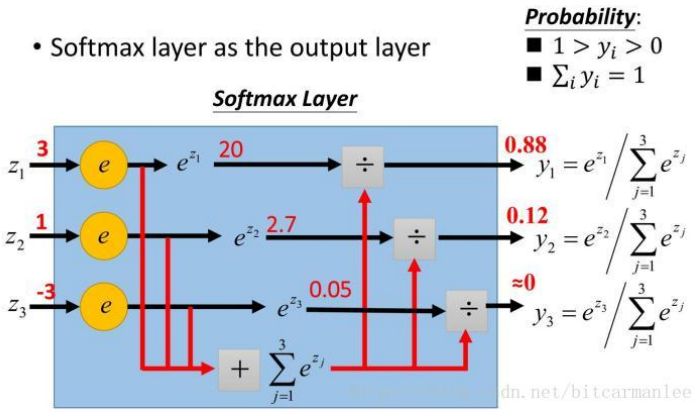

图 **15**　**softmax** 示意图

的节点进行内容嵌入，借鉴自然语言处理的注意力机制进行序列式的策略（下一步选取哪个节点访问）输出，最好通过 softmax 层将模型输出归一化为概率值。

6.2.3 最佳路径策略分布概率

我们以及基于注意的编解码器模型给出策略的计算方法。整个问题实例 s 的解是一系列策略 $p(\pi|s)$ 的序列，**模型的输出定义了一个所有可能策略的概率分布。**

$$p_\theta(\pi \mid s) = \prod_{t=1}^{n} p_\theta\left(\pi_t \mid s, \pi_{1:t-1}\right). \tag{29}$$

其中模型的所有参数总称为 θ。模型的训练过程需要对策略的质量进行评价，我们通过损失函数 $L(\pi)$ 来描述策略的质量。$L(\pi)$ 定义为路径的总长度。但是不同的问题实例 s 的难度不同，为了使得模型对参数的更新不受问题实例的难易程度影响，我们需要一个基准线。在本模型中，我们使用贪婪策略为基准线

$$\nabla L(\theta \mid s) = E_{p_\theta(\pi|s)}\left[(L(\pi) - b(s))\nabla \log p_\theta(\pi \mid s)\right]. \tag{30}$$

基线 $b(s)$ 的目标是估计实例的难度，这样它就可以联合策略的损失 $L(\pi)$ 共同评价为神经网络提供梯度信息。$b(s)$ 可以有效降低梯度方差，从而提高学习速度。优化器算法方面，我们使用 Adam 优化器对神经网络参数进行更新。

6.2.4 移动充电器路径均衡度

由于小车所带的充电能力有限，且为了保证每条路径的负载均衡，我们使用的神经网络训练过程中给定了移动充电器携带的最大充电量 D，传感器节点数量 $n_M = 29$，充电车数量 $n_C = 4$，为了使得 4 个充电车每次经过的节点尽量均衡，我们可以利用载重量约束，给定每个节点一个合适的负载。考虑到使负载均衡，我们给所有的传感器节点确定相同的 δ。δ 的值确定如下：

$$\delta = tolerance\frac{n_C D}{n_M}. \tag{31}$$

其中 $tolerance$ 是一个 0-1 之间的参数，表示对负载不均衡的容忍度。$w(R_i)$ 表示第 i 条路径的路径长度。

$$tolerance = \frac{\max_{i,j} |w(R_i) - w(R_j)|}{\max_j w(R_i)}. \tag{32}$$

6.3 模型求解

我们使用的 cvrp_50 网络最大支持求解 50 个传感器的车辆路径问题。采用如表5所示的超参数训练了 100 个 epoch。

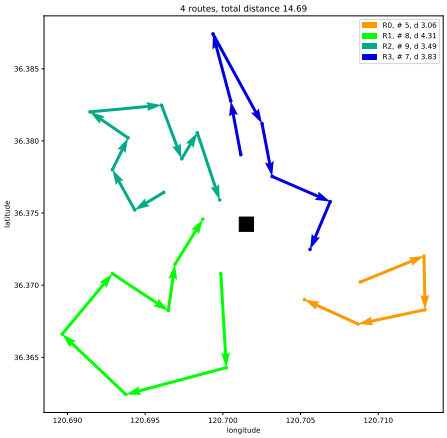

图 **16**　平面路径示意

图 **17**　真实地图路径示意

表 **5**　参数设定

批量大小	编码器层数	model 学习率	critic 学习率	梯度修剪阈值
512	3	0.0001	0.0002	1

模型输出的 4 辆充电车分别的路径长度和总长度间表6所示。在基于第二问中得到的平均参数数据的情况下，我们求得了 4 条路径上传感器设计电池容量的最小值，如表7

表 **6**　**4** 辆充电车的路径长度和总长度

R0	R1	R2	R3	总长
3.06km	4.31km	3.49km	3.83km	14.69km

表 7　4 条路径上传感器电池容量

编号	小车运行周期	传感器电池容量
1	1.36h	13.6 毫安时
2	1.58h	15.8 毫安时
3	1.66h	17.3 毫安时
4	1.48h	12.8 毫安时

6.4 灵敏度分析

我们通过调节车辆路径的负载均衡参数 δ 来观察我们该模型中路径总长度的相关变化。可见路径的总长度随着负载 δ 的增大而出现增长，同时小车的数量则敏感依赖 δ。

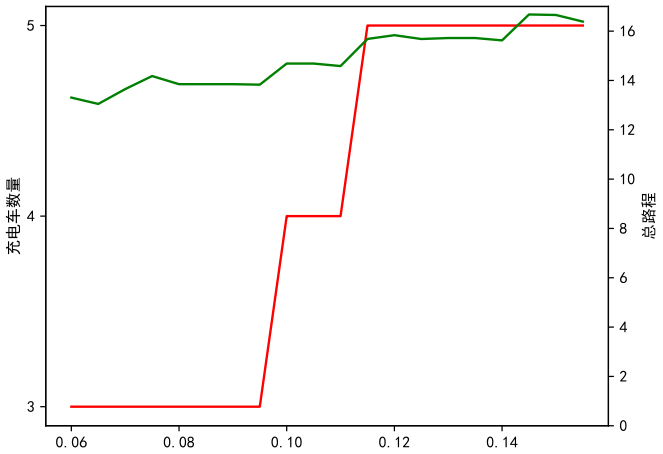

图 18　车辆数目和总路程对 δ 的灵敏度

这与最大携带充电限制的车辆路径问题的特性是相关的，小车数量是给定总载重量 D 和节点的需求量后的输出值。我们调节利用 δ 使得在需要 4 辆充电车的情况下达到最佳负载均衡。

七、 模型评价

7.1 模型的优点

1. 神经网络推理速度快，在节点规模小于模型输入参数的维度时，时间复杂度为 $O(n)$.

2. 模型的可推广性强，可以解决车旅行商问题 (TSP)、辆路径问题 (VRP)、定向问题 (OP) 和随机问题变体的奖品收集 TSP (PCTSP).
3. 注意力机制可以利用人类视觉机制进行直观解释。例如，我们的视觉系统倾向于关注图像中辅助判断的部分信息，并忽略掉不相关的信息.

7.2 模型的缺点

1. 由于模型的最大输入值在训练时就已经确定。所以当节点规模扩大到超出模型输入参数的维度，则需要训练新的模型。

7.3 改进与展望

在考虑相关参数大小对传感器容量设计大小的影响是，只做了电池大小受单个参数数值变化影响的曲线，可以继续做两个甚至个多因素同时变化对电池设计大小的影响，在电池大小设计为一定值后，可以同时调整多个参数，如小车休息时间和充电速度，使得系统的整体运转情况维持原状. 其他参数不变充电速度 4j/s 和电压 17 伏特在加快充电速度到 6j 每秒后电压可以相应减少到 15.6 伏特，电池容量大小可保持不变

参考文献

[1] 崔梦瑶. 可充电无线传感器网络的充电规划设计 [D]. 吉林大学,2018:45-46.

[2] 吕杨. 无线可充电传感器网络充电规划研究 [D]. 吉林大学,2017:34-35.

[3] 俞立春. 智能电网中无线可充电传感器网络充电规划研究 [D]. 上海电机学院,2019:32-35.

[4] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks[J]. In Advances in NeuralInformation Processing Systems, pp. 2692–2700, 2015.

[5] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorialoptimization algorithms over graphs[J].In Advances in Neural Information Processing Systems 30,pp. 6348–6358, 2017.

[6] 胡纯德, 祝延军, 高随祥. 基于人工免疫算法和蚁群算法求解旅行商问题 [J]. 计算机工程与应用,2004(34):60-63.

[7] 陈建锋, 郭子龙, 白林柱, 汪承杰, 郭旻昊, 赵伟. 基于人工免疫算法和模拟退火 TSP 的城市物流空中配送模式优化研究 [J]. 河南科技,2018(05):15-18.

附录 A 代码

A.1 python 源程序

A.2 functions.py

```
import warnings

import torch
import numpy as np
import os
import json
from tqdm import tqdm
from multiprocessing.dummy import Pool as ThreadPool
from multiprocessing import Pool
import torch.nn.functional as F


def load_problem(name):
    from problems import TSP, CVRP, SDVRP, OP, PCTSPDet, PCTSPStoch
    problem = {
        'tsp': TSP,
        'cvrp': CVRP,
        'sdvrp': SDVRP,
        'op': OP,
        'pctsp_det': PCTSPDet,
        'pctsp_stoch': PCTSPStoch,
    }.get(name, None)
    assert problem is not None, "Currently unsupported problem: {}!".format(name)
    return problem


def torch_load_cpu(load_path):
    return torch.load(load_path, map_location=lambda storage, loc: storage) # Load on CPU


def move_to(var, device):
    if isinstance(var, dict):
        return {k: move_to(v, device) for k, v in var.items()}
    return var.to(device)


def _load_model_file(load_path, model):
    """Loads the model with parameters from the file and returns optimizer state dict if it is
        in the file"""
```

```
    # Load the model parameters from a saved state
    load_optimizer_state_dict = None
    print(' [*] Loading model from {}'.format(load_path))

    load_data = torch.load(
        os.path.join(
            os.getcwd(),
            load_path
        ), map_location=lambda storage, loc: storage)

    if isinstance(load_data, dict):
        load_optimizer_state_dict = load_data.get('optimizer', None)
        load_model_state_dict = load_data.get('model', load_data)
    else:
        load_model_state_dict = load_data.state_dict()

    state_dict = model.state_dict()

    state_dict.update(load_model_state_dict)

    model.load_state_dict(state_dict)

    return model, load_optimizer_state_dict


def load_args(filename):
    with open(filename, 'r') as f:
        args = json.load(f)

    # Backwards compatibility
    if 'data_distribution' not in args:
        args['data_distribution'] = None
        probl, *dist = args['problem'].split("_")
        if probl == "op":
            args['problem'] = probl
            args['data_distribution'] = dist[0]
    return args


def load_model(path, epoch=None):
    from nets.attention_model import AttentionModel
    from nets.pointer_network import PointerNetwork

    if os.path.isfile(path):
        model_filename = path
        path = os.path.dirname(model_filename)
    elif os.path.isdir(path):
```

```
        if epoch is None:
            epoch = max(
                int(os.path.splitext(filename)[0].split("-")[1])
                for filename in os.listdir(path)
                if os.path.splitext(filename)[1] == '.pt'
            )
        model_filename = os.path.join(path, 'epoch-{}.pt'.format(epoch))
    else:
        assert False, "{} is not a valid directory or file".format(path)

    args = load_args(os.path.join(path, 'args.json'))

    problem = load_problem(args['problem'])

    model_class = {
        'attention': AttentionModel,
        'pointer': PointerNetwork
    }.get(args.get('model', 'attention'), None)
    assert model_class is not None, "Unknown model: {}".format(model_class)

    model = model_class(
        args['embedding_dim'],
        args['hidden_dim'],
        problem,
        n_encode_layers=args['n_encode_layers'],
        mask_inner=True,
        mask_logits=True,
        normalization=args['normalization'],
        tanh_clipping=args['tanh_clipping'],
        checkpoint_encoder=args.get('checkpoint_encoder', False),
        shrink_size=args.get('shrink_size', None)
    )
    # Overwrite model parameters by parameters to load
    load_data = torch_load_cpu(model_filename)
    model.load_state_dict({**model.state_dict(), **load_data.get('model', {})})

    model, *_ = _load_model_file(model_filename, model)

    model.eval() # Put in eval mode

    return model, args


def parse_softmax_temperature(raw_temp):
    # Load from file
    if os.path.isfile(raw_temp):
        return np.loadtxt(raw_temp)[-1, 0]
```

```
    return float(raw_temp)


def run_all_in_pool(func, directory, dataset, opts, use_multiprocessing=True):
    # # Test
    # res = func((directory, 'test', *dataset[0]))
    # return [res]

    num_cpus = os.cpu_count() if opts.cpus is None else opts.cpus

    w = len(str(len(dataset) - 1))
    offset = getattr(opts, 'offset', None)
    if offset is None:
        offset = 0
    ds = dataset[offset:(offset + opts.n if opts.n is not None else len(dataset))]
    pool_cls = (Pool if use_multiprocessing and num_cpus > 1 else ThreadPool)
    with pool_cls(num_cpus) as pool:
        results = list(tqdm(pool.imap(
            func,
            [
                (
                    directory,
                    str(i + offset).zfill(w),
                    *problem
                )
                for i, problem in enumerate(ds)
            ]
        ), total=len(ds), mininterval=opts.progress_bar_mininterval))

    failed = [str(i + offset) for i, res in enumerate(results) if res is None]
    assert len(failed) == 0, "Some instances failed: {}".format(" ".join(failed))
    return results, num_cpus


def do_batch_rep(v, n):
    if isinstance(v, dict):
        return {k: do_batch_rep(v_, n) for k, v_ in v.items()}
    elif isinstance(v, list):
        return [do_batch_rep(v_, n) for v_ in v]
    elif isinstance(v, tuple):
        return tuple(do_batch_rep(v_, n) for v_ in v)

    return v[None, ...].expand(n, *v.size()).contiguous().view(-1, *v.size()[1:])


def sample_many(inner_func, get_cost_func, input, batch_rep=1, iter_rep=1):
    """
```

```
    :param input: (batch_size, graph_size, node_dim) input node features
    :return:
    """
    input = do_batch_rep(input, batch_rep)

    costs = []
    pis = []
    for i in range(iter_rep):
        _log_p, pi = inner_func(input)
        # pi.view(-1, batch_rep, pi.size(-1))
        cost, mask = get_cost_func(input, pi)

        costs.append(cost.view(batch_rep, -1).t())
        pis.append(pi.view(batch_rep, -1, pi.size(-1)).transpose(0, 1))

    max_length = max(pi.size(-1) for pi in pis)
    # (batch_size * batch_rep, iter_rep, max_length) => (batch_size, batch_rep * iter_rep,
        max_length)
    pis = torch.cat(
        [F.pad(pi, (0, max_length - pi.size(-1))) for pi in pis],
        1
    )  # .view(embeddings.size(0), batch_rep * iter_rep, max_length)
    costs = torch.cat(costs, 1)

    # (batch_size)
    mincosts, argmincosts = costs.min(-1)
    # (batch_size, minlength)
    minpis = pis[torch.arange(pis.size(0), out=argmincosts.new()), argmincosts]

    return minpis, mincosts
```

A.3 main.py

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import os
import numpy as np
import torch
from torch.utils.data import DataLoader
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

```
import matplotlib
from matplotlib import pyplot as plt
from matplotlib.collections import PatchCollection
from matplotlib.patches import Rectangle
from matplotlib.backends.backend_pdf import PdfPages
matplotlib.rcParams["font.sans-serif"] = ["SimHei"]
from geopy.distance import geodesic, great_circle


from utils import load_model
from problems import CVRP



# Code inspired by Google OR Tools plot:
#
    https://github.com/google/or-tools/blob/fb12c5ded7423d524fc6c95656a9bdc290a81d4d/examples/python/cvrptw_plo

def discrete_cmap(N, base_cmap=None):
    """
    Create an N-bin discrete colormap from the specified input map
    """
    # Note that if base_cmap is a string or None, you can simply do
    #    return plt.cm.get_cmap(base_cmap, N)
    # The following works for string, None, or a colormap instance:

    base = plt.cm.get_cmap(base_cmap)
    color_list = base(np.linspace(0, 1, N))
    cmap_name = base.name + str(N)
    return base.from_list(cmap_name, color_list, N)

def plot_vehicle_routes(data, route, ax1, markersize=5, visualize_demands=False,
    demand_scale=1, round_demand=False):
    """
    Plot the vehicle routes on matplotlib axis ax1.
    """

    # route is one sequence, separating different routes with 0 (depot)
    routes = [r[r!=0] for r in np.split(route.cpu().numpy(), np.where(route==0)[0]) if (r !=
        0).any()]
    depot = data['depot'].cpu().numpy()
    locs = data['loc'].cpu().numpy()
    demands = data['demand'].cpu().numpy() * demand_scale
    capacity = demand_scale # Capacity is always 1

    x_dep, y_dep = depot
    ax1.plot(x_dep, y_dep, 'sk', markersize=markersize*4)
```

```
ax1.set_xlabel("longitude")
ax1.set_ylabel("latitude")

legend = ax1.legend(loc='upper center')

cmap = discrete_cmap(len(routes) + 2, 'nipy_spectral')
dem_rects = []
used_rects = []
cap_rects = []
qvs = []
total_dist = 0
for veh_number, r in enumerate(routes):
    color = cmap(len(routes) - veh_number) # Invert to have in rainbow order

    route_demands = demands[r - 1]
    coords = locs[r - 1, :]
    xs, ys = coords.transpose()

    total_route_demand = sum(route_demands)
    #assert total_route_demand <= capacity
    if not visualize_demands:
        ax1.plot(xs, ys, 'o', mfc=color, markersize=markersize, markeredgewidth=0.0)

    dist = 0
    x_prev, y_prev = x_dep, y_dep
    cum_demand = 0
    for (x, y), d in zip(coords, route_demands):
        dist += great_circle((y, x), (y_prev, x_prev)).km
        # dist += np.sqrt((x - x_prev) ** 2 + (y - y_prev) ** 2)

        cap_rects.append(Rectangle((x, y), 0.01, 0.1))
        used_rects.append(Rectangle((x, y), 0.01, 0.1 * total_route_demand / capacity))
        dem_rects.append(Rectangle((x, y + 0.1 * cum_demand / capacity), 0.01, 0.1 * d /
            capacity))

        x_prev, y_prev = x, y
        cum_demand += d

    dist += great_circle((y_dep, x_dep), (y_prev, x_prev)).km
    # dist += np.sqrt((x_dep - x_prev) ** 2 + (y_dep - y_prev) ** 2)
    total_dist += dist
    qv = ax1.quiver(
        xs[:-1],
        ys[:-1],
        xs[1:] - xs[:-1],
        ys[1:] - ys[:-1],
        scale_units='xy',
```

```
            angles='xy',
            scale=1,
            color=color,
            label='R{}, # {}, d {:.2f}'.format(
                veh_number,
                len(r),
                # int(total_route_demand) if round_demand else total_route_demand,
                # int(capacity) if round_demand else capacity,
                dist
            )
        )

        qvs.append(qv)

    ax1.set_title('{} routes, total distance {:.2f}'.format(len(routes), total_dist))
    ax1.legend(handles=qvs)

    pc_cap = PatchCollection(cap_rects, facecolor='whitesmoke', alpha=1.0,
        edgecolor='lightgray')
    pc_used = PatchCollection(used_rects, facecolor='lightgray', alpha=1.0,
        edgecolor='lightgray')
    pc_dem = PatchCollection(dem_rects, facecolor='black', alpha=1.0, edgecolor='black')

    if visualize_demands:
        ax1.add_collection(pc_cap)
        ax1.add_collection(pc_used)
        ax1.add_collection(pc_dem)
    return len(routes), total_dist

def test_custom_data(filename, savename=None, demand=.1):
    assert os.path.splitext(filename)[-1] == '.xlsx'
    df = pd.read_excel(filename)
    normalization = MinMaxScaler()
    dfDescribe = df.describe().loc[["min", "max"], ["longitude", "latitude"]]
    dfDescribe = dfDescribe.T
    dfDescribe["span"] = dfDescribe["max"] - dfDescribe["min"]
    df = normalization.fit_transform(df[["longitude", "latitude"]])
    model, _ = load_model('pretrained/cvrp_50/')
    torch.manual_seed(1234)

    dataset = CVRP.make_custom_dataset(df, demand=demand)
    # Need a dataloader to batch instances
    dataloader = DataLoader(dataset, batch_size=1000)

    # Make var works for dicts
    batch = next(iter(dataloader))
```

```
    # Run the model
    model.eval()
    model.set_decode_type('greedy')
    with torch.no_grad():
        length, log_p, tours = model(batch, return_pi=True)
    print([0] + list(i.item() for i in tours[0].data) + [0])

    # [[0, 22, 28, 24, 23, 21, 0],
    # [0, 17, 29, 26, 25, 18, 19, 20, 1, 0],
    # [0, 9, 7, 6, 11, 14, 15, 8, 12, 2, 0],
    # [0, 10, 16, 27, 13, 5, 3, 4, 0]]
    # Plot the results
    for i, (data, tour) in enumerate(zip(dataset, tours)):
        fig, ax = plt.subplots(figsize=(10, 10))
        data["loc"][:, 0] = dfDescribe.loc["longitude", "span"]*data["loc"][:, 0] +
            dfDescribe.loc["longitude", "min"]
        data["loc"][:, 1] = dfDescribe.loc["latitude", "span"]*data["loc"][:, 1] +
            dfDescribe.loc["latitude", "min"]
        data["depot"][0] = dfDescribe.loc["longitude", "span"]*data["depot"][0] +
            dfDescribe.loc["longitude", "min"]
        data["depot"][1] = dfDescribe.loc["latitude", "span"]*data["depot"][1] +
            dfDescribe.loc["latitude", "min"]
        numV, length = plot_vehicle_routes(data, tour, ax, visualize_demands=False,
            demand_scale=50, round_demand=True)

    if savename is not None:
        pdf = PdfPages("images//cvrp_"+savename+".pdf")
        pdf.savefig()
        pdf.close()
    # else:
    #    plt.show()
    plt.close()
    return numV, length

def plot_(x, y1, y2, savename=None):
    # plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.plot(x, y1, 'r', label="right");
    ax1.set_ylabel('充电车数量');
    ax1.set_yticks([3,4,5])
    ax2 = ax1.twinx() # this is the important function
    ax2.plot(x, y2, 'g', label="left")
    ax2.set_ylabel('总路程');
    ax2.set_xlabel('delta');
    ax2.set_ylim([0, 17]);
```

```
    if savename is not None:
        pdf = PdfPages("images//"+savename+".pdf")
        pdf.savefig()
        pdf.close()

if __name__ == '__main__':
    x = np.arange(0.06, 0.16, 0.005)
    ncs = []
    lens = []
    for d in x:
        nc, length = test_custom_data("../data/data.xlsx", demand=d,
            savename="result03_{:.2f}".format(d))
        ncs.append(nc)
        lens.append(length)
    plot_(x, ncs, lens, "sensitivity")
```

A.4 problemvrp.py

```
from torch.utils.data import Dataset
import torch
import numpy as np
import os
import pickle

from problems.vrp.state_cvrp import StateCVRP
from problems.vrp.state_sdvrp import StateSDVRP
from utils.beam_search import beam_search


class CVRP(object):

    NAME = 'cvrp' # Capacitated Vehicle Routing Problem

    VEHICLE_CAPACITY = 1.0 # (w.l.o.g. vehicle capacity is 1, demands should be scaled)

    @staticmethod
    def get_costs(dataset, pi):
        batch_size, graph_size = dataset['demand'].size()
        # Check that tours are valid, i.e. contain 0 to n -1
        sorted_pi = pi.data.sort(1)[0]

        # Sorting it should give all zeros at front and then 1...n
        assert (
            torch.arange(1, graph_size + 1, out=pi.data.new()).view(1, -1).expand(batch_size,
                graph_size) ==
```

```
            sorted_pi[:, -graph_size:]
        ).all() and (sorted_pi[:, :-graph_size] == 0).all(), "Invalid tour"

        # Visiting depot resets capacity so we add demand = -capacity (we make sure it does not
            become negative)
        demand_with_depot = torch.cat(
            (
                torch.full_like(dataset['demand'][:, :1], -CVRP.VEHICLE_CAPACITY),
                dataset['demand']
            ),
            1
        )
        d = demand_with_depot.gather(1, pi)

        used_cap = torch.zeros_like(dataset['demand'][:, 0])
        for i in range(pi.size(1)):
            used_cap += d[:, i] # This will reset/make capacity negative if i == 0, e.g. depot
                visited
            # Cannot use less than 0
            used_cap[used_cap < 0] = 0
            assert (used_cap <= CVRP.VEHICLE_CAPACITY + 1e-5).all(), "Used more than capacity"

        # Gather dataset in order of tour
        loc_with_depot = torch.cat((dataset['depot'][:, None, :], dataset['loc']), 1)
        d = loc_with_depot.gather(1, pi[..., None].expand(*pi.size(), loc_with_depot.size(-1)))

        # Length is distance (L2-norm of difference) of each next location to its prev and of
            first and last to depot
        return (
            (d[:, 1:] - d[:, :-1]).norm(p=2, dim=2).sum(1)
            + (d[:, 0] - dataset['depot']).norm(p=2, dim=1) # Depot to first
            + (d[:, -1] - dataset['depot']).norm(p=2, dim=1) # Last to depot, will be 0 if depot
                is last
        ), None

    @staticmethod
    def make_dataset(*args, **kwargs):
        return VRPDataset(*args, **kwargs)

    @staticmethod
    def make_custom_dataset(*args, **kwargs):
        return CustomVRPDataset(*args, **kwargs)

    @staticmethod
    def make_state(*args, **kwargs):
        return StateCVRP.initialize(*args, **kwargs)
```

```
    @staticmethod
    def beam_search(input, beam_size, expand_size=None,
                    compress_mask=False, model=None, max_calc_batch_size=4096):

        assert model is not None, "Provide model"

        fixed = model.precompute_fixed(input)

        def propose_expansions(beam):
            return model.propose_expansions(
                beam, fixed, expand_size, normalize=True, max_calc_batch_size=max_calc_batch_size
            )

        state = CVRP.make_state(
            input, visited_dtype=torch.int64 if compress_mask else torch.uint8
        )

        return beam_search(state, beam_size, propose_expansions)


class SDVRP(object):

    NAME = 'sdvrp' # Split Delivery Vehicle Routing Problem

    VEHICLE_CAPACITY = 1.0 # (w.l.o.g. vehicle capacity is 1, demands should be scaled)

    @staticmethod
    def get_costs(dataset, pi):
        batch_size, graph_size = dataset['demand'].size()

        # Each node can be visited multiple times, but we always deliver as much demand as
            possible
        # We check that at the end all demand has been satisfied
        demands = torch.cat(
            (
                torch.full_like(dataset['demand'][:, :1], -SDVRP.VEHICLE_CAPACITY),
                dataset['demand']
            ),
            1
        )
        rng = torch.arange(batch_size, out=demands.data.new().long())
        used_cap = torch.zeros_like(dataset['demand'][:, 0])
        a_prev = None
        for a in pi.transpose(0, 1):
            assert a_prev is None or (demands[((a_prev == 0) & (a == 0)), :] == 0).all(), \
                "Cannot visit depot twice if any nonzero demand"
            d = torch.min(demands[rng, a], SDVRP.VEHICLE_CAPACITY - used_cap)
```

```
            demands[rng, a] -= d
            used_cap += d
            used_cap[a == 0] = 0
            a_prev = a
        assert (demands == 0).all(), "All demand must be satisfied"

        # Gather dataset in order of tour
        loc_with_depot = torch.cat((dataset['depot'][:, None, :], dataset['loc']), 1)
        d = loc_with_depot.gather(1, pi[..., None].expand(*pi.size(), loc_with_depot.size(-1)))

        # Length is distance (L2-norm of difference) of each next location to its prev and of
            first and last to depot
        return (
            (d[:, 1:] - d[:, :-1]).norm(p=2, dim=2).sum(1)
            + (d[:, 0] - dataset['depot']).norm(p=2, dim=1) # Depot to first
            + (d[:, -1] - dataset['depot']).norm(p=2, dim=1) # Last to depot, will be 0 if depot
                is last
        ), None

    @staticmethod
    def make_dataset(*args, **kwargs):
        return VRPDataset(*args, **kwargs)

    @staticmethod
    def make_state(*args, **kwargs):
        return StateSDVRP.initialize(*args, **kwargs)

    @staticmethod
    def beam_search(input, beam_size, expand_size=None,
                    compress_mask=False, model=None, max_calc_batch_size=4096):
        assert model is not None, "Provide model"
        assert not compress_mask, "SDVRP does not support compression of the mask"

        fixed = model.precompute_fixed(input)

        def propose_expansions(beam):
            return model.propose_expansions(
                beam, fixed, expand_size, normalize=True, max_calc_batch_size=max_calc_batch_size
            )

        state = SDVRP.make_state(input)

        return beam_search(state, beam_size, propose_expansions)


def make_instance(args):
    depot, loc, demand, capacity, *args = args
```

```
    grid_size = 1
    if len(args) > 0:
        depot_types, customer_types, grid_size = args
    return {
        'loc': torch.tensor(loc, dtype=torch.float) / grid_size,
        'demand': torch.tensor(demand, dtype=torch.float) / capacity,
        'depot': torch.tensor(depot, dtype=torch.float) / grid_size
    }


class VRPDataset(Dataset):

    def __init__(self, filename=None, size=50, num_samples=1000000, offset=0,
                 distribution=None):
        super(VRPDataset, self).__init__()

        self.data_set = []
        if filename is not None:
            assert os.path.splitext(filename)[1] == '.pkl'

            with open(filename, 'rb') as f:
                data = pickle.load(f)
            self.data = [make_instance(args) for args in data[offset:offset+num_samples]]

        else:

            # From VRP with RL paper https://arxiv.org/abs/1802.04240
            CAPACITIES = {
                10: 20.,
                20: 30.,
                50: 40.,
                100: 50.
            }

            self.data = [
                {
                    'loc': torch.FloatTensor(size, 2).uniform_(0, 1),
                    # Uniform 1 - 9, scaled by capacities
                    'demand': (torch.FloatTensor(size).uniform_(0, 9).int() + 1).float() /
                        CAPACITIES[size],
                    'depot': torch.FloatTensor(2).uniform_(0, 1)
                }
                for i in range(num_samples)
            ]

        self.size = len(self.data)
```

```
    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        return self.data[idx]


class CustomVRPDataset(Dataset):

    def __init__(self, df, demand):
        super(CustomVRPDataset, self).__init__()

        self.data_set = []
        size = len(df) - 1
        locArr = np.array(df)

        # From VRP with RL paper https://arxiv.org/abs/1802.04240
        CAPACITIES = {
            10: 20.,
            20: 30.,
            50: 40.,
            100: 50.
        }

        self.data = [
            {
                'loc': torch.FloatTensor(locArr[1:]),
                'demand': torch.FloatTensor(np.ones(size)*demand),
                'depot': torch.FloatTensor(locArr[0])
            }
        ]
        self.size = len(self.data)
        print("Custon data loaded")

    def __len__(self):
        return self.size

    def __getitem__(self, idx):
        return self.data[idx]

if __name__ == '__main__':
    d = CustomVRPDataset()
    print(d.data)
```