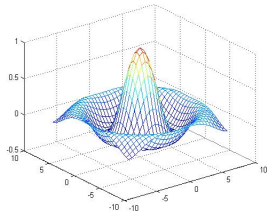
五 一 数 学 建 模 竞 赛



题 目：快递需求分析与数量预测

关键词：粒子群优化投影寻踪法 LSTM 神经网络 贪心算法 极大似然估计

摘 要：

为准确预测快递需求数量，优化快递公式布局仓库站点，节约其仓库的存储存储成本，并规划最佳的运输路线，本文建立了粒子群优化投影寻踪法的综合评价模型，基于 LSTM 神经网络的快递运输数量预测模型，以运输成本最低为目标的贪心算法。

对于问题一，为衡量站点城市重要程度的评价问题，本文使用基于粒子群算法优化的投影寻踪法，以收货量、发货量、发货城市数量、收货城市数量、发货量增幅标准差和收货量增幅标准差作为指标对站点城市的重要程度进行评估，其中收货量为所有站点城市快递收货量之和，发货量为所有站点城市快递发货量之和，最后不同站点城市进行打分，可得到排名前五的城市为 L,G,V,R,X。

对于问题二，先对附件数据进行预处理，将不同站点城市清洗出来，以此建立 LSTM 神经网络模型，将历史数据进行训练，以此预测 2019 年 4 月 18 日和 2019 年 4 月 19 日这两天所有站点城市对之间的快递运输数量，将每一天中所有站点城市对的快递运输数量进行求和，得出 2019 年 4 月 18 日的总快递数为 12243，2019 年 4 月 19 日的总快递数为 12234。

对于问题三，在问题二基础上增加新条件，将所给数据中的无法正常发货的数据用-1000 进行填充，若所预测的快递运输数量为零或负则说该点未能正常发货，为正则说明可正常发货，该路线快递运输数量即为此时的预测值。

对于问题四，需要对 2023 年 4 月 23 日到 2023 年 4 月 27 日的快递运输成本进行计算，考虑使用贪心算法，先将每个站点城市发送给另一站点城市的快递进行拆分，以小件快递形式进行运输。再从所有可达路径中选取至多 5 条路径最短的路径，对其进行加权求解，按照权重分配每个小件快递的数量，最后将通过各个节点到下一节点的快递进行积压，直到所有要通过当前节点的快递均到达，再统一发往下一节点，这即可求出所求那几天的快递最低运输成本，其分别对应为 3306.71，3543.07，3649.35，2895.3，2746.8。

对于问题五，对于固定需求由于其为固定的常数，故本文可使用部分统计描述量对其进行模型建立，经过对比分析，本文最终选用其原数据的均值减去标准差作为其固定需求的数学模型，对于非固定需求，由于其样本数据服从某种概率分布，故本文选用极大似然估计对其概率密度进行求解，然后使用连续型随机变量的方差求解公式对其进行求解。

一、 问题重述

1.1 问题背景

随着信息技术的发展以及互联网的普及，人们的生活方式、出行方式、购物方式等等都在发生着翻天覆地的变化。人们出行选择网络提前叫车来减少自己等待的时间；人们购物会选择“淘宝”、“亚马逊”、“天猫”等等网络交易平台来方便自己的生活。特别是在中国这种人口众多，地域广阔的国家，信息技术的发展和互联网普及的速度之快，引起了许多人的研究和关注。随着现代网络的发展，网络购物已经成为一种必不可少的消费方式，快递行业的需求也随之增加，现已经为我国的经济建设做出重大贡献。如何准确的预测快递需求数量对快递公司布局仓库站点、规划运输路线、节约存储成本等均有重大意义。

随着生活水平的逐渐提高，快递运输的需求只增不减。如何分析快递需求以合理安排快递运输，将会对网络购物和快递服务业的发展产生巨大影响。附件中包含发货日期、发货城市和收货城市。

1.2 问题重述

通过附件 1、附件 2、附件 3 中的快递运输数据，建立数学模型，完成以下问题。

1. 结合附件 1 中的快递运输数据，从收获量、发货量、快递数量增长/减少趋势、相关性分析等多角度考虑，建立数学模型，对各城市的重要程度进行综合排序，并给出重要程度排名前 5 的城市。
2. 使用附件 1 中数据建立数学模型，预测 2019 年 4 月 18 日和 2019 年 4 月 19 日各站点城市之间的快递运输数量和当日所有站点城市之间的总快递运输数

量。

3. 部分城市由于突发事件影响导致城市之间无法正常运输，利用附件 2 中数据建立数学模型，预测 2023 年 4 月 28 日和 2023 年 4 月 29 日可正常发货的站点城市对。判断所给表中指定城市对是否能正常发货，给出可正常发货其对应的快递运输数量。
4. 快递运输时每个城市对之间使用的路径数不超过 5 条。利用附件 1、附件 2、附件 3 中的数据建立数学模型，给出该快递公司的成本最低运输方案，并计算 2023 年 4 月 23-27 日的每日最低运输成本。
5. 快递由固定需求和非固定需求这两部分组成。利用附件 2 中数据建立数学模型，按季度估计固定需求常数，并验证其准确性。将指定季度、指定站点城市对的固定需求常数，当季度所有城市对的固定需求常数总和。
6. 给出非固定需求概率分布估计方法，将指定季度、指定城市对的非固定需求均值、标准差，当季度所有城市对的非固定需求均值总和、非固定需求标准差总和。

二、 问题分析

2.1 问题一的分析

通过分析附件 1 中的数据，选取合适的指标衡量各站点城市的重要程度，最终选取了收货量、发货量、发货城市数量、收货城市数量、发货量增幅标准差和收货量增幅标准差作为指标对站点城市的重要程度进行评估。基于粒子群算法改进的投影寻踪法，本文可以对附件一中 24 个站点城市进行综合评价，得出每个站点城市的重要程度得分，将选出排名前 5 的站点城市作为最重要的站点城市。

2.2 问题二的分析

问题二需要通过附件 1 数据，建立数学模型，预测 2019 年 4 月 18 日和 2019 年 4 月 19 日各“发货-收货”站点城市之间快递运输数量，以及当日所有“发货-收货”站点城市之间的总快递运输数量，根据附件一中的数据类型可知其为连续时间数据类型，对数据进行分析可知部分时间仍有缺失，本问对于缺失数据选择补 0，然后基于 LSTM 神经网络模型对时间序列进行预测。

2.3 问题三的分析

问题三在考虑受到突发事件影响，部分城市之间快递线路无法正常运输，导致站点城市之间无法正常发货或收货的情况下预测 2023 年 4 月 28 日和 2023 年 4 月 29 日“发货-收货”的站点城市是否能发货，以及正常发货的快递数量。对于其中无法正常发货的缺失值，本文采用其所在月份的平均值取负数进行填补，然后基于 LSTM 模型进行分析，对于预测值为负数的路线，则认为未正常发货，对于预测值为正的路线则认为正常发货且快递数量为该预测值。

2.4 问题四的分析

问题四需要对 2023 年 4 月 23 日到 2023 年 4 月 27 日的快递运输成本进行计算，考虑使用贪心算法，先将每个站点城市发送给另一站点城市的快递进行拆分，以小件快递形式进行运输。再从所有可达路径中选取至多 5 条路径最短的路径，对其进行加权求解，按照权重分配小件快递的数量，最后将通过各个节点到下一节点的快递进行积压，直到所有要通过当前节点的快递均到达，再统一发往下一节点，这即可求出所求那几天的快递最低运输成本。

2.5 问题五的分析

问题五需要对固定需求以及非固定需求进行建模，对于固定需求由于其为固定的常数，故本文可使用部分统计描述量对其进行模型建立，经过对比分析，本文最终选用其原数据的均值减去标准差作为其固定需求的数学模型，对于非固定需求，由于其样本数据服从某种概率分布，故本文选用极大似然估计对其概率密度进行求解，然后使用连续型随机变量的方差求解公式对其进行求解。

三、 模型假设

1. 假设快递运输的实际装货量允许超过额定装货量
2. 假设未正常发货的快递运输量为负
3. 假设固定需求为一确定常数，非固定需求服从某概率分布
4. 假设每件快递的重量为单位 1

四、 符号说明

符号	符号说明
$import_i$	*第 i 个站点城市的收货量*
$export_i$	*第 i 个站点城市的发货量*
$amount\_in_i$	*第 i 个站点城市的发货城市数量*
$amount\_out_i$	*第 i 个站点城市的收货城市数量*
$deliver\_out\_\sigma_i$	*发货量增幅标准差*
$deliver\_in\_\sigma_i$	*收货量增幅标准差*
$cost$	*铁路运输成本*
θ	*最大似然估计的估计量*

五、 模型的建立与求解

5.1 问题一模型的建立与求解

5.1.1 评价指标的构建

问题一需要针对附件 1 中的站点城市之间的快递运输数据进行处理，为选择出重要程度排名前 5 的站点城市名称。本文使用收货量、发货量、发货城市数量、收货城市数量、发货量增幅标准差和收货量增幅标准差作为指标对站点城市的重要程度进行评估。

1. 收货量：

当前站点城市会有来自其他已连通站点城市运输过来的快递，运送到当前站点城市的快递数量的总和定义为当前站点城市的收货量$import_i$，收货量越多，说明当前站点城市的重要程度越大，该站点城市就越重要。

$$import_i = \sum_{j=1}^{24} goods\_in_{i,j} \tag{1}$$

其中$goods_in_{i,j}$为 2018 年 4 月 19 日到 2019 年 4 月 17 日这段时间内，第 j 个城市输送给第 i 个城市的快递数量之和，若两个站点城市不连通，则快递数量为 0，$1 \leqslant i \leqslant 24, 1 \leqslant j \leqslant 24$。

2. 发货量：

当前站点城市会发送快递到其他已连通站点城市，运送到其他站点城市的快递数量的总和定义为当前站点城市的发货量$export_i$，发货量越多，说明当前站点城市的重要程度越大，该站点城市就越重要。

$$export_i = \sum_{j=1}^{24} goods\_out_{i,j} \tag{2}$$

其中$goods_out_{i,j}$为 2018 年 4 月 19 日到 2019 年 4 月 17 日这段时间内，第 i 个城市输送给第 j 个城市的快递数量之和，若两个站点城市不连通，则快递数量为 0，$1 \leqslant i \leqslant 24, 1 \leqslant j \leqslant 24$。

3. 发货城市数量：

当前站点城市与其他站点城市连通，且当前城市作为发货城市，则发货城市数量

为$amount_in_i$，发货城市数量越多，说明当前站点城市的重要程度越大，该站点城市就越重要。

$$amount\_in_i = \sum_{j=1}^{24} city_{i,j} \tag{3}$$

其中$city_{i,j}$为 2018 年 4 月 19 日到 2019 年 4 月 17 日这段时间内，第 i 个城市作为发货城市，是否可以运输快递到第 j 个站点城市，可以运输则$city_{i,j}=1$，否则$city_{i,j}=0$，$1 \leqslant i \leqslant 24, 1 \leqslant j \leqslant 24$。

4. 收货城市数量：

当前站点城市与其他站点城市连通，当前城市作为收货城市，则收货城市数量为$amount_out_i$，收货城市数量越多，说明当前站点城市的重要程度越大，该站点城市就越重要。

$$amount\_out_i = \sum_{j=1}^{24} city_{i,j} \tag{4}$$

其中$city_{i,j}$为 2018 年 4 月 19 日到 2019 年 4 月 17 日这段时间内，第 j 个城市作为发货城市，是否可以运输快递到第 i 个站点城市，可以运输则$city_{i,j}=1$，否则$city_{i,j}=0$，$1 \leqslant i \leqslant 24, 1 \leqslant j \leqslant 24$。

5. 发货量增幅标准差：

当前站点城市的相邻两天中，发送快递数量的差值为发货量的增幅$deliver_out$，具体计算公式如下：

$$deliver\_out = deliver\_out\_td - deliver\_out\_ysd \tag{5}$$

其中$deliver_out_td$表示当天的快递的发货量，$deliver_out_ysd$表示前一天的快递发货量。当前站点城市与其他连通城市全部的发货量增幅求出标准差，则发货量增幅标准差为$deliver_out_\sigma_i$，发货量增幅标准差越小，说明当前站点城市的发货量增幅越稳定，发货量稳定增加，城市快递的发展越好，该站点城市的重要程度越大。

$$deliver\_out\_\sigma_i = \sqrt{(x_1-\bar{x})^2+(x_2-\bar{x})^2+\cdots(x_n-\bar{x})^2} \tag{6}$$

其中$x_1, x_2, \cdots, x_n$为发货量的增幅，$\bar{x}$为发货量增幅的平均值，$1 \leqslant i \leqslant 24$。

6. 收货量增幅标准差：

当前站点城市的相邻两天中，接收快递数量的差值为收货量的增幅$deliver_in$，具体计算公式如下：

$$deliver\_in = deliver\_in\_td - deliver\_in\_ysd \tag{7}$$

其中$deliver_in_td$表示当天的快递的收货量，$deliver_in_ysd$表示前一天的快递收货量。当前站点城市与其他连通城市全部的收货量增幅求出标准差，则收货量增幅标准

差为$deliver_in_\sigma_i$，收货量增幅标准差越小，说明当前站点城市的收货量增幅越稳定，收货量稳定增加，城市快递的发展越好，该站点城市的重要程度越大。

$$deliver\_in\_\sigma_i = \sqrt{(x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots (x_n - \bar{x})^2} \tag{8}$$

其中$x_1, x_2, \cdots, x_n$为收货量的增幅，$\bar{x}$为收货量增幅的平均值，$1 \leqslant i \leqslant 24$。

5.1.2 基于投影寻踪法的站点城市重要程度评价模型

投影寻踪法[1]是处理和分析高维数据的一类统计方法，基本思想为将高维数据投影到低维子空间上，寻找出反映原高维数据的结构或特征的投影，以达到研究和分析高维数据的目的。这样得到的结果一定程度上可以解决因某项指标的得分导致评价结果严重改变的问题，从而提高了综合评价问题中各层次的分辨力和评价模型的精度。故投影寻踪法可对对各站点城市的重要程度进行更好综合评价，所得结果可更加可靠与准确。

（1）对呈负向性的指标进行正向化处理

所选的 6 个指标中发货量增幅标准差和收货量增幅标准差均为极小型指标，其余四个指标为极大型指标，对极小型指标进行正向化处理：

$$x_j = max(x_j) - x_j \tag{1}$$

其中x_j表示第 j 列指标值$(j \in \{0, 1, \cdots, 6\})$,$max(x_j)$表示第 j 列指标的最大值。

（2）标准化处理

由于所选取指标的单位不统一，故各指标除了数值影响之外也会有量纲的影响，为了消去量纲采用标准化处理。

$$z_{ij} = x_{ij} \Big/ \sqrt{\sum_{i=1}^{n} x_{ij}^2} \tag{2}$$

其中x_{ij}为第 i 个城市的第 j 列指标$(i \in \{0, 1, \cdots, 24\}, j \in \{0, 1, \cdots, 6\})$，$z_{ij}$表示标准化处理后的结果。

（3）线性投影

从不同方向去观察数据，寻找能最充分挖掘数据特征的最优投影方法。任意选取

若干投影方向$a=(a_1,a_2,\cdots,a_m)$计算投影指标的大小，最大指标的投影解即为最佳投影方向，则第 i 个样本在一维线性空间的投影特征值λ_i表示如下：

$$\lambda_i=\sum_{j=1}^{m}a_j x_{ij} \tag{3}$$

（4）构造投影指标函数

对于最优投影方向的定义，本文希望投影特征值$\lambda_i\,(i\in\{0,1,\cdots,6\})$的分布满足：局部投影点尽可能的密集；投影点团在整体上应当尽可能的散开。则构造出来的目标函数为：

$$max\ Q(a)=S_a D_a \tag{4}$$

其中S_a为投影特征值的标准差，D_a为投影特征值局部密度。其计算公式如下：

$$\begin{aligned} S_a&=\sqrt{\sum_{i=1}^{6}\left(z_i-\overline{z_\alpha}\right)/(n-1)}\\ D_a&=\sum_{i=1}^{n}\sum_{j=1}^{n}(R-r_\alpha)u(R-r_\alpha)\end{aligned} \tag{5}$$

其中r_α为投影特征值之间的距离，且$r_\alpha=|z_i-z_j|,(i,j\in[1,2,\cdots,n])$，$u(t)$为阶跃信号，且$u(t)=\begin{cases}0,t<0\\1,t\geqslant 0\end{cases}$，R 为估计局部散点密度的窗宽参数，按其宽度内至少包含一个散点的原则，应当有$max(r_\alpha)<R<2m,\,(i,k\in[1,2,\cdots,n])$。

综上所述得到针对问题一建立的投影寻踪法的非线性规划模型：

$$max\ Q(a)=S_a D_a$$

$$s.t.\begin{cases} S_a=\sqrt{\sum_{i=1}^{6}\left(z_i-\overline{z_\alpha}\right)/(n-1)}\\ u(t)=\begin{cases}0,t<0\\1,t\geqslant 0\end{cases}\\ D_a=\sum_{i=1}^{n}\sum_{j=1}^{n}(R-r_\alpha)u(R-r_\alpha)\\ r_\alpha=|z_i-z_j|\\ 0<a_j<1\\ \sum_{j=1}^{m}{a_j}^2=1\\ max(r_\alpha)<R<2m\end{cases} \tag{6}$$

5.1.3 粒子群优化投影寻踪法的求解

粒子群算法主要原理是通过自适应权重计算，并结合历史和当前最优个体位置，逐步逼近最优解。其核心思想是利用群体中个体对信息的共享使群体运动在解空间中由无序到有序的演化过程，从而获得问题可行解。相比较模拟退火这种随机模拟算法来说，粒子群算法可达到更高精度的寻优。使用粒子群算法优化投影寻踪法[2]可提高综合评价问题中各层次的分辨力和评价模型的精度，能加快求解速度，更好的找到评价问题的最优解。

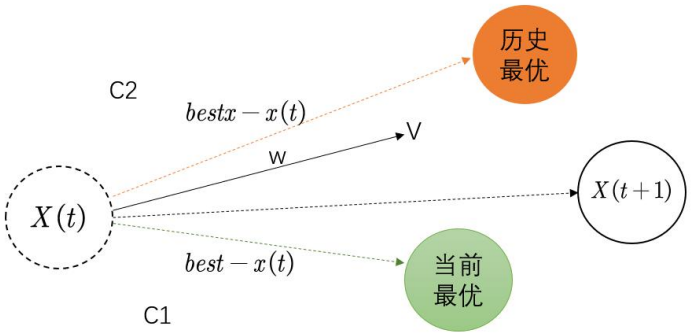


图 1 粒子群算法寻优示意图

取惯性权重的最小值w_{min}为 0.1，惯性权重的最大值w_{max}为 0.5，迭代 50 次粒子群算法优化函数，以 G 函数为目标函数，根据已有约束条件进行解的判定，通过观察粒子群算法对目标函数的迭代结果，此时算法已收敛，粒子群算法求解出的结果即为最优值。

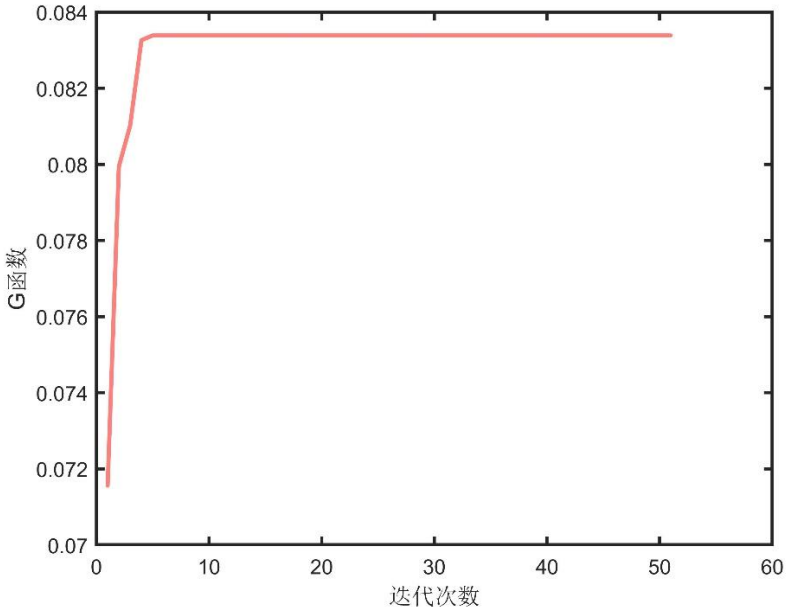


图 2 粒子群算法优化投影寻踪

迭代 50 次粒子群算法优化函数，得到最优投影方向向量为：

$$a=[0.073\ 0.247\ 0.219\ 0.269\ 0.089\ 0.102]$$

该方向投影指标量降序输出得到这 24 个站点城市的排名，完整的排序名单见附录，以下给出重要程度排名前五的站点城市名称（代码见附录）：

表 1 重要性排名前 5 的站点城市

排序	1	2	3	4	5
城市名称	L	G	V	R	X

5.2 问题二模型的建立与求解

5.2.1 数据预处理

附件 2 中城市名已用字母代替，且已剔除了 6 月、11 月、12 月的数据。通过日常经验的研究，快递行业可能会因为“6·18”、“双十一”、“双十二”等大型商家活动

的影响，导致快递数量在这三个月中大幅增加，会导致模型预测的准确性，故只考虑其余 9 个月的快递数量。

首先对附件一的数据进行分析，将单个城市发货-收货路线分离出来，再将其每个快递运输量按时间排列，对于一年的这 9 个月中某一天若没有快递，则将其进行补 0 处理。经过 Excel 对附件数据进行处理，可知共有 91 条线路，画出每一条线路从 2018 年 4 月 19 日到 2019 年 4 月 17 日的快递运输数量的折线图。通过观察图像，可知每条线路在某段时间内的快递数量变化趋势基本类似，且在 2018 年到 2019 年这段时间内均具有周期性。

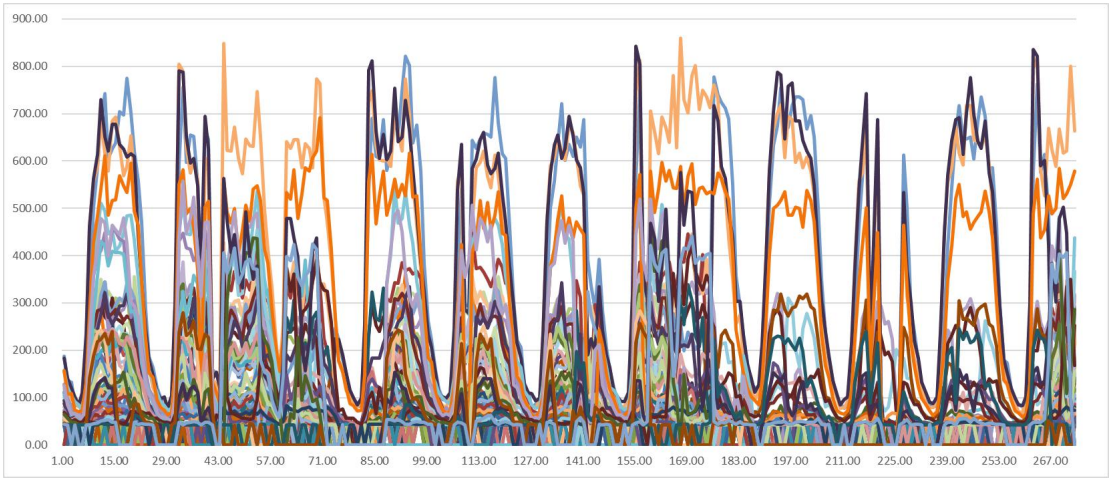


图 3 所有线路的快递数量变化

5.2.2 基于 LSTM 的短期预测模型

LSTM(Long short-term memory)[3]是一种递归神经网络，能够学习长期依赖性，适用在从时间序列中提取时序特征，在时间序列预测方面具有很大的适用性，模型泛化能力强。RNN（循环神经网络）模型，其基本神经元每一个神经元都会接收上一个神经元的输出（且每个神经元状态相同），其状态方程如下：

$$\begin{cases} h' = \sigma(\omega^h h + \omega^i x) \\ y = \sigma(\omega^o h) \end{cases} \tag{1}$$

x 表示当前状态下的输入，y 表示当前节点状态下的输出，h 表示上一个节点的输入，h' 表示传递到下一个节点的输出。

LSTM 神经网络相较于 RNN 模型，分别增加了一个输入状态以及输出状态 C，其状态方程如下：

$$C' = C'^{-1} + a \tag{2}$$

其中 a 为随机常数，通过多次迭代更正其值。

本文使用 LSTM 对各“发货-收货”站点城市之间快递运输数量进行时间序列预测，首先对 EXCEL 中数据进行分析，由图分析站点城市对的时间序列图，观察周期性，可选择以 30 为一个周期性来训练 LSTM 模型，通过选取训练集-测试集对应的比例，即可进行求解。

然后将原数据集进行归一化处理，本文采用 *min-max* 归一化：

$$X' = (X - X_{min})/(X_{max} - X_{min}) \tag{3}$$

X' 为归一化后的快递运输数量。

将时间序列中的每个时期的数据输入，构建一个多层 LSTM 串联网络，最后一层用一层线性层进行输出。通过比较输出与 Y 的误差不断迭代对参数进行优化，得到最终的训练模型。然后通过该训练模型预测 2019 年 4 月 18 日和 2019 年 4 月 19 日各“发货-收货”站点城市之间快递运输数量，以及当日所有“发货-收货”站点城市之间的总快递运输数量。

下图即为 LSTM 模型的实现原理图，*Forget gate* 为遗忘门，*Input gate* 为输入门，*Output gate* 为输出门：

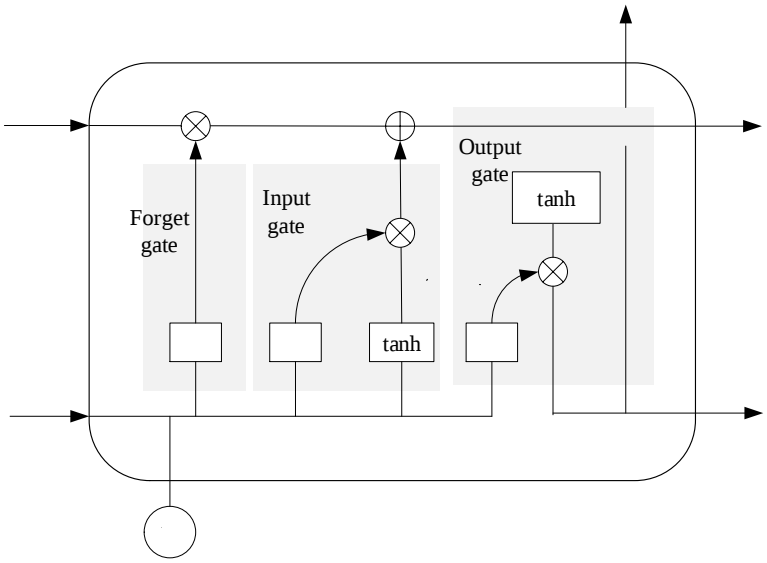


图 4 LSTM 模型结构图

5.2.3 模型的求解

将附件 1 中数据的每一站点城市对在 2018 年 4 月 19 日到 2019 年 4 月 17 日的快递运输数量作为数据集进行训练。将 80%的历史数据划分为训练集，20%的历史数据作为测试集。使用 python 将数据进行归一化处理，导入 LSTM 包进行模型创建与训练，将最终的结果保留小数位为整数位，既可得到最终结果为下表（代码见附录）：

表 2 问题二求解结果

日期	“发货-收货”城市之间的快递运输数量		所有“发货-收货”城市之间的总快递运输数量
2019 年 4 月 18 日	M-U	82	12243
	Q-V	45	
	K-L	101	
	G-V	568	
2019 年 4 月 19	V-G	518	12234
	A-Q	68	

日	D-A	43	
	L-K	88	

5.3 问题三模型的建立与求解

5.3.1 模型的建立

问题三中要求预测每条路线 2020 年 4 月 28 日—2023 年 4 月 27 日的快递运输数量以及是否能正常发货，故本文仍延用问题二的 LSTM 神经网络模型对其进行预测。

首先针对其指定的城市是否能发货进行预测，针对附件二缺失的数据采用本月均值取负进行填补，然后使用 LSTM 神经网络进行预测，如果所预测的快递运输数量为负则说明该发货-收货站点未能正常发货 ,如果预测的快递运输数量为正则说明该发货-收货站点正常发货，且该线路快递运输数量等于该预测值。

5.3.2 模型的求解

将 2023 年 4 月 28 日和 2023 年 4 月 29 日的站点城市对分别进行求解，在问题二的 LSTM 神经网络模型上进行修改。将附件 2 中数据的每一站点城市对在 2020 年 4 月 28 日到 2023 年 4 月 27 日的快递运输数量作为数据集进行训练。将 80%的历史数据划分为训练集，20%的历史数据作为测试集。使用 python 将数据进行归一化处理，导入 LSTM 包进行模型创建与训练，将最终的结果保留小数位为整数位，既可得到最终结果为下表（代码见附录）:

表 3 问题三求解结果

日期	“发货-收货”站点城市对	是否能正常发货 (填写“是”或“否”)	快递运输数量
2023 年 4 月 28 日	I-S	否	0

	M-G	是	38
	S-Q	是	95
	V-A	是	121
	Y-L	是	34
2023 年 4 月 29 日	D-R	否	0
	J-K	是	167
	Q-O	否	0
	U-O	否	0
	Y-W	否	0

5.4 问题四模型的建立与求解

5.4.1 模型的建立

分析附件二中 2023 年 4 月 23 日-2023 年 4 月 27 日每个城市对的数据，可观察出每个城市对有且仅有一个对应的快递运输数量；附件三中所给数据也为所给铁路运输网络[4]中相邻城市对之间对应的固定成本和额定装货量。所给数据不存在缺失或异常，可将 2023 年 4 月 23 日-2023 年 4 月 27 日的数据提取出来进行分别处理。观察附件 3 可看出额定装货量均为 200。

将附件 3 中所给 25 个城市看作 25 个顶点，将固定成本作为相邻两个站点城市的距离，使用 python 对其进行可视化：

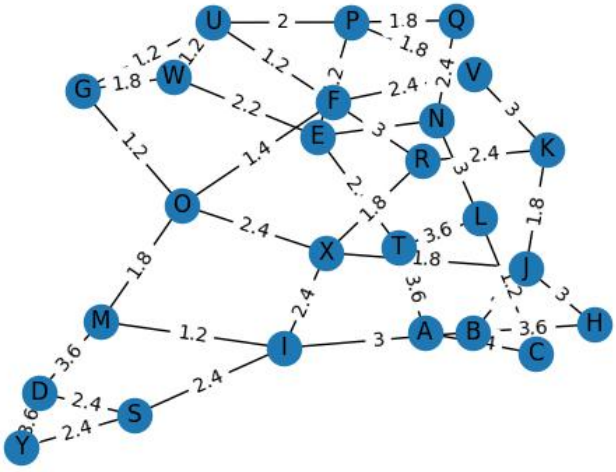


图 5 站点城市间铁路运输网络

铁路运输成本$cost$会影响利润，其计算公式如下：

$$cost = cost\_fix \cdot \left[1 + \left(\frac{actual\_ship}{rate\_ship}\right)^3\right] \tag{1}$$

其中$cost_fix$为铁路运输的固定成本，$actual_ship$为铁路运输的实际装货量，$rate_ship$为铁路运输的额定装货量。

对于站点城市之间运输，本文采用贪心算法对其进行求解：

STEP 1

对于一个站点发送到另一个站点的整个快递，设发送到下一站的固定费用为M_A, M_B, M_C，如果将快递分开为n_1, n_2, n_3进行发送，费用计算为：

$$cost = M_A \cdot \left[1 + \left(\frac{n_1}{200}\right)^3\right] + M_B \cdot \left[1 + \left(\frac{n_2}{200}\right)^3\right] + M_C \cdot \left[1 + \left(\frac{n_3}{200}\right)^3\right] \tag{2}$$

将其简化为$ax^3 + by^3 + cz^3$，在a, b, c相差不大且x, y, z均大于 0 的情况下可将其简化为：

$$ax^3 + by^3 + cz^3 \geqslant a(x + y + z)^3 \tag{3}$$

其中可将不等式右边的a替换为b或c，故应当将发送出去的快递细分为较多的小件快递才能够降低成本。

STEP 2

将快递分为若干个小于等于 5 的小件快递后，对于每个小件快递来说，花费价格为途径每个站点的所有费用相加，如果要求单个小件快递的实际费用，设实际装货量为 n,设通过点A, B, C, D的固定费用为M_A, M_B, M_C, M_D，其费用为：

$$cost = (M_A + M_B + M_C + M_D) \cdot \left[1 + \left(\frac{n}{200}\right)^3\right] \tag{4}$$

在单条路径上，费用与每个站点城市的固定费用成正比。本问要求每个“发货-收货”站点城市对之间使用的路径数不超过 5 条，故可从所有可达路径中选取至多 5 条路径最短（固定费用最少）的路径，设各路径总固定费用分别为 R_1, R_2, R_3, R_4 ，总权重 W 和每条路径分配货量的百分比 $Pert$ 为：

$$\begin{aligned} W &= \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} \\ Pert &= \frac{1}{R_1 W} + \frac{1}{R_2 W} + \frac{1}{R_3 W} + \frac{1}{R_4 W} \end{aligned} \tag{5}$$

后续按照此权重分配快递数量。

STEP 3

对于各个节点来说，根据所有通过当前节点到下一个相同节点的包裹都可以积压，直到所有要通过当前节点的包裹都到达，统一发向下一个节点。

5.4.2 模型的求解

使用穷举法对问题进行求解，其伪代码如下即可得到最低运输成本 $cost$ 为：其中对子函数进行相关封装，列出为主函数调用子函数的主要代码：

Algorithm 1 穷举法伪代码

```
Input:
  Interger start,Integer end,Float weight
Output:
  cost
Initialize parameters
for i do
    for j do
        if !map[i][j] do
            dfs(i,j) //获取前 5 条路径
    end
end

path_weight() //按路径权重分快递
 for i do
    for j do
        package[i].end=end
```

```
            package[i].route=j->second
            package[i].weight=w*p->second
        end
    end
    Return
```

通过 c++编程对其进行求解，先将附件 2，附件 3 数据进行处理，将需要求解的对应日期进行划分。

模型求解过程中选择合适权重，将求解运输成本最低时对应的权重设置为合理权重，经过参数调整，再对路径权重中的分配进行验证，剔除不合理的分配，对应路径数大于 5 条的站点城市对，可对其进行重新求解，重新进行分配，选取较为合理的分配结果充当答案（代码见附录）。

表 4 问题四求解结果

日期	最低运输成本
2023 年 4 月 23 日	3306.71
2023 年 4 月 24 日	3543.07
2023 年 4 月 25 日	3649.35
2023 年 4 月 26 日	2895.3
2023 年 4 月 27 日	2746.8

5.5 问题五模型的建立与求解

5.5.1 数据预处理

将附件 2 中 2022 年第三季度（7 月-9 月）和 2023 年第一季度（1 月-3 月）的数据筛选出来，并将筛选后的数据中每一站点城市对提出出来做进一步分析。将快递运输数量为 0 的数据进行删除。做出 2022 年第三季度 A-O 和 2023 年第一季度 J-I 的快递数量变化图。

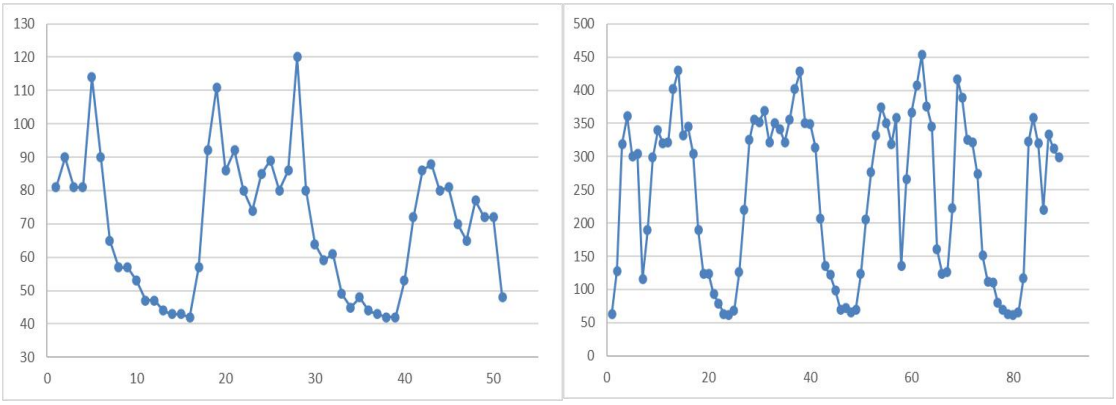


图 6 快递数量变化折线图

5.5.2 模型的建立

通常情况下，快递需求可由固定需求和非固定需求两部分组成，即：

$$\text{快递需求} = \text{固定需求} + \text{非固定需求} \tag{1}$$

根据上图可知在任意一个季度中，其快递运输量仍具有一定的周期性，且波动程度较大，本文采用时间序列的需求分析思维对该数据及进行分析，要求得其固定需求，假设其每日运输量为x_i，则运输量平均值等于：

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2}$$

运输量的标准差σ为：

$$\sigma = \sqrt{S^2} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x}) \tag{3}$$

则假设固定需求为$demand_{fixd}$，则：

$$demand_{fixd} = x_i - \sigma$$

$$demand_{fixd} = \begin{cases} demand_{fixd}, & demand_{fixd} > 0 \\ 0, & demand_{fixd} \leqslant 0 \end{cases} \tag{4}$$

当$demand_{fixd}$大于 0 时，该站点城市的固定需求为$demand_{fixd}$，若其小于 0 时，则说明该站点城市的固定需求为 0。

对其进行准确性检验，将所有的异常值进行增减$0.1\bar{x}$，然后在对其求取其固定需求$demand_{fixd}$，本文对其进行验证后得出异常值变化后的固定值仍未发生变化，可得该准确性检验通过，模型成立。

针对第二小问，首先使用原数据减去固定的需求来求得非固定需求。对于非固定需求，假设其概率密度为$f(x,\theta)$，则该联合密度函数为：

$$L(x_1, x_2, \cdots, x_n, \theta) = \prod_{i=1}^{n} f(x_i, \theta) \tag{5}$$

将这个函数称为非固定需求的似然函数。通过计算求出$\hat{\theta}$在一定范围内的取值使得：

$$L\left(x_1, x_2, \cdots, x_n, \hat{\theta}\right) = max\ L(x_1, x_2, \cdots, x_n, \theta) \tag{6}$$

则$\hat{\theta}$称为参数θ的最大似然值，相应的非固定需求的统计量则称为θ的最大似然估计量。

然后对$L(x_1, x_2, \cdots, x_n, \theta)$进行微分得：

$$\frac{d}{d\theta}L(\theta) \tag{7}$$

令其等于零，则可得到似然方程，并解出概率密度的参数θ。

然后根据连续型随机变量的方差公式：

$$D(x)=\oint_R (x-E(x))^2 f(x)dx \tag{8}$$

即可求得非固定需求方差再对其开根号求得其标准差。

5.5.3 模型的求解

表 5 问题五求解结果

季度	2022 年第三季度 (7—9 月)		2023 年第一季度 (1—3 月)	
“发货-收货”站点城市对	V-N	V-Q	J-I	O-G
固定需求常数	0	0	137	33
非固定需求均值	108.53	92.23	245.32	140.11
非固定需求标准差	71.83	47.90	122.12	74.29
固定需求常数总和	48		757	
非固定需求均值总和	201.34		385.32	
非固定需求标准差总和	120.47		195.54	

六、 模型的评价、改进与推广

6.1 模型的优点

投影寻踪法优点：能够全面、客观、定量地对投影寻踪技术进行评价，相较于传统评价模型如层次分析法，该模型较为客观，主观性不强具有较为准确的评价能力，相较于模糊综合评价该模型的评价结果更具有比较性，相较于灰色综合分析，该方法可以处理较大数据。

LSTM 神经网络优点：LSTM 更适合处理本文中时间序列的预测题型，相较于传

统的 RNN 模型，LSTM 可以处理长期的时间序列问题，且 LSTM 模型改善了 RNN 模型中所存在的长期依赖问题，且作为非线性模型，LSTM 可用于构造更大的深度神经网络，且在本题中作为有监督学习用已有的数据进行模型训练并进行预测更加契合本题的要求。

6.2 模型的缺点

投影寻踪法缺点：评价结果的可靠性极度依赖于数据的准确性和完整性，且本文中指标选取较少，可能会影响评价结果。

LSTM 神经网络缺点 :模型本身结构较为复杂导致训练耗时相对于其他模型较长，且并未完全解决 RNN 模型所存在的长期依赖问题，对于更长的时间序列模型则需要引入更多变体。

6.3 模型的改进

对于第一问中的投影寻踪法，可再引入一些指标使其评价模型更加准确，契合题中的要求。

对于二三问中的 LSTM 神经网络模型，则可引入第二个变体对其进行训练预测，与文中的预测情况相比较，选取最优预测结果。

6.4 模型的推广

投影寻踪法可用于环境的评价，水质评估等需要客观性评价，且数据较为明朗的情况下。

LSTM 神经网络可以应用于疾病、股票等预测，也可以应用于一些周期较长，且呈非线性的数据集的训练。

七、 参考文献

[1] 索瑞霞,马嘉敏.基于投影寻踪法的中国城市低碳发展水平评价[J].调研世界,2023,No.355(04):73-82.DOI:10.13778/j.cnki.11-3705/c.2023.04.008.

[2] 梁苗. 基于粒子群优化投影寻踪模型的大型商场火灾风险评价[D].武汉理工大学,2020.DOI:10.27381/d.cnki.gwlgu.2020.000176.

[3] 韩金磊,熊萍萍,孙继红.基于 LSTM 和灰色模型的股价时间序列预测研究[J/OL].南京信息工程大学学报(自然科学版):1-22[2023-05-01].http://kns.cnki.net/kcms/detail/32.1801.N.20230105.1635.003.html.

[4] 谭存爱,钱颖.基于复杂网络的中国煤炭铁路运输网络研究[J].物流科技,2022,45(15):89-94.DOI:10.13714/j.cnki.1002-3100.2022.15.021.

[5] 魏丹丹,周翊,赵宇.基于最大似然估计的自适应回声消除算法[J].西南大学学报(自然科学版),2023,45(04):210-218.DOI:10.13718/j.cnki.xdzk.2023.04.020.

附录

附录 1

介绍：粒子群优化投影寻踪　　problem1.m

```
clear;clc;
x=[2    31276  4      51708  52.57889936   50.45593111
1       12458  1      15624  29.63853752   42.97382358
4       18361  2      13228  46.10505847   28.48831727
4       31135  4      52401  54.70099154   41.12926916
1       14643  2      20055  29.203883     35.24265243
7       287248 9      273737 291.3801287   283.2866973
3       45866  3      47155  77.30082643   69.59407607
3       45100  3      50855  76.20409793   87.08336269
4       99173  4      98557  120.5682802   119.9666089
3       74841  3      71003  88.96797243   94.82945039
10      296846 10     311289 281.1682951   277.8236723
5       52041  4      47609  75.31307063   76.29544954
3       62359  5      44445  62.05232479   62.71792015
3       64986  6      87091  102.3338257   136.9054998
1       15419  1      13357  21.85998496   19.02082915
5       60654  5      81076  71.82360017   97.9920796
5       109685 5      91333  136.5062515   127.5227486
5       56714  2      37140  69.34058919   52.03018059
2       32819  0      0      79.73535074   0
4       40584  2      24339  60.39116282   39.80198592
6       164403 6      192823 187.3221914   227.2002083
3       69505  3      77336  129.4539603   144.4777508
4       89437  5      84303  127.2938271   136.2121642
3       52758  2      41811  81.5348613    90.09763273];
[n,m] = size(x);
x = x ./ repmat(sum(x.*x) .^ 0.5, n, 1);


%粒子群优化投影方向

num=1000;

iter=50;%迭代次数

n=size(x,2);%指标个数

wmax=0.5;wmin=0.1; %惯性权重
```

```
sub=[0.1 0.2 0.3 0.05 0.05 0.05];%设置六个指标的权重范围
up=[0.3 0.4 0.5 0.5 0.5 0.5];

c=[];
for i=1:100000
    c(i,:)=sub+rand.*(up-sub);
    c(i,:)=c(i,:)./sum(c(i,:));
    y(i,1)=Target(x,c(i,:));
end
[y,b]=sort(y,'descend');
c=c(b,:);
y=y(1:num,1);
c=c(1:num,:);
bestc=c(1,:);
besty=y(1);
trace=besty;
cc=[];
yy=[];
v=rand(num,6);
for i=1:iter
    fave=mean(y);
    fmax=max(y);
    for j=1:num
        if y(j)>fave
            w=wmin+(fmax-y(j))*(wmax-wmin)/(fmax-fave);
        else
            w=wmax;
        end
        v(j,:)=w*v(j,:)+rand*(bestc-c(j,:));
        cc(j,:)=c(j,:)+v(j,:);
        cc(j,:)=max(cc(j,:),sub);
        cc(j,:)=min(cc(j,:),up);
        cc(j,:)=cc(j,:)./sum(cc(j,:));
        yy(j,1)=Target(x,cc(j,:));
    end
    Y=[];
    C=[];
    Y=[y;yy];
    C=[c;cc];
    [Y,b]=sort(Y,'descend');
    C=C(b,:);
    y=Y(1:num,:);
```

```
    c=C(1:num,:);
    bestc=c(1,:);
    besty=y(1);
    trace=[trace,besty];
end

disp('最佳权重')

disp(bestc)

disp('评价值')

P=x*bestc'
[Q,index] = sort(P ,'descend')
figure
plot(trace)

xlabel('迭代次数')


ylabel('G 函数')


title('粒子群算法优化投影寻踪')
```

附录 2

介绍：投影寻踪法 target.m

```
function y=Target(x,a)
[m,n]=size(x);
for i=1:m
    s1=0;
    for j=1:n
        s1=s1+a(j)*x(i,j);
    end
    z(i)=s1;
end
Sz=std(z);
R=0.1*Sz;
s3=0;
for i=1:m
    for j=1:m
        r=abs(z(i)-z(j));
        t=R-r;
        if t>=0
```

```
            u=1;
        else
            u=0;
        end
        s3=s3+t*u;
    end
end
Dz=s3;
y=Sz*Dz;
end
```

附录 3

介绍：贪心求解最低运输成本　　cost.cpp

```
#include <iostream>
#include <set>
#include <list>
#include <map>
#include <cmath>
using namespace std;
double maps[30][30];

bool arrive[30];//已达

struct Route {
	map<int,list<int>> route;

	map<int, double> weight;// 路基对应权重

	void replace(int start,list<int>& list) {
		int ww = getWeight(start, list);
		if (route.size() >= 5) {
			int max = 0;
			int wh = 0;

			// 找出最不划算得路径替换掉

			for (auto i = route.begin(); i != route.end(); i++) {

				// 长度优先

				if (max < i->second.size()) {
					max = i->second.size();
					wh = i->first;
				}

				// 相同长度路径权重比较

				else if (max == i->second.size()) {
					if (weight[i->first] < weight[wh]) {
						max = i->second.size();
						wh = i->first;
					}
				}
			}
			if (max > list.size()) {
				route[wh] = list;
```

```
                                        weight[wh] = ww;
                                }
                                else if (max == list.size()) {
                                        if (weight[wh] < ww) {
                                                route[wh] = list;
                                                weight[wh] == ww;
                                        }
                                }
                        }
                        else {
                                weight[route.size()] = ww;
                                route[route.size()] = list;
                        }
                }
private:
        // 获取总体权重，与实际权重是相对值
        double getWeight(int start,list<int>& list) {
                double all = 0;
                auto wh = list.begin();
                while (wh!=list.end()) {
                        all += maps[start][*wh];
                        start = *wh;
                        wh++;
                }
                return all;
        }
};
struct Package {
        int weight = 0;
        int end = 0;

        list<int> route;// 包裹路径

        bool operator < (const Package& package)const {
                return this->end < package.end;
        }
        bool operator == (const Package& package)const {
                return this->end == package.end;
        }

        void plus(Package& p) {
                if (this->end == p.end) {
                        this->weight += p.end;
```

```
                }
        }

        void plus(int weight) {
                this->weight += weight;
        }
};
struct Node {

        // end,包集合

        map<int,Package> package;

        // end, 可到达选路径

        map<int, Route> route;

        list<Package> routePackage;// 转发包裹列表

};
Node node[30];
void dfs(int start,int next,list<int>& t) {
        if (next == start) {
                return;
        }
        arrive[next] = true;
        list<int> route;
        for (int i : t)route.push_back(i);
        route.push_back(next);
        node[start].route[next].replace(start,route);
        for (int i = 0; i < 30; i++) {
                if (maps[next][i] != 0&&!arrive[i]) {
                        dfs(start, i, route);
                }
        }
        arrive[next] = false;
}
int main() {
        while (true) {
                char start, end;
                double cost;
                cin >> start;
                if (start == 'Z')break;
                cin>> end >> cost;
                maps[start - 'A'][end - 'A'] = cost;
        }
```

```
        while (true) {
                char start, end;
                int weight;
                cin >> start;
                if (start == 'Z')break;
                cin >> end >> weight;
                Package p;
                p.end = end - 'A';
                p.weight = weight;
                node[start-'A'].package[end - 'A'] = p;
        }

        // 获取前 5 条路径

        for (int i = 0;i< 30; i++) {
                for (int j = 0; j < 30; j++) {
                        if (maps[i][j] != 0) {
                                list<int> t;
                                dfs(i, j, t);
                        }
                }
        }

        // 分包

        for (int i = 0; i < 30; i++) {

                // 发送节点 i 的包裹

                for (auto p = node[i].package.begin(); p !=
node[i].package.end(); p++) {

                        Package packages[5];// 分成 5 个包裹


                        // p 是发送得包裹

                        int end = p->second.end;
                        map<int, list<int>> route = node[i].route[end].route;
                        map<int, double> weight = node[i].route[end].weight;

                        int allWeight = 0;// 总权重

                        for (auto i = weight.begin(); i != weight.end(); i++) {
                                allWeight += i->second;
                        }
                        if (allWeight == 0) {
                                continue;
                        }
```

```
                        // 按路径权重分包裹
                        int wp = 0;
                        double maxw = 0;
                        int maxwp = 0;
                        for (auto j = route.begin(); j != route.end() && wp < 5;
j++, wp++) {
                                int w = weight[j->first];// 路径权重
                                if (w > maxw) {
                                        maxw = w;
                                        maxwp = wp;
                                }
                                packages[wp].end = end;
                                packages[wp].route = j->second;
                                packages[wp].weight = w * p->second.weight/
allWeight;
                        }
                        // 多余包裹给到权重最大得包裹
                        int yu = 0;
                        for (int i = 0; i < 5; i++) {
                                yu += packages[i].weight;
                        }
                        if (p->second.weight - yu != 0) {
                                packages[maxwp].weight +=
p->second.weight - yu;
                        }
                        for (int i = 0; i < 4; i++) {
                                node[i].routePackage.push_back(packages[i]);
                        }
                }
        }
        // 分发包裹
        double ans = 0;
        map<int, double> ps[30];// 保存节点发送到同一个节点的总重量
        while (true) {
                bool flag = true;
                for (int nod = 0; nod < 30; nod++) {
                        if (node[nod].routePackage.size() == 0) {
                                continue;
```

```
                        }
                        for (auto p = node[nod].routePackage.begin(); p !=
node[nod].routePackage.end(); p++) {
                            if ((*p).route.size() == 0)continue;
                            flag = false;
                            int next = (*p).route.front();// 下一跳地址
                            (*p).route.pop_front();// 除去第一跳
                            ps[nod][next] += (*p).weight;// 相同点重量相加
                            node[next].routePackage.push_back(*p);// 发
送
                        }
                        node[nod].routePackage.clear();
                    }
                    if (flag)break;
                }
                // 计算合计成本
                for (int i = 0; i <30; i++) {
                    if (ps[i].size() > 0) {
                        for (auto next = ps[i].begin(); next != ps[i].end();
next++) {
                            double w = maps[i][next->first];// 固定成本
                            double weight = next->second;// 重量
                            ans += w * (1 + pow(weight / 200, 3));
                        }
                    }
                }
                cout << ans;

                return 0;
}
```