五一数学建模竞赛

承　诺　书

我们仔细阅读了五一数学建模竞赛的竞赛规则。

我们完全明白，在竞赛开始后参赛队员不能以任何方式（包括电话、电子邮件、网上咨询等）与本队以外的任何人（包括指导教师）研究、讨论与赛题有关的问题。

我们知道，抄袭别人的成果是违反竞赛规则的，如果引用别人的成果或其它公开的资料（包括网上查到的资料），必须按照规定的参考文献的表述方式在正文引用处和参考文献中明确列出。

我们郑重承诺，严格遵守竞赛规则，以保证竞赛的公正、公平性。如有违反竞赛规则的行为，我们愿意承担由此引起的一切后果。

我们授权五一数学建模竞赛组委会，可将我们的论文以任何形式进行公开展示（包括进行网上公示，在书籍、期刊和其他媒体进行正式或非正式发表等）。

参赛题号（从 A/B/C 中选择一项填写）：B

参赛队号：T3838005627393

参赛组别（研究生、本科、专科、高中）：本科

所属学校（学校全称）：中山大学

参赛队员：　队员 1 姓名：杨睿贤

队员 2 姓名：邵若西

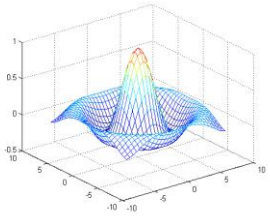
队员 3 姓名：邹书承

联系方式：　Email：joengjeoijin@outlook.com　联系电话：17600715372

日期：　2024　年　5　月　1　日

（除本页外不允许出现学校及个人信息）

五 一 数 学 建 模 竞 赛



题 目：基于优化算法的交通需求规划的研究

关键词： 交通需求规划 凸优化 非线性规划 优化模型 模拟退火 期望可达率最大

摘 要： 随着城市化的不断推进，交通规划在新兴城市建设中越来越重要。未来，自动驾驶汽车能够在无人状态下自动遵循预设路线，将自动驾驶技术整合到一个特定的未来新城的交通规划中有助于形成一个更高效、更可持续的城市交通网络。本文主要研究如何规划交通需求以使所有交通的期望可达率最大，建立了优化模型，讨论了目标函数的相关性质，并使用 KKT 法、“爬山法”以及模拟退火方法对问题进行了良好近似与快速求解。

在问题一中，我们在使用广度优先搜索算法求出可行路径的基础上，将原问题中的不连续目标函数转为连续且凸的二次函数形式，通过凸二次规划问题的 KKT 点来获得目标函数的全局最优解，并使用 Moore-Penrose 广义逆矩阵求出一个特解，得到了网络中任意 1 条路段出现突发状况时使所有交通的期望可达率最大的五条最短路径的交通分配量，见附录表 1 问题 1 结果。

对于问题二，由于路线复杂度提升、题目要求变多、数据量变大，近似目标函数的最优化结果与准确的目标函数优化结果存在差异，所以可以使用分段优化方法。首先使用爬山法优化近似目标函数，随后使用模拟退火方法优化准确的目标函数。使模型快速收敛的同时跳出局部最优解求得了全局最优解，见附录表 2 问题 2 结果。

在问题三中，若将交通量分配的需求看作等式约束，则可行域不存在。因此，我们转换等式约束为不等式约束，通过一次搜索，使用模拟退火算法求出了该问题的数值近似解。见附录表 3 问题 3 结果。

在问题四中，我们使用逐步优化的模拟退火算法，通过定义迫切度指标逐步求解，给出了新建道路的若干方案。并使用模拟退火算法对新建道路后的总体需求期望可达率进行了评估，算法稳定而高效。结果见附录表 4 问题 4 结果。

一、问题的背景与重述

1.1 问题的背景

随着城市化的推进，交通规划在新兴城市建设中尤为关键。未来新城规划中，自动驾驶技术预期成为交通出行的主导模式，这彻底改变出行方式和城市规划的基础理念。拥有先进传感器、智能算法和通信技术的自动驾驶车辆，能够自动遵循预设路线，无需人为操作。将自动驾驶技术整合到一个特定未来新城的交通需求规划中，可以构建更高效、更可持续的城市交通网络。

1.2 问题的重述

问题 1：在图 1 所示的小型交通网络中，附件 1 给定各(起点,终点)对之间的交通需求，现需要规划一个交通流量分配方案，将交通需求分配到相应的路径上，以最大化在任意一条路段发生突发状况时，网络中所有交通需求的期望可达率。

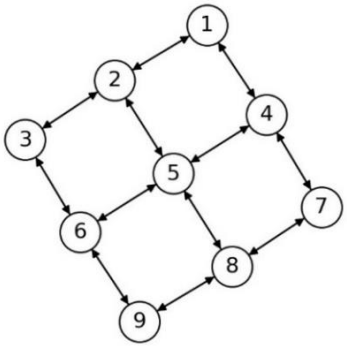


图 1 交通网络 1

问题 2：在图 2 所示的交通网络中，附件 2 给定各(起点,终点)对之间的交通需求。现在需要规划一个交通流量分配方案，将交通需求分配到对应路径上的，以确保在网络中任意 5 条路段同时出现突发状况的情况下，网络中所有交通需求的期望可达率最大。

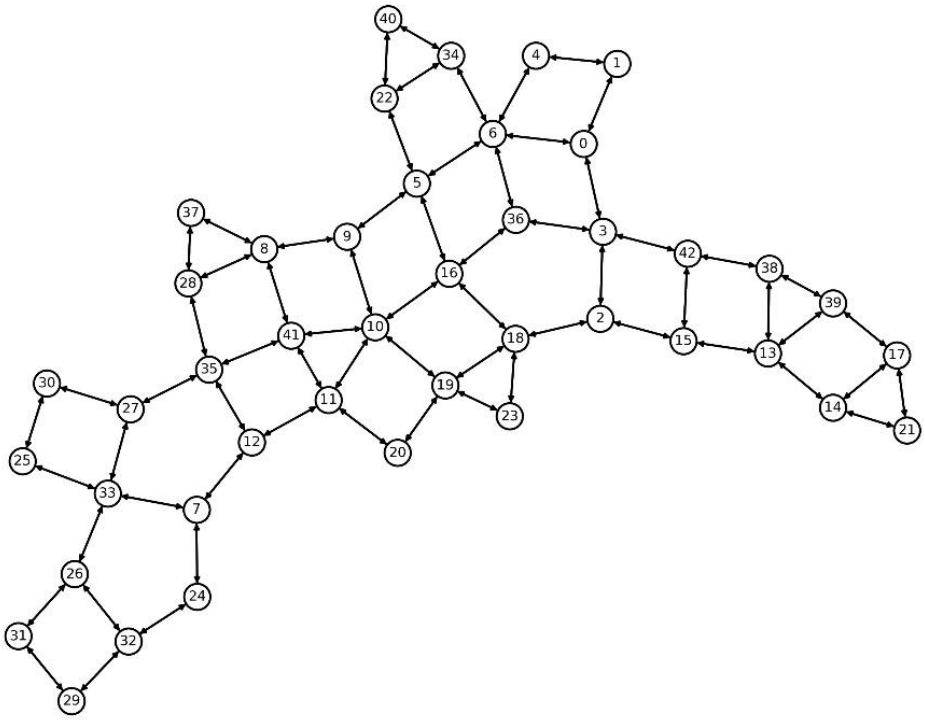


图 2 交通网络 2

问题 3：与问题 2 相比，本题多出了对道路承载力的限制。需要交通需求分配到对应路径上的交通量，使得网络中任意 5 条路段出现突发状况时，网络中所有交通需求的期望可达率最大，且不超过路段容量。

问题 4： 本题在问题 3 的基础上允许在交通网络中修建六条不交叉的新单向路段。优化目标仍然不变，需求出在任意 5 条线路出现突发事故时的最大期望可达率。

二、符号说明

表格 1 论文中使用的符号说明

符号	符号含义及说明
η_i	在第i种情况下的总体交通可达率
（问题分析中的）p_i	第i种情况的发生概率
m	整个网络的期望可达率
t_m	网络中各路段（路段m）通行交通量
R	给定的路径方案矩阵
p	分配到对应规划路径上的交通量

λ^*	Lagrange 乘子
p^*	问题一的最优解
p_{0i}	所有出行方案在未限制道路容量时的被分配的交通量
t_{0j}	未限制道路容量时的交通量
c_j	道路容量

三、问题分析

对本问题我们最初有两种想法：一种想法认为本题所希望求的是在所有道路突发状况下总体交通可达率η_i的平均值，即优化目标为：

$$max = \sum_i \eta_i p_i \tag{3.1}$$

这种理解简单直接，也更加符合一般意义上的期望，但是，对本题而言，使用这种理解会导致如下问题。考虑一个两支路α和β的网络，两支路的突发状况概率分别为q_1和q_2，整个网络的期望可达率记为m，通过支路α的流量占比为a，则通过支路β的占比为$1-a$。我们不难得到以下公式：

$$a(1-q_1)+(1-a)(1-q_2)=m \tag{3.2}$$

化简，我们会得到：

$$m = a(q_2-q_1)+1-q_2 \tag{3.3}$$

显然，为使m最大，当$q_1 > q_2$时，取$a=0$，当$q_1 < q_2$时，取$a=1$，当$q_1 = q_2$时，a可取$[0,1]$之间的任意实数。

同时，根据公式

$$\sum_i \eta_i p_i \tag{3.4}$$

以及前面的讨论，各种交通需求之间互不干扰。这将使得问题 1 和问题 2 的交通需求分配方案在路径长度一致时取值可为任意数。这样的答案近乎无意义，同时与现实生活中的情景严重不符。

另一种理解是，本题所求总体交通需求的期望可达率实际上指在最坏情况下的总体交通需求可达率，亦即交通系统的稳定程度。即优化目标为

$$max = \min(\eta_i), \forall i. \qquad (3.5)$$

在这种情况下，各种交通需求之间不再解耦。而是深度耦合，考虑以下的例子：

如图 3a，假设只有从 A 到 B 和从 C 到 D 的交通需求，此时如果我们分别考虑从 A 到 B 与从 C 到 D 的交通需求，我们将会得到如图 3a 的结果。此时如果 AE 或者 DE 路段出现特殊状况，则会导致整个系统的期望可达率下降，在综合考虑后，如图 3b，我们将流量重新分配，得到一个更优的结果。不必证明图 3b 结果是最优解，已经足以说明将不同交通需求单独考虑无法得到最优解。

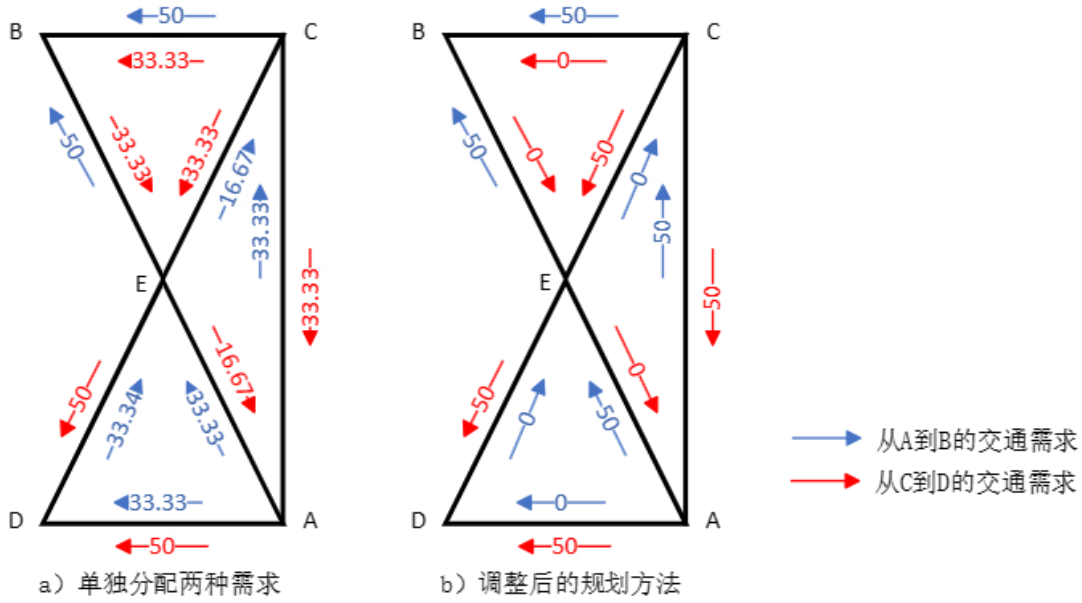


图 3

此时，每条路径的交通需求量均为 50，整个交通系统的期待可达率也提高了，这说明，不同（起点，终点）对之间的交通需求量的分配会影响整个系统的期待可达率。

为了求出两点之间的可能路径，在建模过程中我们使用了广度优先搜索算法。

广度优先搜索算法(Breadth-FirstSearch)，又译作宽度优先搜索，或横向优先搜索，简称 BFS，是最简便的图的搜索算法之一。简单的说，BFS 是从根节点开始，沿着树的宽度遍历树的节点。如果所有节点均被访问，则算法中止。这一算法也是很多重要的图算法的原型。Diikstra 单源最短路径和 Prim 最小生成树算法都采用了和广度优先搜索类似的思想。BFS 是一种盲目搜寻法，目的是系统地展开并检查图中的所有节点，以找寻结果。换句话说，它并不考虑结果的可能位址，彻底地搜索整张图，直到找到结果为止。[1]

比起深度优先搜索算法，广度优先搜索算法在寻找较短的路径上更胜一筹，因为它不像深度优先算法那样先走完一条道路，它是同时搜索多条道路，直到找到最短的五条。随后，我们将使用恰当的方法分配不同路径上的交通量，使之满足题目的要求且使该交通网络中所有交通需求的期望可达率最大。

在问题一二中——我们下面将证明这两个问题可以近似转化为凸优化问题——分别使用 KKT 法和“爬山算法”求出各规划路径上的最优交通量。而对于问题三与问题四，除了上述约束条件外，由于还要求各路段上的交通量不能超过路段容量，我们还将采用模拟退火优化算法进行求解。

计算机模拟退火优化算法的主要作用是在一个大的搜索空间中寻找最优解或近似最优解。它通过模拟退火的方式，跳出局部最优解，从而有机会找到全局最优解。由于问题三四中的模型较为复杂，我们无法肯定其能够被转化为凸优化问题，意味着很可能存在多个局部最优解。计算机模拟退火算法可以帮助跳出局部最优解，并在整个搜索空间中找到全局最优解或近似最优解。[2]

2.1 问题一的分析

在问题一中，需要求出一个小网络的交通需求分配方法。由上述分析，我们将其近似转化为一个凸二次优化问题。随后，我们通过 Lagrange 乘子法求得二次目标函数取最小值的必要条件，并使用 Moore-Penrose 广义逆矩阵求出一个特解。我们将稍后说明其通解的求法。

2.2 问题二的分析

与问题一类似，只是这次的交通线路更复杂且共有 5 条线路出现突发状况。此时，由于我们所要优化的目标函数仍然可以近似转化为一个二次函数。这是一个凸优化问题，其局部最优解即为全局最优解，而由于该问题的数据量比第一题大，我们选择用爬山法来寻找最优解。在此之后，我们使用随机游走的爬山算法进一步寻找使得最坏情况下不能到达目的地的交通量的最小值。从而使解更精确。

爬山法(Hill Climbing，HC)是一种局部择优的贪心搜索算法，该算法每次从当前的节点开始，与周围的邻接点进行比较，若当前节点是最大的，就将当前节点作为最大值；若当前节点是最小的，就用最高的邻接点替换当前节点，从而实现向山峰的高处攀爬的目的。在本题中，由于问题的自由度较高，因此使用了爬山法的随机游走变体以加快计算速度。

2.3 问题三的分析

在问题二的基础上，问题三要求各路段上的交通量不能超过路段容量，因此我们在求解方程时多了一个约束条件。由于本题凸性不明确，为了避免陷入局部最优解，我们决定使用模拟退火的优化算法寻找全局最优解。同时，我们发现现有交通网络即使不发生突发情况也不可能承载所有的交通需求。

2.4 问题四的分析

问题四相比问题三允许多修建六条单向不交叉的道路。可以有效化解问题三中出现的热点区域拥堵问题，具有较为明确的现实意义。经过计算，该问题的穷举规模在10^{18}量级。完整地进行最优化需要进行动态规划，但限于问题的规模，我们选

择使用贪心算法，在修建每一条道路之后重新评估修建新道路的迫切性。从而逼近全局最优解。

四、模型假设

1. 假设每个（起点,终点）对之间使用的路径数不超过 5（各路段长度均为单位 1，优先选择距离短的路径）。（题目给出）

2. 假设交通网络中所有车辆均为无人驾驶车辆，并且所有车辆都服从系统预先规划的路径进行出行。（题目给出）

3.假设图 2 和图 3 中的路段为双向路段。（题目给出）

4.假设每条路段出现突发状况的概率相同。（题目给出）

5.假设新建路段容量足够大，不用考虑路段容量的问题。（题目给出）

五、模型的建立与求解

5.1 问题一的模型建立与求解

向量t表示网络中各路段通行交通量（反映一个路段上的繁忙程度），矩阵R（各元素为r_{mn},m为行数，n为列数）表示一个给定的路径方案矩阵（在本题中由路径搜索算法给出），向量p为分配到对应规划路径上的交通量，因此：

$$\begin{pmatrix} t_1 \\ \vdots \\ t_m \end{pmatrix} = \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & \ddots & \vdots \\ r_{m1} & \cdots & r_{mn} \end{pmatrix} \begin{pmatrix} p_1 \\ \vdots \\ p_n \end{pmatrix} \tag{5.1.1}$$

即

$$t = Rp \tag{5.1.2}$$

决策变量为

$$p_1, p_2, \cdots, p_n$$

约束条件为

$$\begin{cases} p_{in} > 0 \\ t_i > 0 \\ Ap = b \end{cases}$$

优化目标函数t为

$$min = \max\{t_i\} \tag{5.1.3}$$

方便起见，我们将优化目标转化为

$$min = \sum t_i^2 \tag{5.1.4}$$

在结果中我们将会看到，这样的优化目标实际上良好地控制了t_i的最大值。

由于t是一个列向量，因此

$$t^2 = t^T t = (Rp)^T (Rp) = p^T R^T Rp \tag{5.1.5}$$

记

$$Q = R^T R \tag{5.1.6}$$

又由于列向量p满足$Ap = b$（A是一个最简行阶梯型矩阵 ，b是各对之间的交通需求量），即

$$\begin{cases} min = p^T Qp, \\ Ap = b \end{cases}$$

其中Q是正定对称阵。即说明转化后问题是凸二次规划问题。由于在凸二次规划问题中，局部最优解即为全局最优解，所以可以通过求解凸二次规划问题的 KKT 点，间接求得本题的全局最优解[3]。

设p^*为问题的最优解，p^*则满足 KKT 条件：

$$\begin{cases} Qp^* - A^T \lambda^* = 0 \\ Ap^* = b \end{cases} \tag{5.1.7}$$

记λ^*为对应的 Lagrange 乘子。将（5.1.7）中的等式组表示为如下矩阵形式：

$$\begin{pmatrix} P & -A^T \\ A & 0 \end{pmatrix} \begin{pmatrix} p^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} 0 \\ b \end{pmatrix} \tag{5.1.8}$$

则可以求解。

*注意*到，系数矩阵

$$\begin{pmatrix} P & -A^T \\ A & 0 \end{pmatrix}$$

并不是满秩矩阵，这使得该非齐次线性方程的解

$$\begin{pmatrix} p^* \\ \lambda^* \end{pmatrix}$$

不唯一。为求出其中一个特解我们使用 Moore-Penrose 广义逆矩阵求出其一个特解。

记

$$B = \begin{pmatrix} P & -A^T \\ A & 0 \end{pmatrix}$$

则 B 矩阵满足全部 4 个 Moore-Penrose 条件，即 BGB＝B、GBG＝G、（GB）H ＝ GB 、（BG）H ＝ BG 的广义逆矩阵 G，记作 B＋，称为矩阵 B 的加号逆．它对于任意矩阵 B 都存在，而且是唯一的。

Moore-Penrose 广义逆矩阵是满足 Moore-Penrose 条件的矩阵。[4]若矩阵G对一给定的矩阵B具有以下性质：

$$\begin{cases} BGB = B \\ GBG = G \\ (GB)H = GB \\ (BG)H = BG \end{cases}$$

则称G是B的 Moore-Penrose 广义逆矩阵，记作B^+。彭罗斯(R.Penrose)证明了对任一$m \times n$阶矩阵B，都存在惟一的$n \times m$阶矩阵 G 满足上述条件。利用其性质三，我们可以方便地求出无穷多解非齐次线性方程组的一个特解。

于是，$\begin{pmatrix} p^* \\ \lambda^* \end{pmatrix}$的一个特解为:

$$\begin{pmatrix} p^* \\ \lambda^* \end{pmatrix} = \begin{pmatrix} P & -A^T \\ A & 0 \end{pmatrix}^+ \begin{pmatrix} 0 \\ b \end{pmatrix} \qquad (5.1.9)$$

在本题中，由 Moore-Penrose 广义逆求出的p^*恰好全为正数。但是在更一般的情形下，是否存在p^*满足题目要求并且符合 KKT 条件还需要进一步的讨论。

显然，该方程可以通过 Gauss-Jordan 法求出其通解，但该法程序实现较为复杂，我们遂决定使用爬山算法求出其数值解。

最后得出：

不可达需求量：643.5417

总需求量：5330

最大期望可达率：87.93%

5.2 问题二的模型建立与求解

首先创建路径邻接矩阵，其次使用广度优先搜索，搜索出符合条件的五条路径，接着使用“爬山法”寻找局部最优解。

与问题一类似，但是优化目标变为

$$max = \min \eta_{i_1 i_2 i_3 i_4 i_5} \tag{5.2.1}$$

此时

$$\eta_{i_1 i_2 i_3 i_4 i_5} \geq \sum_i p_i - (t_{i_1} + t_{i_2} + t_{i_3} + t_{i_4} + t_{i_5}) \tag{5.2.2}$$

等号成立当且仅当

$$\forall i, j, k, \frac{\partial t_j}{\partial p_i} \frac{\partial t_k}{\partial p_i} = 0 \tag{5.2.3}$$

但考虑到数据规模，这一假设显然不成立且不能近似认为成立。

因此，我们提出如下算法：取t的最大值t_i，考虑所有j满足$R_{ij} \neq 0$，令$p_j = 0$，即

$$\exists i \; st. \forall k \; t_i \geq t_k, \forall j, R_{ij} \neq 0, 令 p_j = 0$$

重新计算t，重复上述步骤五次，剩余的可通行量

$$\eta_{min} = \sum_i t_i$$

即为所希望的最小可通行量——即交通网络在受到最大冲击时仍可满足的通行量。

必须指出，准确地计算最小可通行量η_{min}需要使用动态规划算法。但由于算法速度的考虑，此处使用该种贪心算法求取其最大值。不难发现，使用该种算法所造成的影响不会太大。因此，在综合考虑效率和准确度之后，我们选取如上所述求出η_i作为目标函数。

由于该函数的凸性不明确，该题的求解使用两段优化。可以知道，问题一中的近似目标

$$min = \sum_i t_i^2$$

仍然是本问的低阶近似。在该问中，KKT 生成的系数矩阵

$$\begin{pmatrix} Q & -A^T \\ A & 0 \end{pmatrix}$$

的 Moore-Penrose 广义逆矩阵存在小于零的元素，而求出该方程的通解较为复杂，所以使用随机游走的爬山算法求其近似解。

首先随机分配p_i满足$Ap = b$，然后选取同一（起点，终点）对中的两个交通量p_i和p_j，并产生一随机数ω使得$p_i \geq \omega \geq -p_j$，令$p_i$减去$\omega$，同时令$p_j$加上$\omega$。检查新状态下的目标函数

$$\sum_i t_i'^2$$

若新目标函数值大于原目标函数值，则接受该更改，否则撤销该更改。

随后，使用模拟退火算法优化前文提到的准确的目标函数。与爬山算法类似，该算法选取同一（起点，终点）对中的两个交通量p_i和p_j，并产生一随机数ω使得$\omega \geq p_i$，令p_i减去ω，同时令p_j加上ω。检查新状态下的目标函数值与原目标函数的值之差。若新目标函数值大于原目标函数值，则接受该更改；若新目标函数值小于原目标函数值，则有概率

$$q = e^{\frac{new-old}{T}}$$

接受更改。其中 T为“温度”，通过设置合理的初始温度和降温速度，可以有效跳出可能的局部最优解，达到快速收敛的效果。本题通过模拟退火，优化的结果如下：

不可达的交通量：3900.37

总的交通需求量：8037

可通行量：4136.63

最大期望可达率：51.47%

5.3 问题三的模型建立与求解

问题三与问题二类似，但是在简单尝试后发现，道路总容量不能满足所有交通需求。即，记第i条道路的容量为c_i，有

$$\sum_i c_i < \sum_j t_j, (t = Rp, Ap = b) \qquad (5.3.1)$$

因此，并不是所有交通量都会被分配，因此，第二问中的初始条件不可用。但只需将p_i初始值分配为 1，并将交通需求重分配的方式更改为对单个p_i进行随机游走，并添加限制条件

$$\forall i, t_i < c_i$$

和

$$\forall i, s_i \geq 0, s = b - Ap$$

即可使用准确的目标函数进行优化。优化的结果如下：

可通行量 3764.7525

总的交通需求量：8037

最大期望可达率：46.84%

5.4 问题四的模型建立与求解

在问题四中，题目允许新建道路对热点区域进行疏解和优化。题目给出的交通网络中只有四节点环和五节点环。不难发现，在不考虑新建道路总数的情况下，四节点环的新建道路方案有 7 种，五节点环的新建道路方案有 61 种。该交通网络中四节点环有 15 个，五节点环有 3 个，新建道路的方案规模将来到10^{18}量级，即使考虑新建道路的总数，问题的规模仍然无法穷举。因此，本题将采取逐步规划的方式予以解决。

首先，我们尝试定义新建道路的重要性指标：迫切度（eagerness），它由两部分组成，建设道路后期望从道路上经过的车辆数量，称为愿望通行数（expected flow）和环上的通行缺口（shortness）。愿望通行数可以通过所有出行方案中，经过新建道路起始点α和新建道路终点β的所有出行方案在未限制道路容量时的被分配交通量p_{0i}之和计算（具体算法见附录中 Q4.ipynb 中的代码）。而环上的通行缺口则使用未限制道路容量时的交通量t_{0j}与道路容量c_j之差计算。即

$$shortness = \sum_{\{j|t_{0j}-c_j>0\}} t_{0j} - c_j \qquad (5.4.1)$$

新建道路对环上道路的疏解作用为愿望通行数和通行缺口的较小值。因此，我们将迫切度定为二者当中的较小值。即

$$eagerness = \begin{cases} expect\ flow\ (expect\ flow < shortness), \\ shortness\ (expect\ flow > shortness). \end{cases}$$

随后，我们建设迫切度最高的道路并重新运行迫切度评估，以求出在新状态下新建道路的迫切性。重复该过程直至建设五条道路。随后使用问题三中的评估方式对建设方案进行评估。得到的结果如下

表格 2 各方案的可通行量

方案	可通行量
1	3867.62

2	3863.94
3	3878.59
4	3834.21
5	3886.69

六、模型的评价

6.1 模型的优点

首先，我们对问题进行了有效而良好的近似。一是对目标函数进行了良好的近似，巧妙将原来不连续的最大值函数转化连续的目标函数，大幅提高了对相对简单问题的求解效率。同时对目标函数的凸性研究证明了一些情况下使用爬山法或梯度下降法的可行性，大幅提高了求解速度。二是在一些问题需要使用动态规划或其它算法的子问题中，充分论证了使用贪心算法的局限性和近似性，为使用贪心算法替代动态规划，从而提高问题的求解速度上提供了理论依据。

其次是数学方法的使用。在本文中大量使用了成熟数学理论，不仅优化了运算量，还提高了算法的可靠性。对目标函数的性质研究则表明了算法的鲁棒性。

最后是本文对于道路修建迫切度指标的提出，通过评价-修改的方式对道路修建提出一有效指标，该指标的提出便于迁移和现实应用。

6.2 模型的缺点

首先，时间的不充分使得我们无法深入研究目标函数的性质，并未说明所有问中函数的凸性。使得我们依然使用到了模拟退火算法求解，求解速度仍由提升的空间。

其次，函数中使用的贪心算法近似较多，在提高对目标函数性质研究并提高目标函数求解速度之后可以将问题四中的部分影响较大的贪心算法改为动态规划算法，进一步提高模型的准确度。

七、参考文献

[1] 杨爱民. 并行广度优先搜索算法研究[D].西安电子科技大学,2013.

[2] 陆胜锋.计算机模拟退火优化算法监测模型及模拟试验[J].技术与市场,2024,31(03):87-91.

[3] 陈文标. 线性约束的凸二次规划求解算法研究[D].武汉大学,2023.DOI:10.27379/d.cnki.gwhdu.2022.000274.

[4] 尹钊,贾尚晖.Moore-Penrose 广义逆矩阵与线性方程组的解[J].数学的实践与认识,2009,39(09):239-244.

八、附录

8.1 问题相关表格

附录表 1 问题 1 结果

(起点,终点)	规划路径 (依次给出经过的所有节点，例如:1-2-3-6-9)	分配交通量
(1,9)	1-2-3-6-9	61.44821
	1-2-5-6-9	28.3155
	1-2-5-8-9	28.56502
	1-4-5-6-9	30.71088
	1-4-5-8-9	30.9604
(3,7)	3-2-1-4-7	53.94059
	3-2-5-4-7	18.08653
	3-2-5-8-7	23.44498
	3-6-5-4-7	19.58473
	3-6-5-8-7	24.94317

附录表 2 问题 2 结果

(起点,终点)	规划路径 (依次给出经过的所有节点，例如： 27-35-41-10-16-36-6)	分配交通量
(27,6)	27-35-28-8-9-5-6	27.54576
	27-35-41-8-9-5-6	18.90434
	27-35-41-10-9-5-6	47.79997
	27-35-41-10-16-5-6	13.74438
	27-35-41-10-16-36-6	13.00555
(19,25)	19-10-11-12-7-33-25	9.998465
	19-10-41-35-27-30-25	90.91459
	19-10-41-35-27-33-25	8.38851
	19-20-11-12-7-33-25	5.698437
	/	/

附录表 3 问题 3 结果

(起点,终点)	规划路径 (依次给出经过的所有节点，例如： 27-35-41-10-16-36-6)	分配交通量
(32,39)	32-24-7-12-11-10-16-18-2-15-13-39	32.67651
	32-24-7-12-11-10-16-36-3-42-38-39	2.000417
	32-24-7-12-11-10-19-18-2-15-13-39	2.30175
	32-24-7-12-11-20-19-18-2-15-13-39	6.753713
	/	/
(17,8)	17-14-13-15-2-18-16-5-9-8	2.803369
	17-14-13-15-2-18-16-10-9-8	4.499651
	17-14-13-15-2-18-16-10-41-8	0.243859
	17-14-13-15-2-18-19-10-9-8	0.356546
	17-14-13-15-2-18-19-10-41-8	72.94604

附录表 4 问题 4 结果

	新建路段 1 (给出新建路段的起点和终点，例如 9-16)	新建路段 2 (给出新建路段的起点和终点，例如 9-16)	新建路段 3 (给出新建路段的起点和终点，例如 9-16)	新建路段 4 (给出新建路段的起点和终点，例如 9-16)	新建路段 5 (给出新建路段的起点和终点，例如 9-16)	新建路段 6 (给出新建路段的起点和终点，例如 9-16)	可达率
方案 1	35-33	35-8	32-7	18-10	42-2	16-2	48.123%
方案 2	35-33	35-8	32-7	18-10	42-2	38-3	48.077%
方案 3	35-33	35-8	32-7	18-10	42-2	5-10	48.259%
方案 4	35-33	35-8	32-7	18-10	42-2	10-18	47.707%
方案 5	35-33	35-8	32-7	18-10	42-2	3-6	48.360%

8.2 程序附录

问题一求解过程：Q1. ipynb（6 页）

问题二求解过程：Q2. ipynb（17 页）

问题三求解过程：Q3. ipynb（17 页）

问题四求解过程：Q4. ipynb（24 页）

Q1

2024 年 5 月 4 日

#2024 51MCM-B 题 #

首先创建路径邻接矩阵，1 代表联通，0 代表断开。节点与自身不连通。然后使用广度优先搜索，搜索出符合条件的五条路径

[ ]:
```
import numpy as np

matrix = np.array([[0, 1, 0, 1, 0, 0, 0, 0, 0],[1, 0, 1, 0, 1, 0, 0, 0, 0],[0,␣
↪1, 0, 0, 0, 1, 0, 0, 0],[1, 0, 0, 0, 1, 0, 1, 0, 0],[0, 1, 0, 1, 0, 1, 0, 1,␣
↪0],[0, 0, 1, 0, 1, 0, 0, 0, 1],[0, 0, 0, 1, 0, 0, 0, 1, 0],[0, 0, 0, 0, 1,␣
↪0, 1, 0, 1],[0, 0, 0, 0, 0, 1, 0, 1, 0]])
#print(matrix)

from collections import deque

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, node, neighbors):
        self.graph[node] = neighbors

class ResizableList(object):
    def __init__(self, initial_size=90):
        self.data = [0] * initial_size
        self.size = 0

    def __getitem__(self, index):
        if index >= self.size:
```

```
            self._resize(index + 1)
        return self.data[index]

    def __setitem__(self, index, value):
        if index >= self.size:
            self._resize(index + 1)
        self.data[index] = value
        if index >= self.size:
            self.size = index + 1

    def _resize(self, new_size):
        self.data += [0] * (new_size - len(self.data))
```

[ ]:
```
g = Graph()
g.add_edge(1, [2,4])
g.add_edge(2, [1,3,5])
g.add_edge(3, [2,6])
g.add_edge(4, [1,5,7])
g.add_edge(5, [2,4,6,8])
g.add_edge(6, [3,5,9])
g.add_edge(7, [4,8])
g.add_edge(8, [5,7,9])
g.add_edge(9, [6,8])

def bfs(graph, start, target):
    visited = []
    paths = []
    queue = [[start]]
    if start == target:
        return "Start = target"
    while queue:
        path = queue.pop(0)
        node = path[-1]
        neighbours = graph[node]
        for neighbour in neighbours:
            new_path = list(path)
```

```
            new_path.append(neighbour)
            queue.append(new_path)
            if neighbour == target:
                if len(paths) >=5:return paths # 更严格的限制
                if len(paths) >= 1:
                    if len(new_path) > len(paths[-1]):
                        return paths
                paths.append(new_path)
        visited.append(node)
    return "No path"

print(g.graph)
paths = bfs(g.graph, 3, 7)
print(paths)
```

[ ]:
```
plans = [[1,5,100],
[1,6,150],
[1,8,200],
[1,9,180],
[2,4,160],
[2,6,110],
[2,7,120],
[2,9,140],
[3,4,130],
[3,5,150],
[3,7,140],
[3,8,160],
[4,2,100],
[4,3,170],
[4,8,190],
[4,9,120],
[5,1,160],
[5,3,140],
[5,7,180],
[5,9,100],
[6,1,170],
```

```
[6,2,120],
[6,7,130],
[6,8,200],
[7,2,120],
[7,3,150],
[7,5,190],
[7,6,100],
[8,1,130],
[8,3,180],
[8,4,140],
[8,6,150],
[9,1,170],
[9,2,130],
[9,4,190],
[9,5,160]]

def routenum(i,j):
    return 9 * i + j

def relationmat(route):
    relmat = ResizableList()
    while route:
        if len(route) == 1:
            break
        i = route.pop(0)
        j = route[0]
        relmat[routenum(i,j)] = 1
    return relmat


relmat = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    plan.append(len(routes))
```

```
    for route in routes:
        relmat.append(relationmat(route).data)

nprelmat = np.array(relmat)
nprelmat = nprelmat.T
print(nprelmat)

routess = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    routess.extend(routes)
```

[ ]:
```
%%time

Q = nprelmat.T @ nprelmat
print(Q)

A = []
b = []
i = 0
for plan in plans:
    A.append(np.zeros(i).tolist() + np.ones(plan[3]).tolist() + np.zeros(len(Q)␣
 ↪- plan[3] - i).tolist())
    b.append(plan[2])
    i += plan[3]

A = np.array(A)
print(A)

# 此处论文中的方法不可用，因为原来这个矩阵就已经是最简行阶梯了，前面的 N 列不是满秩
的，所以无法求逆
P = np.append(np.append(Q, -A.T, axis=1), np.append(A, np.zeros((len(A),␣
 ↪len(A))), axis=1), axis=0)
print(P)
```

```
np.linalg.matrix_rank(P)
#res = np.linalg.inv(P) @ np.append(np.zeros(len(Q)), b)
#print(res)
res=np.linalg.pinv(P)@np.append(np.zeros(len(Q)), b)
res = res[:len(Q)]
print(max(nprelmat @ res))
'''
##restable = []
restable.append(routess)
restable.append(res)
restable = np.array(restable)
print(res)
'''
```

[ ]:
```
import csv
# 保存列表到 CSV 文件
with open("E:\\Q1_result_path.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])


#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q1_result_res.csv", res, fmt='%f', delimiter=',')
```

Q2

2024 年 5 月 4 日

#2024 51MCM-B 题 #

首先创建路径邻接矩阵，1 代表联通，0 代表断开。节点与自身不连通。然后使用广度优先搜索，搜索出符合条件的五条路径

[ ]:
```
import numpy as np
from collections import deque
import copy

matrix = np.array([[0, 1, 0, 1, 0, 0, 0, 0, 0],[1, 0, 1, 0, 1, 0, 0, 0, 0],[0,␣
↪1, 0, 0, 0, 1, 0, 0, 0],[1, 0, 0, 0, 1, 0, 1, 0, 0],[0, 1, 0, 1, 0, 1, 0, 1,␣
↪0],[0, 0, 1, 0, 1, 0, 0, 0, 1],[0, 0, 0, 1, 0, 0, 0, 1, 0],[0, 0, 0, 0, 1,␣
↪0, 1, 0, 1],[0, 0, 0, 0, 0, 1, 0, 1, 0]])
#print(matrix)



class Graph:
    def __init__(self):
        self.graph = {0:[]}

    def add_edge(self, node, neighbors):
        self.graph[node]= self.graph.get(node,[]) + [neighbors]

class ResizableList(object):
    def __init__(self, initial_size=1722):
        self.data = [0] * initial_size
        self.size = 0

    def __getitem__(self, index):
```

```
        if index >= self.size:
            self._resize(index + 1)
        return self.data[index]

    def __setitem__(self, index, value):
        if index >= self.size:
            self._resize(index + 1)
        self.data[index] = value
        if index >= self.size:
            self.size = index + 1

    def _resize(self, new_size):
        self.data += [0] * (new_size - len(self.data))
```

[ ]:
```
g = Graph()
g.add_edge(0,1)
g.add_edge(0,3)
g.add_edge(1,0)
g.add_edge(1,4)
g.add_edge(2,3)
g.add_edge(2,18)
g.add_edge(3,0)
g.add_edge(3,2)
g.add_edge(3,36)
g.add_edge(3,42)
g.add_edge(4,1)
g.add_edge(4,6)
g.add_edge(5,6)
g.add_edge(5,9)
g.add_edge(5,16)
g.add_edge(5,22)
g.add_edge(6,4)
g.add_edge(6,5)
g.add_edge(6,34)
g.add_edge(7,12)
g.add_edge(7,24)
```

```
g.add_edge(7,33)
g.add_edge(8,9)
g.add_edge(8,28)
g.add_edge(8,37)
g.add_edge(8,41)
g.add_edge(9,5)
g.add_edge(9,8)
g.add_edge(9,10)
g.add_edge(10,9)
g.add_edge(10,11)
g.add_edge(10,16)
g.add_edge(10,19)
g.add_edge(11,10)
g.add_edge(11,12)
g.add_edge(11,20)
g.add_edge(11,41)
g.add_edge(12,7)
g.add_edge(12,11)
g.add_edge(12,35)
g.add_edge(13,14)
g.add_edge(13,15)
g.add_edge(13,39)
g.add_edge(14,13)
g.add_edge(14,17)
g.add_edge(14,21)
g.add_edge(15,13)
g.add_edge(15,42)
g.add_edge(16,5)
g.add_edge(16,10)
g.add_edge(16,18)
g.add_edge(16,36)
g.add_edge(17,14)
g.add_edge(17,39)
g.add_edge(18,2)
g.add_edge(18,16)
g.add_edge(18,19)
```

```
g.add_edge(18,23)
g.add_edge(19,10)
g.add_edge(19,18)
g.add_edge(19,20)
g.add_edge(19,23)
g.add_edge(20,11)
g.add_edge(20,19)
g.add_edge(21,14)
g.add_edge(22,5)
g.add_edge(22,34)
g.add_edge(22,40)
g.add_edge(23,18)
g.add_edge(23,19)
g.add_edge(24,7)
g.add_edge(24,32)
g.add_edge(25,30)
g.add_edge(25,33)
g.add_edge(26,31)
g.add_edge(26,33)
g.add_edge(27,30)
g.add_edge(27,35)
g.add_edge(28,8)
g.add_edge(28,35)
g.add_edge(29,31)
g.add_edge(29,32)
g.add_edge(30,25)
g.add_edge(30,27)
g.add_edge(31,26)
g.add_edge(31,29)
g.add_edge(32,24)
g.add_edge(32,29)
g.add_edge(33,7)
g.add_edge(33,25)
g.add_edge(33,26)
g.add_edge(34,6)
g.add_edge(34,22)
```

```
g.add_edge(34,40)
g.add_edge(35,12)
g.add_edge(35,27)
g.add_edge(35,28)
g.add_edge(36,3)
g.add_edge(36,16)
g.add_edge(37,8)
g.add_edge(38,39)
g.add_edge(38,42)
g.add_edge(39,13)
g.add_edge(39,17)
g.add_edge(39,38)
g.add_edge(40,22)
g.add_edge(40,34)
g.add_edge(41,8)
g.add_edge(41,11)
g.add_edge(42,3)
g.add_edge(42,15)
g.add_edge(42,38)
g.add_edge(27,33)
g.add_edge(33,27)
g.add_edge(26,32)
g.add_edge(32,26)
g.add_edge(6,0)
g.add_edge(0,6)
g.add_edge(21,17)
g.add_edge(17,21)
g.add_edge(37,28)
g.add_edge(28,37)
g.add_edge(35,41)
g.add_edge(41,35)
g.add_edge(6,36)
g.add_edge(36,6)
g.add_edge(41,10)
g.add_edge(10,41)
g.add_edge(38,13)
```

```
g.add_edge(13,38)
g.add_edge(2,15)
g.add_edge(15,2)
print(g.graph)

def bfs(graph, start, target):
    visited = []
    paths = []
    queue = [[start]]
    if start == target:
        return "Start = target"
    while queue:
        path = queue.pop(0)
        node = path[-1]
        neighbours = graph[node]
        for neighbour in neighbours:
            if(neighbour in path):
                continue
            new_path = list(path)
            new_path.append(neighbour)
            queue.append(new_path)
            if neighbour == target:
                if len(paths) >=5:return paths # 更严格的限制
                if len(paths) >= 1:
                    if len(new_path) > len(paths[0]):
                        return paths
                paths.append(new_path)
        visited.append(node)
    return "No path"

print(g.graph)
paths = bfs(g.graph, 3, 7)
print(paths)
```

[ ]:
```
plans = [[17,6,92],
[22,4,113],
```

```
[6,22,101],
[13,26,84],
[11,28,87],
[4,17,62],
[26,15,25],
[27,26,65],
[23,24,49],
[17,29,64],
[19,33,42],
[23,10,102],
[32,34,61],
[22,10,97],
[32,23,113],
[36,18,80],
[27,8,100],
[9,34,114],
[28,20,107],
[27,40,52],
[14,17,46],
[38,33,85],
[26,40,88],
[35,23,61],
[21,19,121],
[32,20,103],
[9,31,102],
[14,6,58],
[7,16,65],
[33,35,55],
[29,9,24],
[20,25,66],
[10,40,64],
[41,7,65],
[17,8,122],
[39,30,61],
[27,17,66],
[25,20,66],
```

```
[38,20,91],
[27,6,121],
[6,3,63],
[7,26,89],
[35,22,93],
[40,27,90],
[12,24,98],
[32,39,134],
[21,38,95],
[31,15,82],
[28,40,74],
[2,19,63],
[17,29,84],
[8,37,92],
[37,33,98],
[24,28,101],
[5,40,51],
[38,33,103],
[23,31,110],
[15,20,122],
[13,23,66],
[2,22,72],
[19,23,85],
[15,1,84],
[39,11,54],
[39,29,33],
[40,8,83],
[34,18,86],
[13,4,59],
[23,2,55],
[19,25,115],
[21,13,91],
[18,15,77],
[4,15,64],
[0,5,75],
[23,32,65],
```

```
[7,14,89],
[12,3,86],
[40,10,88],
[27,17,52],
[38,14,108],
[24,21,80],
[2,4,85],
[7,26,63],
[8,13,86],
[21,36,82],
[35,39,85],
[4,22,78],
[18,37,63],
[23,16,100],
[8,14,116],
[0,17,73],
[17,0,79],
[0,35,54],
[21,40,103],
[13,23,112],
[28,24,72],
[39,20,97],
[41,21,49],
[7,39,80],
[10,13,47],
[26,28,64]]

def routenum(i,j):
    return 41 * i + j -42

def relationmat(route):
    relmat = ResizableList()
    while route:
        if len(route) == 1:
            break
        i = route.pop(0)
```

```
        j = route[0]
        relmat[routenum(i,j)] = 1
    return relmat

relmat = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    plan.append(len(routes))
    print(len(relmat))
    for route in routes:
        relmat.append(relationmat(route).data)

nprelmat = np.array(relmat)
nprelmat = nprelmat.T
print(nprelmat)
```

[ ]:
```
routess = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    routess.extend(routes)
```

下面使用目标函数 min=^2 进行梯度下降优化

[ ]:
```
%%time

Q = nprelmat.T @ nprelmat
print(Q)

A = []
b = []
i = 0
for plan in plans:
```

```
    A.append(np.zeros(i).tolist() + np.ones(plan[3]).tolist() + np.zeros(len(Q)␣
 ↪- plan[3] - i).tolist())
    b.append(plan[2])
    i += plan[3]

A = np.array(A)
print(A)

# 此处论文中的方法不可用，因为原来这个矩阵就已经是最简行阶梯了，前面的 N 列不是满秩
的，所以无法求逆
P = np.append(np.append(Q, -A.T, axis=1), np.append(A, np.zeros((len(A),␣
 ↪len(A))), axis=1), axis=0)
print(P)

np.linalg.matrix_rank(P)
#res = np.linalg.inv(P) @ np.append(np.zeros(len(Q)), b)
#print(res)
res=np.linalg.pinv(P)@np.append(np.zeros(len(Q)), b)
res = res[:len(Q)]
print("result is:", res)
print("stream:", [x for x in nprelmat @ res if x != 0])
print("Max stream:", max([x for x in nprelmat @ res if x != 0]))
```

[ ]:
```
#np.savetxt("E:\\p.csv", res, delimiter=',')
```

[ ]:
```
def evaluate_max_influence(R, p):
    originload = sum(p)
    for k in range(5):
        t = R @ p
        a, _ = max(enumerate(t), key=lambda x: x[1])
        for j in range(len(R[a])):
            if R[a,j] != 0:
                p[j] = 0
    return originload - sum(p)
```

接下来使用爬山算法进行优化

[ ]:
```
# 首先平均分配交通量
p = np.zeros(len(Q))
i = 0
for plan in plans:
    p[i:i + plan[3]] = plan[2]/plan[3]
    plan.append(i)
    plan.append(i + plan[3])
    i += plan[3]
```

[ ]:
```
%%time
import random
latestmax = p.T @ Q @ p
for i in range(100000):
    plan_tmp = plans[random.randint(0, len(plans) - 1)]
    fromchange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    tochange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    changeamount = 0.5 * (random.random() - 0.5) * max(p[fromchange],␣
 ↪p[tochange])
    if p[tochange] + changeamount < 0 or p[fromchange] - changeamount < 0:␣
 ↪continue
    p[fromchange] -= changeamount
    p[tochange] += changeamount
    tmpmax = p.T @ Q @ p
    if tmpmax > latestmax:
        p[fromchange] += changeamount
        p[tochange] -= changeamount
        continue
    if tmpmax < latestmax:
        latestmax = tmpmax
        print("cycle:", i, "max:", evaluate_max_influence(nprelmat, copy.
 ↪deepcopy(p)))
```

[ ]:
```
Changerate = 2
```

[ ]:
```
%%time
latestmax = evaluate_max_influence(nprelmat, copy.deepcopy(p))
```

```
for i in range(10000):
    Changerate = Changerate * 0.9999
    plan_tmp = plans[random.randint(0, len(plans) - 1)]
    fromchange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    tochange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    changeamount = Changerate * (random.random() - 0.5) *  max(p[fromchange],␣
 ↪p[tochange])
    if p[tochange] + changeamount < 0 or p[fromchange] - changeamount < 0:␣
 ↪continue
    p[fromchange] -= changeamount
    p[tochange] += changeamount
    tmpmax = evaluate_max_influence(nprelmat, copy.deepcopy(p))
    if tmpmax > latestmax:
        p[fromchange] += changeamount
        p[tochange] -= changeamount
        continue
    if tmpmax < latestmax:
        latestmax = tmpmax
        print("cycle:", i, "max:", evaluate_max_influence(nprelmat, copy.
 ↪deepcopy(p)))
```

[ ]:
```
import csv
# 保存列表到 CSV 文件
with open("E:\\Q2_result_path_notoptimized.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])


#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q2_result_res_notoptimized.csv", p, fmt='%f', delimiter=',')
```

[ ]:
```
restable = []
restable.append(routess)
restable.append(p)
```

```
print(p)
```

[ ]:
```
def largest_n_indices(lst, n):
    return sorted(range(len(lst)), key=lambda x: lst[x])[-n:]
def smallest_n_indices(lst, n):
    return sorted(range(len(lst)), key=lambda x: lst[x])[:n]
```

[ ]:
```
routess = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    routess.extend(routes)


p = p.tolist()
```

[ ]:
```
i = 0
for plan in plans:
    if plan[3] > 5:
        #j = sorted(largest_n_indices(p[i:i + plan[3]], plan[3] - 5),␣
 ↪reverse=True)
        j = sorted(np.random.randint(0, plan[3] - 1, plan[3] - 5), reverse=True)
        #raise ValueError("error")
        for n in j:
            routess.pop(n + i)
            p.pop(n + i)
            print(n + i)
        plan[3] = 5
        #raise ValueError("error")
    plan[4] = i
    i = i + plan[3]
    plan[5] = i
'''
relmat = []
for route in routess:
    relmat.append(relationmat(route).data)
'''
```

```
nprelmat = np.array(relmat)
nprelmat = nprelmat.T
Q = nprelmat.T @ nprelmat
```

[ ]:
```
# 首先平均分配交通量
p = np.zeros(len(Q))
i = 0
for plan in plans:
    p[i:i + plan[3]] = plan[2]/plan[3]
    i += plan[3]
```

[ ]:
```
%%time
import random
latestmax = p.T @ Q @ p
for i in range(100000):
    plan_tmp = plans[random.randint(0, len(plans) - 1)]
    fromchange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    tochange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    changeamount = 0.5 * (random.random() - 0.5) * max(p[fromchange],␣
 ↪p[tochange])
    if p[tochange] + changeamount < 0 or p[fromchange] - changeamount < 0:␣
 ↪continue
    p[fromchange] -= changeamount
    p[tochange] += changeamount
    tmpmax = p.T @ Q @ p
    if tmpmax > latestmax:
        p[fromchange] += changeamount
        p[tochange] -= changeamount
        continue
    if tmpmax < latestmax:
        latestmax = tmpmax
        print("cycle:", i, "max:", evaluate_max_influence(nprelmat, copy.
 ↪deepcopy(p)))
```

[ ]:
```
Changerate = 0.5
```

[ ]:
```
Changerate = 2
```

[ ]:
```
%%time
latestmax = evaluate_max_influence(nprelmat, copy.deepcopy(p))
for i in range(10000):
    Changerate = Changerate * 0.9999
    plan_tmp = plans[random.randint(0, len(plans) - 1)]
    fromchange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    tochange = random.randint(plan_tmp[4], plan_tmp[5] - 1)
    changeamount = Changerate * (random.random() - 0.5) *  max(p[fromchange],␣
 ↪p[tochange])
    if p[tochange] + changeamount < 0 or p[fromchange] - changeamount < 0:␣
 ↪continue
    p[fromchange] -= changeamount
    p[tochange] += changeamount
    tmpmax = evaluate_max_influence(nprelmat, copy.deepcopy(p))
    if tmpmax > latestmax:
        p[fromchange] += changeamount
        p[tochange] -= changeamount
        continue
    if tmpmax < latestmax:
        latestmax = tmpmax
        print("cycle:", i, "max:", evaluate_max_influence(nprelmat, copy.
 ↪deepcopy(p)))
```

[ ]:
```
import csv
# 保存列表到 CSV 文件
with open("E:\\Q2_result_path.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])



#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q2_result_res.csv", p, fmt='%f', delimiter=',')
```

[ ]:
```
load = [x for x in nprelmat @ p if x != 0]
print("stream:", load)
import copy
print(evaluate_max_influence(nprelmat, copy.deepcopy(p)))
```

Q3

2024 年 5 月 4 日

#2024 51MCM-B 题 #

首先创建路径邻接矩阵，1 代表联通，0 代表断开。节点与自身不连通。然后使用广度优先搜索，搜索出符合条件的五条路径

[ ]:
```
import numpy as np
from collections import deque
import copy

matrix = np.array([[0, 1, 0, 1, 0, 0, 0, 0, 0],[1, 0, 1, 0, 1, 0, 0, 0, 0],[0,␣
↪1, 0, 0, 0, 1, 0, 0, 0],[1, 0, 0, 0, 1, 0, 1, 0, 0],[0, 1, 0, 1, 0, 1, 0, 1,␣
↪0],[0, 0, 1, 0, 1, 0, 0, 0, 1],[0, 0, 0, 1, 0, 0, 0, 1, 0],[0, 0, 0, 0, 1,␣
↪0, 1, 0, 1],[0, 0, 0, 0, 0, 1, 0, 1, 0]])
#print(matrix)



class Graph:
    def __init__(self):
        self.graph = {0:[]}

    def add_edge(self, node, neighbors):
        self.graph[node]= self.graph.get(node,[]) + [neighbors]

class ResizableList(object):
    def __init__(self, initial_size=1764):
        self.data = [0] * initial_size
        self.size = 0

    def __getitem__(self, index):
```

```
        if index >= self.size:
            self._resize(index + 1)
        return self.data[index]

    def __setitem__(self, index, value):
        if index >= self.size:
            self._resize(index + 1)
        self.data[index] = value
        if index >= self.size:
            self.size = index + 1

    def _resize(self, new_size):
        self.data += [0] * (new_size - len(self.data))
```

[ ]:
```
g = Graph()
g.add_edge(0,1)
g.add_edge(0,3)
g.add_edge(1,0)
g.add_edge(1,4)
g.add_edge(2,3)
g.add_edge(2,18)
g.add_edge(3,0)
g.add_edge(3,2)
g.add_edge(3,36)
g.add_edge(3,42)
g.add_edge(4,1)
g.add_edge(4,6)
g.add_edge(5,6)
g.add_edge(5,9)
g.add_edge(5,16)
g.add_edge(5,22)
g.add_edge(6,4)
g.add_edge(6,5)
g.add_edge(6,34)
g.add_edge(7,12)
g.add_edge(7,24)
```

```
g.add_edge(7,33)
g.add_edge(8,9)
g.add_edge(8,28)
g.add_edge(8,37)
g.add_edge(8,41)
g.add_edge(9,5)
g.add_edge(9,8)
g.add_edge(9,10)
g.add_edge(10,9)
g.add_edge(10,11)
g.add_edge(10,16)
g.add_edge(10,19)
g.add_edge(11,10)
g.add_edge(11,12)
g.add_edge(11,20)
g.add_edge(11,41)
g.add_edge(12,7)
g.add_edge(12,11)
g.add_edge(12,35)
g.add_edge(13,14)
g.add_edge(13,15)
g.add_edge(13,39)
g.add_edge(14,13)
g.add_edge(14,17)
g.add_edge(14,21)
g.add_edge(15,13)
g.add_edge(15,42)
g.add_edge(16,5)
g.add_edge(16,10)
g.add_edge(16,18)
g.add_edge(16,36)
g.add_edge(17,14)
g.add_edge(17,39)
g.add_edge(18,2)
g.add_edge(18,16)
g.add_edge(18,19)
```

```
g.add_edge(18,23)
g.add_edge(19,10)
g.add_edge(19,18)
g.add_edge(19,20)
g.add_edge(19,23)
g.add_edge(20,11)
g.add_edge(20,19)
g.add_edge(21,14)
g.add_edge(22,5)
g.add_edge(22,34)
g.add_edge(22,40)
g.add_edge(23,18)
g.add_edge(23,19)
g.add_edge(24,7)
g.add_edge(24,32)
g.add_edge(25,30)
g.add_edge(25,33)
g.add_edge(26,31)
g.add_edge(26,33)
g.add_edge(27,30)
g.add_edge(27,35)
g.add_edge(28,8)
g.add_edge(28,35)
g.add_edge(29,31)
g.add_edge(29,32)
g.add_edge(30,25)
g.add_edge(30,27)
g.add_edge(31,26)
g.add_edge(31,29)
g.add_edge(32,24)
g.add_edge(32,29)
g.add_edge(33,7)
g.add_edge(33,25)
g.add_edge(33,26)
g.add_edge(34,6)
g.add_edge(34,22)
```

```
g.add_edge(34,40)
g.add_edge(35,12)
g.add_edge(35,27)
g.add_edge(35,28)
g.add_edge(36,3)
g.add_edge(36,16)
g.add_edge(37,8)
g.add_edge(38,39)
g.add_edge(38,42)
g.add_edge(39,13)
g.add_edge(39,17)
g.add_edge(39,38)
g.add_edge(40,22)
g.add_edge(40,34)
g.add_edge(41,8)
g.add_edge(41,11)
g.add_edge(42,3)
g.add_edge(42,15)
g.add_edge(42,38)
g.add_edge(27,33)
g.add_edge(33,27)
g.add_edge(26,32)
g.add_edge(32,26)
g.add_edge(6,0)
g.add_edge(0,6)
g.add_edge(21,17)
g.add_edge(17,21)
g.add_edge(37,28)
g.add_edge(28,37)
g.add_edge(35,41)
g.add_edge(41,35)
g.add_edge(6,36)
g.add_edge(36,6)
g.add_edge(41,10)
g.add_edge(10,41)
g.add_edge(38,13)
```

```
g.add_edge(13,38)
g.add_edge(2,15)
g.add_edge(15,2)
print(g.graph)

def bfs(graph, start, target):
    visited = []
    paths = []
    queue = [[start]]
    if start == target:
        return "Start = target"
    while queue:
        path = queue.pop(0)
        node = path[-1]
        neighbours = graph[node]
        for neighbour in neighbours:
            if(neighbour in path):
                continue
            new_path = list(path)
            new_path.append(neighbour)
            queue.append(new_path)
            if neighbour == target:
                if len(paths) >=5:return paths # 更严格的限制

                if len(paths) >= 1:
                    if len(new_path) > len(paths[0]):
                        return paths

                paths.append(new_path)
        visited.append(node)
    return "No path"

print(g.graph)
paths = bfs(g.graph, 3, 7)
print(paths)
```

[ ]:
```
plans = [[17,6,92],
[22,4,113],
[6,22,101],
[13,26,84],
[11,28,87],
[4,17,62],
[26,15,25],
[27,26,65],
[23,24,49],
[17,29,64],
[19,33,42],
[23,10,102],
[32,34,61],
[22,10,97],
[32,23,113],
[36,18,80],
[27,8,100],
[9,34,114],
[28,20,107],
[27,40,52],
[14,17,46],
[38,33,85],
[26,40,88],
[35,23,61],
[21,19,121],
[32,20,103],
[9,31,102],
[14,6,58],
[7,16,65],
[33,35,55],
[29,9,24],
[20,25,66],
[10,40,64],
[41,7,65],
[17,8,122],
[39,30,61],
```

```
[27,17,66],
[25,20,66],
[38,20,91],
[27,6,121],
[6,3,63],
[7,26,89],
[35,22,93],
[40,27,90],
[12,24,98],
[32,39,134],
[21,38,95],
[31,15,82],
[28,40,74],
[2,19,63],
[17,29,84],
[8,37,92],
[37,33,98],
[24,28,101],
[5,40,51],
[38,33,103],
[23,31,110],
[15,20,122],
[13,23,66],
[2,22,72],
[19,23,85],
[15,1,84],
[39,11,54],
[39,29,33],
[40,8,83],
[34,18,86],
[13,4,59],
[23,2,55],
[19,25,115],
[21,13,91],
[18,15,77],
[4,15,64],
```

```
[0,5,75],
[23,32,65],
[7,14,89],
[12,3,86],
[40,10,88],
[27,17,52],
[38,14,108],
[24,21,80],
[2,4,85],
[7,26,63],
[8,13,86],
[21,36,82],
[35,39,85],
[4,22,78],
[18,37,63],
[23,16,100],
[8,14,116],
[0,17,73],
[17,0,79],
[0,35,54],
[21,40,103],
[13,23,112],
[28,24,72],
[39,20,97],
[41,21,49],
[7,39,80],
[10,13,47],
[26,28,64]]

l_org = [[0,1,315],
[0,3,393],
[1,0,200],
[1,4,231],
[2,3,667],
[2,18,790],
[3,0,615],
```

```
[3,2,424],
[3,36,730],
[3,42,618],
[4,1,200],
[4,6,246],
[5,6,405],
[5,9,545],
[5,16,282],
[5,22,529],
[6,4,200],
[6,5,403],
[6,34,520],
[7,12,829],
[7,24,512],
[7,33,395],
[8,9,517],
[8,28,321],
[8,37,200],
[8,41,276],
[9,5,678],
[9,8,392],
[9,10,389],
[10,9,205],
[10,11,495],
[10,16,574],
[10,19,479],
[11,10,541],
[11,12,561],
[11,20,606],
[11,41,261],
[12,7,758],
[12,11,707],
[12,35,519],
[13,14,419],
[13,15,868],
[13,39,227],
```

```
[14,13,513],
[14,17,200],
[14,21,200],
[15,13,598],
[15,42,200],
[16,5,328],
[16,10,554],
[16,18,473],
[16,36,469],
[17,14,290],
[17,39,517],
[18,2,635],
[18,16,576],
[18,19,497],
[18,23,397],
[19,10,453],
[19,18,388],
[19,20,522],
[19,23,506],
[20,11,240],
[20,19,368],
[21,14,210],
[22,5,377],
[22,34,305],
[22,40,230],
[23,18,480],
[23,19,467],
[24,7,472],
[24,32,325],
[25,30,200],
[25,33,316],
[26,31,237],
[26,33,442],
[27,30,350],
[27,35,831],
[28,8,371],
```

```
[28,35,418],
[29,31,200],
[29,32,200],
[30,25,289],
[30,27,200],
[31,26,200],
[31,29,200],
[32,24,323],
[32,29,266],
[33,7,200],
[33,25,293],
[33,26,505],
[34,6,279],
[34,22,270],
[34,40,340],
[35,12,410],
[35,27,767],
[35,28,606],
[36,3,501],
[36,16,557],
[37,8,272],
[38,39,407],
[38,42,821],
[39,13,375],
[39,17,315],
[39,38,406],
[40,22,247],
[40,34,200],
[41,8,280],
[41,11,494],
[42,3,974],
[42,15,200],
[42,38,460],
[27,33,442],
[33,27,340],
[26,32,200],
```

```
[32,26,289],
[6,0,226],
[0,6,370],
[21,17,321],
[17,21,200],
[37,28,200],
[28,37,272],
[35,41,637],
[41,35,521],
[6,36,200],
[36,6,298],
[41,10,475],
[10,41,483],
[38,13,365],
[13,38,434],
[2,15,688],
[15,2,921]]



def routenum(i,j):
    return 41 * i + j - 1

def relationmat(route):
    relmat = ResizableList()
    while route:
        if len(route) == 1:
            break
        i = route.pop(0)
        j = route[0]
        relmat[routenum(i,j)] = 1
    return relmat
```

[ ]:
```
l = np.zeros(1764)
for l_i in l_org:
```

```
    l[routenum(l_i[0],l_i[1])] = l_i[2]



relmat = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    plan.append(len(routes))
    print(len(relmat))
    for route in routes:
        relmat.append(relationmat(route).data)

nprelmat = np.array(relmat)
nprelmat = nprelmat.T
print(nprelmat)
```

[ ]:
```
routess = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    routess.extend(routes)
```

[ ]:
```
%%time

Q = nprelmat.T @ nprelmat
print(Q)

A = []
b = []
i = 0
for plan in plans:
    A.append(np.zeros(i).tolist() + np.ones(plan[3]).tolist() + np.zeros(len(Q)␣
 ↪- plan[3] - i).tolist())
    b.append(plan[2])
    i += plan[3]
```

```
A = np.array(A)
print(A)

# 此处论文中的方法不可用，因为原来这个矩阵就已经是最简行阶梯了，前面的 N 列不是满秩的，所以无法求逆
P = np.append(np.append(Q, -A.T, axis=1), np.append(A, np.zeros((len(A),
 ↪len(A))), axis=1), axis=0)
print(P)

np.linalg.matrix_rank(P)
#res = np.linalg.inv(P) @ np.append(np.zeros(len(Q)), b)
#print(res)
res=np.linalg.pinv(P)@np.append(np.zeros(len(Q)), b)
res = res[:len(Q)]
print("result is:", res)
print("stream:", [x for x in nprelmat @ res if x != 0])
print("Max stream:", max([x for x in nprelmat @ res if x != 0]))
```

[ ]:
```
#np.savetxt("E:\\p.csv", res, delimiter=',')
```

[ ]:
```
def evaluate_max_influence(R, p):
    originload = sum(p)
    for k in range(5):
        t = R @ p
        a, _ = max(enumerate(t), key=lambda x: x[1])
        for j in range(len(R[a])):
            if R[a,j] != 0:
                p[j] = 0
    return originload - sum(p)
```

[ ]:
```
# 首先平均分配交通量
p = np.ones(len(Q))
i = 0
for plan in plans:
    p[i:i + plan[3]] = plan[2]/plan[3]
    plan.append(i)
```

```
    plan.append(i + plan[3])
    i += plan[3]
```

[ ]:
```
%%time
import random
import math
def list_exp(l):
    return [math.exp(x) for x in l]
latestmax = sum(list_exp(- l + nprelmat @ p))
p_best = np.ones(len(Q))
```

[ ]:
```
%%time
import random
p = p_best
latestmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))
max_best = latestmax
T = 0.01
for i in range(100000):
    fromchange = random.randint(0,len(p) - 1)
    changeamount = 0.5 * (random.random() - 0.5) * (p[fromchange]+1)
    if p[fromchange] + changeamount < 0:
        continue

    p[fromchange] += changeamount

    if min(l - nprelmat @ p) < 0:
        p[fromchange] -= changeamount
        continue

    if max(A @ p - b) > 0:
        p[fromchange] -= changeamount
        continue
    tmpmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))
    if random.random() > np.exp((tmpmax - latestmax) / T):
        p[fromchange] -= changeamount
        latestmax = tmpmax
        print("cycle:", i, "tmp:", tmpmax)
```

```
        continue
    if tmpmax > latestmax:
        latestmax = tmpmax
        print("cycle:", i, "max:", latestmax)

        if latestmax > max_best:
            max_best = latestmax
            p_best = copy.deepcopy(p)
    T = 0.999 * T
print("result is:", max_best)
```

[ ]:
```
import csv
# 保存列表到 CSV 文件
with open("E:\\Q3_result_path_notoptimized.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])


#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q3_result_res_notoptimized.csv", p_best, fmt='%f',␣
 ↪delimiter=',')
```

[ ]:
```
load = [x for x in nprelmat @ p if x != 0]
print("stream:", load)
import copy
print(evaluate_max_influence(nprelmat, copy.deepcopy(p)))
```

[ ]:
```
load = nprelmat @ p
loadmap = np.zeros(len(nprelmat))
for i in range(len(load)):
    j = (i + 1) % 41
    k = i + 1 - 41 * j
    loadmap[j,k] = load[i]
```

Q4

2024 年 5 月 4 日

#2024 51MCM-B 题 #

首先创建路径邻接矩阵，1 代表联通，0 代表断开。节点与自身不连通。然后使用广度优先搜索，搜索出符合条件的五条路径

[ ]:
```
import numpy as np
from collections import deque
import copy

matrix = np.array([[0, 1, 0, 1, 0, 0, 0, 0, 0],[1, 0, 1, 0, 1, 0, 0, 0, 0],[0,␣
↪1, 0, 0, 0, 1, 0, 0, 0],[1, 0, 0, 0, 1, 0, 1, 0, 0],[0, 1, 0, 1, 0, 1, 0, 1,␣
↪0],[0, 0, 1, 0, 1, 0, 0, 0, 1],[0, 0, 0, 1, 0, 0, 0, 1, 0],[0, 0, 0, 0, 1,␣
↪0, 1, 0, 1],[0, 0, 0, 0, 0, 1, 0, 1, 0]])
#print(matrix)



class Graph:
    def __init__(self):
        self.graph = {0:[]}

    def add_edge(self, node, neighbors):
        self.graph[node]= self.graph.get(node,[]) + [neighbors]

class ResizableList(object):
    def __init__(self, initial_size=1764):
        self.data = [0] * initial_size
        self.size = 0

    def __getitem__(self, index):
```

```
        if index >= self.size:
            self._resize(index + 1)
        return self.data[index]

    def __setitem__(self, index, value):
        if index >= self.size:
            self._resize(index + 1)
        self.data[index] = value
        if index >= self.size:
            self.size = index + 1

    def _resize(self, new_size):
        self.data += [0] * (new_size - len(self.data))
```

[ ]:
```
g = Graph()
g.add_edge(0,1)
g.add_edge(0,3)
g.add_edge(1,0)
g.add_edge(1,4)
g.add_edge(2,3)
g.add_edge(2,18)
g.add_edge(3,0)
g.add_edge(3,2)
g.add_edge(3,36)
g.add_edge(3,42)
g.add_edge(4,1)
g.add_edge(4,6)
g.add_edge(5,6)
g.add_edge(5,9)
g.add_edge(5,16)
g.add_edge(5,22)
g.add_edge(6,4)
g.add_edge(6,5)
g.add_edge(6,34)
g.add_edge(7,12)
g.add_edge(7,24)
```

```
g.add_edge(7,33)
g.add_edge(8,9)
g.add_edge(8,28)
g.add_edge(8,37)
g.add_edge(8,41)
g.add_edge(9,5)
g.add_edge(9,8)
g.add_edge(9,10)
g.add_edge(10,9)
g.add_edge(10,11)
g.add_edge(10,16)
g.add_edge(10,19)
g.add_edge(11,10)
g.add_edge(11,12)
g.add_edge(11,20)
g.add_edge(11,41)
g.add_edge(12,7)
g.add_edge(12,11)
g.add_edge(12,35)
g.add_edge(13,14)
g.add_edge(13,15)
g.add_edge(13,39)
g.add_edge(14,13)
g.add_edge(14,17)
g.add_edge(14,21)
g.add_edge(15,13)
g.add_edge(15,42)
g.add_edge(16,5)
g.add_edge(16,10)
g.add_edge(16,18)
g.add_edge(16,36)
g.add_edge(17,14)
g.add_edge(17,39)
g.add_edge(18,2)
g.add_edge(18,16)
g.add_edge(18,19)
```

```
g.add_edge(18,23)
g.add_edge(19,10)
g.add_edge(19,18)
g.add_edge(19,20)
g.add_edge(19,23)
g.add_edge(20,11)
g.add_edge(20,19)
g.add_edge(21,14)
g.add_edge(22,5)
g.add_edge(22,34)
g.add_edge(22,40)
g.add_edge(23,18)
g.add_edge(23,19)
g.add_edge(24,7)
g.add_edge(24,32)
g.add_edge(25,30)
g.add_edge(25,33)
g.add_edge(26,31)
g.add_edge(26,33)
g.add_edge(27,30)
g.add_edge(27,35)
g.add_edge(28,8)
g.add_edge(28,35)
g.add_edge(29,31)
g.add_edge(29,32)
g.add_edge(30,25)
g.add_edge(30,27)
g.add_edge(31,26)
g.add_edge(31,29)
g.add_edge(32,24)
g.add_edge(32,29)
g.add_edge(33,7)
g.add_edge(33,25)
g.add_edge(33,26)
g.add_edge(34,6)
g.add_edge(34,22)
```

```
g.add_edge(34,40)
g.add_edge(35,12)
g.add_edge(35,27)
g.add_edge(35,28)
g.add_edge(36,3)
g.add_edge(36,16)
g.add_edge(37,8)
g.add_edge(38,39)
g.add_edge(38,42)
g.add_edge(39,13)
g.add_edge(39,17)
g.add_edge(39,38)
g.add_edge(40,22)
g.add_edge(40,34)
g.add_edge(41,8)
g.add_edge(41,11)
g.add_edge(42,3)
g.add_edge(42,15)
g.add_edge(42,38)
g.add_edge(27,33)
g.add_edge(33,27)
g.add_edge(26,32)
g.add_edge(32,26)
g.add_edge(6,0)
g.add_edge(0,6)
g.add_edge(21,17)
g.add_edge(17,21)
g.add_edge(37,28)
g.add_edge(28,37)
g.add_edge(35,41)
g.add_edge(41,35)
g.add_edge(6,36)
g.add_edge(36,6)
g.add_edge(41,10)
g.add_edge(10,41)
g.add_edge(38,13)
```

```
g.add_edge(13,38)
g.add_edge(2,15)
g.add_edge(15,2)


# 新添加的路线
g.add_edge(35,33)
g.add_edge(35, 8)
g.add_edge(32,7)
g.add_edge(18,10)
g.add_edge(42,2)

print(g.graph)

def bfs(graph, start, target):
    visited = []
    paths = []
    queue = [[start]]
    if start == target:
        return "Start = target"
    while queue:
        path = queue.pop(0)
        node = path[-1]
        neighbours = graph[node]
        for neighbour in neighbours:
            if(neighbour in path):
                continue
            new_path = list(path)
            new_path.append(neighbour)
            queue.append(new_path)
            if neighbour == target:
                if len(paths) >=5:return paths # 更严格的限制

                if len(paths) >= 1:
                    if len(new_path) > len(paths[0]):
                        return paths
```

```
                paths.append(new_path)
        visited.append(node)
    return "No path"
```

[ ]:
```
plans = [[17,6,92],
[22,4,113],
[6,22,101],
[13,26,84],
[11,28,87],
[4,17,62],
[26,15,25],
[27,26,65],
[23,24,49],
[17,29,64],
[19,33,42],
[23,10,102],
[32,34,61],
[22,10,97],
[32,23,113],
[36,18,80],
[27,8,100],
[9,34,114],
[28,20,107],
[27,40,52],
[14,17,46],
[38,33,85],
[26,40,88],
[35,23,61],
[21,19,121],
[32,20,103],
[9,31,102],
[14,6,58],
[7,16,65],
[33,35,55],
[29,9,24],
```

```
[20,25,66],
[10,40,64],
[41,7,65],
[17,8,122],
[39,30,61],
[27,17,66],
[25,20,66],
[38,20,91],
[27,6,121],
[6,3,63],
[7,26,89],
[35,22,93],
[40,27,90],
[12,24,98],
[32,39,134],
[21,38,95],
[31,15,82],
[28,40,74],
[2,19,63],
[17,29,84],
[8,37,92],
[37,33,98],
[24,28,101],
[5,40,51],
[38,33,103],
[23,31,110],
[15,20,122],
[13,23,66],
[2,22,72],
[19,23,85],
[15,1,84],
[39,11,54],
[39,29,33],
[40,8,83],
[34,18,86],
[13,4,59],
```

```
[23,2,55],
[19,25,115],
[21,13,91],
[18,15,77],
[4,15,64],
[0,5,75],
[23,32,65],
[7,14,89],
[12,3,86],
[40,10,88],
[27,17,52],
[38,14,108],
[24,21,80],
[2,4,85],
[7,26,63],
[8,13,86],
[21,36,82],
[35,39,85],
[4,22,78],
[18,37,63],
[23,16,100],
[8,14,116],
[0,17,73],
[17,0,79],
[0,35,54],
[21,40,103],
[13,23,112],
[28,24,72],
[39,20,97],
[41,21,49],
[7,39,80],
[10,13,47],
[26,28,64]]

l_org = [[0,1,315],
[0,3,393],
```

```
[1,0,200],
[1,4,231],
[2,3,667],
[2,18,790],
[3,0,615],
[3,2,424],
[3,36,730],
[3,42,618],
[4,1,200],
[4,6,246],
[5,6,405],
[5,9,545],
[5,16,282],
[5,22,529],
[6,4,200],
[6,5,403],
[6,34,520],
[7,12,829],
[7,24,512],
[7,33,395],
[8,9,517],
[8,28,321],
[8,37,200],
[8,41,276],
[9,5,678],
[9,8,392],
[9,10,389],
[10,9,205],
[10,11,495],
[10,16,574],
[10,19,479],
[11,10,541],
[11,12,561],
[11,20,606],
[11,41,261],
[12,7,758],
```

```
[12,11,707],
[12,35,519],
[13,14,419],
[13,15,868],
[13,39,227],
[14,13,513],
[14,17,200],
[14,21,200],
[15,13,598],
[15,42,200],
[16,5,328],
[16,10,554],
[16,18,473],
[16,36,469],
[17,14,290],
[17,39,517],
[18,2,635],
[18,16,576],
[18,19,497],
[18,23,397],
[19,10,453],
[19,18,388],
[19,20,522],
[19,23,506],
[20,11,240],
[20,19,368],
[21,14,210],
[22,5,377],
[22,34,305],
[22,40,230],
[23,18,480],
[23,19,467],
[24,7,472],
[24,32,325],
[25,30,200],
[25,33,316],
```

```
[26,31,237],
[26,33,442],
[27,30,350],
[27,35,831],
[28,8,371],
[28,35,418],
[29,31,200],
[29,32,200],
[30,25,289],
[30,27,200],
[31,26,200],
[31,29,200],
[32,24,323],
[32,29,266],
[33,7,200],
[33,25,293],
[33,26,505],
[34,6,279],
[34,22,270],
[34,40,340],
[35,12,410],
[35,27,767],
[35,28,606],
[36,3,501],
[36,16,557],
[37,8,272],
[38,39,407],
[38,42,821],
[39,13,375],
[39,17,315],
[39,38,406],
[40,22,247],
[40,34,200],
[41,8,280],
[41,11,494],
[42,3,974],
```

```
[42,15,200],
[42,38,460],
[27,33,442],
[33,27,340],
[26,32,200],
[32,26,289],
[6,0,226],
[0,6,370],
[21,17,321],
[17,21,200],
[37,28,200],
[28,37,272],
[35,41,637],
[41,35,521],
[6,36,200],
[36,6,298],
[41,10,475],
[10,41,483],
[38,13,365],
[13,38,434],
[2,15,688],
[15,2,921],

[35,33,10000],
[35,8,10000],
[32,7,10000],
[18,10,10000],
[42,2,10000]
]

possible_links = [[1,6],
                  [6,1],
[0,4],
[4,0],
[22,6],
[6,22],
```

```
[5,34],
[34,5],
[13,17],
[17,13],
[14,39],
[39,14],
[13,42],
[42,13],
[15,38],
[38,15],
[2,42],
[3,6],
[6,3],
[0,36],
[36,0],
[2,36],
[36,2],
[36,18],[18,36],
[3,18],[38,3],
[3,16],[16,3],
[2,16],[16,2],
[5,36],[36,5],
[6,16],[16,6],
[10,18],
[5,10],[10,5],
[9,16],[16,9],
[8,10],[10,8],
[9,41],[41,9],
[10,20],[20,10],
[11,19],[19,11],
[8,35],
[11,35],[35,11],
[12,41],[41,12],
[7,35],[35,7],
[33,35],
[33,12],[12,33],
```

```
[30,33],[33,30],
[25,27],[27,25],
[7,32],
[7,26],[26,7],
[33,32],[32,33],
[31,32],[31,32],
[26,29],[29,26],
]

'''
round 1
[35,33],
[12,27],[27,12],
[27,7],[7,27],

round 2
[35,8],
[28,41],[41,28],

round 3
[32,7],
[26,24],[24,26],
[24,33],[33,24],

round 4
[18,10],
[16,19],[19,16],

round 5
[42,2],
[3,15],[15,3],
'''
```

[ ]:
```
def routenum(i,j):
    return 41 * i + j -42
```

```
def relationmat(route):
    relmat = ResizableList()
    while route:
        if len(route) == 1:
            break
        i = route.pop(0)
        j = route[0]
        relmat[routenum(i,j)] = 1
    return relmat
```

[ ]:
```
def largest_n_indices(lst, n):
    return sorted(range(len(lst)), key=lambda x: lst[x])[-n:]
def smallest_n_indices(lst, n):
    return sorted(range(len(lst)), key=lambda x: lst[x])[:n]
```

[ ]:
```
l = np.zeros(1764)
for l_i in l_org:
    l[routenum(l_i[0],l_i[1])] = l_i[2]



relmat = []
for plan in plans:
    i = plan[0]
    j = plan[1]
    routes = bfs(g.graph, i, j)
    plan.append(len(routes))
    print(len(relmat))
    for route in routes:
        relmat.append(relationmat(route).data)

nprelmat = np.array(relmat)
nprelmat = nprelmat.T
print(nprelmat)
```

[ ]:
```
routess = []
for plan in plans:
    i = plan[0]
```

```
    j = plan[1]
    routes = bfs(g.graph, i, j)
    routess.extend(routes)
```

下面使用目标函数 min=~2 进行梯度下降优化

[ ]:
```
%%time

Q = nprelmat.T @ nprelmat
print(Q)

A = []
b = []
i = 0
for plan in plans:
    A.append(np.zeros(i).tolist() + np.ones(plan[3]).tolist() + np.zeros(len(Q)␣
 ↪- plan[3] - i).tolist())
    b.append(plan[2])
    i += plan[3]
A = np.array(A)
```

[ ]:
```
def evaluate_max_influence(R, p):
    originload = sum(p)
    for k in range(5):
        t = R @ p
        a, _ = max(enumerate(t), key=lambda x: x[1])
        for j in range(len(R[a])):
            if R[a,j] != 0:
                p[j] = 0
    return originload - sum(p)
```

[ ]:
```
p = np.ones(len(Q))
i = 0
for plan in plans:
    p[i:i + plan[3]] = plan[2]/plan[3]
    plan.append(i)
    plan.append(i + plan[3])
```

```
    i += plan[3]
```

[ ]:
```
%%time
import random
import math
def list_exp(l):
    return [math.exp(x) for x in l]
#p_best = np.ones(len(Q))
```

[ ]:
```
%%time
import random
def back_fire(nprelmat, p_best):
    p = p_best
    latestmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))
    max_best = latestmax
    T = 0.01
    for i in range(50000):
        fromchange = random.randint(0,len(p) - 1)
        changeamount = 0.5 * (random.random() - 0.5) * (p[fromchange]+1)
        if p[fromchange] + changeamount < 0:
            continue

        p[fromchange] += changeamount
        if min(l - nprelmat @ p) < 0:
            p[fromchange] -= changeamount
            continue
        if max(A @ p - b) > 0:
            p[fromchange] -= changeamount
            continue
        tmpmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))

        if random.random() > np.exp((tmpmax - latestmax) / T):
            p[fromchange] -= changeamount
            latestmax = tmpmax
            #print("cycle:", i, "tmp:", tmpmax)
            continue
        if tmpmax > latestmax:
```

```
            latestmax = tmpmax

            if latestmax > max_best:
                max_best = latestmax
                p_best = copy.deepcopy(p)
                print("cycle:", i, "max:", latestmax)
        T = 0.999 * T
    return np.append(max_best, p_best)


def back_fire_Q2(nprelmat, p_best):
    p = p_best
    latestmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))
    max_best = latestmax
    T = 0.01
    for i in range(20000):
        fromchange = random.randint(0,len(p) - 1)
        changeamount = 0.5 * (random.random() - 0.5) * (p[fromchange]+1)
        if p[fromchange] + changeamount < 0:
            continue
        p[fromchange] += changeamount
        # 此处删去了限制不超过通行能力的限制
        if max(A @ p - b) > 0:
            p[fromchange] -= changeamount
            continue
        tmpmax = sum(p) - evaluate_max_influence(nprelmat, copy.deepcopy(p))

        if random.random() > np.exp((tmpmax - latestmax) / T):
            p[fromchange] -= changeamount
            latestmax = tmpmax
            #print("cycle:", i, "tmp:", tmpmax)
            continue
        if tmpmax > latestmax:
            latestmax = tmpmax
```

```
            if latestmax > max_best:
                max_best = latestmax
                p_best = copy.deepcopy(p)
                print("cycle:", i, "max:", latestmax)
        T = 0.999 * T
    return np.append(max_best, p_best)
```

[ ]:
```
%%time
import copy

p0 = back_fire(nprelmat, np.ones(len(Q)))
p0 = p0.tolist()
effectiveness = p0.pop(0)

len_possible_links = len(possible_links)
hope_passes = np.zeros(len_possible_links)
for k in range(len_possible_links):
    possible_link = possible_links[k]
    for i in range(len(routess)):
        for j in range(len(routess[i]) - 2):
            if routess[i][j] == possible_link[0]:
                if routess[i][j+2] == possible_link[1]:
                    hope_passes[k] += p0[i]
                break

p_norestrict = back_fire_Q2(nprelmat, copy.deepcopy(p0))
p_norestrict = p_norestrict.tolist()
effectiveness_norestrict = p_norestrict.pop(0)
shortness_map = 1 - nprelmat @ p_norestrict

shortness_links = np.zeros(len_possible_links)
for i in range(len(possible_links)):
    routes = bfs(g.graph, possible_links[i][0], possible_links[i][1])
    for route in routes:
        for j in range(len(route) - 1):
            shortness_links[i] += shortness_map[routenum(route[j],route[j+1])]
```

```
neednesses = np.zeros(len_possible_links)
for i in range(len_possible_links):
    neednesses[i] = min(shortness_links[i], hope_passes[i])

print(max(neednesses), largest_n_indices(neednesses, 5))
for i in largest_n_indices(neednesses, 5):
    print(possible_links[i])
%%time
import copy

p0 = back_fire(nprelmat, np.ones(len(Q)))
p0 = p0.tolist()
effectiveness = p0.pop(0)

len_possible_links = len(possible_links)
hope_passes = np.zeros(len_possible_links)
for k in range(len_possible_links):
    possible_link = possible_links[k]
    for i in range(len(routess)):
        for j in range(len(routess[i]) - 2):
            if routess[i][j] == possible_link[0]:
                if routess[i][j+2] == possible_link[1]:
                    hope_passes[k] += p0[i]
                break

p_norestrict = back_fire_Q2(nprelmat, copy.deepcopy(p0))
p_norestrict = p_norestrict.tolist()
effectiveness_norestrict = p_norestrict.pop(0)
shortness_map = l - nprelmat @ p_norestrict

shortness_links = np.zeros(len_possible_links)
for i in range(len(possible_links)):
    routes = bfs(g.graph, possible_links[i][0], possible_links[i][1])
    for route in routes:
        for j in range(len(route) - 1):
```

```
            shortness_links[i] += shortness_map[routenum(route[j],route[j+1])]

neednesses = np.zeros(len_possible_links)
for i in range(len_possible_links):
    neednesses[i] = min(shortness_links[i], hope_passes[i])

print(max(neednesses), largest_n_indices(neednesses, 5))
for i in largest_n_indices(neednesses, 5):
    print(possible_links[i])
```

[ ]:
```
%%time
import copy

p0 = back_fire(nprelmat, np.ones(len(Q)))
p0 = p0.tolist()
effectiveness = p0.pop(0)

len_possible_links = len(possible_links)
hope_passes = np.zeros(len_possible_links)
for k in range(len_possible_links):
    possible_link = possible_links[k]
    for i in range(len(routess)):
        for j in range(len(routess[i]) - 2):
            if routess[i][j] == possible_link[0]:
                if routess[i][j+2] == possible_link[1]:
                    hope_passes[k] += p0[i]
                break

p_norestrict = back_fire_Q2(nprelmat, copy.deepcopy(p0))
p_norestrict = p_norestrict.tolist()
effectiveness_norestrict = p_norestrict.pop(0)
shortness_map = 1 - nprelmat @ p_norestrict

shortness_links = np.zeros(len_possible_links)
for i in range(len(possible_links)):
    routes = bfs(g.graph, possible_links[i][0], possible_links[i][1])
```

```
    for route in routes:
        for j in range(len(route) - 1):
            shortness_links[i] += shortness_map[routenum(route[j],route[j+1])]

neednesses = np.zeros(len_possible_links)
for i in range(len_possible_links):
    neednesses[i] = min(shortness_links[i], hope_passes[i])

print(max(neednesses), largest_n_indices(neednesses, 5))
for i in largest_n_indices(neednesses, 5):
    print(possible_links[i])
```

[ ]:
```
import csv
# 保存列表到 CSV 文件
with open("E:\\Q4_result_path_notoptimized5.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])


#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q4_result_res_notoptimized5.csv", p0, fmt='%f', delimiter=',')
```

[ ]:
```
'''
import csv
# 保存列表到 CSV 文件
with open("E:\\Q4_result_path0.csv", 'w', newline='') as f:
    writer = csv.writer(f)
    for routes in routess:
        writer.writerow([routes])


#np.savetxt("E:\\Q1_result_path.csv", routess, fmt='%d', delimiter=',')
np.savetxt("E:\\Q4_result_res0.csv", p, fmt='%f', delimiter=',')
'''
```

[ ]:
```
'''
load = [x for x in nprelmat @ p if x != 0]
print("stream:", load)
import copy
print(evaluate_max_influence(nprelmat, copy.deepcopy(p)))
'''
```