**重 庆 大 学**

**学 生 实 验 报 告**

**实验课程名称 数学实验**

**开课实验室 数学实验教学实验中心**

**学 院 计算机学院 年级 2022**

**专业班**

**学生姓名**

**学 号**

**开 课 时 间 2023 至 2024 学年第 二 学期**

|  |  |
| --- | --- |
| **总 成 绩** |  |
| **教师签名** |  |

**数 学 与 统 计 学 院 制**

**开课学院、实验室：数学与统计学院、数学实验教学实验中心 实验时间 ：2024年 4月27 日**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **课程**  **名称** | 数学实验 | **实验项目**  **名 称** | 合作实验：图论模型及算法的实验 | **实验项目类型** | | | | |
| **验证** | **演示** | **综合** | **设计** | **其他** |
| **指导**  **教师** | 龚劬 | **成 绩** |  |  |  | **√** |  |  |
| 应用实验（或综合实验）  一、问题重述  **某出版社的销售经理，需要从重庆飞往北京、上海、深圳、广州、杭州、武汉、兰州的分部去办事，最后回到重庆，请通过建立TSP模型的方法为他制定最廉价的旅行路线。**  二、问题分析  **为了规划最经济的出差路线，我们将采用TSP模型，这是一个经典的组合优化问题，其目标是找到一条从起点出发，经过指定的一系列城市后再返回起点的最短路径。在这个问题中，我们的目标是从重庆出发，途径北京、上海、深圳、广州、杭州、武汉、兰州七个城市，并最终回到重庆，以最小的成本完成所有出差任务。**  **首先，我们需要获取各个城市之间的交通信息和费用数据。然后，我们将这些信息构建成一个带权重的图，其中每个城市作为节点，城市之间的费用作为边的权重。**  **接下来，我们将使用TSP算法对这个图进行求解，以找到最优的旅行路线。**  **最后，我们会分析得到的旅行路线，计算出各个城市之间的实际出行成本，并提供给销售经理最佳的出差方案。**  三、数学模型的建立与求解  根据上传的图片，显示了旅行商问题（TSP）的数学模型，我们可以对该问题的模型建立与求解过程进行详细解释。  **TSP 问题的数学模型**   1. **目标函数:**   最小化旅行的总成本，即路径的总长度。数学上表示为：    其中 是城市i 到城市j 之间的距离， 是一个决策变量，当从城市 i 到城市 j 有路径时取 1，否则取 0。  **2. 约束条件:**  每个城市只能离开一次:    每个城市只能被访问一次:  避免子循环: 这意味着解决方案不能有分离的循环，这是通过添加多余的约束来完成的，称为 subtour elimination constraints (SEC)：    二进制决策变量: 确保 只能取 0 或 1，对应于没有旅行或有旅行从城市i到城市j。  **求解过程:**  1. 模型建立:  我们首先需要确定所有城市之间的距离矩阵 ，其中 表示从城市 i 到城市j的距离。  2. 定义决策变量:  创建一个的决策变量矩阵 ，其中 表示是否存在从城市i 到城市j的路径。  3. 设置目标函数与约束:  将上述目标函数和约束条件放入一个数学优化模型中。这通常涉及使用线性规划、整数规划或混合整数线性规划技术。  4. 解释结果:  一旦找到了解决方案，我们可以解释结果为旅行的具体路径以及总成本。  5. 实施路径:  将求解得到的路径转换为实际的旅行计划。  建模如下：  设dij是i与j之间的距离，xij=0或1（1表示连线，0表示不连线）。  ;  s.t.  四、实验结果及分析  输出为  X( 1, 5) 1.000000 350.0000  X( 2, 8) 1.000000 500.0000  X( 3, 2) 1.000000 570.0000  X( 4, 6) 1.000000 940.0000  X( 5, 4) 1.000000 500.0000  X( 6, 3) 1.000000 110.0000  X( 7, 1) 1.000000 650.0000  X( 8, 7) 1.000000 1200.000  所以最佳路线为1-5-4-6-3-2-8-7-1，即重庆-广州-深圳-杭州-上海-北京-兰州-武汉-重庆，总费用为4820元  五、附录  **MODEL:**  **sets:**  **cities/1..8/:level; !level(i)= the level of city;**  **link(cities, cities):**  **distance, !The distance matrix;**  **x; ! x(i,j)=1 if we use link i,j;**  **endsets**  **data: !Distance matrix, it need not be symmetirc;**  **distance =**  **0 600 378 260 350 320 200 570**  **600 0 495 700 799 330 510 500**  **490 570 0 500 550 110 480 510**  **880 1796 1200 0 1030 940 1018 1498**  **900 1467 1174 500 0 790 1001 750**  **700 957 110 600 400 0 500 580**  **650 1087 886 1099 1014 1085 0 1325**  **1120 1596 1398 1360 2710 1069 1200 0;**  **enddata**  **n=@size(cities); !The model size;**  **! Minimize total distance of the links;**  **min=@sum(link(i,j)|i #ne# j: distance(i,j)\*x(i,j));**  **!For city i;**  **@for(cities(i) :**  **! It must be entered;**  **@sum(cities(j)| j #ne# i: x(j,i))=1;**  **! It must be departed;**  **@sum(cities(j)| j #ne# i: x(i,j))=1;**  **! level(j)=levle(i)+1, if we link j and i;**  **@for(cities(j)| j #gt# 1 #and# j #ne# i :**  **level(j) >= level(i) + x(i,j)**  **- (n-2)\*(1-x(i,j)) + (n-3)\*x(j,i);**  **);**  **);**  ! Make the x's 0/1;  **@for(link : @bin(x));**  **! For the first and last stop;**  **@for(cities(i) | i #gt# 1 :**  **level(i)<=n-1-(n-2)\*x(1,i);**  **level(i)>=1+(n-2)\*x(i,1);**  **);**  **END**  **方法2对角线完全算法**  **求解过程**  我们另外还使用python实现了对角线完全算法解决tsp问题。算法步骤如下：  输入：图的距离矩阵D  （1）求D的简化矩阵D’以及各行各列的约数R(i)，R’(j)，罚数P(i)，P’(j)  （2）计算在简化矩阵中零元素所在行与列的罚数和，即P(i,j)=P(i)+P’(j)。将P(i,j)由大到小排列后，依次选取可作为可行部分路的边（i,j）。这些边对应的零元素记为0\*。用这些选择出来的边构成可行部分路。  （3）构造新的距离矩阵称为重构距离矩阵：按上述可行部分路的顶点序重新排列简化距离D’的行，列也按使上述所有“0\*”位于对角线上的次序重新排列。  （4）产生D的子阵：设重构矩阵对角线上m个非零元素对应的边为(i1,j1),（i2,j2），…，（im,jm），则从D中取出相应的m行，m列构成一个m×m子阵D1。为保证选出的边与原来的可行部分路不形成子循环，有m条边不能选择，将其对应的元素置为∞。并将列作适当调整使对角线元素为∞。  （5）对D1重复（1）—（4）步，直到重构矩阵对角线上的元素全为0为止，这时便可得到一个H圈。  **实验结果及分析：**    使用对角线完全算法求出的结果为重庆-武汉-北京-兰州-杭州-上海-广州-深圳-重庆，费用为4896。由于tsp问题为np hard问题，所使用的算法都是启发式算法，均只能接近最优值。  附录：  import numpy as np  from dioganal import diagonal\_complete  distance\_matrix = np.array([  [0, 600, 378, 260, 350, 320, 200, 570], # 重庆到其他城市的距离  [600, 0, 495, 700, 799, 330, 510, 500], # 北京到其他城市的距离  [490, 570, 0, 500, 550, 110, 480, 510], # 上海到其他城市的距离  [880, 1796, 1200, 0, 1030, 940, 1018, 1498], # 深圳到其他城市的距离  [900, 1467, 1174, 500, 0, 790, 1001, 750], # 广州到其他城市的距离  [700, 957, 110, 600, 400, 0, 500, 580], # 杭州到其他城市的距离  [650, 1087, 886, 1099, 1014, 1085, 0, 1325], # 武汉到其他城市的距离  [1120, 1596, 1398, 1360, 2710, 1069, 1200, 0] # 兰州到其他城市的距离  ])  diagonal\_complete(distance\_matrix)  import numpy as np  def diagonal\_complete(D):  D = D.astype('float') # D是带权邻接矩阵  # 为了算法需要，将D的对角线元素均置为无穷大  inf = np.inf  for i in range(D.shape[0]):  D[i, i] = inf  H = [] # 初始化最终H圈  row\_extract, column\_extract = range(D.shape[0]), range(D.shape[0])  D\_extract = D.copy() # 初始化算法迭代时考虑的子阵  k\_iter = 1 # 初始化迭代次数  while 1:  N = D\_extract.shape[0] # 子阵的行列数  # 求出简化矩阵和各行各列的约数罚数  Ri = [] # 初始化各行约数  for i in range(N):  Ri.append(np.min(D\_extract[i])) # 求出每行的最小值，即行的约数  D\_extract[i] -= Ri[i] # 子阵每一行减去此行的最小值  Rj = [] # 初始化各列约数  for i in range(N):  Rj.append(np.min(D\_extract[:, i])) # 在上述基础上求出每列的最小值，即列的约数  D\_extract[:, i] -= Rj[i] # 处理后的子阵每一列再减去此列的最小值  Pi, Pj = [], [] # 初始化各行各列的罚数  for i in range(N):  t = sorted(D\_extract[i])  Pi.append(t[1] - t[0]) # 求出每一行的罚数  t = sorted(D\_extract[:, i])  Pj.append(t[1] - t[0]) # 求出每一列的罚数  # 求出简化阵中零元素对应的罚数并按从大到小的次序排列  zero\_list = [] # 初始化存放 (零元素行，零元素列，罚数)的列表  for i in range(N):  for j in range(N):  if D\_extract[i, j] == 0:  zero\_list.append((row\_extract[i], column\_extract[j], Pi[i] + Pj[j]))  zero\_list = sorted(zero\_list, key=lambda k: k[2], reverse=True) # 按照罚数由大到小排序  for item in zero\_list:  # 刚开始没有路径情况的特殊处理  if len(H) == 0:  H.append([item[0], item[1]])  else:  # 否则判断目前考虑的边的两个端点是否在以及在哪个路径里  pos\_1, pos\_2 = [item[0] in k for k in H], [item[1] in k for k in H]  # 分别找端点1，2是否在已有子路径内，如果在，找出在哪条子路径  try:  pos\_1 = pos\_1.index(True)  except ValueError:  pos\_1 = []  try:  pos\_2 = pos\_2.index(True)  except ValueError:  pos\_2 = []  # 如果目前考虑边的两个端点都不在任一子路径内，则设为另一新的子路径  if pos\_1 == [] and pos\_2 == []:  H.append([item[0], item[1]])  elif pos\_1 != [] and pos\_2 == []:  # 确保不会使得该边加进去后子路径某一点出次大于1  if H[pos\_1][-1] == item[0]:  H[pos\_1].append(item[1])  elif pos\_1 == [] and pos\_2 != []:  # 确保不会使得该边加进去后子路径某一点入次大于1  if H[pos\_2][0] == item[1]:  H[pos\_2] = [item[0]] + H[pos\_2]  else:  # 两个端点都含于同一子路径内，则舍弃考虑该边  # 两个端点分别含于不同子路径内，则要求出点在子路1末，入点在子路2头  # 这时合并两条路径  if pos\_1 != pos\_2 and H[pos\_1][-1] == item[0] and H[pos\_2][0] == item[1]:  t = H[pos\_1] + H[pos\_2]  H = [H[k] for k in range(len(H)) if (k != pos\_1 and k != pos\_2)]  H.append(t)  if len(H) == 1:  # H里面只含有一条路径，则首尾相连即得到最终结果  H = H[0] + [H[0][0]]  break  else:  # 提取应该考虑的子阵的行和列  row\_extract, column\_extract = [], []  for item in H:  row\_extract.append(item[-1])  column\_extract.append(item[0])  # 建立新的子阵  D\_extract = D[row\_extract, :]  D\_extract = D\_extract[:, column\_extract]  # 将导致形成子循环的元素置为无穷大  for i in range(len(H)):  D\_extract[i, i] = inf  print('完成', k\_iter, '次迭代')  k\_iter += 1  # 求所得H圈的权和  W = 0  for i, j in zip(H[:-1], H[1:]):  W += D[i, j]  print('所求得的H圈为:', [i + 1 for i in H], '权和为:', W)  教师签名  年 月 日 | | | | | | | | |