基于集合的序和蚁群算法的公交司机排班模型

摘要

公交司机排班是在保证公交计划通勤任务的正常完成的条件下，通过科学合理地排班，降低机组人力资源成本。本文考虑了司机排班的工作时间、工作里程和人力成本，求出最小不可行子链数，再考虑司机出勤率的要求构造了模型。

问题一要求在只考虑时空连接约束、换乘时间及地点约束、连续驾驶时长约束的条件下找到最小不可行子链的数量。可以转换为在一个部分有序集合里搜寻**最大有序子集**的问题。我们首先根据时空约束和换乘条件建立了**时空约束矩阵**，通过矩阵运算得到有序集合，再利用有序集合的性质，分别讨论最小不可行子链去掉首尾的情况并**分解子链**，然后重复此过程将子链数分解为不可再分形式，最后利用**递推关系**求解。我们求得最小不可行子链数共有 **920138 条**，并分析了该子链数的产生。

问题二要求额外考虑司机每日最大工作时长，司机出勤率等要素设计司机排版表，本质是用子集覆盖全集的**集合覆盖问题**。我们首先列出约束，利用问题一中的子链缩减规划域。因为约束和优化目标都有非线性因子，我们考虑使用**蚁群算法**求解，并且引入了暂停机制以防止算法陷入局部最优解。结果显示只需要 **42 名司机**就可以完成一天内行程的全覆盖。

问题三要求在问题一，二的基础上进行分析和深化，我们从司机身心健康和公交系统效率出发，提出了 5 个指标：小休次数、驾驶时长、休息时长、工作里程、换乘次数。我们可视化了指标并以指标分析了问题二中的排班计划，接着以这些指标改进模型，**修正优化目标**，并给出了蚁群算法的**后续改进可能**，包括**自适应调整，精英策略**等。结果显示司机的驾驶时长在 **5 到 6 小时之间**和工作里程在 170 公里左右，较为合理，小休次数 **2 到 3 次**需要适当放宽。

对于大规模的子链问题和排班问题，我们基于公交排班的特点和所建立的数学模型也可以加速求解，极大地降低了计算时间，并且具有良好的鲁棒性。总体而言，我们的模型可以有效地处理机组排班问题，对于小规模数据能够给出满意的解

关键字： **集合的序，极小元递推，集合覆盖，蚁群算法**

一、 问题重述

1.1 问题背景

公共交通在生活中扮演着重要的角色。而庞大的民众需求对运营与调度提出了更高的要求。司机是公交系统最重要的人力资源之一, 对系统效率产生直接的影响。好的调度排班（也称为轮值表）应该使总的司机数最少且满足所有公交发车的约束。近年来事故调查结果显示, 司机的不良状态如精神疲劳、协调缺失等是引起人为错误的重要因素。由此, 我们看到的确有必要在机组人员排班过程中充分考虑并协调各环节中能够影响到机组人员身心健康的因素, 如休息时间, 并积极采取措施减少此类消极因素的影响。即轮值表应该综合考虑运营成本、运营效果和司机健康。

1.2 问题的提出

乘务排班计划是城市轨道交通运营管理的重要环节，充分利用乘务资源进行合理计划是轨道交通企业降低运营成本、提高运营效率，以及保证列车运行安全的关键措施。然而，由于车次数量多、运行间隔密、运行路线复杂多变、法规条例严格等因素，优化乘务排班计划具有相当难度。某地铁公司运营部门决定优化目前的乘务排班流程，需要决策咨询顾问运用运筹学技术制定最优的地铁乘务人员排班方案。

城市地铁通常存在多条线路，针对每条线路一般都有固定的乘务人员。地铁的乘务人员包含列车司机、安全巡视员等，在这里主要的是考虑列车司机的排班要求。一名司机（A）通常会在指定站点出勤签到，按照分配的任务顺序开始一天的工作，从接车站点驾驶某车次的列车在规定时间到达下车站点，这样的一个任务称为一个值乘片段。到达下车站点后，他会与一名已在站台等候的司机（B）快速完成交接，然后由司机 B 驾驶该列车继续出发，司机 A 在站台等待任务表中下一班列车到达，并依次执行，这样的一次下车和接车过程称为换乘。因此，对于每位司机而言，一天的工作任务流程安排可以形象地看成一条任务链，又叫轮值任务卡，所有司机的轮值任务卡构成了排班人员视角里的轮值表。

列车司机的排班工作一般分为两步：第一步：根据列车运行图或时刻表，综合考虑车次的接车下车时间地点、换乘约束以及班制运营要求，编制列车司机的轮值表；

第二步：基于轮值表，按照特定的班组转换模式偏好指派列车司机执行对应任务，从而编制列车司机的排班母表。

本题仅针对第一步编制轮值表进行研究。轮值表模型中，核心决策为一天的值乘片段组合（也就是任务链），主要考虑换乘规则、里程工时上限等业务规则，以决策完成每日值乘任务所需人数及具体工作安排。

1. **问题一：**请结合附件中的数据，在满足时空连接约束和换乘时间及地点约束的前提下，仅针对连续驾驶时长约束进行进一步分析，搜索不满足该条约束的全部最小不可行任务子链，并输出最小不可行子链的数量。
2. **问题二：**结合题目中涉及的约束条件，考虑优化目标为最小车次任务链数量（任务链数量越少，意味着需要安排的列车司机人数就越少），建立该地铁运营公司的列车司机排班模型，并设计算法进行求解。输出最小的轮值任务链数量及每条链的任务执行顺序。
3. **问题三：**请基于第 2 问找到的最优列车司机排班方案，分析当前方案还有哪些不足，比如在任务均衡性、合理性等方面。请自行定义衡量指标（例如：累计工作时长、累计驾驶时长、累计行驶里程、累计休息时长等），可以对均值、标准差等统计指标进行分析，并通过可视化作图形式解释选择原因。根据数据分析的结果，从模型或算法角度提出可行的改进方案。

二、 问题分析

2.1 问题一

问题在常规的时空衔接约束和换乘时间约束上，加上了最大工作时间的限制，包含总工作时间和单次工作时间两方面。由题目可知线路两端终点站 PA 和 PC 以及换乘站点 PB 均可以进行换乘。事实上取消了一般轮值表问题中对换乘地点的限制。问题要求出所有最小不可行子链的数量。我们把所有任务链，包括可行和不可行子链当作一个集合，将约束转化为元素之间的偏序关系，这个问题等价为对于一个部分有序集合，在固定约束下的完全集合分割问题。我们首先将元素之间有序关系转换为邻接矩阵，分别构建地点约束邻接矩阵和时间约束邻接矩阵，然后对矩阵进行操作以剔除不满足驾驶时长约束的任务。司机小休的存在实际上允许两个有序集合克服一些有序关系拼接在一起，我们先不考虑小休求子链数及这时的集合，再求最后一段不可行子链数和对应集合，最后通过两个集合和时空邻接矩阵之间的操作得到总的最大不可行子链数。

2.2 问题二

在司机排班调度问题中，排班是根据已确定的公交路线来生成固定周期内从某一地出发并且能够返回任务链，排班需要满足时间空间一致性、人员单次最大工作时间、最大总工作时间和休息时间的规定；同时确保这些排班能覆盖所有计划的路线，不应使任何需要值勤的路线未分配给司机。这是一个集合覆盖问题，我们结合问题一中最小不可行子链，构建最长任务子链集合并以总公交数最少为目标，把问题转换为一个规划问题。然后给出目标函数和飞行时间、执勤时间、休息时间等约束的显示表达式，最后利用蚁群算法求解。

2.3 问题三

问题三要求深入分析问题二中的排班，分析指标并提出算法改进方法。我们从司机的身体心理健康出发，同时考虑公交系统效率而提出了 5 个指标：小休次数，司机驾驶时长，休息时长，工作里程，换乘次数。统计了每名司机的五个指标并进行了可视化。这些指标可以作为改进模型的基础，我们主要考虑指标对优化目标的修正，并且给出了蚁群算法后续的可能改进。

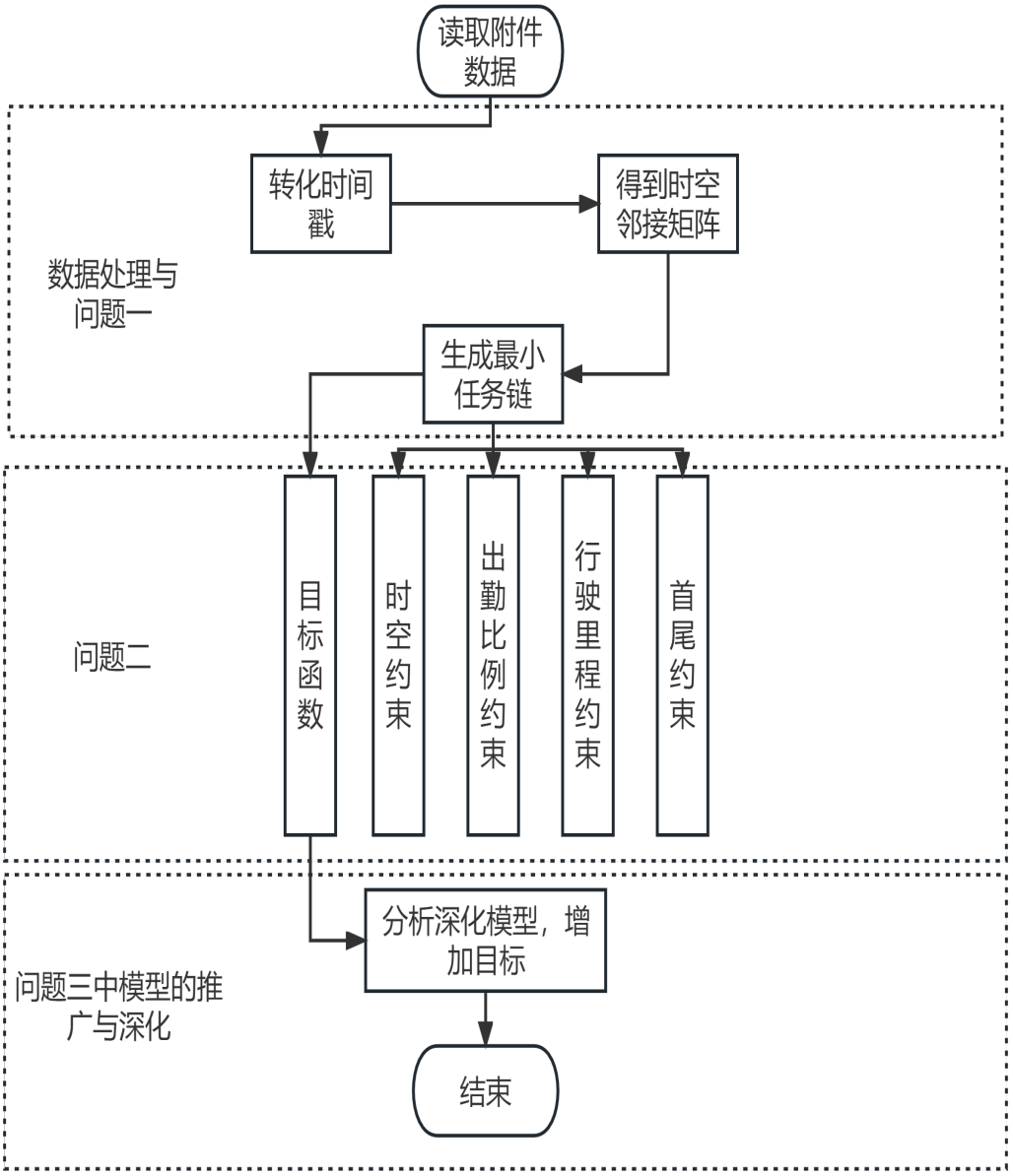


图 1　问题思路

三、 模型假设

为简化模型，分析问题本质，我们提出如下假设并给出其的依据：

1. 假设司机在单次连续任务中，两次驾驶公交的间隙或换乘时间不计入休息时间。
   依据： 虽然司机等待公交和换乘的时间并不算入工作时间，但是司机的精神仍然保持半工作状态。

四、 符号说明

表 1　符号说明

符号	意义
Pe	地点邻接矩阵
Tm	时间邻接矩阵
C	时空邻接矩阵
t_i^d	第 i 条公交线路出发时间
t_i^a	第 i 条公交线路到达时间
A_i^d	第 i 条公交线路出发地点
A_i^a	第 i 条公交线路出发地点
T	最小不可行子链去掉最后一个元素
R	最小不可行子链去掉第一个元素
m	最小不可行子链数
S	子链集合
x_k^i	司机 k 是否在公交航线 i 上工作
V_k	司机 k 在当天执勤中总工作时长

五、 数据处理

为方便后续处理，我们将每条公交线路的时间戳转化为从 0：00 开始计时的小时数，即

$$t = HH + MM \times \frac{100}{60} \tag{1}$$

HH 是时间戳的小时（以 24 小时表示形式），MM 是分钟数。利用此公式可以将时间戳 06：57 变为 6.95。

六、 模型建立

6.1 问题一：基于序数的时空约束子链数目模型

6.1.1 时空关系的建立

根据时空约束有公交线路图

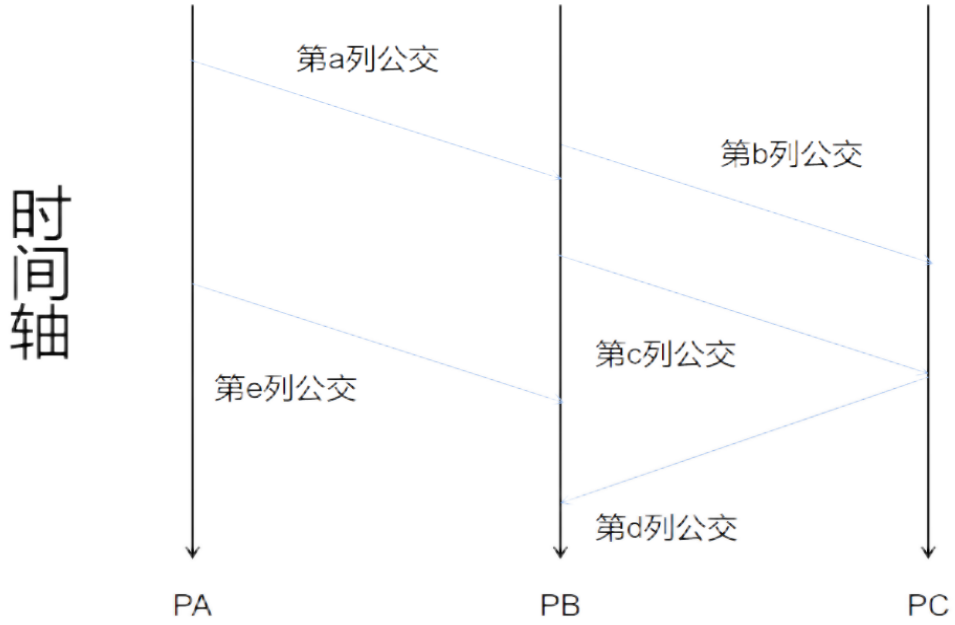


图 2　公交时空网络示意图

这里只给出部分抽象后的公交线路，时空邻接矩阵 C 内的元素赋值如下表所示

表 2　构造时空邻接矩阵示例

公交对	是否在集合中	原因
a,b	否	不满足时间约束
b,a	是	不满足换乘时间约束
a,c	否	满足所有约束
e,d	否	不满足地点约束

我们通过矩阵运算得到时空邻接矩阵。首先根据附件数据分别构建地点邻接矩阵 Pe 和时间邻接矩阵 Tm。其中

$$P_{242\times 2}=\begin{bmatrix} B & B & \cdots \\ C & A & \cdots \end{bmatrix} \tag{2}$$

地点矩阵 P 的每一列代表一条公交线路，第一行是出发点 A_i^d，第二行是终点 A_i^a。依次两两遍历这个矩阵的所有元素，构建邻接矩阵 Pe

$$\begin{cases} Pe_{ij}=1 & \text{if } P_{2i}=P_{1j} \\ P_{ij}=0 & \text{else} \end{cases} \tag{3}$$
$$i,j=1\ldots 242$$

对于时间矩阵 T

$$T_{242\times 3}=\begin{bmatrix} 6.05 & 6.07 & \cdots \\ 6.97 & 6.67 & \cdots \\ 1 & 2 & \cdots \end{bmatrix} \tag{4}$$

时间矩阵 T 的每一列仍然代表一条公交线路，第一行是出发时间 t^d，第二行是达到时间 t^a，构建时间邻接矩阵还需要考虑反向换乘时间不小于 10 分钟的约束。过程如下

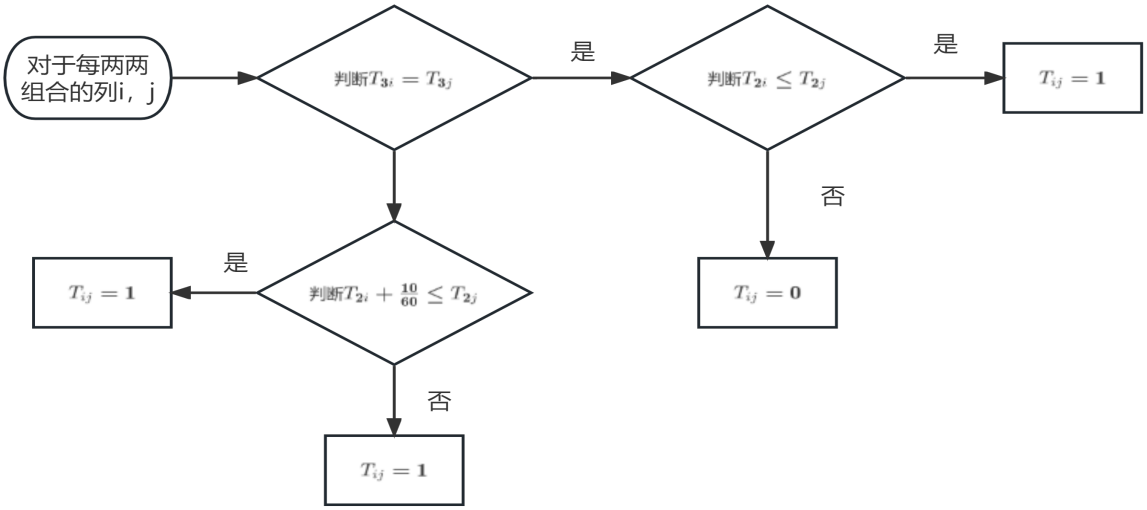


图 3　构建时间邻接矩阵

然后将两个矩阵做与操作，得到时空邻接矩阵 C

$$C = Pe\&Tm \tag{5}$$

6.1.2 基于序数的子链数目求解

每一条公交线路可以看作一个二维有序空间中的点，一个任务链即是有序空间中的可达路径。在不考虑小休的情况下只考虑一个维度时，空间是良序的，虽然在二维上只是部分有序。但在引入了小休的概念后，任务链的长度的约束变松，而因为反向换乘时间不能再以上文的方式构建矩阵，原有的一些可达有序点变成了无序点。这样集合就变成了二维无序集。不妨将小休前后视为两端独立的任务链，由整体到部分，由一般到特殊求解。

我们先不考虑小休，因为题目中的最小不可行子链只针对任务链的第一个和最后一个元素，图三中的时间约束修改为

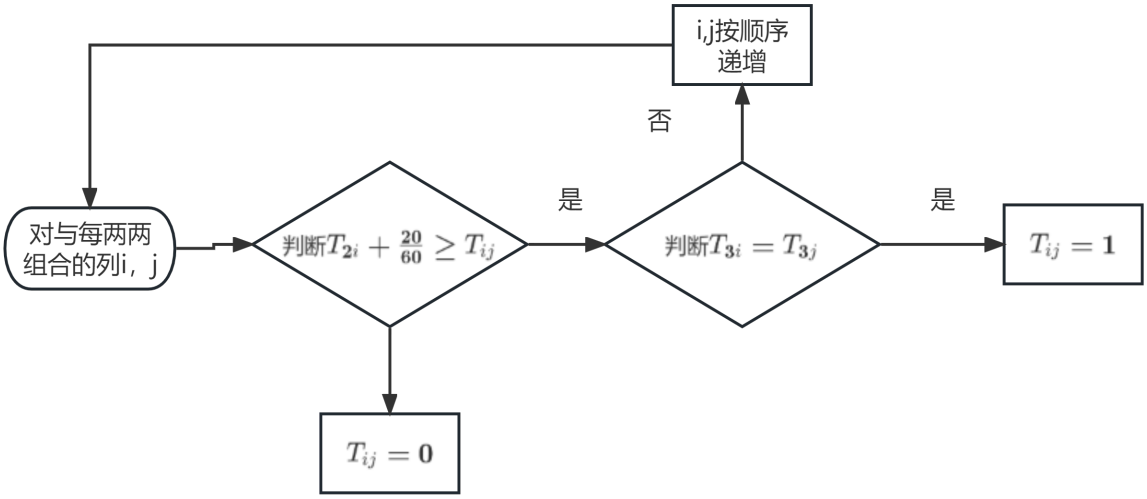


图 4　构建修改约束后的时间邻接矩阵

这里计算了 20 分钟可达的站点，记修改后的时间邻接矩阵为 tm，可得修改后的时空邻接矩阵 C_1

$$C_1 = Pe\&tm \tag{6}$$

然后小休后可达站点矩阵 C_2 为

$$C_2 = C - C_1 \tag{7}$$

现在求最后一段不可行子链的数目，考虑一条不可行子链，它明显是有限的，而且剔除任意元素，剩下的集合在其序下有极小元（即最小不可行任务链去掉首或尾或只剩一个元素），那么它一定是良序的。根据良序定理，

定理 1 (良序定理) 设集合 $(S,\leq)$ 为一全序集，$\leq$ 是其全序关系，若对任意的 S 的非空子集，在其序下都有极小元，则称 $\leq$ 为良序关系，$(S,\leq)$ 为良序集。

$\leq$ 在这里规定为时间空间顺序，由 6.1.1 里的方法，确定任意元素和其他元素的时空顺序。

在这条子链上指定一个点，可以发现它由两部分组成，一部分是指定点衍生出去的不可行子链，一部分是延后的不可行子链。如下图所示

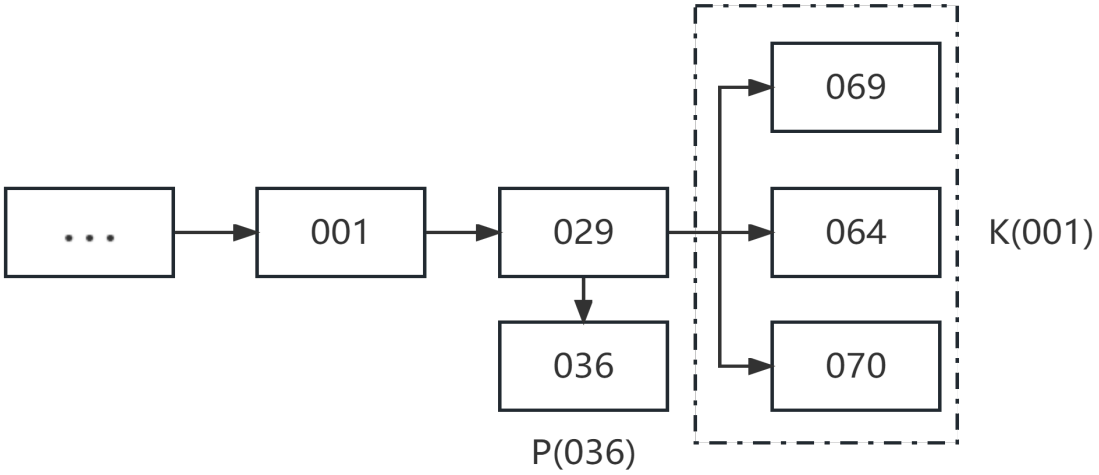


图 5　不可行子链分段

以路线 001，029 进行小休后，再以 036，071 为新起点的部分记为 $P(001)$，不进行小休，连续工作两小时以上的不可行子链记为 $K(001)$。由良序定理一定可以将子链分为极小元求解，实际上等价于将乱序集按某种方式分割为两个不同的良序集，对不同的良序集分别求其分割上界。采取分而治之和递归的方法，形如

$$\begin{aligned} &T\left(001\right) \\ &= K\left(001\right) + P\left(001\right) + P\left(029\right) \\ &P\left(001\right) \\ &= T\left(036\right) + \cdots + T\left(001\right) \\ &P\left(029\right) \\ &= T\left(071\right) + \cdots \end{aligned} \tag{8}$$

将式（8）推广到所有点

$$T\left(i\right) = K\left(i\right) + \sum P\left(j\right) \tag{9}$$

其中 j 是 i 站点为起点可连续驾驶不超过 2 小时的元素。

$$P\left(j\right) = \sum T\left(l\right) \tag{10}$$

l 是 C_2 中第 j 行不为 0 的列序号。

接着求解延后子链 K，仍然将它分解为当前序列的极小元，流程图如下

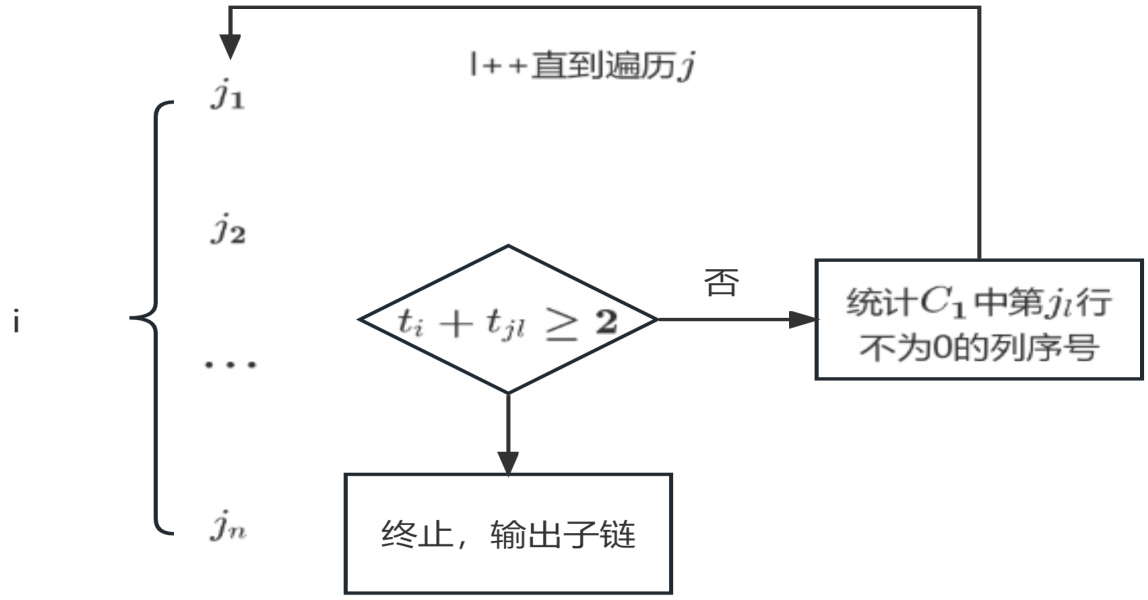


图 6　求 K 类型子链流程

其中 $j_1, j_2, \cdots, j_n$ 是 C_1 中第 i 行不为 0 的列序号。这样子链的求解过程像从树的枝干搜索到叶子。

对于某一任务链，去掉第一段求子链数同以上过程，只不过将有关行处理变为列处理，列处理变为行处理。

6.1.3 结果求解与分析

当然，只有一个元素的集合中 T, P, K 恒为 0。理论上对于一个有 n 个可达邻接点的点 O，记它后面还有 m 个点，因为 P 链被不停的累加，可以估计以点 O 为起点的子链不会超过

$$\alpha \frac{n!}{2m^n} \tag{11}$$

α 跟集合的稀疏性有关，子链总数记为 m，将 R 和 T 加在一起并减去一个 k，有

$$m = \sum_{i=1}^{242} R(i) + T(i) - k(i) \tag{12}$$

初步估计第一个点的子链数数量级为 10 万，总数量级为百万。最后计算得 m=920138。这里给出三条子链，分别为

- P1,P29,P64
- P100,P125,P145,P174
- P1,P41,P91,P125,P145,P179,P202

绘制时间图像如下

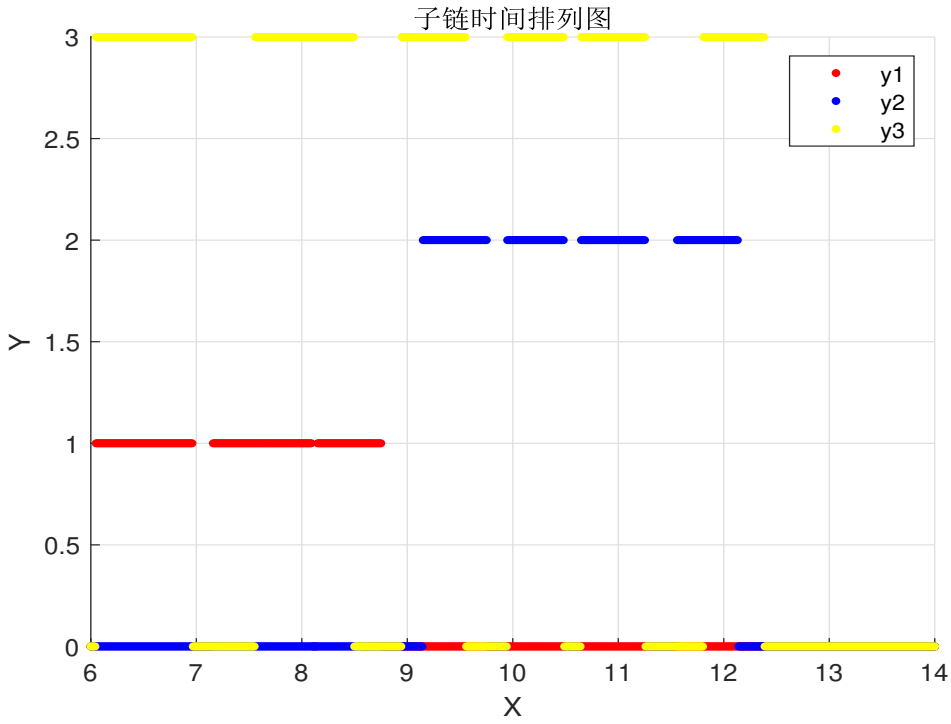
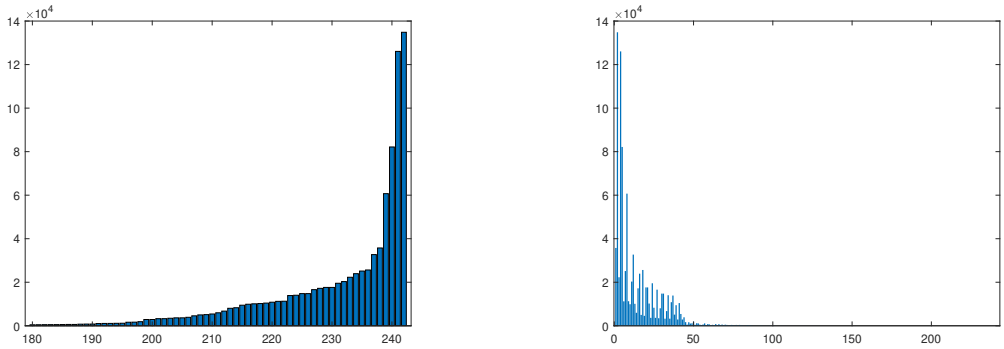


图 7　三条子链

横坐标是时间，不同的 Y 值代表不同的子链，Y=0 的线是所有子链的集合。然后按不同公交线路的子链数的多少排序，有



(a) 子链数从低到高排序　　**(b) 横坐标对应的子链数**

图 8　子链数目与公交序号的关系

图 a 横坐标是公交线路的序号，纵坐标是子链的个数，这里并未可视化子链数在 180 条以下的公交线路。可以看出子链呈现长尾分布，最大值达到 10^5 数量级。以网络的角度来看，不同公交线路的联通性（邻接点个数）不平衡，少部分点的联通性尤其强，衍生的子链数可占总数的 10% 以上。从图 b 可以看出序号越小的公交线路子链越可能大，因为在时空邻接矩阵中它们的邻接点最多。

6.2 问题二：基于蚁群算法的集合覆盖求解模型

6.2.1 基于子链集合的规划域确定

一般的排班问题直接应用列生成法求解，虽然列生成法具有模型简单、易于处理各种复杂约束以及结果效果好等优点，但由于模型本身缺乏各种规章约束参数的显性表达，计算时间相等较长，而且一般的灵敏度分析方法并不适用。我们根据第一问中的子链为基础缩小规划域

由于需要的任务链要尽可能少，我们首先考虑工作时长达到最大或工作距离达到最大的无衔接站点的最长任务子集，记出发站点 A_i^d 构成的子链集合为 S_i，根据以下步骤求 S_i。

Algorithm 1 子链集合的计算

出发站点 A_i^d，时空邻接矩阵 C，考虑小休的时空邻接矩阵 C_2
for C 中的每一行 **do**
  for 每个元素 **do**
    if 遇到元素为一 **then**
      返回初始 FOR 语句，行数加 1，将该公交信息加入任务链。
    end if
    if 时间大于 2 **then**
      需要小休，下一次遍历 C_2 中对应行。
    end if
    if 工作时间或工作时长达到最大或工作距离达到最大 **then**
      跳出这一行
    end if
  end for
end for
返回 S_i

这个算法统计了每个出发站点的最长子链的的集合，极大的缩小了规划域，方便后续求解。

6.2.2 基于蚁群算法的集合覆盖求解

用子集去覆盖全集，这是一个集合覆盖问题，而且允许子集有交集，首先，我们确立了约束

- **时空约束：**由时空约束矩阵给出。

- **总工作时长约束：**一个司机的总工作时长为

$$V_k = \sum x_i(t_i^a - t_i^d) \tag{13}$$

有

$$V_k \leq 8 \tag{14}$$

- **行驶里程约束：**

$$\sum \rho x_i(A_i^a - A_i^d) \leq 200 \tag{15}$$

其中 $\rho(A_i^a - A_i^d)$ 可以认为是两地之间的里程数，ρ 视为一种权重。

- **出勤率约束：**

$$\sum \frac{x_k^A}{x_k^B} \geq 50\% \tag{16}$$

其中 x_k^A 表示从某个出发点是 A 的出勤的司机。

目标为

$$\min \sum k \tag{17}$$

建立约束和目标后，利用蚁群算法求解，算法流程如下

Algorithm 2 蚁群算法（Ant Colony Optimization, ACO）

```
初始化蚂蚁群和信息素矩阵
for 迭代次数小于 200 do
  for S_i 中每一个元素 do
    将 m 个蚂蚁均匀地分配给 n 个子集，根据信息素和启发式信息选择下一节点
    if 完成一次完整的路径 then
      更新当前蚂蚁的路径
      if 当前路径长度小于历史最优路径长度 then
        更新历史最优路径和长度
      end if
    end if
  end for
  if 连续 n 次迭代没有发现更好的解 then
    交换信息素
  end if
  根据所有蚂蚁的路径长度更新信息素
end for
返回历史最优路径
```

这种算法通过模拟蚂蚁通过信息素的交流和合作寻找食物的行为来求解问题。在这个过程中，每一只蚁都可以通过踪迹信息素来感知其他蚂蚁的行为，并根据这些信息选择自己的行动。这种机制使得整个蚁群能够在一定的时间内找到最优的解决方案。

具体的，在选择下一节点时，每一只蚂蚁按照概率 P 选择 S_i 中的一个子集 $S_i^{'}$，P 为

$$P=\begin{cases}\text{argmax}\left\{\left|\tau^{\alpha}\right|\cdot\left|\eta^{\beta}\right|\right\}, & \text{if } q\leqslant q_0\\ \frac{\tau^{\alpha}\eta^{\beta}}{\sum_{h\in S_i}\tau_h^{\alpha}\eta_h^{\beta}}, & \text{else}\end{cases} \tag{18}$$

其中 α 为子集合上残留信息的重要程度，β 为启发信息的重要程度，q 为一个在区间 [0,1] 内的随机数，q_0 是预先设定的值，在 [0,1] 内。η 是启发信息，反映了解是最优解的可能程度，η 由以下公式计算

$$\eta=a\frac{1}{P\left(S_i^{'}\right)-1} \tag{19}$$

$P\left(S_i^{'}\right)$ 是选取的子集 $S_i^{'}$ 的覆盖度，a 是可变参数。覆盖度定义为子集占全集的长度

$$P\left(S_i^{'}\right)=\frac{S_i-S_i^{'}}{S_i^{'}} \tag{20}$$

信息素是蚁群算法的核心，信息素更新按下式进行

$$\begin{aligned}&\tau_j(t+1)=(1-\rho)\tau_j(t)+\textstyle\sum_K\Delta\tau_j^k\\&\Delta\tau_j^k=\begin{cases}Q/|S|,\ \text{如果蚂蚁}k\ \text{选择了集合}C_j\\0,\qquad\qquad\qquad\text{否则}\end{cases}\end{aligned} \tag{21}$$

Q 为一个常数，其中 $0\leq\rho\leq1$，该参数确定信息素的消逝速度。$|S|$ 是蚂蚁 k 寻找到的子集个数。当蚁群算法连续多次迭代仍然寻不到更好的解时

$$\tau_k=\begin{cases}\tau_{\text{min}},\ \text{if}\ \tau_k=\tau_{\text{max}}\\\tau_{\text{max}},\ \text{if}\ \tau_k=\tau_{\text{min}}\end{cases} \tag{22}$$

上式可以有效防止算法陷入局部最优解。

6.2.3 结果分析

考虑蚁群算法在此问题的复杂度，设有 m 只蚂蚁，6.2.1 中子集里有 n 个元素

1. 初始化参数的时间复复杂度为 $O(n^2+m)$。
2. 每只蚂蚁单独构造解的时间复杂度为 $O(n^2m)$。
3. 解的评价和轨迹更新量的计算 $O(n^2m)$。
4. 信息素更新的时间复杂度为 $O(n^2)$。

5. 判断是否终止的时间复杂度为 $O(nm)$。当 n 足够大时，低次幂的影响可以忽略不计，基本蚁群算法中 m 只蚂蚁遍历 n 个城市，这里经过 200 次循环，则整个蚁群算法的时间复杂度为

$$T(n) = O(200n^2m) \tag{23}$$

基本蚁群算法的空间复杂度为

$$S(n) = O(n^2) + O(nm) \tag{24}$$

由此可见，基本蚁群算法是易于存储易于实现的。

设置参数为：蚂蚁个数 50，信息素重要程度 $\alpha = 1$，启发式因子重要程度 $\beta = 5$，信息素蒸发因子 $\rho = 0.1$，最大迭代次数 200 次，结果如下

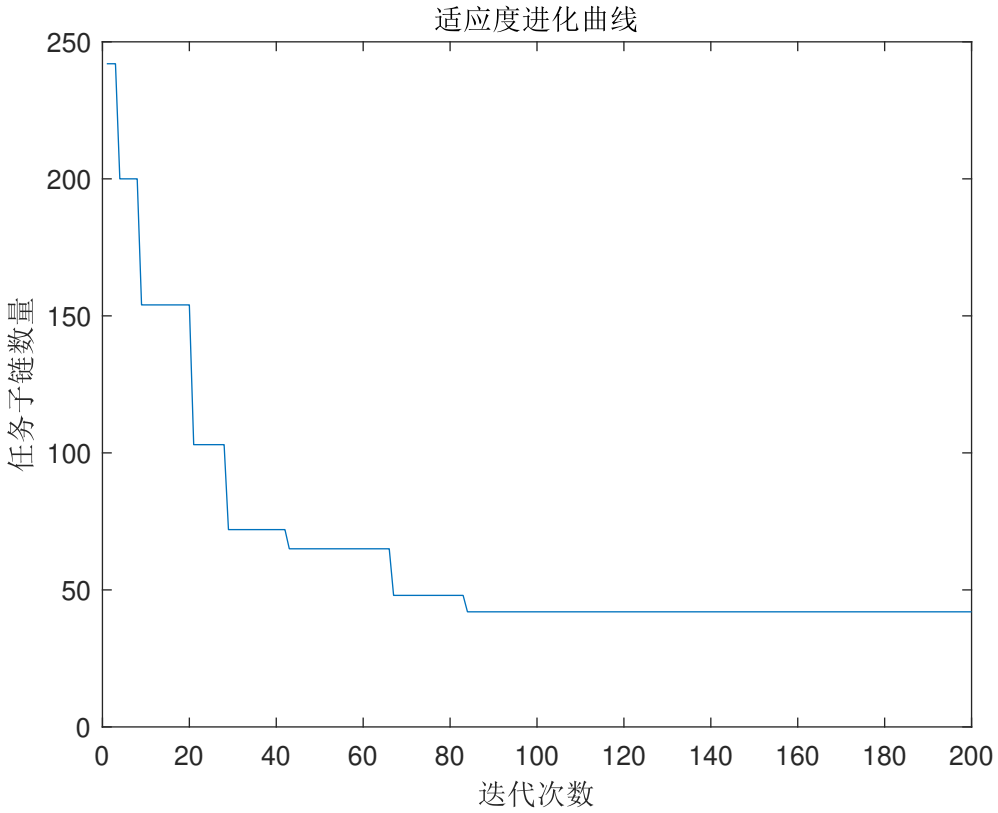


图 9　蚁群算法参数改变对任务链的影响

即最多只需要 42 名司机就可以覆盖所有公交线路。还可以看出算法在迭代次数 85 次左右时已经收敛，证明它的计算速度是比较快的。结果放在附录。

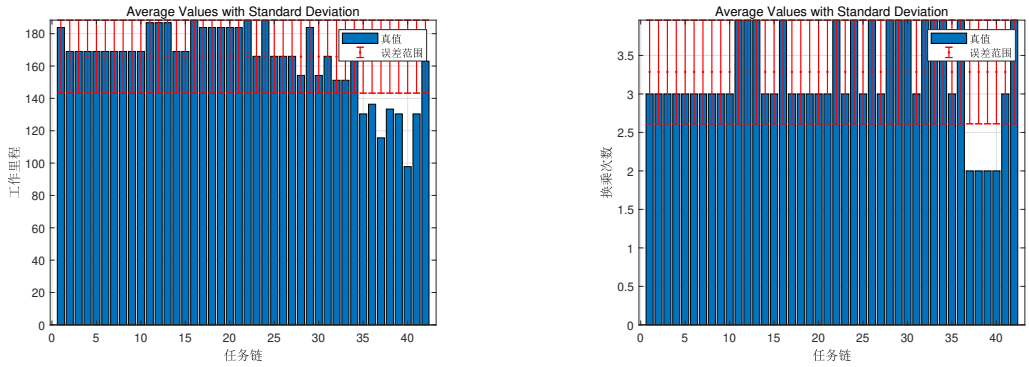
6.3 问题三：考虑多指标的改进模型

6.3.1 多指标的模型分析

为了对当前的最优列车司机排班方案进行全面的分析，我们需要考虑多个衡量指标。主要从司机的身体心理健康和公交运行的效率出发，以下是我们选取的指标：

- **小休次数：** 司机在一天内的小休次数。长时间的旅途奔波易使机组人员精神和体力疲劳, 合理的小休安排有助于缩短机组人员的环境适应过程以帮助其更好地休息, 从而能够促进其疲劳恢复, 保证后续驾驶的安全性
- **司机驾驶时长：** 司机在一天内的总驾驶时间。这是一个重要的安全指标。长时间的驾驶可能会增加司机疲劳和疏忽的风险，因此需要限制驾驶时长，保证司机的休息和安全。
- **休息时长：** 司机在一天内的总休息时间。考虑到司机的每一次小休并不一定只有 20 分钟，过高的小休影响公交系统效率，增加人力成本，过低的小休时间有可能导致司机精神不适。因此，需要合理安排休息时间，以保证司机能够得到充分的休息，同时不影响公交系统的效率。
- **工作里程：** 司机一天内的总行驶里程。这个指标反映了司机的劳动强度和工作量，过高的工作里程可能会导致司机疲劳和疏忽，增加安全风险。因此，需要合理安排工作里程，保证司机的劳动强度和工作量适中。
- **换乘次数：** 司机在一天内更换驾驶车辆的次数。过多的换乘次数会增加司机的工作量和疲劳程度，同时也会影响公交系统的效率。因此，需要合理安排换乘次数，保证司机的工作量和安全。

在分析这些指标时，我们计算每个指标的均值和标准差，以评估其分布的均衡性。均值可以反映整体水平，而标准差可以反映个体差异的大小。可视化指标如下

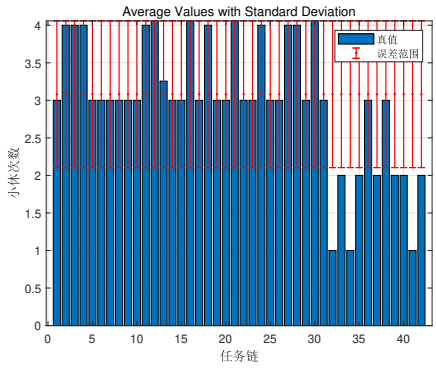


(a) 工作里程 **(b) 换乘次数**

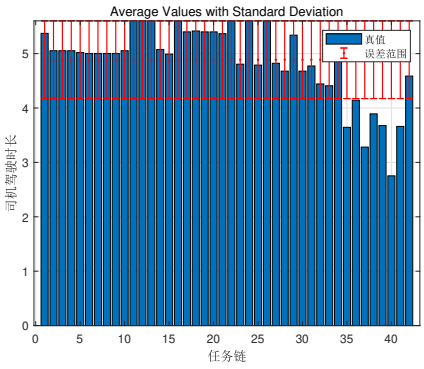
图 10　司机的后两个指标

司机每日工作里程在 170 左右，既保证了司机的健康，又保证了整个公交系统的效

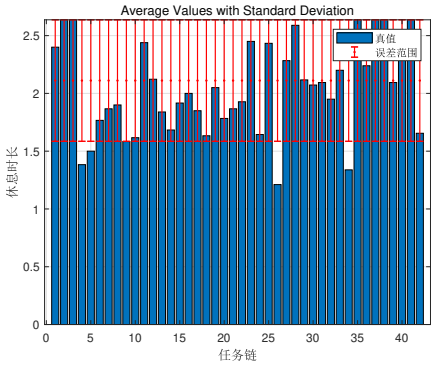
率。换乘次数符合日常经验，可以认为较为合理。



(a) 小休次数



(b) 司机驾驶时长



(c) 休息时长

图 11　司机的前三个指标

红色区域中的点是均值，宽度代表了标准差。可以看出不同司机的各指标之间略有变化幅度，大部分人的指标在均值附近，可以以均值为基准实施差异化管理，发挥不同司机的特长和能力。司机的驾驶时长普遍不长，较为合理，小休次数不多但是每次的休息时间可观

6.3.2 基于多指标的模型修正

根据以上的多个指标，我们可以改进问题二中的优化目标和蚁群算法，将单目标优化转变为多目标优化，假设第 k 个目标用 D_k 表示，我们可以将式 (17) 修正为

$$\min \sum_k \lambda_k D_k \tag{25}$$

其中 λ_k 是第 k 的目标的权重，应该有

$$\sum_k \lambda_k = 1 \tag{26}$$

λ 越大，与其对应的目标也就越重要，这样问题二中的算法可以推广到任意多目标系统中。

对于蚁群算法本身，我们可以引入动态自适应信息素机制，令

$$\Delta\tau_i^k(t) = f(t) = \frac{Q(t)}{L_k} \tag{27}$$

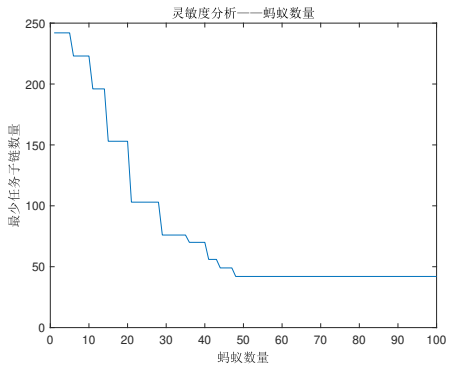
这里 $Q(t)$ 是随时间的函数，一般设为单调函数，比如

$$Q(t) = be^{ct} \tag{28}$$

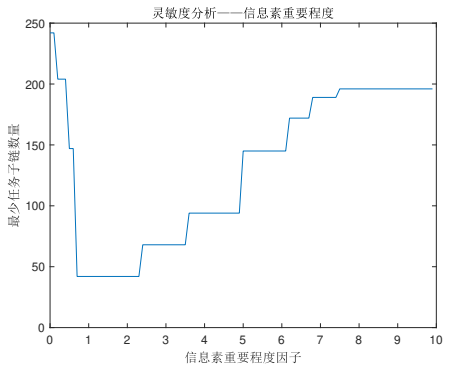
b, c 是参数。还可以将信息素按高速分布在蚁群内传播，或引入精英机制，或者结合遗传算法让蚂蚁不断产生下一代，这些算法改进过程较为繁琐冗长，这里不予赘述。

七、 灵敏性分析

我们的模型里只有蚁群算法是依赖预先设置的参数，接下来我们把蚂蚁个数，信息素蒸发系数和信息素重要程度参数改变，观察最小任务子链数量的变化情况，有



(a) 修改蚂蚁数量



(b) 修改信息素重要因子

图 12　关于蚁群算法的灵敏性分析

可以看到在较大程度的改变蚂蚁参数后，需要的任务子链数量仍然没有改变，即算法的鲁棒性非常好，改变蚂蚁参数并不会使结果有明显变化，这可能是由于引入式 (23) 以跳出局部最优解，而信息素重要因子会影响算法的收敛速度，过大的值将导致在几次迭代后收敛，而过小会导致混沌现象，但是可以看到在常规取值段 [1,2.5] 之间计算结果并未变化，这也证明模型是可依赖的，可推广的。

八、 模型总结

8.1 模型优点

- 利用集合中序的概念简化问题，使用矩阵之间的运算简化了数值求解过程，大大加快了求解的速度。

- 使用方法求解集合覆盖问题，与传统的列生成法或分支定界法相比减少了计算开销。
- 目标优化中考虑了司机的身体和精神健康，让公交线路设计在多个角度得到优化。

8.2 模型缺点

- 模型求解的结果对数据精确度要求高，遇到公交延误等特殊情况难以进行有效调整。
- 改进模型的指标并不能完全反映排班方案的合理性。还需要考虑其他因素，例如工作强度、连续工作时间、休息时间的质量等。

参考文献

[1] 葛洪伟, 高阳. 基于蚁群算法的集合覆盖问题 [J]. 计算机工程与应用,2007,43(4):49-50,105.

附录 A　任务链

表 3　任务链表 1

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
1	P016	06:47:10	PA	07:21:11	PB	1
1	P035	07:21:11	PB	08:15:43	PC	1
1	P078	08:35:43	PC	09:30:32	PB	2
1	P116	09:45:32	PB	10:21:10	PA	2
1	P148	10:45:10	PA	11:18:11	PB	1
1	P165	11:18:11	PB	12:12:43	PC	1
1	P232	13:37:43	PC	14:33:32	PB	1
2	P002	06:04:32	PB	06:40:10	PA	2
2	P024	07:02:10	PA	07:37:11	PB	1
2	P043	07:37:11	PB	08:31:43	PC	1
2	P087	08:51:43	PC	09:45:32	PB	2
2	P118	09:48:32	PB	10:24:10	PA	2
2	P154	10:54:10	PA	11:28:11	PB	1
2	P224	13:23:11	PB	14:17:43	PC	1
3	P004	06:11:32	PB	06:47:10	PA	2
3	P027	07:07:10	PA	07:41:11	PB	1
3	P046	07:41:11	PB	08:35:43	PC	1
3	P089	08:55:43	PC	09:51:32	PB	2
3	P123	09:54:32	PB	10:30:10	PA	2
3	P156	11:00:10	PA	11:33:11	PB	1
3	P230	13:33:11	PB	14:27:43	PC	1

表 4　任务链表 2

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
4	P005	06:16:32	PB	06:52:10	PA	2
4	P030	07:12:10	PA	07:45:11	PB	1
4	P051	07:49:11	PB	08:43:43	PC	1
4	P097	09:07:43	PC	10:03:32	PB	2
4	P128	10:03:32	PB	10:39:10	PA	2
4	P160	11:09:10	PA	11:43:11	PB	1
4	P178	11:48:11	PB	12:42:43	PC	1
5	P008	06:26:32	PB	07:02:10	PA	2
5	P031	07:17:10	PA	07:49:11	PB	1
5	P063	08:09:11	PB	09:03:43	PC	1
5	P104	09:23:43	PC	10:18:32	PB	2
5	P141	10:30:32	PB	11:06:10	PA	2
5	P172	11:29:10	PA	12:03:11	PB	1
5	P186	12:03:11	PB	12:57:43	PC	1
6	P012	06:36:32	PB	07:12:10	PA	2
6	P037	07:25:10	PA	07:57:11	PB	1
6	P069	08:17:11	PB	09:11:43	PC	1
6	P107	09:27:43	PC	10:21:32	PB	2
6	P149	10:45:32	PB	11:21:10	PA	2
6	P174	11:34:10	PA	12:08:11	PB	1
6	P195	12:28:11	PB	13:22:43	PC	1
7	P015	06:45:32	PB	07:21:10	PA	2
7	P039	07:33:10	PA	08:05:11	PB	1
7	P076	08:33:11	PB	09:27:43	PC	1
7	P114	09:39:43	PC	10:33:32	PB	2
7	P157	11:03:32	PB	11:39:10	PA	2

表 5　任务链表 3

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
8	P020	06:53:32	PB	07:29:10	PA	2
8	P049	07:45:10	PA	08:17:11	PB	1
8	P082	08:41:11	PB	09:35:43	PC	1
8	P121	09:51:43	PC	10:45:32	PB	2
8	P163	11:13:32	PB	11:49:10	PA	2
8	P184	11:59:10	PA	12:33:11	PB	1
8	P211	12:53:11	PB	13:47:43	PC	1
9	P021	06:57:32	PB	07:33:10	PA	2
9	P049	07:45:10	PA	08:17:11	PB	1
9	P082	08:41:11	PB	09:35:43	PC	1
9	P124	09:55:43	PC	10:49:32	PB	2
9	P151	10:49:32	PB	11:25:10	PA	2
9	P188	12:04:10	PA	12:38:11	PB	1
9	P201	12:38:11	PB	13:32:43	PC	1
10	P033	07:17:32	PB	07:53:10	PA	2
10	P066	08:13:10	PA	08:45:11	PB	1
10	P088	08:53:11	PB	09:47:43	PC	1
10	P133	10:11:43	PC	11:08:32	PB	2
10	P166	11:18:32	PB	11:54:10	PA	2
10	P196	12:29:10	PA	13:03:11	PB	1
10	P214	13:03:11	PB	13:57:43	PC	1

表 6　任务链表 4

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
11	P007	06:23:43	PC	07:21:32	PB	2
11	P038	07:25:32	PB	08:01:10	PA	2
11	P073	08:29:10	PA	09:01:11	PB	1
11	P094	09:01:11	PB	09:55:43	PC	1
11	P136	10:19:43	PC	11:13:32	PB	2
11	P168	11:23:32	PB	11:59:10	PA	2
11	P199	12:34:10	PA	13:08:11	PB	1
11	P239	13:53:32	PB	14:29:10	PA	2
12	P011	06:35:43	PC	07:33:32	PB	2
12	P044	07:37:32	PB	08:13:10	PA	2
12	P075	08:33:10	PA	09:05:11	PB	1
12	P099	09:09:11	PB	10:03:43	PC	1
12	P139	10:27:43	PC	11:23:32	PB	2
12	P176	11:43:32	PB	12:19:10	PA	2
12	P203	12:39:10	PA	13:13:11	PB	1
12	P237	13:48:32	PB	14:24:10	PA	2
13	P013	06:43:43	PC	07:41:32	PB	2
13	P047	07:41:32	PB	08:17:10	PA	2
13	P080	08:37:10	PA	09:09:11	PB	1
13	P102	09:13:11	PB	10:07:43	PC	1
13	P147	10:43:43	PC	11:38:32	PB	2
13	P182	11:53:32	PB	12:29:10	PA	2
13	P210	12:49:10	PA	13:23:11	PB	1
13	P234	13:38:32	PB	14:14:10	PA	2
14	P022	06:57:43	PC	07:53:32	PB	2
14	P059	08:01:32	PB	08:37:10	PA	2

表 7　任务链表 5

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
15	P025	07:03:43	PC	07:57:32	PB	2
15	P064	08:09:32	PB	08:45:10	PA	2
15	P096	09:05:10	PA	09:37:11	PB	1
15	P113	09:37:11	PB	10:31:43	PC	1
15	P155	10:59:43	PC	11:53:32	PB	2
15	P202	12:38:32	PB	13:14:10	PA	2
15	P226	13:24:10	PA	13:58:11	PB	1
16	P001	06:03:11	PB	06:57:43	PC	1
16	P029	07:09:43	PC	08:05:32	PB	2
16	P071	08:25:32	PB	09:01:10	PA	2
16	P105	09:24:10	PA	09:57:11	PB	1
16	P126	09:57:11	PB	10:51:43	PC	1
16	P161	11:11:43	PC	12:08:32	PB	2
16	P206	12:43:32	PB	13:19:10	PA	2
16	P229	13:29:10	PA	14:03:11	PB	1
17	P003	06:09:11	PB	07:03:43	PC	1
17	P036	07:21:43	PC	08:17:32	PB	2
17	P084	08:45:32	PB	09:21:10	PA	2
17	P109	09:33:10	PA	10:05:11	PB	1
17	P140	10:29:11	PB	11:23:43	PC	1
17	P177	11:47:43	PC	12:43:32	PB	2
17	P209	12:48:32	PB	13:24:10	PA	2
18	P006	06:21:11	PB	07:15:43	PC	1
18	P041	07:33:43	PC	08:29:32	PB	2
18	P086	08:51:32	PB	09:27:10	PA	2
18	P117	09:48:10	PA	10:21:11	PB	1

表 8　任务链表 6

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
19	P009	06:27:11	PB	07:21:43	PC	1
19	P045	07:39:43	PC	08:33:32	PB	2
19	P091	08:57:32	PB	09:33:10	PA	2
19	P120	09:51:10	PA	10:25:11	PB	1
19	P150	10:48:11	PB	11:42:43	PC	1
19	P183	11:57:43	PC	12:53:32	PB	2
19	P222	13:18:32	PB	13:54:10	PA	2
20	P010	06:33:11	PB	07:27:43	PC	1
20	P048	07:43:43	PC	08:39:32	PB	2
20	P095	09:03:32	PB	09:39:10	PA	2
20	P125	09:57:10	PA	10:29:11	PB	1
20	P153	10:53:11	PB	11:47:43	PC	1
20	P189	12:07:43	PC	13:03:32	PB	2
20	P217	13:08:32	PB	13:44:10	PA	2
21	P014	06:45:11	PB	07:39:43	PC	1
21	P053	07:51:43	PC	08:45:32	PB	2
21	P100	09:09:32	PB	09:45:10	PA	2
21	P129	10:06:10	PA	10:38:11	PB	1
21	P146	10:43:11	PB	11:37:43	PC	1
21	P185	12:02:43	PC	12:58:32	PB	2
21	P225	13:23:32	PB	13:59:10	PA	2
22	P017	06:49:11	PB	07:43:43	PC	1
22	P054	07:55:43	PC	08:51:32	PB	2
22	P106	09:24:32	PB	10:00:10	PA	2
22	P137	10:21:10	PA	10:53:11	PB	1
22	P159	11:08:11	PB	12:02:43	PC	1

表 9　任务链表 7

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
23	P019	06:53:11	PB	07:47:43	PC	1
23	P056	07:59:43	PC	08:54:32	PB	2
23	P127	10:00:32	PB	10:36:10	PA	2
23	P164	11:15:10	PA	11:48:11	PB	1
23	P181	11:53:11	PB	12:47:43	PC	1
23	P219	13:12:43	PC	14:08:32	PB	2
24	P023	07:01:11	PB	07:55:43	PC	1
24	P067	08:15:43	PC	09:09:32	PB	2
24	P100	09:09:32	PB	09:45:10	PA	2
24	P130	10:09:10	PA	10:43:11	PB	1
24	P162	11:13:11	PB	12:07:43	PC	1
24	P194	12:27:43	PC	13:23:32	PB	2
24	P228	13:28:11	PB	14:22:43	PC	1
25	P026	07:05:11	PB	07:59:43	PC	1
25	P065	08:11:43	PC	09:06:32	PB	2
25	P132	10:09:32	PB	10:45:10	PA	2
25	P167	11:21:10	PA	11:53:11	PB	1
25	P190	12:13:11	PB	13:07:43	PC	1
25	P223	13:22:43	PC	14:18:32	PB	2
26	P028	07:09:11	PB	08:03:43	PC	1
26	P072	08:27:43	PC	09:21:32	PB	2
26	P106	09:24:32	PB	10:00:10	PA	2
26	P137	10:21:10	PA	10:53:11	PB	1
26	P153	10:53:11	PB	11:47:43	PC	1
26	P189	12:07:43	PC	13:03:32	PB	2
26	P216	13:08:11	PB	14:02:43	PC	1

表 10　任务链表 8

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
27	P032	07:17:11	PB	08:11:43	PC	1
27	P083	08:43:43	PC	09:39:32	PB	2
27	P134	10:15:32	PB	10:51:10	PA	2
27	P169	11:25:10	PA	11:58:11	PB	1
27	P192	12:18:11	PB	13:12:43	PC	1
27	P227	13:27:43	PC	14:23:32	PB	2
28	P018	06:52:10	PA	07:25:11	PB	1
28	P052	07:49:32	PB	08:25:10	PA	2
28	P081	08:41:10	PA	09:13:11	PB	1
28	P115	09:45:11	PB	10:39:43	PC	1
28	P158	11:07:43	PC	12:03:32	PB	2
28	P209	12:48:32	PB	13:24:10	PA	2
28	P231	13:34:10	PA	14:08:11	PB	1
29	P034	07:21:10	PA	07:53:11	PB	1
29	P070	08:21:32	PB	08:57:10	PA	2
29	P098	09:09:10	PA	09:41:11	PB	1
29	P131	10:09:11	PB	11:03:43	PC	1
29	P170	11:27:43	PC	12:23:32	PB	2
29	P205	12:43:11	PB	13:37:43	PC	1
29	P238	13:52:43	PC	14:48:32	PB	1
30	P039	07:33:10	PA	08:05:11	PB	1
30	P074	08:29:32	PB	09:05:10	PA	2
30	P112	09:36:10	PA	10:09:11	PB	1
30	P135	10:17:11	PB	11:11:43	PC	1
30	P173	11:32:43	PC	12:28:32	PB	2
30	P209	12:48:32	PB	13:24:10	PA	2

表 11　任务链表 9

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
31	P042	07:37:10	PA	08:09:11	PB	1
31	P090	08:57:11	PB	09:51:43	PC	1
31	P136	10:19:43	PC	11:13:32	PB	2
31	P181	11:53:11	PB	12:47:43	PC	1
31	P213	12:57:43	PC	13:53:32	PB	2
31	P239	13:53:32	PB	14:29:10	PA	2
32	P050	07:49:10	PA	08:21:11	PB	1
32	P079	08:36:32	PB	09:12:10	PA	2
32	P138	10:24:10	PA	10:58:11	PB	1
32	P162	11:13:11	PB	12:07:43	PC	1
32	P191	12:17:43	PC	13:13:32	PB	2
32	P221	13:18:11	PB	14:12:43	PC	1
33	P057	08:01:10	PA	08:33:11	PB	1
33	P085	08:48:32	PB	09:24:10	PA	2
33	P144	10:36:10	PA	11:08:11	PB	1
33	P159	11:08:11	PB	12:02:43	PC	1
33	P197	12:32:43	PC	13:28:32	PB	2
33	P235	13:43:11	PB	14:37:43	PC	1
34	P062	08:09:10	PA	08:41:11	PB	1
34	P090	08:57:11	PB	09:51:43	PC	1
34	P133	10:11:43	PC	11:08:32	PB	2
34	P171	11:28:11	PB	12:22:43	PC	1
34	P200	12:37:43	PC	13:33:32	PB	2
34	P235	13:43:11	PB	14:37:43	PC	1

表 12　任务链表 10

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
35	P060	08:03:43	PC	08:57:32	PB	2
35	P122	09:53:11	PB	10:47:43	PC	1
35	P193	12:22:43	PC	13:18:32	PB	2
35	P230	13:33:11	PB	14:27:43	PC	1
36	P061	08:05:11	PB	08:59:43	PC	1
36	P092	08:59:43	PC	09:54:32	PB	1
36	P143	10:33:32	PB	11:09:10	PA	2
36	P174	11:34:10	PA	12:08:11	PB	1
36	P215	13:03:32	PB	13:39:10	PA	2
36	P240	13:54:10	PA	14:28:11	PB	1
37	P068	08:17:10	PA	08:49:11	PB	1
37	P103	09:17:11	PB	10:11:43	PC	1
37	P204	12:42:43	PC	13:38:32	PB	2
37	P242	13:58:11	PB	14:52:43	PC	1
38	P058	08:01:11	PB	08:55:43	PC	1
38	P101	09:11:43	PC	10:06:32	PB	2
38	P145	10:39:32	PB	11:15:10	PA	2
38	P220	13:14:10	PA	13:48:11	PB	1
38	P242	13:58:11	PB	14:52:43	PC	1
39	P055	07:57:11	PB	08:51:43	PC	1
39	P108	09:31:43	PC	10:27:32	PB	2
39	P175	11:43:11	PB	12:37:43	PC	1
39	P207	12:47:43	PC	13:43:32	PB	2

表 13　任务链表 11

任务链序号	车次	接车时间	接车站点	下车时间	下车站点	流向
40	P111	09:35:43	PC	10:30:32	PB	2
40	P195	12:28:11	PB	13:22:43	PC	1
40	P241	13:57:43	PC	14:53:32	PB	2
41	P058	08:01:11	PB	08:55:43	PC	1
41	P101	09:11:43	PC	10:06:32	PB	2
41	P208	12:48:11	PB	13:42:43	PC	1
41	P241	13:57:43	PC	14:53:32	PB	2
42	P040	07:33:11	PB	08:27:43	PC	1
42	P083	08:43:43	PC	09:39:32	PB	2
42	P119	09:49:11	PB	10:43:43	PC	1
42	P170	11:27:43	PC	12:23:32	PB	2
42	P211	12:53:11	PB	13:47:43	PC	1

附录 B　代码

```
   #include<bits/stdc++.h>
using namespace std;
// const
const int INF = 0x3f3f3f3f;
#define sqr(x) ((x)*(x))
#define eps 1e-8
//variables
string file_name;
int type;// type == 1 全矩阵, type == 2 二维欧拉距离
int N;//地铁数量
double **dis;//地铁间距离
double **pheromone;//信息素
double **herustic;//启发式值
double **info;// info = pheromone ^ delta * herustic ^ beta
double pheromone_0;//pheromone初始值, 这里是1 / (avg * N)其中avg为图网中所有边边权的平均数。
int m;//种群数量
int delta, beta;//参数
```

```
double alpha;
int *r1, *s, *r;//agent k的出发地铁，下一个点，当前点。
int MAX, iteration;//最大迭代次数，迭代计数变量
set<int> empty, *J;
struct vertex{
                  double x, y;// 地铁坐标
                  int id;// 地铁编号
                  int input(FILE *fp){
                  return fscanf(fp, "%d %lf %lf", &id,                  &x, &y);
}
}*node;

typedef pair<int, int> pair_int;
struct Tour{//路径
      vector<pair_int> path;//path[i]，存储一              条边(r,s)
     double L;
     void clean(){
            L = INF;
            path.clear();
            path.shrink_to_fit();
      }//清空
     void calc(){
            L = 0;
            int sz = path.size();
            for (int i = 0; i < sz; i ++){
            L += dis[path[i].first][path[i].second];
            }
       }//计算长度
     void push_back(int x, int y){
            path.push_back(make_pair(x, y));
       }
     int size(){
            return (int)path.size();
       }
     int r(int i){
            return path[i].first;
       }
     int s(int i){
            return path[i].second;
       }
     void print(FILE *fp){
            int sz = path.size();
            for (int i = 0; i < sz; i ++){
            fprintf(fp, "%d->", path[i].first + 1);
       }
            fprintf(fp, "%d\n", path[sz - 1].second +           1);
       }
```

```
    bool operator <(const Tour &a)const{
            return L < a.L;
     }//重载
} *tour, best_so_far;

double EUC_2D(const vertex &a, const vertex &b){
    return sqrt(sqr(a.x - b.x) + sqr(a.y - b.y));
}

void io(){//输入
    printf("input file_name and data type\n");
    cin >> file_name >> type;
    FILE *fp = fopen(file_name.c_str(), "r");
    fscanf(fp, "%d", &N);
    node = new vertex[N + 5];
    dis = new double*[N + 5];
    double tmp = 0;
    int cnt = 0;
    if (type == 1){
            for (int i = 0; i < N; i ++){
                dis[i] = new double[N];
                for (int j = 0; j < N; j ++){
                    fscanf(fp, "%lf", &dis[i][j]);
                    tmp += i != j ? dis[i][j] : 0;// i == j的                 时候
                        dis不存在，所以不考虑。
                    cnt += i != j ? 1 : 0;// i == j的时候
                        dis不存在，所以不考虑。
                }
            }
     }else{
          for (int i = 0; i < N; i ++)
          node[i].input(fp);
          for (int i = 0; i < N; i ++){
                dis[i] = new double[N];
                for (int j = 0; j < N; j ++){
                    dis[i][j] = EUC_2D(node[i],                             node[j]);//
                        计算距离
                    tmp += i != j ? dis[i][j] : 0;// i == j的                 时候
                        dis不存在，所以不考虑。
                    cnt += i != j ? 1 : 0;// i == j的时候
                        dis不存在，所以不考虑。
                 }
          }
    }
     pheromone_0 = (double)cnt / (tmp *                 N);//pheromone初始值，这里是1 / (avg
         * N)其中avg为图网中所有边边权的平均数。
     fclose(fp);
```

```
        return;
}

void init(){//初始化
        alpha = 0.1;//evaporation parameter, 挥发参数, 每次信息素要挥发的量
        delta = 1;
        beta = 6;// delta 和 beta分别表示pheromones和herustic的比重
        m = N;
        pheromone = new double*[N + 5];
        herustic = new double*[N + 5];
        info = new double*[N + 5];
        r1 = new int[N + 5];
        r = new int[N + 5];
        s = new int[N + 5];
        J = new set<int>[N + 5];
        empty.clear();
        for (int i = 0; i < N; i ++){
              pheromone[i] = new double[N + 5];
              herustic[i] = new double[N + 5];
              info[i] = new double[N + 5];
              empty.insert(i);
              for (int j = 0; j < N; j ++){
                    pheromone[i][j] = pheromone_0;
                    herustic[i][j] = 1 / (dis[i][j] + eps);//加
                        一个小数eps, 防止被零除
               }
         }
         best_so_far.clean();
         iteration = 0;
         MAX = N * N;
}

double power(double x, int y){
         double ans = 1;
         while (y){
                if (y & 1) ans *= x;
                x *= x;
                y >>= 1;
          }
         return ans;
}

void reset(){
         tour = new Tour[m + 5];
         for (int i = 0; i < N; i ++){
               tour[i].clean();
               r1[i] = i;//初始化出发地铁,
```

```
            J[i] = empty;
            J[i].erase(r1[i]);//初始化agent i需要访问的首发接车地点
            r[i] = r1[i];//当前在出发点
        }
        for (int i = 0; i < N; i ++)
        for (int j = 0; j < N; j ++){
            info[i][j] = power(pheromone[i][j], delta) *        power(herustic[i][j], beta);
        }//选择公式
}

int select_next(int k){
        if (J[k].empty()) return r1[k]; //如果J是空的，那么返回出发点地铁
       double rnd = (double)(rand()) /
           (double)RAND_MAX;//产生0..1的随机数
       set<int>::iterator it = J[k].begin();
       double sum_prob = 0, sum = 0;
       for (; it != J[k].end(); it ++){
            sum += info[r[k]][*it];
       }//计算概率分布
       rnd *= sum;
       it = J[k].begin();
       for (; it != J[k].end(); it ++){
           sum_prob += info[r[k]][*it];
           if (sum_prob >= rnd){
                return *it;
           }
        }//依照概率选取下一步地铁
}

void construct_solution(){
      for (int i = 0; i < N; i ++){
          for (int k = 0; k < m; k ++){
                int next = select_next(k);//选择下一步的              最优决策
                J[k].erase(next);
                s[k] = next;
                tour[k].push_back(r[k], s[k]);
                r[k] = s[k];
           }
      }
}

void update_pheromone(){
      Tour now_best;
      now_best.clean();//初始化
      for (int i = 0; i < m; i ++){
          tour[i].calc();
          if (tour[i] < now_best)
```

```
            now_best = tour[i];//寻找当前迭代最优解
        }
        if (now_best < best_so_far){
            best_so_far = now_best;//更新最优解
        }
        for (int i = 0; i < N; i ++)
        for (int j = 0; j < N; j ++)
        pheromone[i][j] *= (1 - alpha);//信息素挥发
        int sz = now_best.size();
        for (int i = 0; i < sz; i ++){
            pheromone[now_best.r(i)][now_best.s(i)] +=     1. / (double)now_best.L;
            pheromone[now_best.s(i)][now_best.r(i)] =
                pheromone[now_best.r(i)][now_best.s(i)];//       对称
        }//更新信息素含量
}

int main(){
        srand((unsigned) time(0));//初始化随机种子
        io();
        init();
        double last = INF;
        int bad_times = 0;
        for (; iteration < MAX; iteration ++){
            if (bad_times > N) break;//进入局部最优
            reset();//初始化agent信息
            construct_solution();//对于所有的agent构造        一个完整的tour
            update_pheromone();//更新信息素
            printf("iteration %d:best_so_far = %.2lf\n",       iteration, best_so_far.L);
            if (last > best_so_far.L)
            last = best_so_far.L, bad_times = 0;
            else bad_times ++;//记录当前未更新代数，若       迭代多次未更新，认为进入局部最优
        }
        printf("best_so_far = %.2lf\n", best_so_far.L);// 输出目标值
        best_so_far.print(stdout);//输出路径
}
```

```
% 计算均值和标准差
average_data = mean(Total(:, 5));
std_data = std(Total(:, 5));

% 绘制柱状图
figure;
bar(1:1:42, Total(:, 5)');
hold on;
```

```
% 添加误差线（标准差）
errorbar(1:42, average_data * ones(1, 42), std_data * ones(1, 42), 'r.', 'LineWidth', 1);

% 设置坐标轴标签
xlabel('任务链');
ylabel('小休次数');

% 添加标题和图例
title('Average Values with Standard Deviation');
legend('真值', '误差范围');

% 设置图形的其他属性
grid on;
ylim([0, max(average_data) + std_data]);

% 结束绘图
hold off;
```