厦門大學

本 科 课 程 论 文（设 计）

（校选课程）

基于便利原则的校园图书智慧配送问题解决

Solve the problem of intelligent distribution of campus books based on the principle of convenience

姓　　名：盛焕新

学　　号：15220202202189

学　　院：经济学院

专　　业：统计学

年　　级：2020 级

校内指导教师：吴云峰　副教授

二〇二三年六月二十四日

基于便利原则的校园图书智慧配送问题解决

盛焕新 15220202202189

2023 年 6 月 24 日

摘要

人工智能为人类社会的智慧未来点亮了无数可能，而大学校园中的图书借还过程存在一些问题影响了效率和便利性。基于人工智能的智慧校园图书配送系统应运而生，通过人工智能图书配送机器人进行工作，需要规划用时最短的工作路线。本文基于便利原则构建数学模型，使用 Python 编程实现了机器人借还书工作的路线规划，对于任务一和任务二的规划均耗时 4:45:45。为展示简洁，最优结果的工作路线在 txt 附件中进行展示。

目录

1 引言与背景

人工智能为人类社会的智慧未来点亮了无数可能。随着科技的不断发展和人们对智能化生活的追求，人工智能逐渐从科幻片中走向现实生活，并在这个过程中不断地影响着人类社会的方方面面。在交通出行方面，通过人工智能技术的应用，可以实现更加智能、高效的城市交通系统；在环境保护方面，人工智能可以协助自然资源的管理和保护，实现资源的合理分配和可持续利用。此外，人工智能的科技发展，还可以持续促进人类智能水平和劳动生产力的不断提高，为人类社会不断带来新的进步和发展机遇。

大学校园中的图书借还过程存在一些问题影响了效率和便利性。首先，借还书的流程需要学生们来回奔波。通常，学生们需要亲自前往图书馆，租借书籍并在应还日期前再次亲自来馆还书。如果学生们忘记还书日期或者没有时间去还书，那么就会面临额外的罚款和延迟归还的问题，这显然不方便且浪费时间。其次，在大学校园内的图书馆配送方面存在着不少问题。传统的图书馆配送方式需要通过人工方式进行书籍的借还和传递，这过程费时又费力，特别是在大学校园内，图书馆和学生宿舍之间的距离往往较远，这就使得书籍的借还和传送过程更加费力耗时。并且，大学校园内的图书馆配送问题还存在着书籍数量过大、管理不够便利等问题。大学图书馆需要管理大量的书籍，每日都有不少的学生前来借书还书，需要进行统计和管理，但传统的管理方式十分耗时且容易出现错误，需要花费大量精力和时间来对图书进行跟踪和管理，而且有时还可能出现书籍遗漏或信息错误的情况。

基于以上问题，基于人工智能的智慧校园图书配送系统应运而生。校图书馆为了方便师生借阅和及时归还图书，上线了智慧图书配送模块，通过配送机器人进行图书的归还回收和借阅发放，这本质上是通过使用人工智能方法来提升校园图书借还工作的效率和便利性。基于智慧校园图书配送系统，学生可实时获取机器人的位置、路线和时间，更加方便快捷的实现书籍归还和借阅服务。同时，图书配送系统采用了配送机器人，无需学生和工作人员来回奔波，并且通过智能规划来实现快速有效的书籍管理、借阅归还处理等功能，大幅提高了图书馆配送效率。还可以输出工作日志以便查询任何一本书籍的借出和归还时间与地点，节省了图书馆管理人员的工作量和时间，并进一步确保了图书借出与归还的安全性。

为了实现这样的智慧配送系统，本文基于便利性原则建立数学模型，设计了配送机器人的借还书工作算法，能使配送机器人在给定借还书需求和校园地图的情况下尽快地完成借还书工作，并能输出工作日志以供查询。

2 问题重述及方法分析

为了实现大学校园图书的智慧配送问题，需要使用 Python 编程指导配送机器人进行路线规划和配送工作，尽快完成回收还书和发放借书的任务，自充电驻地出发并最终返回，写明配送机器人各自行驶的具体路线和骑行总时间。

其中，任务一只考虑将各宿舍楼的学生还书带回图书馆，任务二不仅需要实现学生还书需求还要实现借书需求。在分别解决两个问题之后，需要比较两个问题的模型，分析不同问题模型的异同和各自优缺点。

本问题为厦门大学 2023 年春季大数据建模与工业应用课程的期末项目问题，考察了学生根据实际问题构建数学模型以实现最优规划的能力。

2.1 配送要求

配送机器人进行归还回收和借阅发放的具体做法如下：

(1) 各个楼宇的师生将需要归还的图书统一放在楼宇的门口，并上报借阅需求（见表 1）。各楼宇的图书存放处只允许存放本楼宇的还书或借书；

(2) 现有配送机器人 A 和 B 分别从充电驻地出发（出发前均未装载书本），随后到达各处楼宇，然后将回收的归还书本带回图书馆，随后再次出发回收还书或发放借书；

(3) 每个配送机器人单独行动，暂不考虑行驶途中因为相遇而互换图书；

(4) 在完成图书馆委托的任务后，配送机器人须返回各自初始出发的驻地进行充电。

每个楼宇的地点位置和实际距离分别如图 1 和表 2 所示。每个配送机器人最多可装载 10 本书。配送机器人 A 的平均行驶速度是 8 km/h，配送机器人 B 的平均行驶速度是 10 km/h 。

2.2 问题分析

本问题有趣且开放，能够充分发挥学生根据数学建模解决实际问题的能力。

首先，观察地图可知，两个配送机器人分别位于校园地图的西北角和东南角，两个充电站没有借还书的需求。同时，重要节点图书馆位于地图的中央，周边有两个红绿灯环岛，分别命名为红绿灯 1 和红绿灯 2，也没有借还书需求。并且，地图上共有 6×6 共 36 个地点，道路规划横平竖直，除了外

围地点，每个位置都与东西南北四个方向的位置相连，只有 20 号地点朱华楼和 21 号地点红绿灯 2 没有道路相连。

图 1: 学校各建筑与交通路线示意图（各道路实际距离见附表 2）

其次，从各楼宇的还书借书需求数量表格中可知，还书需求较大且较为分散，但单个地点的还书需求没有超出配送机器人的最大载荷量。借书需求较少，有借书需求的地点都有还书需求，因此根据便利性原则，可以让配送机器人去某个有借书还书需求的地点前带上该地点的借书，再把该地点的还书归还到图书馆。

并且，考虑到两个机器人的速度差异，较远的路线任务应交由机器人 B 执行，为了保证效率，机器人 A 的总路程应该大致为机器人 B 的 0.8 倍，与二者速度成比例。如果能够在每一轮出发都规划出理想比例的路径，可以保证每一次规划都大致是同步规划，因此最终路程也成比例，耗费时间大致相等，尽可能调用了最大效率。

表 1 各楼宇的还书和借书的需求数量

楼宇名称	还书数	借书数	楼宇名称	还书数	借书数
四美楼	3	0	耸翠楼	4	0
芳邻楼	4	1	朱华楼	2	3
凌云楼	2	0	绿竹楼	5	0
远山楼	5	2	沉璧楼	6	0
重霄楼	6	0	雅望楼	5	2
北辰楼	6	2	朝晖楼	3	0
弥津楼	5	0	流丹楼	8	0
映雪楼	2	0	俊采楼	3	0
高洁楼	4	0	汀兰楼	6	2
南溟楼	3	1	兰亭楼	2	0
长洲楼	6	0	郁青楼	3	0
秋水楼	4	3	星耀楼	4	0
临川楼	3	0	万千楼	2	2
景明楼	6	0	东隅楼	1	0
清风楼	2	0	长风楼	5	0
皓月楼	7	2			

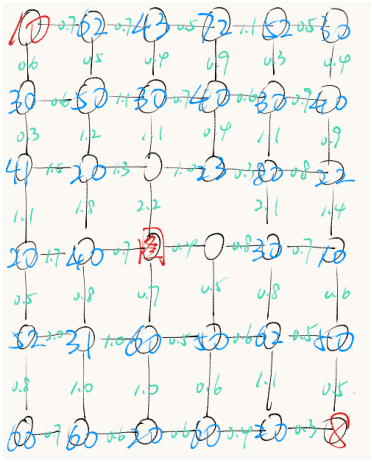

图 2: 综合借书还书需求及路线路程的抽象地图

为了便于进一步分析，作者将原始地图抽象为 6×6 的点图，并根据地图路线情况连接每一个点。作者将各个地点的借书还书需求写于各个地点的圆圈上，用蓝色笔迹表示，每个点有两个数字，左侧的表示还书数，右侧表示借书数。在路径上用青色笔迹表出路径的路程，并用红色笔迹标出图书馆的位置和两个机器人的速度。根据表 1 顺序和地图记录楼宇号数，如充电站 B 为 0 号地点，芳邻楼为 2 号地点，凌云楼为 3 号地点，长风楼为 34 号地点，充电站 A 为 35 号地点。

表 2 各条道路的实际距离

道路名称	实际距离(km)	道路名称	实际距离(km)	道路名称	实际距离(km)
昆仑路	0.6	云南路	1.0	江西路	0.8
峨眉道	0.3	贵州路	0.6	广东路	0.6
陕西路	1.1	恒山道	0.4	香港路	0.4
蜀山道	0.5	五台道	1.1	龙虎道	0.3
青城道	0.8	山西路	2.2	泰山道	1.1
新疆路	0.7	黄山路	0.7	山东路	2.1
甘肃路	0.6	华容道	1.0	武夷道	0.8
青海路	1.5	吉林路	0.5	岭南道	1.1
宁夏路	1.7	河北路	0.7	黑龙江路	0.5
四川路	2.0	河南路	1.0	北京路	0.9
西藏路	0.7	安徽路	0.4	上海路	0.8
敦煌道	0.5	广西路	0.5	浙江路	0.7
骊山道	1.2	澳门路	0.6	福建路	0.5
华山道	1.8	武当道	0.9	台湾路	0.3
衡山道	0.8	嵩山道	0.4	齐云道	0.4
武陵道	1.0	庐山路	0.5	天柱道	0.9
内蒙古路	0.7	中山路	0.6	雁荡道	1.4
湖北路	1.1	辽宁路	1.1	普陀道	0.6
湖南路	1.3	天津路	0.6	终南道	0.5
重庆路	0.7	江苏路	0.3		

2.3 进一步分析

首先明确首轮从充电站出发，中间首尾经过图书馆，最后回站，首轮可以执行还书，返回时可以执行借书。

考虑首轮和返程的工作路径。分析首轮，机器人可以经由从充电站至图书馆的最短路径并沿路把还书收回，也可以找到一条路程较长的路径把距离充电站较近而距离图书馆较远的还书收回，直至达到满载 10 本书。分析返程，返程一般是从图书馆到充电站的最短路径，但也可以是一条路程较长的路径以实现较多借书需求并最终到达充电站。鉴于便利性原则，本文最终选用了最短路径作为首轮和返程的路径。

分析中间轮次，可以用随机过程中的马尔科夫链思想进行建模。所谓马尔科夫链 (Markov Chain) 模型，即由多个状态节点顺序联系的一条链，在链中每一个状态节点有且仅有前一个状态与之相关依赖。即在完成中间轮次的某一轮之后，不妨说第三轮，则第四轮的路线规划及需求分布有多种选择，但均只由第三轮的路线规划及借书后的需求分布以某一概率决定，与第三轮以前的其他轮次状态（第一轮、第二轮）无关。这就要求在实现中间轮次编程过程中，要实时更新地图上图书借还的需求状态更改，并且要考虑路径规划的随机性，若由主观判定路径规划，难以实现效率的最大化。

基于便利性原则进行问题解决的构思。鉴于作者并没有修读过最优规划相关的课程，即便知道一些最优化算法，如鸟群算法、贪心算法、爬山算法和遗传算法等，限于时间约束，也没有想出如何将这些方法应用于本问题。因此，作者从便利性原则角度出发，在中间轮次这一部分思考出解决办法，即先实现距离图书馆最远位置的借还书需求，在最短路径的返回路径中沿途收回还书，保证距离较远的位置能尽早地解决借还书需求。这一想法在现实当中也是符合需要的，较远位置的工作往往消耗更多的效率，配送机器人应优先解决这一痛点问题，至于距离图书馆较近的地区，有借还书需求的师生也可以选择自行前往办理。

2.4 可能存在的问题及解决办法

这一方法可能会出现问题，作者根据思考提出了解决办法。

首先，设计拐弯机制以降低效率浪费。规划出了一条前往最远位置的路径，但是这条路径上除了最远位置基本上就没有还书需求了，这就造成了效率的浪费。因此作者设计了一个拐弯机制，在从最远位置沿着最短路径进行还书工作时，每经过一个位置并完成还书需求后，机器人将检查关于路径上

暂未经过的地点还有没有需要归还的图书，若有则沿原路径继续前进，否则将进行拐弯。所谓拐弯，即在地图上找到一个最近的有还书需求的位置，再沿着返回图书馆的最短路径沿途收回还书，这个位置需要保证距离目前位置比图书馆更近，否则就会变成为了还书而绕了远路，降低总体还书效率。

其次，随机规划路径提高总体效率。如果机器人每次出发都是前往目前距离图书馆最远的位置进行图书配送，可能无法保证可以实现最高的效率。有些位置虽然远，但是在第一轮便前往解决并不一定是整体效率最高的方法。同时，设计机器人 A 的路径规划时，作者选用的是找到一条路程约为机器人 B 路径路程 0.8 倍的路径。这一方法的优点是能够保证两个机器人的总路程与各自速度成比例，可以实现每一轮出发的同步规划，也能够使两个机器人耗费的时间大致相同。但也有缺点，若固定以这一方法进行路径规划，可能也会造成效率的降低。因此作者在路径规划时添加了随机性，即在机器人 B 进行路径规划时，可以随机选择最远的几个位置之一，在机器人 A 进行路径规划时，也会随机选择离目标路程最近的几个位置之一。这样可以让路径规划从固定变得随机，可以经过多次试验得到最优结果。本文对于任务一和任务一都进行了 1000 次试验，以得到最优的结果。

3 编程解决实现过程

3.1 初始准备

首先输入数据并创建距离矩阵，使用 Floyd 算法计算最短路径，得到距离矩阵和最短路径。

距离矩阵：

	0	1	2	3	4	5	6	7	8	9	...	26	27	28	29	\
0	0.0	0.6	0.9	2.0	2.5	3.3	0.7	1.2	2.4	3.7	...	3.2	5.3	5.9	5.6	
1	0.6	0.0	0.3	1.4	1.9	2.7	1.1	0.6	1.8	3.1	...	3.1	5.0	5.3	5.0	
2	0.9	0.3	0.0	1.1	1.6	2.4	1.4	0.9	1.5	2.8	...	3.4	4.7	5.0	4.7	
3	2.0	1.4	1.1	0.0	0.5	1.3	2.5	2.0	2.6	1.7	...	4.5	3.6	3.9	3.6	
4	2.5	1.9	1.6	0.5	0.0	0.8	3.0	2.5	3.1	2.2	...	5.0	4.1	3.9	3.1	
5	3.3	2.7	2.4	1.3	0.8	0.0	3.8	3.3	3.9	2.5	...	5.8	3.8	3.1	2.3	
6	0.7	1.1	1.4	2.5	3.0	3.8	0.0	0.5	1.7	3.5	...	2.5	4.6	5.4	6.1	
7	1.2	0.6	0.9	2.0	2.5	3.3	0.5	0.0	1.2	3.0	...	2.5	4.6	5.2	5.6	
8	2.4	1.8	1.5	2.6	3.1	3.9	1.7	1.2	0.0	1.8	...	2.6	3.7	4.0	4.4	
9	3.7	3.1	2.8	1.7	2.2	2.5	3.5	3.0	1.8	0.0	...	4.0	1.9	2.2	2.6	
10	4.5	3.9	3.6	2.5	2.0	1.7	4.3	3.8	2.6	0.8	...	4.8	2.7	2.1	2.5	
11	4.0	3.4	3.1	2.0	1.5	0.7	4.5	4.0	3.6	1.8	...	5.2	3.1	2.4	1.6	
12	1.4	1.8	2.1	3.2	3.7	4.5	0.7	1.2	2.4	4.2	...	1.8	3.9	4.7	5.4	
13	1.8	1.7	2.0	3.1	3.6	4.4	1.1	1.1	2.3	4.0	...	1.4	3.5	4.3	5.0	
14	2.9	2.8	2.8	3.9	4.4	5.2	2.2	2.2	1.3	2.9	...	1.3	3.4	3.7	4.1	
15	4.4	3.8	3.5	2.4	2.9	3.0	4.2	3.7	2.5	0.7	...	3.3	1.2	1.5	1.9	
16	5.1	4.5	4.2	3.1	3.0	2.3	4.9	4.4	3.2	1.4	...	3.9	1.8	1.1	1.5	
17	4.6	4.0	3.7	2.6	2.1	1.3	5.1	4.6	4.2	2.4	...	4.6	2.5	1.8	1.0	
18	1.9	2.3	2.6	3.7	4.2	5.0	1.2	1.7	2.9	4.7	...	1.6	3.7	4.5	5.2	
19	2.5	2.4	2.7	3.8	4.3	5.1	1.8	1.8	2.7	4.3	...	0.7	2.8	3.6	4.3	
20	2.9	2.8	3.1	4.2	4.7	5.5	2.2	2.2	2.3	3.9	...	0.3	2.4	3.2	3.9	
21	4.8	4.2	3.9	2.8	3.3	3.0	4.6	4.1	2.9	1.1	...	2.9	0.8	1.1	1.5	
22	5.3	4.7	4.4	3.3	3.3	2.5	5.1	4.6	3.4	1.6	...	3.4	1.3	0.6	1.0	
23	5.2	4.6	4.3	3.2	2.7	1.9	5.7	5.2	4.0	2.2	...	4.0	1.9	1.2	0.4	

图 3: 最短路径距离矩阵

路径：

	0
0	0, 0
1	0, 1
2	0, 1, 1, 2
3	0, 1, 1, 2, 2, 3
4	0, 1, 1, 2, 2, 3, 3, 4
...	...
1291	35, 34, 34, 33, 33, 32, 32, 31
1292	35, 34, 34, 33, 33, 32
1293	35, 34, 34, 33
1294	35, 34
1295	35, 35

[1296 rows x 1 columns]

图 4: 最短路径

所谓 Floyd 算法，是求解最短路径矩阵的一种经典方法，能够应用本问题的无向有环图中，求出地图上任意两个地点之间的最短距离以及最短路径。其基本思想就是在这两个地点之间寻找一个结点使得经过该结点能得到更短的距离，通过循环来实现。具体算法如伪代码算法一所示，dist 是一个 n*n 的矩阵，dist[i][j] 代表从 i 到 j 的距离，除了直接连接的道路初始值都设为无穷 inf。n 是图中节点数。该算法的时间复杂度为 $O(n^3)$，空间复杂度也为 $O(n^3)$。

Algorithm 1 Floyd 算法

Require:

距离矩阵 dist;

位置个数 n.

for k in range(n) **do**
  for i in range(n) **do**
    for j in range(n) **do**
      if $dist[i][j] > dist[i][k] + dist[k][j]$ **then**
        $dist[i][j] \leftarrow dist[i][k] + dist[k][j]$
      end if
    end for
  end for
end for

在求解出最短距离矩阵和最短路径之后，可以着手解决任务问题。首先对于首轮工作，即机器人从充电站运行到图书馆，此时配送机器人应该是满电状态，在从充电站到图书馆的最短路径上沿途将需要归还的图书归还到图书馆，只要能够满足负载数量以内。

```
机器人 A 第 1 轮工作日志如下：
本轮工作路线为：['35', '29', '29', '23', '23', '22', '22', '21', '21', '15']
08:00:00: 机器人 A 在 35 号地点 充电站A 拿了 0 本书，目前载荷量为 0 ，准备前往 29 号地点 兰亭楼 。
08:02:15: 机器人 A 在 29 号地点 兰亭楼 拿了 2 本书，目前载荷量为 2 ，准备前往 23 号地点 沉碧楼 。
08:05:15: 机器人 A 在 23 号地点 沉碧楼 拿了 6 本书，目前载荷量为 8 ，准备前往 22 号地点 绿竹楼 。
08:09:45: 机器人 A 在 22 号地点 绿竹楼 拿了 2 本书，目前载荷为 10 已满，剩余 3 本，直接前往 图书馆 。
08:16:30: 机器人 A 到达 15 号地点图书馆，第 1 轮工作结束，整装待发。

机器人 B 第 1 轮工作日志如下：
本轮工作路线为：['0', '1', '1', '2', '2', '3', '3', '9', '9', '15']
08:00:00: 机器人 B 在 0 号地点 充电站B 拿了 0 本书，目前载荷量为 0 ，准备前往 1 号地点 四美楼 。
08:03:36: 机器人 B 在 1 号地点 四美楼 拿了 3 本书，目前载荷量为 3 ，准备前往 2 号地点 芳邻楼 。
08:05:24: 机器人 B 在 2 号地点 芳邻楼 拿了 4 本书，目前载荷量为 7 ，准备前往 3 号地点 凌云楼 。
08:12:00: 机器人 B 在 3 号地点 凌云楼 拿了 2 本书，目前载荷量为 9 ，准备前往 9 号地点 高洁楼 。
08:22:12: 机器人 B 在 9 号地点 高洁楼 拿了 1 本书，目前载荷为 10 已满，剩余 3 本，直接前往 图书馆 。
08:26:24: 机器人 B 到达 15 号地点图书馆，第 1 轮工作结束，整装待发。
```

图 5: 首轮工作日志：从充电站前往图书馆

第一轮和中间轮次使用的是相同的算法但是也有区别。首先，区别在于起点不同，第一轮的起点是充电站，因此只要随着最短路径一路沿途回收还书一路前往图书馆即可。而中间轮次的起点是图书馆，这需要配送机器人沿途将借书交到各楼栋，同时前往规划路线中的目标地点。其次，区别在于路线不同。第一轮的路线是确定的，即从充电站到达图书馆的最短路径。而中间轮次是使用自编函数进行确定的，规划机器人 B 前往的是某一个最远的目的地，而机器人 A 前往的是路程大致为 B 路程 0.8 倍的某个目的地。在路线规划之后，需要判定两条道路是否存在重合，如果存在重合就可能会造成效率的降低，因此会进行一次重新规划来作为本轮的规划路线。为什么只判断一次呢？因为有时可能路线重合也有可能是最终的最优解，而一次判定已经足够避开大部分低效率的重合情况了。关于中间轮次的算法，由于其间使用了很多自编函数，为展示简洁，使用中文式代码进行展示。

3.2 核心算法

```
def robot_return_books(robot_name, count, path, time_count, lend):
#robot_name 机器人型号
#count 机器人运行轮次计数
#path 规划好的路线
#time_count 机器人运行计时
#lend 是否执行借书操作

if (从图书馆出发):
  if (执行借书):
        检查路径上是否需要借书，共需借多少本
        if (这条路径需要借书):
          for i in range(路径上的节点):
                if (这个节点不需要借书):
                  就到下一个节点
                else:
                  把书借出，清除需求

  #完成路径上借书工作之后开始还书
  直接去路径上最远的点

else: #如果不是从图书馆出发但也遇到了借书需求(比如中途绕行了)
        if (执行借书):
                检查路径上是否需要借书，共需借多少本
                if (这条路径需要借书):
                        for i in range(路径上的节点):
                                if (这个节点不需要借书):
                                        就到下一个节点
                                else:
                                        if (目前载荷+借书量 <= 最大载荷):
                                                把书借出，清除需求
                                        else:
                                                能借多少是多少
```

```
for i in range(路径上的节点):
        得到目前节点的信息
        if (路径上已经没书要还了):
                找到最近的绕行节点
                if (绕行节点就是图书馆):
                        直接回图书馆
                else:
                        执行绕行，调用自身
                        return time_count, count

        if (机器人拿得下这个地点要还的所有书):
                把这个地点要还的书都拿上
                if (刚好拿满了) or (下一站就是图书馆):
                        直接回图书馆
                        return time_count, count

                #若拿完还没拿满而且还没到图书馆
                if (路径上已经没书要还了):
                        找到最近的绕行节点
                        if (绕行节点就是图书馆):
                                直接回图书馆
                        else:
                                执行绕行，调用自身
                                return time_count, count
                else: #路径上还有书要还
                        去下一个节点执行还书

        else: #机器人拿不下这个节点要还的所有书
                能拿多少拿多少
                直接回图书馆

                return time_count, count
```

3.3 全程算法

根据这一算法，就可以执行中间轮次中某一个轮次的借还书操作。要实现全程的工作，要需要再编制出一个算法来实现。

```
def return_work(count_A, path_A, time_count_A,
                count_B, path_B, time_count_B, lend):
#count_A  机器人A目前已执行的轮次
#path_A  机器人A本轮的规划路线
#time_count_A  机器人A目前用时
#count_B  机器人B目前已执行的轮次
#path_B  机器人B本轮的规划路线
#time_count_B  机器人B目前用时
#lend  是否执行借书操作
i = 0
while i < 100:  #这个算法是对中间轮次和最终返回的结合
        if (还有图书没有归还):
          对机器人A执行函数 robot_return_books
        else:
          加上第三轮的时间，取最长耗时作为最终时间
          if (执行借出操作):
            检查是否还有图书没有借出，如果有就派用时最短的机器人去借
          break

        if (还有图书没有归还):
          对机器人B执行函数 robot_return_books
        else:
          加上第三轮的时间，取最长耗时作为最终时间
          if (执行借出操作):
            检查是否还有图书没有借出，如果有就派用时最短的机器人去借
        break

        if (还有图书没有归还):
          根据目前的归还情况规划路线
return final_time_count
```

对于任务一和二，调用 return_work，即可实现全程算法。任务一和任务二各对应一个函数，两个函数本质相同，唯一差别是后者中间轮次和返回轮次 lend 的取值不同。由于任务二中，机器人 B 在返回轮次中将要途径芳邻楼，可以将芳邻楼的借书需求安排在此时完成，因此在第二轮开始之前先行将芳林楼的借书需求消除。

```
def thorough_return()    #对于任务一:
def thorough_both()      #对于任务二:
        初始化数据: 借还书需求数量, 轮次, 计时
        完成第一轮: 从充电站前往图书馆, 沿途还书
        #若为任务二, 把芳邻楼的借书需求消除
        完成中间轮次和最终返回
return final_time_count
```

由此，执行 thorough_return() 便能完成一次任务一，在执行过程中会输出某一时间下某个机器人的运行情况，即为工作日志，以及最终使用的时间，实现任务要求。同理，执行 thorough_both() 便能完成一次任务二，也有相同的输出实现任务要求。

4 随机搜索与模型比较

在路径规划中我们增添了随机性：根据目前剩下的还没有归还图书的位置个数，取 0.1 倍的个数作为随机窗宽 window，并在 [0, 2*window] 中取一个随机数作为机器人 B 应前往的目标节点，即若 window 为 2，随机数取得 3，则本轮次机器人 B 将前往距离图书馆第 4 远的目标节点，并规划前往的路线。同理，机器人 A 的路径规划中，将在 [0, window] 中取一个随机数作为机器人 A 应前往的目标节点，即若随机数取得 0，则本轮次机器人 A 将前往距离 0.8 倍机器人 B 路程最近的目标节点，即在目前存在还书需求的节点中，找到一个节点使得从图书馆出发前往的路程最接近于已规划的机器人 B 的路线路程，并规划前往的路线。

基于这样的算法思路，由于随机性，每一次执行任务一和任务二的全程算法，结果都有很大可能会不一样。为了找到用时最短的配送方案，本文对全程算法分别执行一千次，并找到最好的结果。搜索结果显示，任务一和任务二的最短用时相等，为 4:45:45，但是工作日志显示两个任务的工作历程，

除了第一轮出发和最终返回，中间轮次并不相同，可以参见附件查看二者的完整工作日志。

机器人 A 第 2 轮工作日志如下：

本轮工作路线为：['15', '14', '14', '13', '13', '12']

此路径上第 4 个地点，即 12 号地点 秋水楼，借书需求为 3 本。

此路径为从 15 号地点 图书馆 到 12 号地点 秋水楼 的最短路径，共经过 3 个地点。

本次借书工作一共将借出 3 本书。

机器人 A 从图书馆出发开始借书工作：

08:44:15: 机器人 A 途径 12 号地点 秋水楼 为图书馆借出图书 3 本。

机器人 A 从图书馆出发，前往 12 号地点 秋水楼 ，08:44:15到达并开始还书工作：

08:44:15: 机器人 A 在 12 号地点 秋水楼 拿了 4 本书，目前载荷量为 4 ，准备前往 13 号地点 临川楼 。

08:47:15: 机器人 A 在 13 号地点 临川楼 拿了 3 本书，目前载荷量为 7 ，准备前往 14 号地点 红绿灯1 。

路径上剩余位置都没有要还的书

决定转移路径到 19 号地点 耸翠楼

机器人 A 第 3 轮工作日志如下：

本轮工作路线为：['19', '20', '20', '14', '14', '15']

此路径上第 2 个地点，即 20 号地点 朱华楼，借书需求为 3 本。

此路径为从 19 号地点 耸翠楼 到 15 号地点 图书馆 的最短路径，共经过 3 个地点。

本次借书工作一共将借出 3 本书。

机器人 A 执行借书工作。

09:11:15: 机器人 A 途径 20 号地点 朱华楼 为图书馆借出图书 3 本。

08:47:15: 机器人 A 在 19 号地点 耸翠楼 拿了 3 本书，目前载荷为 10 已满，剩余 1 本，直接前往 图书馆 。

09:14:15: 机器人 A 到达 15 号地点图书馆，第 3 轮工作结束，整装待发。

图 6: 任务二最优结果工作日志：机器人第一次从图书馆出发执行任务

任务一的最优路线为：

机器人 A['35', '29', '29', '23', '23', '22', '22', '21', '21', '15'] 到达图书馆——前往 7 号地点 ['7', '8', '8', '9', '9', '15'] 到达图书馆——前往 12 号地点 ['12', '13', '13', '14', '14', '15'] 在 13 号时绕路 19 号地点 ['19', '20', '20', '14', '14', '15'] 到达图书馆——前往 26 号地点 ['26', '27', '27', '21', '21', '15'] 到达图书馆——前往 4 号地点 ['4', '3', '3', '9', '9', '15'] 在 4 号地点绕路 5 号地点 ['5', '11', '11', '17', '17', '16', '16', '15'] 到达图书馆——前往 17 号地点 ['17', '16', '16', '15'] 到达图书馆——前往 28 号地点 ['28', '22', '22', '21', '21', '15'] 在 28 号地点绕路 34 号地点 ['34', '28', '28', '22', '22', '21', '21', '15'] 在 34 号地点绕路 27 号地点 ['27', '21', '21', '15'] 到达图书馆——返回

充电站 A。

机器人 B['0', '1', '1', '2', '2', '3', '3', '9', '9', '15'] 到达图书馆——前往 30 号地点 ['30', '31', '31', '32', '32', '33', '33', '27', '27', '21', '21', '15'] 到达图书馆——前往 18 号地点 ['18', '12', '12', '13', '13', '14', '14', '15'] 在 18 号地点绕路 19 号地点 ['19', '20', '20', '14', '14', '15'] 到达图书馆——前往 24 号地点 ['24', '25', '25', '19', '19', '20', '20', '14', '14', '15'] 在 25 号地点转移到 6 号地点 ['6', '7', '7', '8', '8', '9', '9', '15'] 到达图书馆——前往 6 号地点 ['6', '7', '7', '8', '8', '9', '9', '15'] 在 6 号地点转移到 5 号地点 ['5', '11', '11', '17', '17', '16', '16', '15'] 到达图书馆——前往 11 号地点 ['11', '17', '17', '16', '16', '15'] 在 11 号地点绕路 10 号地点 ['10', '9', '9', '15'] 在 10 号地点绕路 22 号地点 ['22', '21', '21', '15'] 在 22 号地点绕路 28 号地点 ['28', '22', '22', '21', '21', '15'] 到达图书馆——返回充电站 B。

任务二的最优路线同理，详情可参照附件的工作日志，会更易于理解。

比较两个任务使用的模型，实际上相似性非常的大，但也有少数不同。相同之处在于，二者都根据便利性原则，在主要的优化过程——中间轮次中优先选择较远的目的地进行路线规划，并且在返回的过程中沿途回收还书，同时也可以实现绕路操作以实现效率的提升。二者的不同之处在于，任务二的算法需要实现借书操作，由于本题的借书需求较少，所以就还是根据还书需求进行规划，在前往目标节点的过程中实现借书需求，并且拐弯绕路也可以根据载荷情况实现借书需求。

5 结论与反思

最终结论为，任务一和任务二的最优用时均为 4:45:45，二者工作路线并不相同，由于工作路线太长不便展示，工作路线以工作日志的形式在附件中进行展示。

对于本文解决方法的反思，由于便利性原则，本文可以尽快地解决问题，即建模和编程工作总体需要的实现时间较短，图书馆智能配送系统的开发人员可以采用类似本文的方法在较短的时间内解决智能配送的路径规划问题。但本文方法的缺点也比较明显，即外推性较差，同时依赖于本问题的数据特点，即还书数多、借书数少、基本上所有楼宇都有还书需求且有借书需求的楼宇刚好都有还书需求，因此对于新的数据并不能直接适应求解，还需要根据数据特点对算法进行修改以提升算法效果。