队伍编号	MC2201805
题号	（B）

基于改进遗传算法的无人仓多 AGV 防碰撞的路径规划研究

摘 要

针对无人仓的多搬运机器人的路径规划问题，本文综合运用了**多目标规划、遗传算法和交通管制法**等方法，充分发挥了**线性加权和 banach 空间中的向量范数**等思想的优势，根据题意构建具体的线性规划模型，并借助 python 软件编程求解。

首先我们对给出的数据进行数据预处理：1）根据 map.csv 中的仓库地图数据，基于每一个节点的坐标画出无向图，基于节点关系构建出**邻接矩阵**；2）根据邻接矩阵，运用 **Floyd 算法**求解每个点到其他点的最短距离，得到**距离矩阵**和**路由矩阵**；3）通过 Python 软件以栅格的形式进行可视化呈现。通过可视化的栅格仓库图，删除不符合实际的数据，因此，在该无人仓中**实际可用的小车数量为 19 辆**。

针对问题一，本题要求满足以下约束：**1）满足所有的订单需求，即实现全部拣选工位都空闲；2）保证在搬运过程中每个机器人尽可能忙；3）每个拣选工位同时容纳的托盘数不超过 3 个停靠位**，最终实现最小化全部搬运机器人的行走路径的目标函数。因此，根据题意，我们构建**单目标规划模型**，基于托盘搬运顺序进行编码，并采用**改进的遗传算法**和**模拟退火**两种方法进行对比求解。通过改进的遗传算法求解后可以得到，所有 AGV 的路径总长度为 **7316**；通过模拟退火算法求解后可以得到所有 AGV 的路径总长度为 **7428**。通过两种方式的比较，可以得到改进遗传算法使得 AGV 以更短的路径搬运完所需要的商品。

针对问题二，本题需要满足以下约束：**1）每个拣选工位对应托盘的商品总量尽可能地平均；2）最小化全部托盘到其默认拣选工位距离总和；3）对仓库内储位上的每个托盘，都指定一个由第一阶段的目标规划唯一确定的默认拣选工位，**最终实现全部搬运机器人的行为总路径最小的目标函数。本题在问题一的基础上，需要对仓库进行合理分区。因此，本题建立了**两阶段的目标规划模型**，其中约束条件 1）和 2）属于第一阶段**动态分区规划模型**，目的是为了实现仓库的合理分区，约束条件 3）则是在第一阶段的基础上新增加的约束条件，属于第二阶段的**路径规划模型**。本题设计一种具有**新型编码方式的遗传算法**来实现储位的分区，通过求解后可以得到在分区情况下所有 AGV 的移动**总路径长度为 7453**。

针对问题三，本题在满足问题一和问题二的基础上，需要进一步考虑搬运机器人的**碰撞和拥堵**问题。首先本文确定了路径冲突的三种类型：**垂直冲突、相向冲突和追尾冲突**，以及通过**向量内积**进行路径冲突类型的判断。其次，我们建立了**交通管制法**和**优先级规划法模型**，并在 **AGV 为 12-19 辆的不同情况**下均进行 100 次随机实验，最终得到最优的 AGV 数量规划，防止多 AGV 在仓库内的碰撞和拥堵问题。通过模型的求解可以得到在**交通管制法下**所有 AGV 的总路径最短，且**当 AGV 数量减少为 15** 时，减少碰撞和拥堵情况是最佳的。

关键词：多 AGV 路径规划，改进遗传算法，目标规划，交通管制法，优先级规划法

目录

一、 问题重述与分析

1.1 问题背景

随着电商的兴起，无人仓逐渐作为自动化仓储物流系统的发展方向和目标。仓库管理(也叫仓储管理)是对仓库内货物的接收、存储、发货等一系列活动进行有效的控制管理，维护仓库货物并确保日常经营活动正常进行。传统的仓库管理模式大多是人工管理模式，而这种管理方式有着不少问题，特别是因为人力效率低下的因素导致的“爆仓”现象。最早的包括 Amazon 的 Kiva 机器人，以及后来 AGV 搬运机器人、SHUTTLE 货架穿梭车、DELTA 分拣机器人等各式各样的、高度自动化的机器人都是为仓库的无人化量身定制的。而无人仓内搬运机器人的调度问题是其中的核心问题。

在对问题进行分析之前，首先要了解一下概念。（1）搬运机器人，Automated Guided Vehicle (AGV)，通过特殊地标导航自动将物品运输至指定地点。它是执行仓库内任务的主要单元，可以通过指令指派一个 AGV 去取一个货架到仓库内指定地点，比如拣选工位，储位，回收处等；（2）托盘和储位。仓库内绝大部分区域都是放置托盘的储位，而托盘上摆放着待出库商品。有的仓库里托盘用多层的货架代替，无论货架还是托盘都可以被 AGV 搬运；（3）拣选工位。由分拣机器人拣选商品，完成打包后通过传送带输送出库，包括了多个放置托盘的停靠位，分拣机器人以及商品出库的传送带；（4）托盘回收处。仓库内用来回收空托盘的固定区域，一般仓库内会指定 2-4 个固定位置为托盘回收处。

1.2 研究现状

随着计算机集成制造技术的快速发展，传统的制造业发生了重大而深刻的变革，以工业机器人的核心的智能制造技术开始应用于企业生产。国外在 20 世纪 50 年代开始，一种柔性化的智能搬运机器人 AVG 就已经在制造业、港口、码头等领域得到普遍应用[1]。亚马逊最先使用了基于 Kiva 的多机器人仓储系统，大幅加快了简练工作的速度[2]。然而在我国搬运机器人的发展较为迟缓，直到 1988 年，北京邮政科学技术研究所研发了第一套邮政枢纽搬运机器人系统，我国的智能搬运机器人才起步 [3] 。国内搬运机器人的运用主要有农业生产方面、航空领域，自动化领域，以及物流领域等。在农业生产方面，多数搬运机器人的研究重点在于机器人本身所具有的功能的设置，如机器人的动力装置[4]，机器人的硬件结构等[5]。在航空领域，学者提出了针对棒料搬运和无人机翼下弹药挂装，设计了一种基于双自由度结构的双臂搬运机器人，依旧侧重在硬件结构的设置方面[6]。在自动化领域，为了保证机器人系统运行稳定，在硬件结构动力电芯上，实现了将动力电芯从定制料框中抓取上线到输送线定位托盘中的要求，设计了一种三自由度的直角坐标型搬运机器人[7]。

在仓储搬运机器人的研究中，AGV 的路线规划已经成为重要的方向之一[8]。在以往的仓储实际应用上，单 AGV 已经不能满足现代流通和仓库商品出库的要求，因此，多 AGV 仓库路径优化成为当下的热点[9]。在多 AGV 路径规划中，以往研究常用到 A*算法、神经网络、强化学习以及启发式算法等。其中，A*算法通常运用在单 AGV 情况下[10]，改进后的 A*算法对多 AGV 环境下也能很好的使用，对可能出现的碰撞情况有很好的处理[11]。在多 AGV 情况下的路径规划问题中，国内外更多采用的智能算法进行求解。通过改进的蚁群算法的路径规划很好的满足了多 AGV 的情况，且能够有效的避开障碍物[12]；在作业车间调度问题上，使用混合选择策略、自适应伪随机比例规则和改进信息素更新规则优化的蚁群算法，有效的完成搬运机器人的分配和搬运工序的排序[13]。除此以外，更为常见的是基于遗传算法的启发式算法的应用[14],通过改变遗传算法中的交叉变异规则，以满足不同的环境需求，最终实现多 AGV 路径优化的最佳调度[15][16][17]。

1.3 问题重述

本题属于无人仓管理中搬运机器人 AGV 的调度问题。一般的，无人仓可以简化为

一个 $n \times m$ 的矩阵图，如图 1-1。

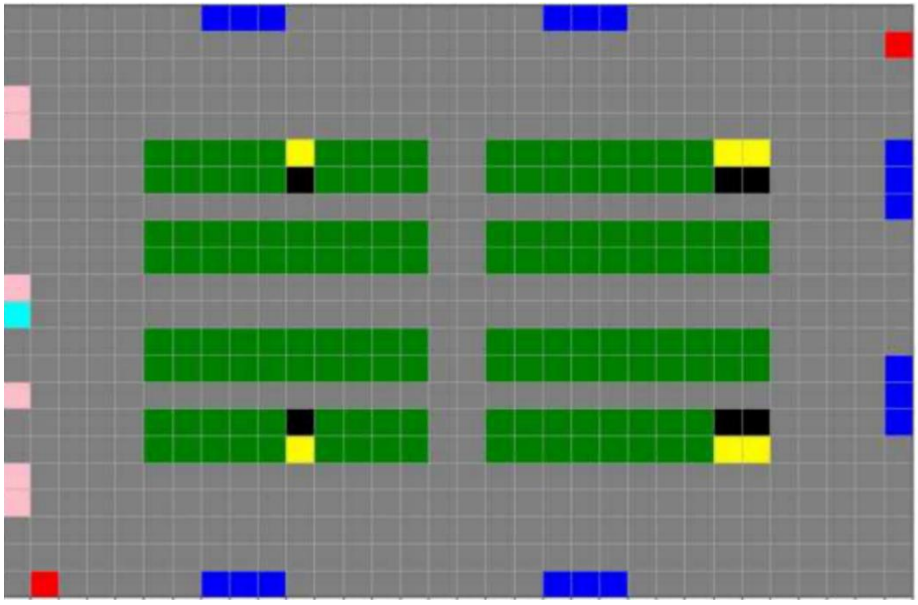


图 1-1 仓库简化图

在这个简化图中，共有 7 种不同类型的方框，表示仓库中不同作用的设施。其中，灰色的方框为路径节点，在该部分中，AGV 可以自由通行；绿色部分为储位节点，该部分为放置托盘的区域，也是 AGV 去取货的地方；蓝色部分为拣选工作节点，AGV 将托盘搬运至此处后打包出库，一般一个拣选工作节点可以放置多个托盘；粉色部分为补货位节点，一般为如果拣选工作节点处商品不足，则通过传输带将补货位的获取传送到拣选工作节点；红色部分为空托盘回收节点，AGV 将空托盘放置此处。

一般的，我们对仓库和 AGV 做了以下限制：AGV 在仓库中一次只能搬运一个托盘，抵达拣选工作节点之后可以直接执行下一个任务，且 AGV 的运动轨迹只能是垂直方向或者水平方向，不能斜着移动；一般的，一个订单内只包含一种商品以及相应的数量。同时，AGV 的任务只包括 3 类，出库（AGV 搬运载有商品的托盘到空闲拣选工位）、回库（AGV 搬运拣选完成的托盘从工位回到仓库内空储位）、和回收（AGV 搬运拣选完成的空托盘从工位到托盘回收处）。

基于以上背景，有以下三个问题。

1.3.1 问题一的解析

问题一在不考虑 AGV 机器人存在可能碰撞的问题，设计多 AGV 的调度算法，需要满足以下目标：（1）满足所有订单；（2）让每个机器人尽量忙；（3）全部搬运机器人的行走总路径最小；（4）最忙拣选工位的工作时长最短；（5）在最短的时间内满足订单需求。通过本题的求解，我们可以得到一个合理的多 AGC 的路径规划。具体如图 1-2，红色圆圈表示 AGV,绿色方块表示货架或者托盘，蓝色 X 表示拣选工位，左图为初始各点的分布状态，右图为匹配了最近的 AGV、托盘和工位，使得指定 AGV 去取托盘后再送到工位拣选。

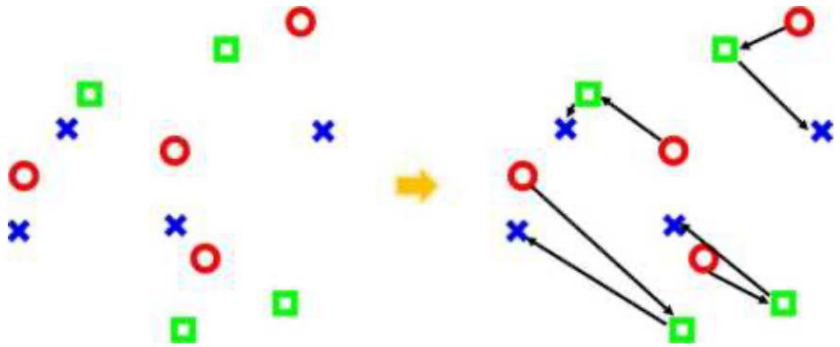


图 1-2 AGV 匹配调度示例图

1.3.2 问题二的解析

为了更好地平衡拣选工位的负载，同时预防搬运机器人的局部拥堵，根据拣选工位和库存商品数量对仓库地图进行动态分区。也就是对仓库内储位上的每个托盘，都指定一个默认拣选工位。本题要在问题一的基础上，对 AGV 调度算法进行优化，使得每个拣选工位对应托盘的商品总量尽可能平均，同时要求最小化全部托盘到其默认拣选工位距离总和，更加合理地均衡每个拣选工位在某段时间内的工作量。如图 1-3，对储位区域进行分区，为其分配相对应的默认拣选工位。

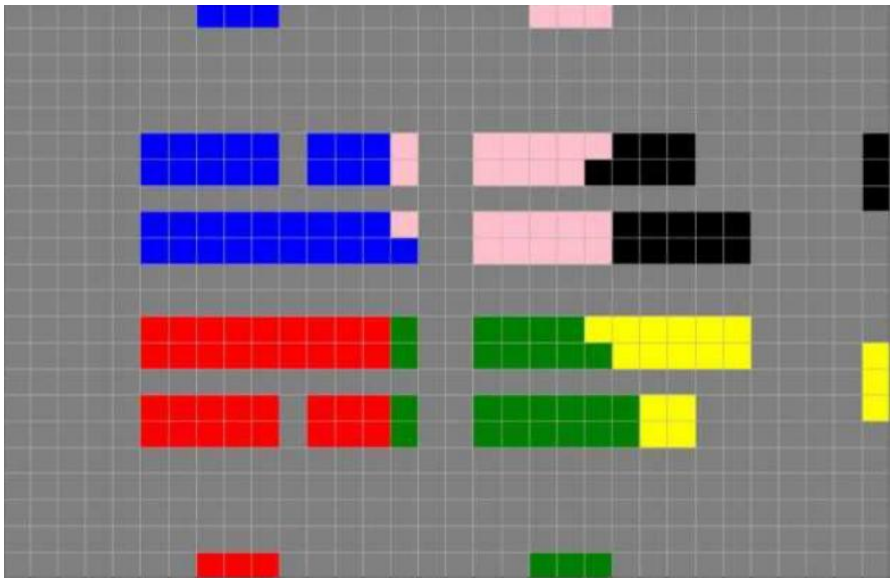


图 1-3 默认拣选工位分配示例图

1.3.3 问题三的解析

本题考虑了 AGV 的碰撞和拥堵问题，当仓库内同时有多个 AGV 在执行任务时，不可避免有些 AGV 在某个路径节点上相遇，甚至在一些特殊节点处（如托盘回收处），可能还会出现多个 AGV 的拥堵甚至死锁场景。在这样的情况旨在，势必需要有 AGV 进行避让，因此本题在问题一和问题二的基础上，考虑可能出现的碰撞情形，在合理的假设下，优化算法，使其具有一定的防碰撞能力。

1.4 研究思路与框架图

本文以问题为驱动，建立一种基于改进的遗传算法的多 AGV 调度模型，具体方法和结构如图 1-4。

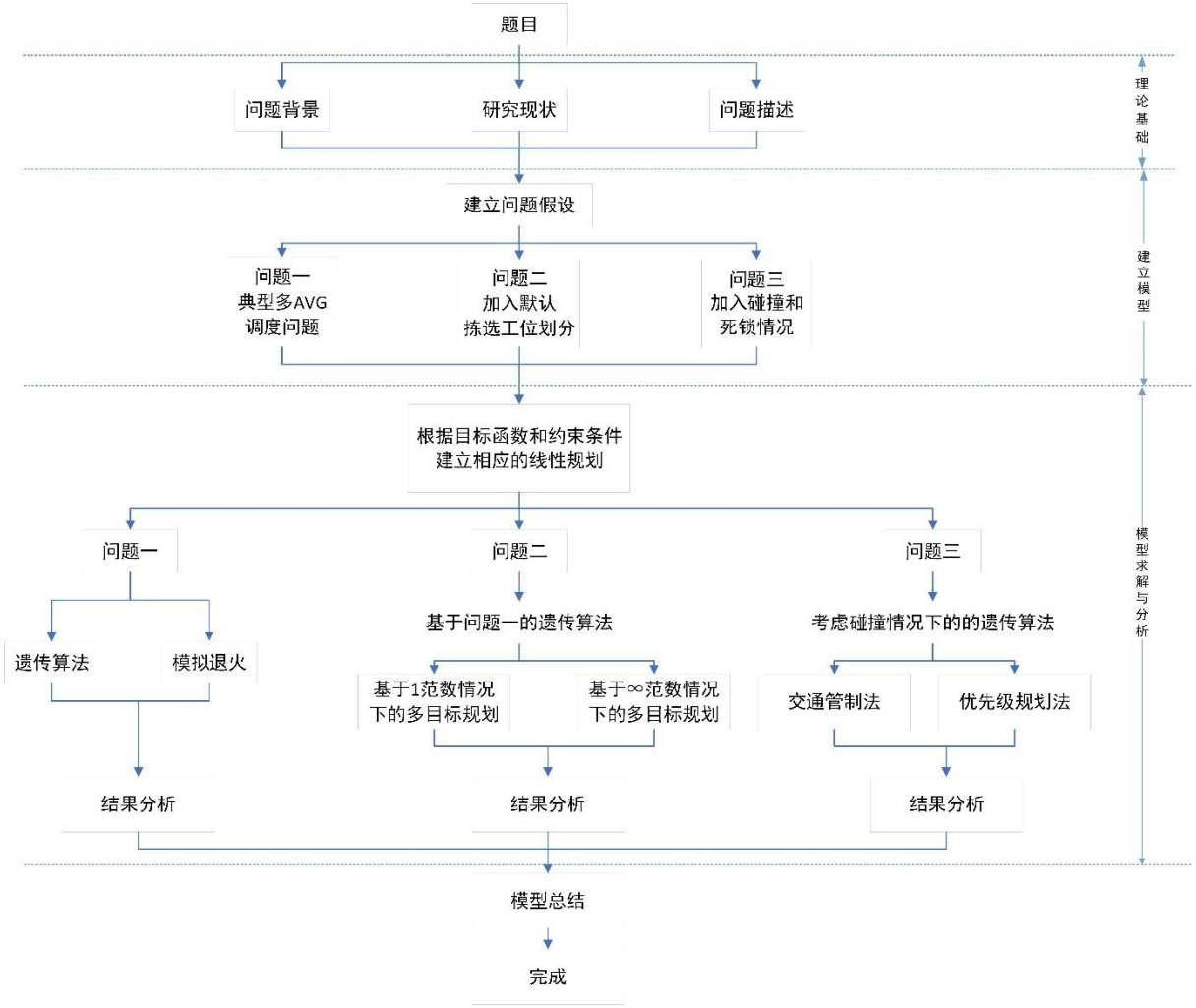

题目
问题背景
研究现状
问题描述
理论基础
建立问题假设
问题一
典型多AVG
调度问题
问题二
加入默认
拣选工位划分
问题三
加入碰撞和
死锁情况
建立模型
根据目标函数和约束条件
建立相应的线性规划
问题一
问题二
问题三
遗传算法
模拟退火
基于问题一的遗传算法
考虑碰撞情况下的的遗传算法
基于1范数情况
下的多目标规划
基于∞范数情况
下的多目标规划
交通管制法
优先级规划法
结果分析
结果分析
结果分析
模型求解与分析
模型总结
完成

图 1-4 本文流程图

二、 模型准备

2.1 假设说明

1. 搬运机器人 AGV 完成出库任务后不在停靠位等候，立刻进行回库或者回收任务（且下一个托盘到达拣选工位节点时，前一个托盘必定已完成拣选工作），若拣选工位无托盘，则直接进行下一次出库任务。
2. 满足所有的订单需求时，托盘可置于拣选工位，搬运机器人 AGV 不需要进行回库或者回收任务。
3. 保留节点，视为柱子节点，即障碍物，搬运机器人 AGV 不能到达。
4. 若订单中的货物量超过储位的总量，由补货位节点从高密度区补货的商品放置点通过传送带进行输送，此时不增加搬运机器人 AGV 的工作负担。
5. 出库托盘在拣选工位需要停留一段时间后，等拣选机器人打包发货后才能进行后续的回库或者回收任务。
6. 搬运机器人 AGV 在工作时进行匀速运动，且各相邻栅格间的距离相等，搬运机器人 AGV 通过一个栅格均需要一个单位时间。
7. 假设搬运机器人 AGV 不再进行新任务，但全部拣选工位未都空闲为止时，将搬运机器人 AGV 从拣选工位节点人工移出。

2.2 数据预处理

2.3.1 邻接矩阵构建

根据仓库地图数据，我们基于节点位置的 X 轴坐标与 Y 轴坐标构建平面坐标系，并以节点为顶点 V，E 为边，可构建无向图 G=(V,E)，并如图 2-1 所示。

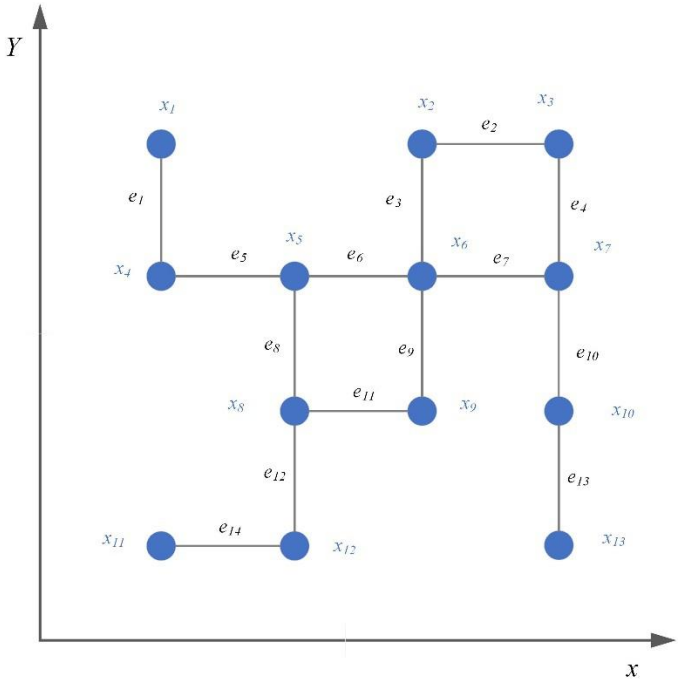


图 2-1 无向图示意

因此，可用一个一维数组存放图中所有顶点数据；用一个二维数组存放顶点间关系（边或弧）的数据，这个二维数组称为邻接矩阵。邻接矩阵又分为有向图邻接矩阵和无向图邻接矩阵，在本文中，我们所使用的邻接矩阵为无向图邻接矩阵，其中无向图的邻接矩阵是对称的。我们按横坐标、纵坐标的次序，对所有节点进行编号，有 0-703 依次的节点编号，根据这些节点间的邻接关系，有如图 2-2 所示的邻接矩阵。

	0	1	2	3	4	5	6	...	697	698	699	700	701	702	703
0	0.0	1.0	inf	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	inf	inf
1	1.0	0.0	1.0	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	inf	inf
2	inf	1.0	0.0	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	inf	inf
3	inf	inf	inf	0.0	inf	inf	inf	...	inf	inf	inf	inf	inf	inf	inf
4	inf	inf	inf	inf	0.0	inf	inf	...	inf	inf	inf	inf	inf	inf	inf
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
699	inf	inf	inf	inf	inf	inf	inf	...	inf	inf	0.0	inf	inf	inf	inf
700	inf	inf	inf	inf	inf	inf	inf	...	inf	inf	inf	0.0	inf	inf	inf
701	inf	inf	inf	inf	inf	inf	inf	...	inf	inf	inf	inf	0.0	inf	inf
702	inf	inf	inf	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	0.0	inf
703	inf	inf	inf	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	inf	0.0

图 2-2 邻接矩阵

2.3.2 距离矩阵与路由矩阵

根据 2.3.1 所构建的邻接矩阵，我们利用 Floyd 算法来求解邻接图中每对顶点间的最短距离。其中 Floyd 算法的时间复杂度为 O（n^3），为多项式时间复杂度算法，在本文中适用。经求解有如图 2-3 和图 2-4 所示的距离矩阵和路由矩阵。

	0	1	2	3	4	5	...	698	699	700	701	702	703
0	0.0	1.0	2.0	5.0	6.0	7.0	...	19.0	20.0	21.0	22.0	23.0	inf
1	1.0	0.0	1.0	4.0	5.0	6.0	...	20.0	21.0	22.0	23.0	24.0	inf
2	2.0	1.0	0.0	3.0	4.0	5.0	...	21.0	22.0	23.0	24.0	25.0	inf
3	5.0	4.0	3.0	0.0	3.0	4.0	...	22.0	23.0	24.0	25.0	26.0	inf
4	6.0	5.0	4.0	3.0	0.0	3.0	...	23.0	24.0	25.0	26.0	27.0	inf
..	...	...	...	...	...	...	...	...	...	...	...	...	...
699	20.0	21.0	22.0	23.0	24.0	25.0	...	3.0	0.0	3.0	4.0	5.0	inf
700	21.0	22.0	23.0	24.0	25.0	26.0	...	4.0	3.0	0.0	3.0	4.0	inf
701	22.0	23.0	24.0	25.0	26.0	27.0	...	5.0	4.0	3.0	0.0	3.0	inf
702	23.0	24.0	25.0	26.0	27.0	28.0	...	6.0	5.0	4.0	3.0	0.0	inf
703	inf	inf	inf	inf	inf	inf	...	inf	inf	inf	inf	inf	0.0

图 2-3 距离矩阵

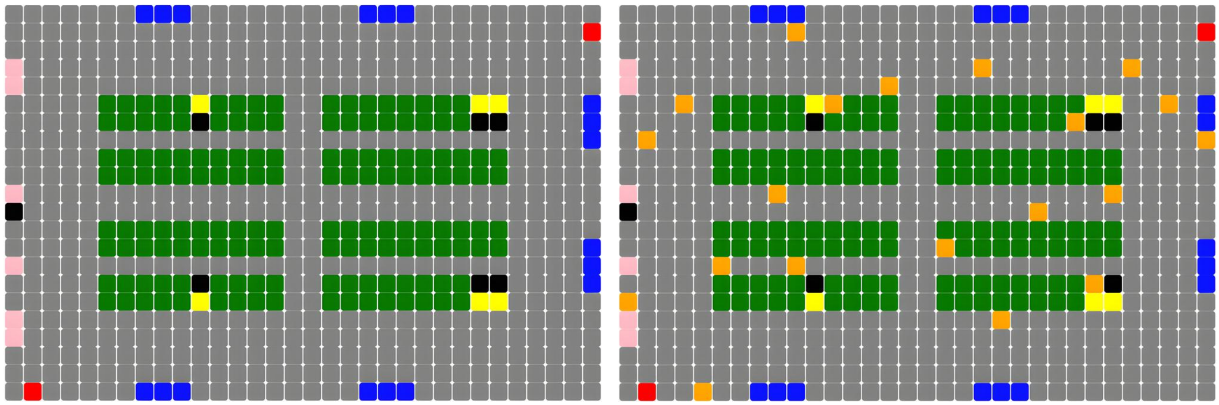
	0	1	2	3	4	5	6	...	697	698	699	700	701	702	703
0	0	1	1	1	1	1	1	...	29	29	29	29	29	29	703
1	0	0	2	2	2	2	2	...	0	0	0	0	0	0	703
2	1	1	0	31	31	31	31	...	1	1	1	1	1	1	703
3	32	32	32	0	32	32	32	...	32	32	32	32	32	32	703
4	33	33	33	33	0	33	33	...	33	33	33	33	33	33	703
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
699	677	677	677	677	677	677	677	...	677	677	0	677	677	677	703
700	678	678	678	678	678	678	678	...	678	678	678	0	678	678	703
701	679	679	679	679	679	679	679	...	679	679	679	679	0	679	703
702	680	680	680	680	680	680	680	...	680	680	680	680	680	0	703
703	0	1	2	3	4	5	6	...	697	698	699	700	701	702	0

图 2-4 路由矩阵

2.3.2 仓库地图数据可视化

为了更加直观的呈现仓库地图的示意图，通过 Python 软件以栅格的形式进行可视

化呈现。在图 2-5 中用灰色表示路径节点，绿色表示储位节点，黄色表示保留节点，黑色表示柱子节点，蓝色表示拣选工位节点，粉色表示补货位节点，红色表示空托盘回收节点，橙色表示搬运机器人 AGV 的初始位置，其中对比图 2-5 中的（a 无搬运机器人 AGV）和图 2-5（b 有搬运机器人 AGV），可判定有一个搬运机器人 AGV 处于柱子节点，即该搬运机器人 AGV 的坐标数据有误，故将其清除，本题**只考虑 19 个搬运机器人 AGV**。



a无搬运机器人AGV　　b有搬运机器人AGV

图 2-5 仓库栅格地图

另外，路径的通行方式对 AGV 的运行效率有非常大的影响，主要包括栅格的通行方向和栅格的连通性两部分。同时规定当两个相邻栅格具有相同的允许通行方向的时候，定义这两个栅格在这个允许通行方向上具有连通关系，如图 2-6 所示为各栅格间的连通关系。

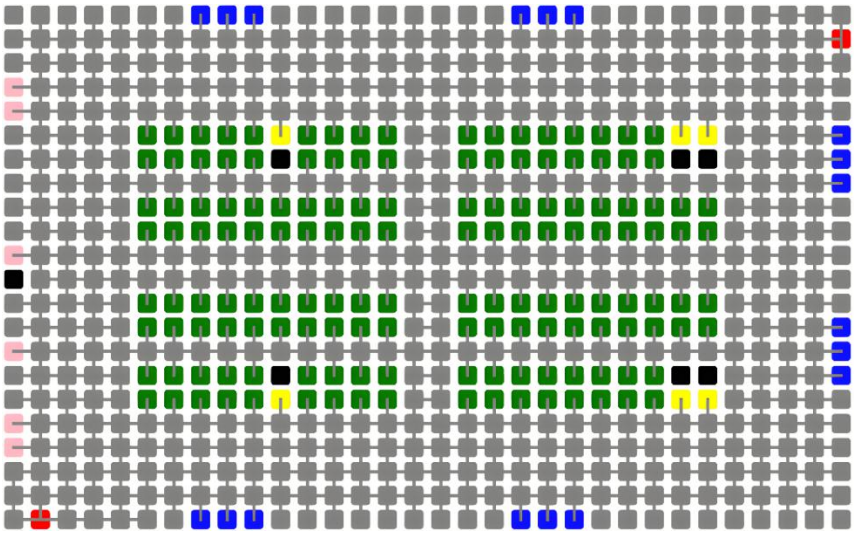


图 2-6 各栅格间的连通关系

三、 问题一：基于改进遗传算法和模拟退火的无人仓多 AGV 路径规划问题

3.1 问题一分析

在本问中假设先不考虑搬运机器人在执行任务时可能的碰撞问题，故考虑所有搬运机器人均进入场地进行工作。故分析问题一所提需求，我们需要考虑满足以下约束：1）满足所有的订单需求，即实现全部拣选工位都空闲；2）保证在搬运过程中每个搬运机器人尽可能忙；3）每个拣选工位同时容纳的托盘数不超过 3 个停靠位。同时实现最小化全部搬运机器人的行走总路径。

考虑在本题中存在补货位节点，补货位节点是从高密度区补货的商品放置点通过传送带进行输送，此时应不增加搬运机器人 AGV 的工作负担。故我们认为在储位库存量低于订单总量时，不足的订单需求由补货位节点直接通过传送带供应，若储位库存量不低于订单总量时，则在满足订单需求的时刻，停止新增搬运机器人的工作（即正在工作路径上的搬运机器人继续工作，直至完成当前工作）。

为保证在搬运过程中每个搬运机器人尽可能忙，在本文中考虑先到达拣选工位节点的搬运机器人先离开，即立刻进行回库或者回收任务，若拣选工位无托盘，则直接进行下一次出库任务。

考虑每个拣选工位同时容纳的托盘数不超过 3 个停靠位，故我们将该约束等价为在每一时刻保证各拣选工位节点的累计工作量的极差尽可能小，即在每一时刻各拣选工位的累计工作量应均衡。

在实际搬运机器人工作过程中，我们认为搬运机器人从起点出发到达拣选工位节点为一条通路（初始起点为搬运机器人的初始位置，而后的起点为搬运机器人上一次到达的拣选工位节点）。而基于图 2-3 和图 2-4 的距离矩阵与路由矩阵，我们可以确定通路的最优路径及最短距离。

进而，为实现最小化全部搬运机器人的行走总路径，我们只需要确定最优总路径情况下储位节点的搬运顺序。因此在问题一中，我们只需满足以上的约束和目标，按照题意建立单目标函数，构建规划模型。为此，我们基于储位节点的搬运顺序进行编码，采用遗传算法和模拟退火算法两种方式进行对比求解。

3.2 变量和符号说明

表 3-1 问题一变量和符号说明列表

符号或者变量	说明
$\boldsymbol{EMPTY}$	按横坐标、纵坐标的次序排列的空储位节点集合，$Empty=(Empty_1,Empty_2\cdots\cdots,Empty_n)^T$
$\boldsymbol{GREEN}$	按横坐标、纵坐标的次序排列的储位节点集合，$Green=(Green_1,Green_2\cdots\cdots,Green_n)^T$
$\boldsymbol{BLUE}$	按横坐标、纵坐标的次序排列的拣选工位节点集合，$Blue=(Blue_1,Blue_2\cdots\cdots,Blue_n)^T$
$\boldsymbol{RED}$	按横坐标、纵坐标的次序排列的空托盘回收节点集合，$Pink=(Red_1,Red_2\cdots\cdots,Red_n)^T$
$\boldsymbol{D}$	各储位节点$green_i$对应的搬运机器人在该储位节点通路行走的最短距离集合，$D=(d_{green_1},d_{green_2}\cdots\cdots,d_{green_n})^T$

$\boldsymbol{GREEN_I}$	储位节点集合 $Green$ 中的第 i 个节点
$\boldsymbol{D_{GREEN_I}}$	储位节点集合 $Green$ 中的第 i 个节点对应的搬运机器人在该储位节点通路行走的距离，若该节点不进行搬运，则其对应的通路行走的距离为 0
$\boldsymbol{DIS}$	利用 Floyd 算法得到的各节点间的距离矩阵。
$\boldsymbol{DIS_{IJ}}$	第 i 个节点到第 j 个节点的最短路径
$\boldsymbol{F}(\Theta)$	表示一个判别函数，若$\theta > 0$，则 $f(\theta)$=1；若$\theta = 0$，则 $f(\theta)$=0
$\boldsymbol{QUANTITY_I}$	储位节点集合 $Green$ 中的第 i 个节点对应的及存放的各个商品 SKU 和数量集合。
$\boldsymbol{QUANTITY_{需求}}$	所有的订单需求集合
$\boldsymbol{YX_{GREEN_I,BLUE_J}}$	储位节点集合 $Green$ 中的第 i 个节点选择第 j 个拣选工位节点时，其所处的优先级（优先级即该储位托盘进入该拣选工位节点的顺序）
$\boldsymbol{M}$	在储位库存量低于订单总量时，不足的订单需求由补货位节点直接通过传送带供应的量。
$\boldsymbol{K_I}$	表示遗传算法中交叉算子和变异算子的设定参数，$i = 1,2,3,4$

3.3 问题一目标规划模型建立

在问题一中，我们的目标是实现最小化全部搬运机器人的行走总路径，并且尽可能少用补货位节点补货。根据两者量纲，我们将其简化为如下单目标函数：

$$min\ z = Green^T D + M \tag{3-1}$$

考虑以下三个约束条件：1）满足所有的订单需求，即实现全部拣选工位都空闲；2）保证在搬运过程中每个搬运机器人尽可能忙；3）每个拣选工位同时容纳的托盘数不超过 3 个停靠位，我们有如下约束：

$$\sum_{i=1}^{n} QUANTITY_{Green_i} * f\left(d_{Green_i}\right) + M - QUANTITY_{需求} \geq 0 \tag{3-2}$$

$$\Delta_t = QUANTITY_{需求} - \sum_{i=1}^{t<n} QUANTITY_{Green_i} * f\left(d_{Green_i}\right) - QUANTITY_{Green_{t+1}} \tag{3-3}$$

$$Diss_{Green_i,Blue_j} = Dis_{Green_i,Blue_j} + Dis_{Green_i,Empty_t} + f(\Delta_t) * Dis_{Blue_t,Empty_t} + (1 - f(\Delta_t)) * \left(Dis_{Red_t,Blue_t} + Dis_{Green_i,Empty_t}\right) + Dis_{Green_i,Blue_t} \tag{3-4}$$

$$YX_{Green_i,Blue_j} + Diss_{Green_i,Blue_j} \leq YX_{Green_i,Blue_k} + Diss_{Green_i,Blue_k}$$

$$其中，Blue_j \in Blue, \forall Blue_k \in Blue \tag{3-5}$$

$$d_{green_i} = Diss_{Green_i,Blue_j} \tag{3-6}$$

其中式（3-2）保证了满足所有的订单需求；式（3-3）定义了某次搬运机器人在搬运储位节点托盘至拣选工位节点拣选完成后，该托盘上剩余的商品情况；式（3-4）定义了储位节点集合 $Green$ 中的第 i 个节点选择第 j 个拣选工位节点时的通路行走距离，根据式（3-3）中托盘上剩余的商品情况进行如图 3-1 所示的 AGV 匹配调度示例图；式（3-5）保证了每次储位节点选择优先级较高（即队列序号较小）并且搬运该节点托盘搬运机器人的行走路径的拣选工位节点，从而保证每个拣选工位容纳的托盘数尽可能均衡，即满足容纳的托盘数不超过 3 个停靠位，同时搬运机器人的行走路径尽可能小；式（3-6）定义了各储位节点$green_i$对应的搬运机器人在该储位节点通路行走的最短距离集合。

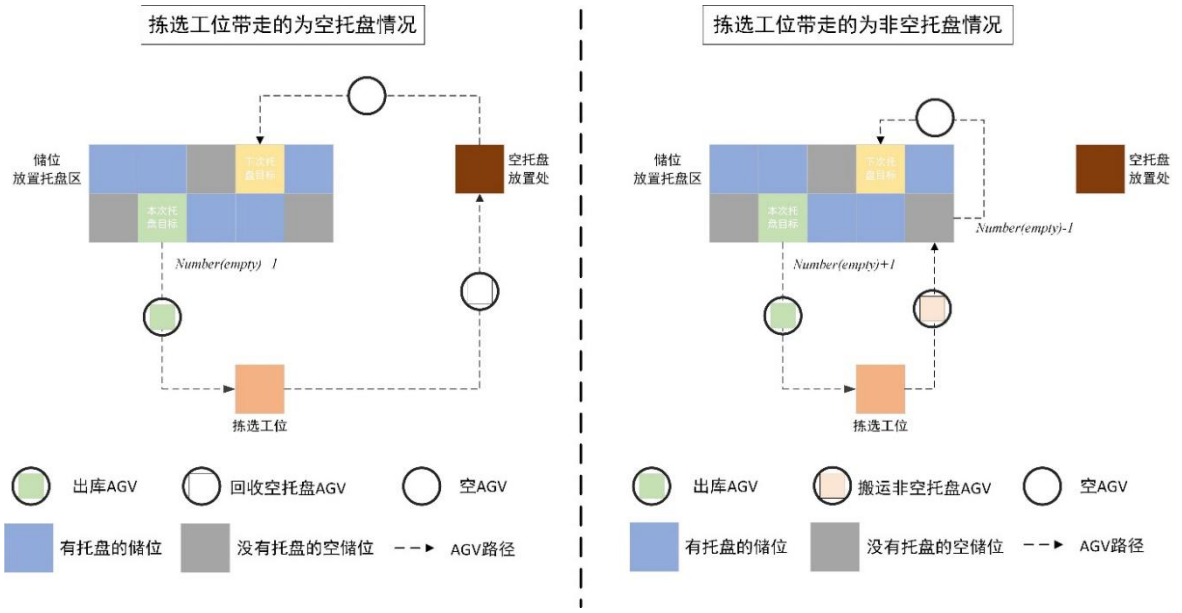


图 3-1 AGV 匹配调度示例图

3.4 问题一改进遗传算法和模拟退火算法设计

3.4.1 基于改进遗传算法的算法设计

因本题的规划无法在多项式时间内解决，故我们可以得到对应的规划问题是一个 NP 难问题，对于该问题，如果使用枚举法求解精确解，其计算量往往呈爆炸增长，陷入维数灾难。针对本题的目标规划，我们考虑使用改进的遗传算法在 python 软件下进行迭代运算求解。

遗传算法的实现过程主要通过对问题的每一个参数进行编码，并产生初始种群，计算适应度，然后进行复制、交叉、变异，如此循环往复 N 代，产生最优的子代，以此确定为全局的最优解。标准遗传算法操作的流程图如图 3-2 所示。

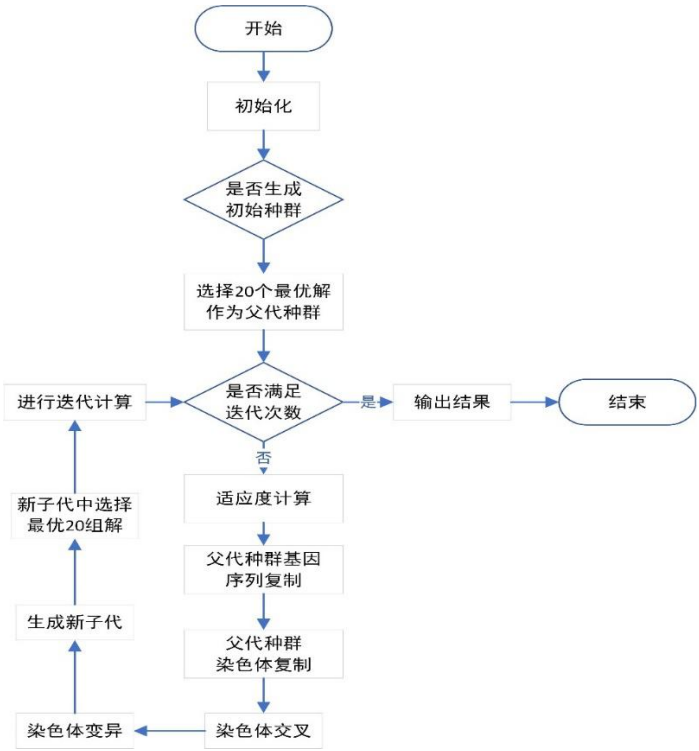


图 3-2 遗传算法流程图

步骤一：融入“储位节点搬运判别要素”的基因编码

遗传算法的实现过程中要对问题的每一个参数进行编码，这里往往采用二进制编码、浮点数编码、混合编码以及实数编码，本题我们采用融合二进制编码和整数编码的混合编码。我们根据储位节点搬运判别要素，有以下规定：若该储位节点在整个工作周期内被搬运，则记为 0;：若该储位节点在整个工作周期内被搬运，则记为 1。基于储位节点搬运顺序进行排列，则有储位节点搬运顺序编码，这里的各个基因为储位节点的编号。例如：1 34|0 13|1 23|1 87（其中有 4 个储位节点）这一串基因，表示该场地内有 4 个储位节点，其中搬运机器人在整个工作周期内依次进行 34 号、23 号、87 号储位节点的托盘搬运，不搬运 13 号储位节点。

步骤二：初始种群生成

种群大小对于遗传算法的收敛性是非常重要的。如果种群规模太小，遗传算法容易收敛到局部最优解，相反，如果种群规模过大，则遗传算法的计算速度会降低。种群的大小与变量 N 相关，适当的种群大小应控制在 4N 和 6N 之间。故在本文中设定初始种群数量为 500。

步骤三：适应度设计

选择亦称再生或复制，选择过程是个体按照其适应度进行择优复制。在本题中我们将目标函数值认定为适应度函数 fitness。按照适应度概率挑选优秀的子代进行复制，淘汰效果不佳的子代，以便于之后的子代都普遍更接近于最优值。其中我们往往通过随机方法来实现选择的操作. 因本题具体计算的基因长度过长，故我们基于上述编码中的例子示意，我们的选择过程如下：

（1）由第二步得如下 4 个个体，他们各自对应的基因串为：

1 56|1 33|1 53|1 57；1 64|0 13|0 93|1 107；1 34|0 13|1 23|1 87；1 84|0 103|1 63|1 55

假设他们各自代入目标函数中所得适应度为 80 94 111 453；则有如下表 3-2 中的种群的初始串及对应的适应度：

表 3-2 种群初始串及对应适应度

序号	串	适应度	百分占比%	期望选择数	实际选择数
1	1 56|1 33|1 53|1 57	80	10.8	0.43	0
2	1 64|0 13|0 93|1 107	94	12.7	0.51	1
3	1 34|0 13|1 23|1 87	111	15.0	0.60	1
4	1 84|0 103|1 63|1 55	453	61.5	2.46	2
总计		738	100.0	4.00	4
平均		184.5	25.0	1.00	1
最大值		453	61.5	2.46	2

对应的轮盘赌转盘：

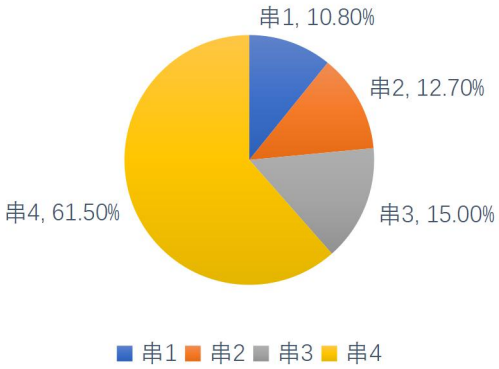


图 3-3 轮盘赌转盘

串 1 所占轮盘的比例为 10.8%，因此每转动一次轮盘落入串 1 所在区域的概率就为 0.108，串 2 所占轮盘的比例为 12.7%，因此每转动一次轮盘落入串 2 所在区域的概率就为 0.127，串 3 所占轮盘的比例为 15.0%，因此每转动一次轮盘落入串 3 所在区域的概率就为 0.150，串 4 所占轮盘的比例为 61.5%，因此每转动一次轮盘落入串 4 所在区域的概率就为 0.615。所有转动 4 次转盘，就可复制出 4 个个体得到新种群，其中有的串被复制多次，有的没有被复制，从而淘汰。在本例中，串 4 被复制了两次，串 2 和串 3 被复制了一次，串 1 被淘汰。

终止条件：迭代 N 代（本题选取 1000 代）。

步骤四：交叉算子设计

因本题的个体基因序列中不能出现同样的染色体片段，即储位节点的托盘不进行二次搬运。故交叉处理时考虑在新选择产生的匹配池中的成员中随机选择个体进行复制并逆序排列，进行两两染色体匹配，然后以一定的概率P_c进行交叉繁殖的过程，具体过程如下例：

（1） 随机从匹配池中选取一个成员进行复制

（2） 将复制的染色体序列进行逆序排列

（3） 将原染色体与逆序排列的染色体这两个成员对应匹配

（4） 首先给个体染色体中的 10 个空隙编号 1-10 号。在概率P_c下这两个成员会发生交叉，这时依均匀分布随机取两个不相同的数字 N,M∈[1，10]，插入个体染色体中对应的 1-10 号空隙。在 N，M 之间，两个个体的染色体发生交叉，如图 3-4：

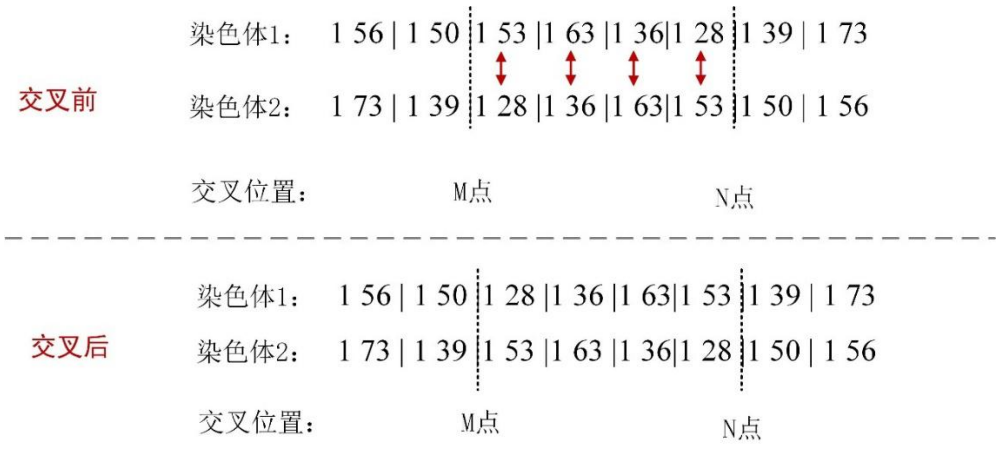


图 3-4 交叉前后染色体变化示意图

为保证个体交叉的合理性，我们规定交叉概率P_c以式（3-7）自适应的方式得到，

从而保证当种群中的个体适应度值趋于一致时，则会增加交叉运算的概率，当个体适应度值趋于分散的时候，则减小交叉运算的概率。并且该方法使得适应度高的个体对应交叉的概率减小，适应度低的个体对应交叉概率增大，有助于保护高适应性的个体，淘汰低适应性的个体，能够更好地找到最优解。

$$P_C=\begin{cases}\dfrac{k_1(\mathrm{f}_{\max}-\mathrm{f}')}{f_{\max}-f_{avg}}, \mathrm{f}'\geq f_{avg}\\ k_2, \mathrm{f}'<f_{avg}\end{cases} \tag{3-7}$$

其中 $\mathrm{f}_{\max}$ 为群体中最大适应度值；f_{avg} 为每代群体的最平均适应度值；f' 为需要交叉个体的适应度值

步骤五：变异算子设计

变异是以很小的概率随机地改变一个储位节点搬运判别要素串位的值，能够防止丢失一些有用的遗传因子。变异的具体处理如下图 3-5：

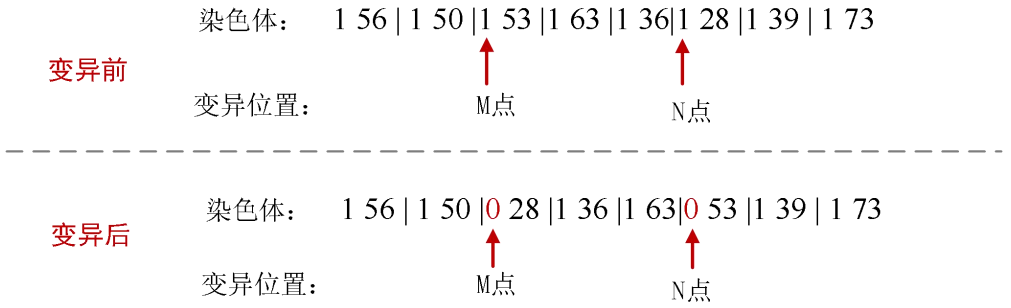


图 3-5 变异具体操作示意

同时我们规定变异概率P_m以式（3-8）自适应的方式得到，从而保证当种群中的个体适应度值趋于一致时，则会增加变异运算的概率，当个体适应度值趋于分散的时候，则减小变异运算的概率。并且该方法使得适应度高的个体对应变异的概率减小，适应度低的个体对应变异概率增大，有助于保护高适应性的个体，淘汰低适应性的个体，能够更好地找到最优解。

$$P_m=\begin{cases}\dfrac{k_3(\mathrm{f}_{\max}-\mathrm{f}')}{f_{\max}-f_{avg}}, \mathrm{f}'\geq f_{avg}\\ k_4, \mathrm{f}'<f_{avg}\end{cases} \tag{3-8}$$

其中 $\mathrm{f}_{\max}$ 为群体中最大适应度值；f_{avg} 为每代群体的最平均适应度值；f' 为需要变异个体的适应度值。

3.4.2 基于模拟退火算法的算法设计

模拟退火（Simulate Anneal，SA）是由 S.Kirkpatrick，C.D.Gelatt 和 M.P.Vecchi 在 1983 年所发明的，该算法的出发点是基于物理中固体物质的退火过程与一般组合优化问题之间的相似性。模拟退火算法来源于固体退火原理，是一种基于概率的算法，模拟退火算法基于这样一种物理原理：一个高温物体降至常温，温度越高时降温在概率越大（降温

越快），温度越低时降温的概率越小（降温越慢）。模拟退火算法基于此种思想搜索，即多次降温（迭代），直到获得一个可行解。

模拟退火基于贪心算法。与同样基于贪心算法的爬山算法（Hill Climbing）不同的是，模拟退火算法在一些情况下可能会克服陷入局部最优解而非找到全局最优解的情况。故针对本题的目标规划，我们考虑使用模拟退火算法在 python 软件下进行迭代运算求解。

模拟退火算法新解的产生和接受可分为如下步骤：

(1) 初始化：初始温度 T(充分大)，初始解状态 S(是算法迭代的起点)，每个 T 值的迭代次数 L

(2) 对 $k = 1,2,3\cdots\cdots,L$ 做第(3)至第 6 步：

(3) 产生新解S'

(4) 计算增量$\Delta T = C(S') - C(S)$，其中 $C(S)$为评价函数，在本问题中即目标函数

(5) 根据 Metropolis 准则，若$\Delta T < 0$,则接受S'作为新的当前解，否则以概率 $exp\left(\frac{\Delta T}{T}\right)$接受$S'$作为新的当前解

(6) 如果满足终止条件则输出当前解作为最优解，结束程序。

终止条件取为连续若干个新解都没有被接受时终止算法。

(7)若程序没有停止，则 T 逐渐减少，且$\Delta T \to 0$，然后转第 2 步。

在这里我们根据储位节点搬运判别要素，有以下规定：若该储位节点在整个工作周期内被搬运，则记为 0;：若该储位节点在整个工作周期内被搬运，则记为 1。基于储位节点搬运顺序进行排列，则有储位节点搬运顺序编码，这里的各个编码为储位节点的编号。例如：1 34|0 13|1 23|1 87（其中有 4 个储位节点）为一个可行解，表示该场地内有 4 个储位节点，其中各搬运机器人在整个工作周期内依次进行 34 号、23 号、87 号储位节点的托盘搬运，不搬运 13 号储位节点。

为此，我们产生新解的方式可通过如图 3-6 所示的编码片段交换获得。

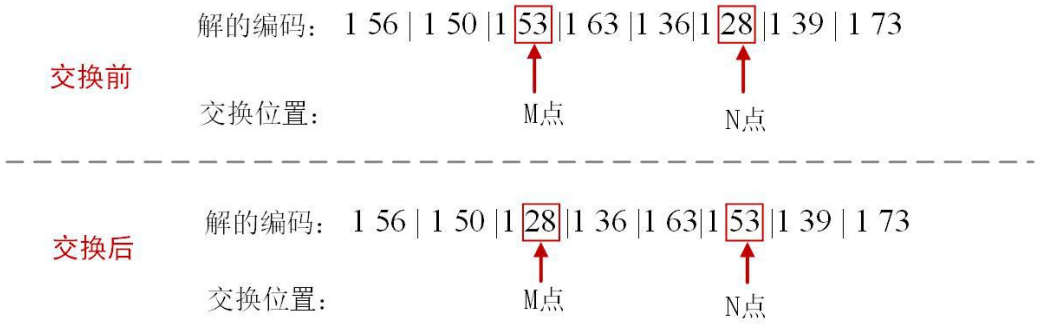


图 3-6 模拟退火编码片段交换

3.5 问题一的求解与分析

3. 5. 1 基于改进遗传算法的算法实现

本题基于上述的模型与遗传算法，通过 Python 语言进行编程实现，最终计算结果为：**所有搬运机器人的总路径为 7316，对应的通过补货节点的补货总量为 307，其中，不进行搬运的储位节点坐标为（5，13）(5,12) (10,12)。**

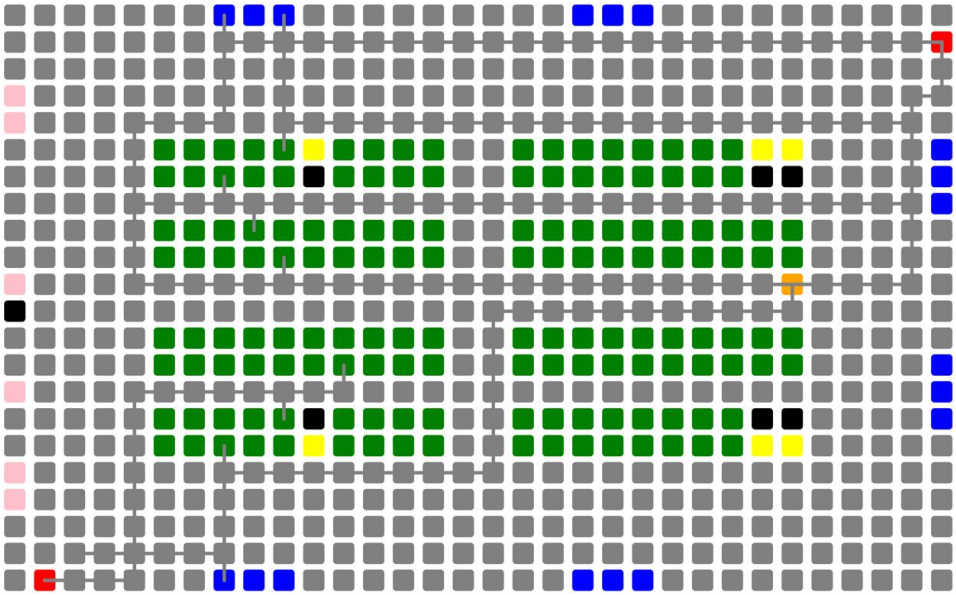


图 3-7 遗传算法下的一个 AGV 的搬运路径

(26, 11)→ (26, 10)→ (25, 10)→ (24, 10)→ (23, 10)→ (22, 10)→ (21, 10)→ (20, 10)→ (19, 10)→ (18, 10)→ (17, 10)→ (16, 10)→ (16, 9)→ (16, 8)
→ (16, 7)→ (16, 6)→ (16, 5)→ (16, 4)→ (15, 4)→ (14, 4)→ (13, 4)→ (12, 4)→ (11, 4)→ (10, 4)→ (9, 4)→ (8, 4)→ (7, 4)→ (7, 5)→ (7, 4)→
(7, 3)→ (7, 2)→ (7, 1)→ (7, 0)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→
(4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (5, 7)→ (6, 7)→ (7, 7)→ (8, 7)→ (9, 7)→ (9, 6)→ (9, 7)→ (8, 7)→ (7, 7)→ (6, 7)→ (5, 7)→ (4, 7)→
(4, 6)→ (4, 5)→ (4, 4)→ (4, 3)→ (4, 2)→ (4, 1)→ (5, 1)→ (6, 1)→ (7, 1)→ (7, 0)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→
(1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→ (4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (5, 7)→ (6, 7)→ (7, 7)→ (8, 7)→ (9, 7)→ (10, 7)
→ (11, 7)→ (11, 8)→ (11, 7)→ (10, 7)→ (9, 7)→ (8, 7)→ (7, 7)→ (6, 7)→ (5, 7)→ (4, 7)→ (4, 6)→ (4, 5)→ (4, 4)→ (4, 3)→ (4, 2)→ (4, 1)→
(5, 1)→ (6, 1)→ (7, 1)→ (7, 0)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→
(4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (4, 8)→ (4, 9)→ (4, 10)→ (5, 10)→ (6, 10)→ (7, 10)→ (8, 10)→ (9, 10)→ (10, 10)→ (10, 9)→ (10, 1
0)→ (9, 10)→ (8, 10)→ (7, 10)→ (6, 10)→ (5, 10)→ (4, 10)→ (4, 9)→ (4, 8)→ (4, 7)→ (4, 6)→ (4, 5)→ (4, 4)→ (4, 3)→ (4, 2)→ (4, 1)→ (5,
1)→ (6, 1)→ (7, 1)→ (7, 0)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→
(30, 16)→ (30, 15)→ (30, 14)→ (30, 13)→ (30, 12)→ (30, 11)→ (29, 11)→ (28, 11)→ (27, 11)→ (26, 11)→ (25, 11)→ (24, 11)→ (23, 11)→ (22, 1
1)→ (21, 11)→ (20, 11)→ (19, 11)→ (18, 11)→ (17, 11)→ (16, 11)→ (15, 11)→ (14, 11)→ (13, 11)→ (12, 11)→ (11, 11)→ (10, 11)→ (9, 11)→
(9, 12)→ (9, 11)→ (8, 11)→ (7, 11)→ (6, 11)→ (5, 11)→ (4, 11)→ (4, 12)→ (4, 13)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→ (6, 17)→
(7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (1
6, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)
→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (2
5, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (13, 14)→ (12, 14)
→ (11, 14)→ (10, 14)→ (9, 14)→ (8, 14)→ (8, 13)→ (8, 14)→ (7, 14)→ (6, 14)→ (5, 14)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→ (6, 1
7)→ (7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)
→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (2
9, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)
→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (13, 14)→ (1
2, 14)→ (11, 14)→ (10, 14)→ (9, 14)→ (8, 14)→ (7, 14)→ (7, 15)→ (7, 14)→ (6, 14)→ (5, 14)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→
(6, 17)→ (7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15,
20)→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→
(29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (29, 17)→ (28, 17)→ (27, 17)→ (26, 17)→ (25, 17)→ (24, 17)→ (23, 1
7)→ (22, 17)→ (21, 17)→ (20, 17)→ (19, 17)→ (18, 17)→ (17, 17)→ (16, 17)→ (15, 17)→ (14, 17)→ (13, 17)→ (12, 17)→ (11, 17)→ (10, 17)→
(9, 17)→ (9, 16)→ (9, 17)→ (9, 18)→ (9, 19)→ (9, 20)→ (9, 21)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (1
6, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)

图 3-8 遗传算法下该 AGV 的具体行动路径

如图 3-7，该图为在所有搬运机器人总路径最短的情况下，其中一个 AGV 在仓库的行动轨迹，且图 3-8 为该 AGV 的具体行动路径。

通过计算，在目标函数最优的情况下，每一个拣选工位的工作量如下表所示。

表 3-3 基于遗传算法的拣选工位工作量

工位节点坐标	工作量
（7，0）、（8，0）、（9，0）	25
（19，0）、（20，0）、（21，0）	26
（31，6）、（31，7）、（31，8）	30
（31，14）、（31，15）、（31，16）	14
（7，21）、（8，21）、（9，21）	26
（19，21）、（20，21）、（21，21）	26

3.5.2 基于模拟退火算法的算法实现

本题基于上述的模型与模拟退火算法，通过 Python 语言进行编程实现，最终计算结果为：**所有搬运机器人的总路径为 7428，对应的通过补货节点的补货总量为 348，其中，不进行搬运的储位节点坐标有（5，13）(5,12) (10,12)。**

在目标函数最优的情况下，每一个拣选工位的工作量分别为：

表 3-4 基于模拟退火的拣选工位工作量

工位节点坐标	工作量
（7，0）、（8，0）、（9，0）	26
（19，0）、（20，0）、（21，0）	26
（31，6）、（31，7）、（31，8）	29
（31，14）、（31，15）、（31，16）	14
（7，21）、（8，21）、（9，21）	26
（19，21）、（20，21）、（21，21）	26

3.5.3 两种算法求解结果的比较分析

本题运用了两种启发式算法进行求解，分别是改进的遗传算法和模拟退火算法。从运算结果来看，在改进的遗传算法的求解下，所有 AGV 的总路径最短，需要补货数量最少。因此可以得到**改进的遗传算法明显优于模拟退火算法**。

四、 问题二：基于两阶段的目标规划的多 AGV 路径规划问题

4.1 问题二分析

问题二在问题一的基础上，为了更好地平衡拣选工位的负载，同时预防搬运机器人的局部拥堵，根据拣选工位和库存商品数量对仓库地图进行动态分区。也就是对仓库内储位上的每个托盘，都指定一个默认拣选工位。并使得每个拣选工位对应托盘的商品总量尽可能地平均，同时要求最小化全部托盘到其默认拣选工位距离总和。因此，我们将问题二拆分为两阶段的目标规划问题，如图 4-1 所示，其中第一阶段为对根据拣选工位和库存商品数量对仓库地图进行动态分区，建立多目标规划模型。第二阶段为问题一中所提的 AGV 调度问题，该层问题基于第一阶段的动态分区结果进行求解。相较于问题一，在第二阶段的问题中，仓库内储位上的每个托盘，都指定一个默认拣选工位，即不需要对储位上的每个托盘进行拣选工位选择。

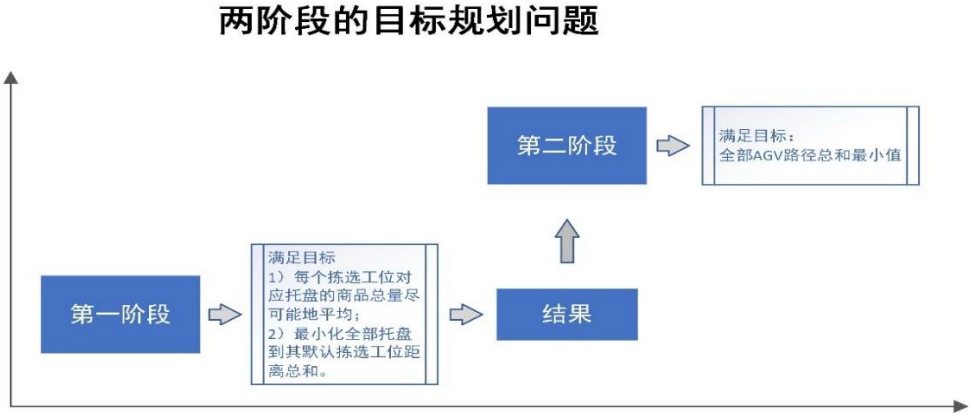


图 4-1 两阶段的目标规划问题

为解决第一阶段的目标规划，我们规定全部托盘到各拣选工位的最短距离由 2.3.2 中的距离矩阵和路由矩阵唯一确定。并有以下两个目标：1）每个拣选工位对应托盘的商品总量尽可能地平均；2）最小化全部托盘到其默认拣选工位距离总和。并针对目标 1 提出基于向量范数中的 1-范数和∞-范数的两种等价目标。

为解决第二阶段的目标规划，我们在满足问题一中各约束的基础上，新增一条约束，即对仓库内储位上的每个托盘，都指定一个由第一阶段的目标规划唯一确定的默认拣选工位。并有同时实现最小化全部搬运机器人的行走总路径的目标函数。

针对这个两阶段目标规划问题，我们沿用问题一中所设计的遗传算法进行第二阶段目标规划的求解，并设计一种具有新型编码方式的遗传算法来实现第一阶段目标规划的求解。

4.2 变量和符号说明

表 4-1 问题二变量和符号说明列表

变量或符号	说明
$\boldsymbol{EMPTY}$	按横坐标、纵坐标的次序排列的空储位节点集合，$Empty = (Empty_1, Empty_2 \cdots\cdots, Empty_n)^T$
$\boldsymbol{GREEN}$	按横坐标、纵坐标的次序排列的储位节点集合，$Green = (Green_1, Green_2 \cdots\cdots, Green_n)^T$
$\boldsymbol{BLUE}$	按横坐标、纵坐标的次序排列的拣选工位节点集合，$Blue =$

	$(Blue_1,Blue_2\cdots\cdots,Blue_n)^T$
$\boldsymbol{RED}$	按横坐标、纵坐标的次序排列的空托盘回收节点集合，$Pink=(Red_1,Red_2\cdots\cdots,Red_n)^T$
$\boldsymbol{D}$	各储位节点$green_i$对应的搬运机器人在该储位节点通路行走的最短距离集合,$D=(d_{green_1},d_{green_2}\cdots\cdots,d_{green_n})^T$
$\boldsymbol{GREEN_I}$	储位节点集合 $Green$ 中的第 i 个节点
$\boldsymbol{D_{GREEN_I}}$	储位节点集合 $Green$ 中的第 i 个节点对应的搬运机器人在该储位节点通路行走的距离，若该节点不进行搬运，则其对应的通路行走的距离规定为 0
$\boldsymbol{DIS}$	利用 Floyd 算法得到的各节点间的距离矩阵。
$\boldsymbol{DIS_{IJ}}$	第 i 个节点到第 j 个节点的最短路径
$\boldsymbol{F}(\Theta)$	表示一个判别函数，若$\theta>0$，则 $f(\theta)$=1；若$\theta=0$，则 $f(\theta)$=0
$\boldsymbol{QUANTITY_I}$	储位节点集合 $Green$ 中的第 i 个节点对应的及存放的各个商品 SKU 和数量。
$\boldsymbol{QUANTITY_{需求}}$	所有的订单需求集合
$\boldsymbol{YX_{GREEN_I,BLUE_J}}$	储位节点集合 $Green$ 中的第 i 个节点选择第 j 个拣选工位节点时，其所处的优先级（优先级即该储位进入该拣选工位节点的顺序）
$\boldsymbol{M}$	在储位库存量低于订单总量时，不足的订单需求由补货位节点直接通过传送带供应的量。
$\boldsymbol{K_I}$	表示遗传算法中交叉算子和变异算子的设定参数，$i=1,2,3,4$
$\boldsymbol{B_J}$	表示第 j 个储位节点所对应的商品总量
$\boldsymbol{A_{IJ}}$	表示从储位节点能否前往拣选工位节点的判断元素，若储位节点 j 能前往拣选工位节点 i，则$a_{ij}=1$，否则$a_{ij}=0$
$\boldsymbol{A_{QJ}}$	表示从储位节点能否前往拣选工位节点的判断元素，若储位节点 j 能前往拣选工位节点 q，则$a_{qj}=1$，否则$a_{qj}=0$
$\boldsymbol{B_J}$	由动态分区规划模型所得的与拣选工位节点 j 相匹配的拣选工位节点与储位节点集合

4.3 问题二的动态分区和路径规划模型建立

4.3.1 动态分区规划模型

在动态分区规划模型中，我们的目标有：1）每个拣选工位对应托盘的商品总量尽可能地平均；2）最小化全部托盘到其默认拣选工位距离总和。为实现每个拣选工位对应托盘的商品总量尽可能地平均，我们定义两种平衡评价目标函数，分别基于向量范数中的 1-范数和∞-范数。

4.3.1.1 基于 1−范数的动态分区规划模型

为实现每个拣选工位对应托盘的商品总量尽可能地平均，我们提出一种基于向量范数中的 1-范数的等价目标，即任意两个拣选工位对应托盘的商品总量的差的绝对值之和最小，因此有目标函数：

$$min\sum_{i\in Blue}\sum_{q\in Blue}\left|\sum_{j\in Green}a_{ij}\,b_j-\sum_{j\in Green}a_{qj}b_j\right| \tag{4-1}$$

同时，为了保证最小化全部托盘到其默认拣选工位距离总和，我们有目标函数：

$$min\sum_{i\in Blue}\sum_{j\in Green}a_{ij}\,Dis_{ij} \tag{4-2}$$

根据线性加权法思路可把多目标模型简化为单目标模型：

$$min\sum_{j=1}^{n}w_j\varphi_j(x) \tag{4-3}$$

$$s.t.\quad g_k(x)\le 0,k=1,2,\ldots,l \tag{4-4}$$

基于这种思想，将上述提出的式（4-1）、式（4-2）目标函数引入，并赋予相应的权重因子，便构造出效用函数：

$$W=\sum_{j=1}^{c}w_j\varphi_j \tag{4-5}$$

但各目标函数具有不同的量纲，不能简单通过设置权重来得到联立后的目标函数，需要进一步对式（4-5）函数进行归一化。

设$\varphi_j^0=max\{|\varphi_j|\}$，且$\left|\varphi_j^0\right|\neq 0$，则经过归一化处理的效用函数为：

$$W=\sum_{j=1}^{c}\frac{w_j\varphi_j}{\varphi_j^0} \tag{4-6}$$

4.3.1.2 基于∞范数的动态分区规划模型

为实现每个拣选工位对应托盘的商品总量尽可能地平均，我们提出一种基于向量范数中的∞-范数的等价目标，即任意两个拣选工位对应托盘的商品总量的差的绝对值的最大值最小，因此有目标函数：

$$min\left(max\left|\sum_{j\in Green}a_{ij}b_j-\sum_{j\in Green}a_{qj}b_j\right|\right)，\ \forall i\in Blue，\ \forall q\in Blue \tag{4-7}$$

同时，为了保证最小化全部托盘到其默认拣选工位距离总和，我们有目标函数：

$$min\sum_{i\in Blue}\sum_{j\in Green}a_{ij}\,Dis_{ij} \tag{4-8}$$

根据线性加权法思路可把多目标模型简化为单目标模型：

$$min\sum_{j=1}^{n}w_j\varphi_j(x) \tag{4-9}$$

$$s.t.\quad g_k(x)\le 0,k=1,2,\ldots,l \tag{4-10}$$

基于这种思想，将上述提出的式（4-7）、式（4-8）目标函数引入，并赋予相应的权重因子，便构造出如式（4-6）所示的效用函数。

4.3.2 行走路径规划模型

行走路径规划问题即问题一中所提的 AGV 调度问题，该问题基于 4.3.1 的动态分区问题的动态分区结果进行求解。相较于问题一，在本问题中，仓库内储位上的每个托

盘，都指定一个默认拣选工位，即不需要对储位上的每个托盘进行拣选工位选择。

在行走路径规划问题中，我们的目标是实现最小化全部搬运机器人的行走总路径，并且尽可能少用补货位节点补货。根据两者量纲，我们将其简化为如下单目标函数：

$$min\ z = Green^T D + M \tag{4-11}$$

考虑以下三个约束条件：1）满足所有的订单需求，即实现全部拣选工位都空闲；2）保证在搬运过程中每个搬运机器人尽可能忙；3）每个拣选工位同时容纳的托盘数不超过 b 个停靠位，我们有如下约束：

$$\sum_{i=1}^{n} QUANTITY_{Green_i} * f\left(d_{Green_i}\right) + M - QUANTITY_{需求} \geq 0 \tag{4-12}$$

$$= QUANTITY_{需求} - \sum_{i=1}^{t<n} QUANTITY_{Green_i} * f\left(d_{Green_i}\right) - QUANTITY_{Green_{t+1}} \tag{4-13}$$

$$Diss_{Green_i,Blue_j} = Dis_{Green_i,Blue_j} + f(\Delta_t) * Dis_{Green_i,Empty_t} + (1 - f(\Delta_t)) * \left(Dis_{Red_t,Blue_t} + Dis_{Green_i,Red_t}\right) + Dis_{Green_i,Blue_t} \tag{4-14}$$

$$Green_i \cap Blue_j \in B_j \tag{4-15}$$

$$d_{green_i} = Diss_{Green_i,Blue_j} \tag{4-16}$$

其中式（4-12）保证了满足所有的订单需求；式（4-13）定义了某次搬运机器人在搬运储位节点托盘至拣选工位节点拣选完成后，该托盘上剩余的商品情况；式（4-14）定义了储位节点集合 $Green$ 中的第 i 个节点选择第 j 个拣选工位节点时的通路行走距离，根据式（4-13）中托盘上剩余的商品情况进行 AGV 匹配调度；式（4-15）约束了储位节点$Green_i$和拣选工位节点$Blue_j$满足动态分区规划的解；式（4-16）定义了各储位节点$green_i$对应的搬运机器人在该储位节点通路行走的最短距离集合。

4.4 问题二的改进遗传算法设计

针对这个两阶段目标规划问题，我们沿用问题一中所设计的遗传算法进行行走路径目标规划的求解，并设计一种具有**新型编码方式的遗传算法**来实现动态分区目标规划的求解。

4.4.1 步骤一：动态分区目标规划的编码方式

遗传算法的实现过程中要对问题的每一个参数进行编码，这里往往采用二进制编码、浮点数编码、混合编码以及实数编码，本题我们采用整数编码，根据储位节点的编号顺序进行编码。例如：53241240（其中有 8 个储位节点）这一串基因，表示该场地内的 8 个连续储位节点，其中搬运机器人在整个工作周期内默认依次托盘搬运至 5，3，2，4，1，2，4，0 号拣选工位。

4.4.2 步骤二：动态分区目标规划的交叉算子

因本题的个体基因序列中仓库内储位上的每个托盘，都指定唯一一个默认拣选工位，即储位节点的托盘不进行二次搬运。故交叉处理时考虑在新选择产生的匹配池中的成员中随机选择个体进行复制并逆序排列，进行两两染色体匹配，然后以一定的概率P_c进行交叉繁殖的过程，具体过程如下例：

（1）随机从匹配池中选取两个成员进行染色体匹配

（2）首先给个体染色体中的 10 个空隙编号 1-10 号。在概率P_c下这两个成员会发生交叉，这时依均匀分布随机取两个不相同的数字 N,M∈[1，10]，插入个体染色体中对应的 1-10 号空隙。在 N，M 之间，两个个体的染色体发生交叉，如图 4-2：

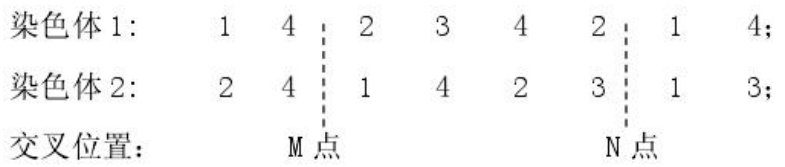


图 4-2 M、N 染色体交叉图

交叉以后形成两条新的染色体如图 4-3：

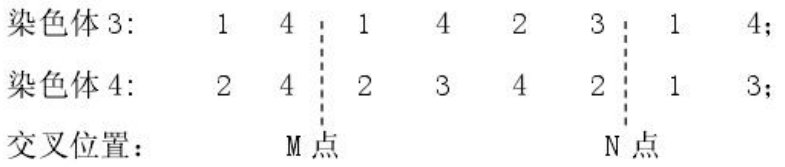


图 4-3 交叉后形成新染色体

为保证个体交叉的合理性，我们规定交叉概率P_c以式（4-17）自适应的方式得到，从而保证当种群中的个体适应度值趋于一致时，则会增加交叉运算的概率，当个体适应度值趋于分散的时候，则减小交叉运算的概率。并且该方法使得适应度高的个体对应交叉的概率减小，适应度低的个体对应交叉概率增大，有助于保护高适应性的个体，淘汰低适应性的个体，能够更好地找到最优解。

$$P_C=\begin{cases}\dfrac{k_1(\mathrm{f}_{\max}-\mathrm{f}')}{f_{\max}-f_{avg}}, \mathrm{f}'\geq f_{avg}\\ k_2, \mathrm{f}'<f_{avg}\end{cases} \tag{4-17}$$

其中$\mathrm{f}_{\max}$为群体中最大适应度值；f_{avg}为每代群体的最平均适应度值；f'为需要交叉个体的适应度值。

4.4.3 步骤三：动态分区目标规划的变异算子

变异是以很小的概率随机地改变一个串位的值，能够防止丢失一些有用的遗传因子。变异的具体步骤如下：对于交叉产生的子代个体，顺序检查交叉之后的所有编码，若满足所有约束条件则该位编码不变，否则按初始种群生成的规则依概率P_m重新为当前编码位取值，从而保证交叉后的子代依旧为可行解。

若子代为可行解，则我们可规定变异概率P_m以式（4-18）自适应的方式得到，从而保证当种群中的个体适应度值趋于一致时，则会增加变异运算的概率，当个体适应度值趋于分散的时候，则减小变异运算的概率。并且该方法使得适应度高的个体对应变异的概率减小，适应度低的个体对应变异概率增大，有助于保护高适应性的个体，淘汰低适应性的个体，能够更好地找到最优解。

$$P_m=\begin{cases}\dfrac{k_3(\mathrm{f}_{\max}-\mathrm{f}')}{f_{\max}-f_{avg}}, \mathrm{f}'\geq f_{avg}\\ k_4, \mathrm{f}'<f_{avg}\end{cases} \tag{4-18}$$

其中 f_{max} 为群体中最大适应度值；f_{avg} 为每代群体的最平均适应度值；f' 为需要变异个体的适应度值。

4.5 问题二基于改进遗传算法的求解与分析

4.5.1 动态分区目标规划求解

4.5.1.1 基于 1-范数的动态分区规划求解

本题基于上述的模型与算法，通过 Python 语言进行编程实现，最终计算结果为：可以得到如图 4-4 的储位分区。

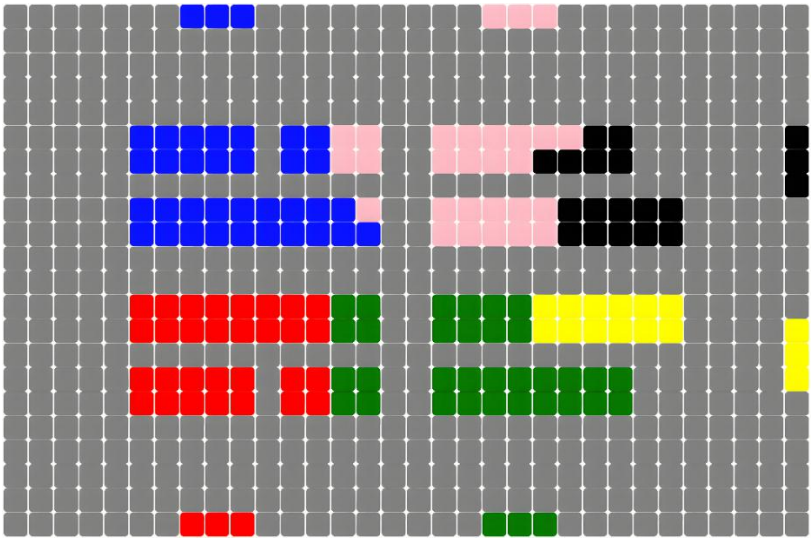


图 4-4 基于 1-范数的动态分区图

根据上图，我们可以得到如上图的分区方案，相同颜色的托盘将默认送至相同颜色的拣选工位。

4.5.1.2 基于∞-范数的动态分区规划求解

本题基于上述的模型与算法，通过 Python 语言进行编程实现，最终计算结果为：可以得到如图 4-5 的储位分区：

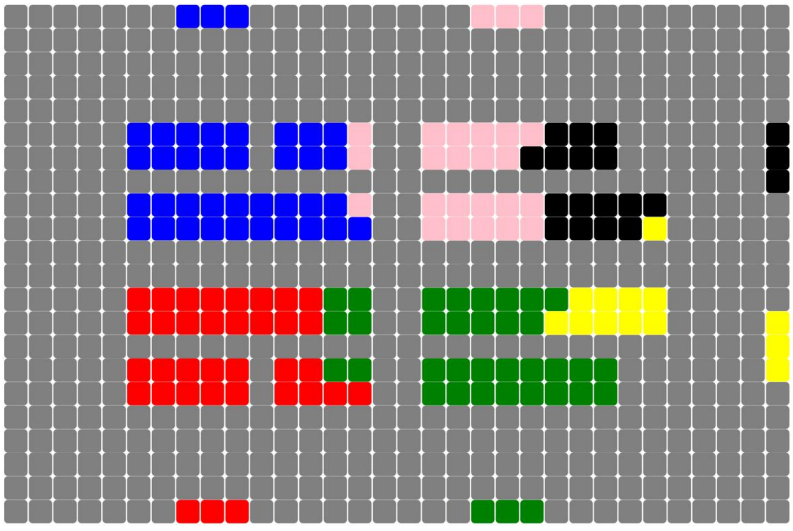


图 4-5 基于 1-范数的动态分区图

根据上图，我们可以得到如上图的分区方案，相同颜色的托盘将默认送至相同颜色的拣选工位。

4.5.2 行走路径目标规划求解

本题基于上述的模型与算法，通过 Python 语言进行编程实现，最终计算结果为：所有**搬运机器人的总路径为 7453，对应的通过补货节点的补货总量为 268，其中，不进行搬运的储位节点坐标为(10,9) (5,9),(22,12)。**

如图 4-6，该图为在所有搬运机器人总路径最短的情况下，其中一个 AGV 在仓库的行动轨迹，且如图 4-7，该图为在所有搬运机器人总路径最短的情况下，该 AGV 在仓库的行动路径的部分截图。

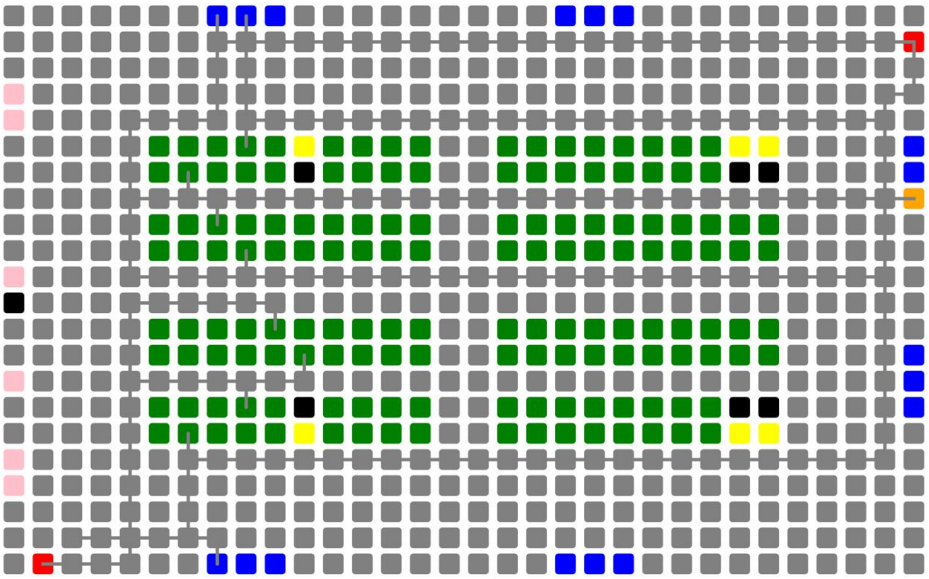


图 4-6 AGV 的移动路径图

(7, 21)→ (7, 20)→ (7, 19)→ (7, 18)→ (7, 17)→ (6, 17)→ (5, 17)→ (4, 17)→ (4, 16)→ (4, 15)→ (4, 14)→ (4, 13)→ (4, 12)→ (4, 11)→ (4, 10)→ (4, 9)→ (4, 8)→ (4, 7)→ (4, 6)→ (4, 5)→ (4, 4)→ (5, 4)→ (6, 4)→ (7, 4)→ (8, 4)→ (9, 4)→ (9, 5)→ (9, 4)→ (9, 3)→ (9, 2)→ (9, 1)→ (9, 0)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→ (4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (5, 7)→ (6, 7)→ (7, 7)→ (8, 7)→ (9, 7)→ (10, 7)→ (11, 7)→ (12, 7)→ (12, 6)→ (12, 7)→ (13, 7)→ (14, 7)→ (15, 7)→ (15, 6)→ (15, 5)→ (15, 4)→ (15, 3)→ (15, 2)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (9, 0)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→ (4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (5, 7)→ (6, 7)→ (7, 7)→ (8, 7)→ (9, 7)→ (10, 7)→ (11, 7)→ (12, 7)→ (13, 7)→ (13, 8)→ (13, 7)→ (14, 7)→ (15, 7)→ (15, 6)→ (15, 5)→ (15, 4)→ (15, 3)→ (15, 2)→ (15, 1)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (4, 1)→ (4, 2)→ (4, 3)→ (4, 4)→ (4, 5)→ (4, 6)→ (4, 7)→ (4, 8)→ (4, 9)→ (4, 10)→ (5, 10)→ (6, 10)→ (7, 10)→ (8, 10)→ (9, 10)→ (10, 10)→ (11, 10)→ (12, 10)→ (12, 9)→ (12, 10)→ (13, 10)→ (14, 10)→ (15, 10)→ (15, 9)→ (15, 8)→ (15, 7)→ (15, 6)→ (15, 5)→ (15, 4)→ (15, 3)→ (15, 2)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (9, 0)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (30, 13)→ (30, 12)→ (30, 11)→ (29, 11)→ (28, 11)→ (27, 11)→ (26, 11)→ (25, 11)→ (24, 11)→ (23, 11)→ (22, 11)→ (21, 11)→ (20, 11)→ (19, 11)→ (18, 11)→ (17, 11)→ (16, 11)→ (15, 11)→ (14, 11)→ (13, 11)→ (12, 11)→ (11, 11)→ (11, 12)→ (11, 11)→ (10, 11)→ (9, 11)→ (8, 11)→ (7, 11)→ (6, 11)→ (5, 11)→ (4, 11)→ (4, 12)→ (4, 13)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→ (6, 17)→ (7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (13, 14)→ (12, 14)→ (11, 14)→ (10, 14)→ (10, 13)→ (10, 14)→ (9, 14)→ (8, 14)→ (7, 14)→ (6, 14)→ (5, 14)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→ (6, 17)→ (7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (14, 15)→ (14, 16)→ (14, 17)→ (14, 16)→ (14, 15)→ (14, 14)→ (13, 14)→ (12, 14)→ (11, 14)→ (10, 14)→ (9, 14)→ (9, 15)→ (9, 14)→ (8, 14)→ (7, 14)→ (6, 14)→ (5, 14)→ (4, 14)→ (4, 15)→ (4, 16)→ (4, 17)→ (5, 17)→ (6, 17)→ (7, 17)→ (7, 18)→ (7, 19)→ (7, 20)→ (7, 21)→ (7, 20)→ (8, 20)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (29, 17)→ (28, 17)→ (27, 17)→ (26, 17)→ (25, 17)→ (24, 17)→ (23, 17)→ (22, 17)→ (21, 17)→ (20, 17)→ (19, 17)→ (18, 17)→ (17, 17)→ (16, 17)→ (15, 17)→ (14, 17)→ (13, 17)→ (12, 17)→ (12, 16)→ (12, 17)→ (11, 17)→ (10, 17)→ (9, 17)→ (9, 18)→ (9, 19)→ (9, 20)→ (9, 21)→ (9, 20)→ (10, 20)→ (11, 20)→ (12, 20)→ (13, 20)→ (14, 20)→ (15, 20)→ (16, 20)→ (17, 20)→ (18, 20)→ (19, 20)→ (20, 20)→ (21, 20)

图 4-7 AGV 的具体移动路径

在目标函数最优的情况下，每一个拣选工位的工作量分别为：

表 4-2 拣选工位的工作量

工位节点坐标	工作量
（7，0）、（8，0）、（9，0）	22
（19，0）、（20，0）、（21，0）	30
（31，6）、（31，7）、（31，8）	22
（31，14）、（31，15）、（31，16）	25
（7，21）、（8，21）、（9，21）	22
（19，21）、（20，21）、（21，21）	25

五、 问题三：基于交通管制法和优先级规划法的无人仓多 AGV 防碰撞冲突的路径规划问题

5.1 问题三分析

在问题一和问题二的基础上，进一步考虑搬运机器人的碰撞和拥堵问题。当仓库内同时有多个 AGV 在执行任务时，不可避免有些 AGV 在某个路径节点上相遇。特别地，如果两个 AGV 在一条货架窄巷道上相遇，那么需要其中一个 AGV 避让。

根据图 2-6 的仓库节点连通图，我们确定路径冲突有以下三种类型：垂直冲突、相向冲突、追尾冲突。

1. 垂直冲突

当两辆 AGV 从垂直方向同时经过同一个十字栅格时，AGV 将会发生垂直冲突，有可能会导致某一辆 AGV 出现侧翻的情况。如图 5-1 所示。

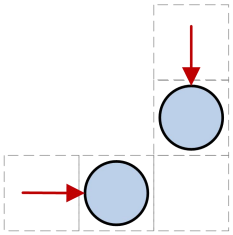


图 5-1 垂直冲突示意图

2. 相向冲突

当两辆 AGV 在一条直线路径上相向而行的时候，将会在两辆 AGV 之间的某一个栅格发生相向冲突。其中相向冲突又有一种特殊情形，即窄路相向冲突，如图 5-2 所示

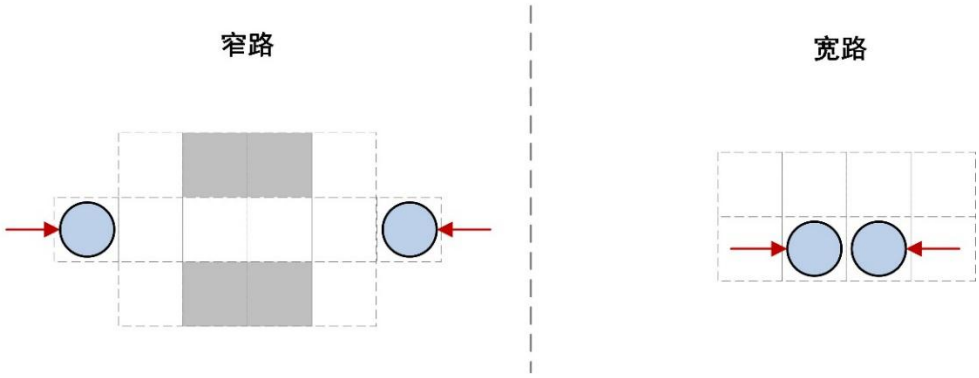


图 5-2 窄路和宽路冲突示意图

3. 追尾冲突

当两辆同向行驶的 AGV 中的前面一辆发生故障时，后面一辆 AGV 将对前面一辆 AGV 发生追尾冲突。另外如果有空闲的 AGV 停留在一辆行驶中的 AGV 的路径的前方时，也会发生追尾冲突。如图 5-3 所示。

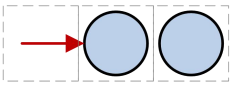


图 5-3 追尾冲突示意图

为避免出现以上三种冲突出现，本题我们考虑交通管制法和优先级规划法两种方式解决。同时 AGV 投放太多的话，出库效率不但不会改善，反而增加 AGV 拥堵的可能性，为此我们选择分别**以 12-19 个搬运机器人 AGV** 来进行问题的模拟求解，其中不同数量搬运机器人情况下对机器人的选择随机生成。为避免实验的偶然性，我们分别以 12-19 个搬运机器人 AGV 来进行问题的模拟求解各 100 次。

5.2 变量和符号说明

表 5-1 问题三变量和符号说明列表

变量或符号	说明
$\boldsymbol{PR_I}$	机器人 i 的优先级，对应机器人下一次到达的拣选工位节点的队列长度，队列越长优先级越低，队列越短，优先级越高；
$\boldsymbol{G_{I,T}}$	机器人 i 在 t 时刻所处的位置节点；
$\boldsymbol{(G_{I,T-1},G_{I,T})}$	机器人 i 在 t-1 时刻所处的位置节点前往 t 时刻所处的位置节点的方向向量；
Z	窄路对应的节点集合；

5.3 问题三模型建立

5.3.1 交通管制法

在交通管制法中，机器人通过和相邻机器人互相协调而实现避碰和避障。在交通管制法中先独立规划各个机器人的运动路径，然后在运动过程中通过改变机器人的运动速度或方向实现避碰和避障。对于问题三提到的三种冲突类型，都可以通过交通管制法实现机器人之间的避碰和避障。

垂直冲突通过停止等待让行进行避让，即低优先级的机器人直接减速为零，停止让高优先级的机器人先过去，然后低优先级的机器人再重新自行启动；相向冲突可以通过机器人在运动方向的垂直方向移动进行避让，即低优先级的机器人在冲突前若干单位处向运动方向的垂直方向移动若干单位后返回，若冲突类型为窄路相向冲突，则低优先级的机器人在进去窄路前向运动方向的垂直方向移动若干单位后返回；追尾冲突也可以通过减速停止进行避障，即低优先级的机器人直接减速为零，停止让高优先级的机器人先过去，然后低优先级的机器人再重新自行启动。交通管制法在多机器人运动中实现比较简单，也易于管理。

遍历所有机器人在 t 时刻所处的位置节点，若存在两个机器人在同一时刻的位置节点重合，则有如下判别：

$$(G_{i,t-1},G_{i,t})\cdot(G_{j,t-1},G_{j,t})>0\rightarrow \textit{追尾冲突} \tag{5-1}$$

$$(G_{i,t-1},G_{i,t})\cdot(G_{j,t-1},G_{j,t})=0\rightarrow \textit{垂直冲突} \tag{5-2}$$

$$(G_{i,t-1},G_{i,t})\cdot(G_{j,t-1},G_{j,t})<0\rightarrow \textit{相向冲突} \tag{5-3}$$

其中在相向冲突中，若$G_{j,t}=G_{i,t}\in Z$，则认为本次冲突为窄路相向冲突。

对于各种类型的冲突，有图 5-4 的解决方案：

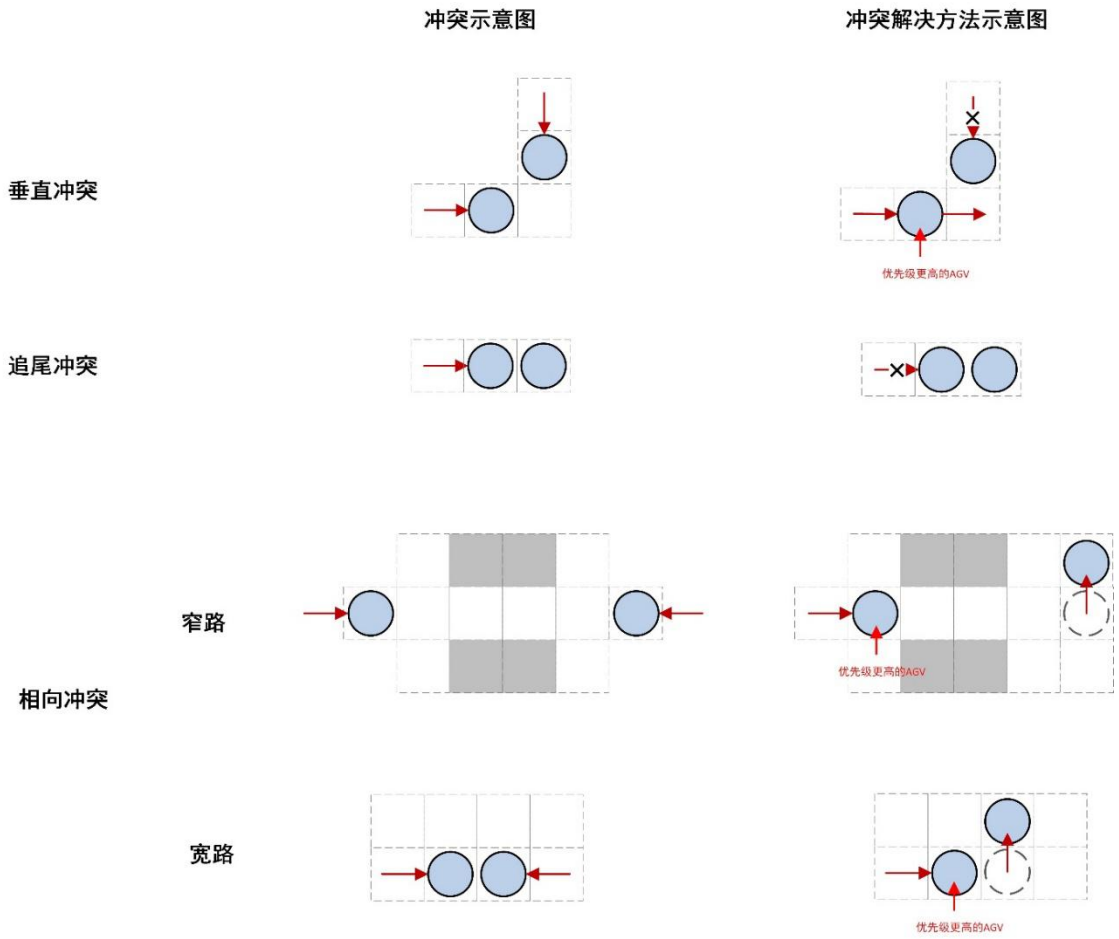


图 5-4 各种冲突类型及其解决方法

5.3.2 优先级规划法

优先级规划法是多机器人路径规划中非常重要的一个方法，在仓储物流中应用也比较广泛。该方法通过为每一个机器人设置一个唯一的优先级，然后根据机器人的优先级从高到低对机器人进行路径规划，并且优先级低的机器人所规划的路径必须避开优先级高的机器人。在此过程中高优先级的机器人可以优先选择路径，而低优先级的机器人必须要根据高优先级的机器人已经规划好的路径来调整自己的路径，从而实现机器人之间的避碰和避障。

另外，为避免优先级规划法中可能会出现的路径死锁现象，对算法作了如下规定：在对高优先级的机器人进行路径规划时，应该避开未进行路径规划的低优先级的机器人的工作节点及其连通节点，防止低优先级的机器人发生路径死锁。另外机器人在运动过程中必须服从低优先级的机器人避让高优先级的机器人的原则。

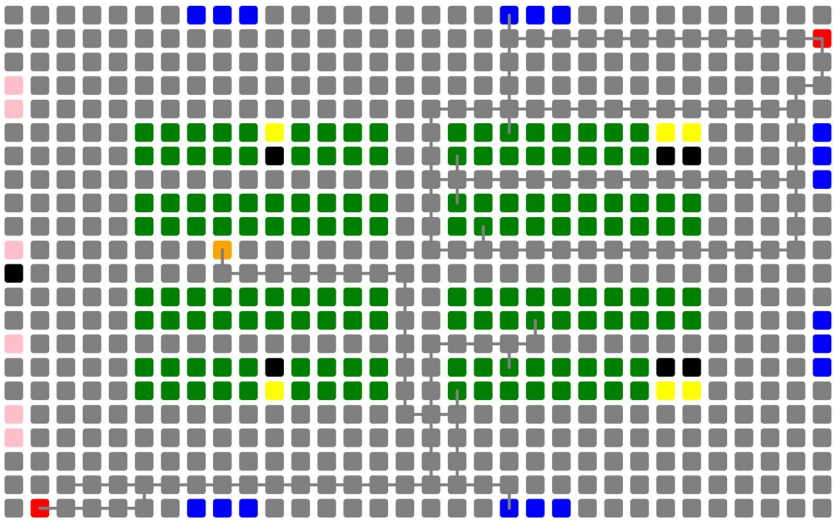
5.4 问题三的求解

根据问题二的模型与算法，通过 Python 语言编程可以得到，在原先的约束条件之下，**AGV 之间的碰撞次数为 103 次**。

5.4.1 基于交通管制法的求解

本题基于交通管制法，最终计算结果为：所有搬运机器人的总路径为 7643。

如图 5-5，该图为在所有搬运机器人总路径最短且相互之间不会碰撞的情况下，其中一个 AGV 在仓库的行动轨迹，图 5-6 为该 AGV 的具体移动的路径坐标。



图表 5-5 交通管制下的一个 AGV 的运动轨迹

(8, 11)→ (8, 10)→ (9, 10)→ (10, 10)→ (11, 10)→ (12, 10)→ (13, 10)→ (14, 10)→ (15, 10)→ (15, 9)→ (15, 8)→ (15, 7)→ (15, 6)→ (15, 5)→ (15, 4)→ (16, 4)→ (17, 4)→ (17, 5)→ (17, 4)→ (17, 3)→ (17, 2)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (5, 0)→ (5, 1)→ (6, 1)→ (7, 1)→ (8, 1)→ (9, 1)→ (10, 1)→ (11, 1)→ (12, 1)→ (13, 1)→ (14, 1)→ (15, 1)→ (16, 1)→ (16, 2)→ (16, 3)→ (16, 4)→ (16, 5)→ (16, 6)→ (16, 7)→ (17, 7)→ (18, 7)→ (19, 7)→ (20, 7)→ (20, 8)→ (20, 7)→ (19, 7)→ (18, 7)→ (17, 7)→ (16, 7)→ (16, 6)→ (16, 5)→ (16, 4)→ (16, 3)→ (16, 2)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (5, 0)→ (5, 1)→ (6, 1)→ (7, 1)→ (8, 1)→ (9, 1)→ (10, 1)→ (11, 1)→ (12, 1)→ (13, 1)→ (14, 1)→ (15, 1)→ (16, 1)→ (16, 2)→ (16, 3)→ (16, 4)→ (16, 5)→ (16, 6)→ (16, 7)→ (16, 8)→ (16, 9)→ (16, 10)→ (17, 10)→ (18, 10)→ (19, 10)→ (19, 9)→ (19, 10)→ (18, 10)→ (17, 10)→ (16, 10)→ (16, 9)→ (16, 8)→ (16, 7)→ (16, 6)→ (16, 5)→ (16, 4)→ (16, 3)→ (16, 2)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (30, 13)→ (30, 12)→ (30, 11)→ (29, 11)→ (28, 11)→ (27, 11)→ (26, 11)→ (25, 11)→ (24, 11)→ (23, 11)→ (22, 11)→ (21, 11)→ (20, 11)→ (19, 11)→ (18, 11)→ (18, 12)→ (18, 11)→ (17, 11)→ (16, 11)→ (16, 12)→ (16, 13)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (17, 13)→ (17, 14)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (17, 15)→ (17, 14)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (30, 19)→ (30, 18)→ (30, 17)→ (29, 17)→ (28, 17)→ (27, 17)→ (26, 17)→ (25, 17)→ (24, 17)→ (23, 17)→ (22, 17)→ (21, 17)→ (20, 17)→ (19, 17)→ (19, 16)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)

图 5-6 交通管制下的一个 AGV 移动的坐标

在目标函数最优的情况下，**所有 AGV 的碰撞次数为 18 次**，相较于在没有交通管制的情况下**减少了 85 次**。因此，我们可以得出交通管制法有效减少小车之间的碰撞次数。

5.4.2 基于优先级规划法的求解

本题基于优先级规划法，最终计算结果为：所有搬运机器人的总路径为 7718。

如图 5-7，该图为在所有搬运机器人总路径最短且相互之间不会碰撞的情况下，其中一个 AGV 在仓库的行动轨迹，图 5-8 为该 AGV 的具体移动的路径坐标。

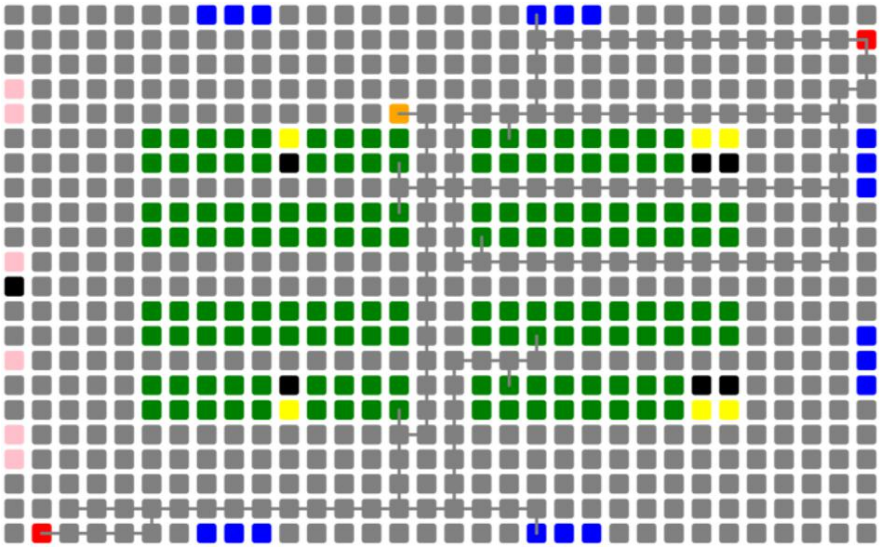


图 5-7 基于优先级规划法的一个 AGV 运动轨迹

(14, 17)→ (15, 17)→ (15, 16)→ (15, 15)→ (15, 14)→ (15, 13)→ (15, 12)→ (15, 11)→ (15, 10)→ (15, 9)→ (15, 8)→ (15, 7)→ (15, 6)→ (15, 5)→ (15, 4)→ (14, 4)→ (14, 5)→ (14, 4)→ (14, 3)→ (14, 2)→ (14, 1)→ (15, 1)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (5, 0)→ (5, 1)→ (6, 1)→ (7, 1)→ (8, 1)→ (9, 1)→ (10, 1)→ (11, 1)→ (12, 1)→ (13, 1)→ (14, 1)→ (15, 1)→ (16, 1)→ (16, 2)→ (16, 3)→ (16, 4)→ (16, 5)→ (16, 6)→ (16, 7)→ (17, 7)→ (18, 7)→ (19, 7)→ (19, 8)→ (19, 7)→ (18, 7)→ (17, 7)→ (16, 7)→ (16, 6)→ (16, 5)→ (16, 4)→ (16, 3)→ (16, 2)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (1, 0)→ (2, 0)→ (3, 0)→ (4, 0)→ (5, 0)→ (5, 1)→ (6, 1)→ (7, 1)→ (8, 1)→ (9, 1)→ (10, 1)→ (11, 1)→ (12, 1)→ (13, 1)→ (14, 1)→ (15, 1)→ (16, 1)→ (16, 2)→ (16, 3)→ (16, 4)→ (16, 5)→ (16, 6)→ (16, 7)→ (16, 8)→ (16, 9)→ (16, 10)→ (17, 10)→ (18, 10)→ (18, 9)→ (18, 10)→ (17, 10)→ (16, 10)→ (16, 9)→ (16, 8)→ (16, 7)→ (16, 6)→ (16, 5)→ (16, 4)→ (16, 3)→ (16, 2)→ (16, 1)→ (17, 1)→ (18, 1)→ (19, 1)→ (19, 0)→ (19, 1)→ (18, 1)→ (17, 1)→ (16, 1)→ (15, 1)→ (14, 1)→ (13, 1)→ (12, 1)→ (11, 1)→ (10, 1)→ (9, 1)→ (8, 1)→ (7, 1)→ (6, 1)→ (5, 1)→ (4, 1)→ (3, 1)→ (2, 1)→ (1, 1)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (30, 13)→ (30, 12)→ (30, 11)→ (29, 11)→ (28, 11)→ (27, 11)→ (26, 11)→ (25, 11)→ (24, 11)→ (23, 11)→ (22, 11)→ (21, 11)→ (20, 11)→ (19, 11)→ (18, 11)→ (17, 11)→ (17, 12)→ (17, 11)→ (16, 11)→ (16, 12)→ (16, 13)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (14, 13)→ (14, 14)→ (15, 14)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (30, 16)→ (30, 15)→ (30, 14)→ (29, 14)→ (28, 14)→ (27, 14)→ (26, 14)→ (25, 14)→ (24, 14)→ (23, 14)→ (22, 14)→ (21, 14)→ (20, 14)→ (19, 14)→ (18, 14)→ (17, 14)→ (16, 14)→ (15, 14)→ (14, 14)→ (14, 15)→ (14, 14)→ (15, 14)→ (16, 14)→ (16, 15)→ (16, 16)→ (16, 17)→ (17, 17)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)→ (31, 20)→ (31, 19)→ (31, 18)→ (30, 18)→ (30, 17)→ (29, 17)→ (28, 17)→ (27, 17)→ (26, 17)→ (25, 17)→ (24, 17)→ (23, 17)→ (22, 17)→ (21, 17)→ (20, 17)→ (19, 17)→ (18, 17)→ (18, 16)→ (18, 17)→ (19, 17)→ (19, 18)→ (19, 19)→ (19, 20)→ (19, 21)→ (19, 20)→ (20, 20)→ (21, 20)→ (22, 20)→ (23, 20)→ (24, 20)→ (25, 20)→ (26, 20)→ (27, 20)→ (28, 20)→ (29, 20)→ (30, 20)

图 5-8 交通管制下的一个 AGV 移动的坐标

在目标函数最优的情况下，**所有 AGV 的碰撞次数为 8 次**，相较于在没有交通管制的情况下**减少了 95 次**。因此，我们可得出优先级规划法有效减少小车之间的碰撞次数。

5.4.3 AGV 数量不同情况下的求解

根据上述的模型与算法，本题随机选择 12-19 个 AGV，在 AGV 数量不同的情况下各进行 100 次实验，将实验结果汇总在下表中。

表 5-2 AGV 数量不同情况下的试验结果

AGV 数量	*最短总路径*	*补货总量*	*碰撞次数*
12	7756	314	4
13	7743	322	4
14	7617	309	5
15	7587	295	5
16	7604	287	8
17	7662	304	13
18	7685	298	15
19	7718	328	18

如表 5-2，我们可以得到，当 AGV 数量为 15 的时候，我们可以得到所有搬运机器人的总路径为 7587，补货总量为 295，以及碰撞次数为 5。通过上述的比较，可以得出，当 AGV 数量进行适当的删减之后，可以有效避免 AGV 碰撞情况的出现。

六、 模型评价

6.1 模型优点

（1）针对无人仓管理中的多 AGV 路线规划问题中，本文提出了一个基于改进遗传算法下的多 AGV 路径规划和调度模型。通过分别融入“储位节点搬运判别要素”的基因编码，采用二进制编码和整数编码的基因序列，进行遗传算法的迭代运算。

（2）针对多 AGV 的碰撞和拥堵问题，本文提出了基于交通管制法和优先级规划法的防冲突和拥堵模型，针对不同的冲突和优先级提出不同的解决方案；同时在不同数量的搬运机器人的情况下，根据问题二改进的遗传算法进行进一步求解，以得到最佳搬运机器人的运行数量，有效防止了搬运机器人的拥堵现象。

（3）本文在研究问题是，综合比较了多种启发式算法的运行结果。通过综合运用对比这些算法，能够有效的提高运行的准确性。同时，根据算法的有效性快速选择最合适的算法。

（4）在问题二的两阶段目标规划模型的构件中，我们考虑使用了 banach 空间中的范数相关知识，分别通过 1 范数和∞范数建立有效规划模型。

6.2 模型改进

本文最主要的目标函数是最小化所有搬运机器人的总路径，在这个过程中，对订单满足只是一个约束条件。根据题意，若没有满足订单需求的，均通过补货工位将货物通过传送带运出，在某程度上简化了实际运输机器人的工作。因此，在后续的研究中，还应该考虑如果一次搬运无法满足订单情况时，搬运机器人需要如何规划如何满足后续商品，并实现成本最小化的搬运过程。除此之外，本文假设一个托盘上的商品只有一种，商品的大小一定固定的，在实际情况下，托盘上的商品不一定一致，且大小也不一定相同。因此，后续研究中，针对托盘上不同尺寸和类型的商品，如何统筹搬运，会使得搬运成本最低，效率最高也是未来深入研究的方向之一。

参考文献

[1]纪寿文,李克强.智能化的物流搬运机器人-AGV[J].中国物流与采购,2004(02):56-57.

[2]Vis I F A. Survey of research in the design and control of automated guided vehicle systems[J]. European Journal of Operational Research, 2006, 170(3): 677-709.

[3]代东阁,黄彪,刘雄,周仁宇,李小虎,蒋顺鹏.搬运机器人发展现状及展望[J].人工智能与机器人研究,2021,10(2):144-153

[4]宋敏, 吴豪杰. 浅析燃料电池驱动农业搬运机器人[J]. 南方农机, 2021, 52(3): 52-54+70.

[5]程麒文, 邱白晶. 箱式农作物搬运机器人的设计[J]. 中国农机化学报, 2019, 40(11): 176-180.

[6]曹东江,王强,王宁.双臂搬运机器人结构设计与动态仿真分析[J].测控技术,2021,40(02):32-36.DOI:10.19708/j.ckjs.2020.12.328.

[7]王大治,石刚,李永庆.动力电芯上线搬运机器人的设计[J].制造业自动化,2020,42(09):62-64+88.

[8]Zhu Q, Hu J, Cai W, et al. A new robot navigation algorithm for dynamic unknown environments based on dynamic path re-computation and an improved scout ant algorithm[J]. Applied Soft Computing, 2011, 11(8): 4667-4676.

[9] Małopolski W. A sustainable and conflict-free operation of AGVs in a square topology[J]. Computers & Industrial Engineering, 2018, 126: 472-481.

[10]Kim C W, Tanchoco J M A. Conflict-free shortest-time bidirectional AGV routeing[J]. The International Journal of Production Research, 1991, 29(12): 2377-2391.

[11]闫妍, 陶美春. 基于 A~* 算法的搬运机器人路径规划——以菜鸟智能仓为例[J]. 物流科技, 2020.

[12]王辉,王景良,朱龙彪,邵小江,王恒.基于改进蚁群算法的泊车系统路径规划[J].控制工程,2018,25(02):253-258.DOI:10.14107/j.cnki.kzgc.151237.

[13]杨煜俊,陈业.求解柔性机器人车间调度问题的混合蚁群算法[J].计算机工程与应用,2018,54(13):160-167.

[14] Mousavi M, Yap H J, Musa S N, et al. Multi-objective AGV scheduling in an FMS using a hybrid of genetic algorithm and particle swarm optimization[J]. PloS one, 2017, 12(3): e0169817.

[15]李孟思. 面向无人化仓储的多机器人调度与优化研究[D].重庆邮电大学,2020.DOI:10.27675/d.cnki.gcydx.2020.000498.

[16]龙传泽,杨煜俊.基于遗传算法的柔性机器人制造单元调度问题研究[J].组合机床与自动化加工技术,2015(11):141-144+148.DOI:10.13462/j.cnki.mmtamt.2015.11.039.

[17]王豪,赵学军,袁修久.基于改进自适应遗传算法的机器人路径规划[J/OL].电光与控制:1-7[2022-04-15].http://kns.cnki.net/kcms/detail/41.1227.TN.20220105.1448.015.html