队伍编号	MC2200562
题号	B

复杂无人仓搬运机器人协同调度机制与方法研究

摘 要

本文对复杂无人仓搬运机器人（AGV）优化调度问题进行研究，梳理 AGV 优化调度全流程，在全面把握无人仓运行相关特征的基础上，综合考虑商品托盘与 AGV 等调度实体，从“机制”与“方法”两个视角研究 AGV 优化调度问题，设计多主体协同的 AGV 调度机制，在此基础上建立多约束受限、多目标一体、多阶段决策的“商品托盘-AGV”协同调度模型，设计求解算法；基于 AGV 运行多维时空调度结果，搭建 AGV “冲突探测-解脱”模型，设计改进的启发式算法，对 AGV 运行调度结果进行二次优化，减少 AGV 运行冲突，以上研究解决了如下问题：

问题一：立足于题目中的相关描述，分析 AGV 优化调度流程，全面把握“商品托盘-AGV”协作调度的显著特点，明确 AGV 优化调度与商品托盘资源调度之间的相关性。在此基础上，构建“商品托盘-AGV”多主体协同调度机制：首先以商品托盘为主体，建立托盘行走路径总长度的关键指标，并以此作为模型的优化目标，考虑托盘中商品数量信息、回库托盘储位限制等约束条件，构建商品托盘指派调度模型，生成商品托盘调度任务集；然后以 AGV 为主体，将上一阶段托盘调度任务集作为输入，以最小机器人空闲时间、机器人行走路径总长度为目标，考虑 AGV 执行任务时间间隔限制条件，构建 AGV 优化调度模型。优化调度机制不仅能完成 AGV 调度任务，还可同时实现无人仓储位管理、拣选工位管理以及托盘回收工位管理。针对模型约束众多、变量空间大、维数高等特点，设计自适应大邻域搜索与模拟退火（Neighborhood Search-Simulated Annealing, NS-SA）相结合的求解算法，通过算例分析可知，算法可在 1200 次迭代后收敛，搬运机器人行走路径总长度为 6273 个单位长度，与初始可行解相比，优化了 34.7%，最大任务完成时间在第 347 个单位时间，与初始可行解相比，优化了 33.3%，优化效果显著。

问题二：依托于问题一中提出的 AGV 调度机制，在商品托盘调度模型中，加入拣选工位分区变量，以拣选工位区域工作时间差异最小与托盘行走路径总长度最小作为优化目标，以优化订单数目作为仓库动态分区的触发条件，采用 NS-SA 算法求解模型，得到 AGV 运行多维时空调度结果，从而动态划分仓库区域。

问题三，在 AGV 运行多维时空调度结果的基础上，定义 AGV 运行冲突类型，采用网格冲突探测法探测 AGV 间潜在冲突；同时，采用相关时空数据结构（Relational Time-Space Data Structure，RTSDS）提升冲突探测效率，减少计算机内存占用。考虑 AGV 优化调度问题的复杂性，设计基于动态分组的变种群规模合作协同进化算法（Adaptive Population Size Cooperative Co-evolution Algorithm based on Dynamic Grouping，APCCDG），采用将相互影响的 AGV 放入同一小组进行优化的策略，降低问题维度。将 APCCDG 与基于随机分组的合作协同进化算法（Cooperative Co-evolution Algorithm based on Random Grouping，CCRG）、遗传算法（Genetic Algorithm，GA）进行对比，实验结果证明，APCCDG 具有更快的收敛速度，可以得到无冲突的 AGV 运行路径，且解脱成本优化了约 6.3%。

关键词：AGV 协同调度机制；邻域搜索；相关时空数据结构；APCCDG；自适应遗传算子

目　　录

1 问题重述

1.1 问题背景

随着互联网时代的到来，激增的商品物流订单对仓库管理提出了更高的要求。智能化的无人仓逐渐取代了传统仓库。特别是在商品搬运过程中搬运机器人及其相互间的协同运行机制和方法成为无人仓运行中不可或缺的一部分。搬运机器人(Automated Guided Vehicle，AGV)又称智能移动机器人，本问题背景是基于无人仓内的订单货物商品，通过 AGV 搬运货物，将传统的“人到货”拣选模式转变为“货到人”模式[1]。在全局统筹优化 AGV 调度的问题上，围绕多主体协同下的最优路径规划问题、任务均衡区域划分问题和避障及拥堵问题，重点研究复杂无人仓内 AGV 的协同运行机制和方法，探寻避免碰撞和拥堵的优化方法及求解算法，使 AGV 能够高效地执行商品订单任务，大大提高无人仓无人管理的效率和质量，并对整个系统的运行效率个经济效益产生深远影响[2]。

1.2 问题描述

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。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 AGV 的调度问题是其中的核心问题。其中，多 AGV 协同路径规划技术是实现多 AGV 在复杂仓库环境中协同完成搬运任务，最大化订单出库率的关键技术[3]。

假设仓库的地图可以抽象成由工位、储位等节点构成的图，再按照 AGV 能到达的节点添加图的边，搬运机器人 AGV 一次只能搬运一个托盘（带有多种商品），能执行从一个地图节点 n_a 移动到 n_b 的路径指令，并且其中每一步只能移动到有边相连的地图节点，不能斜着移动。每个在储位的托盘上叠放着多种商品，无人仓流程是根据给定的一段时间内订单数据流，结合当前库存情况，统筹安排搬运机器人从储位搬运有需求商品的托盘到附近的拣选工位（即出库任务），拣选完成后需安排搬运工位处的非空托盘到空储位（即回库任务），或者安排搬运工位处的空托盘到托盘回收处（即回收任务）等。本题只考虑这三种主要任务场景，即出库、回库、回收任务。

首先，对于出库任务，搬运机器人 AGV 把一个托盘搬运到拣选工位。但是对于同一个工位来说，同时能容纳放置的托盘数目是有限的。假设每个拣选工位有 b 个停靠位，也就是能同时最多分派 b 个出库任务到同一个拣选工位，直到执行回库或者回收任务，有空停靠位后才能容纳新的出库托盘。其次，出库任务完成后，搬运机器人处于空闲状态，可以被安排执行下一个任务，而不需要在停靠位等候着。假设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。这里假设停留时间固定为 t_0，也就是说，无论需要拣选多少商品，都简化为停留时间 t_0 后，该托盘可以被执行后续的回库和回收任务。

简而言之，无人仓的主要运行场景就是安排搬运机器人 AGV 执行如下三种任务：

（1）出库：AGV 搬运载有商品的托盘到空闲拣选工位；

（2）回库：AGV 搬运拣选完成的托盘从工位回到仓库内空储位；

（3）回收：AGV 搬运拣选完成的空托盘从工位回到托盘回收处；

在以上几个约束条件下，主要解决以下三类场景中的问题：**多约束受限、多主体协同、多资源耦合的两阶段优化调度模型问题、任务均衡区域划分问题问题、AGV 避免碰撞和拥堵问题。**本文技术路线图如图 1-1 所示。

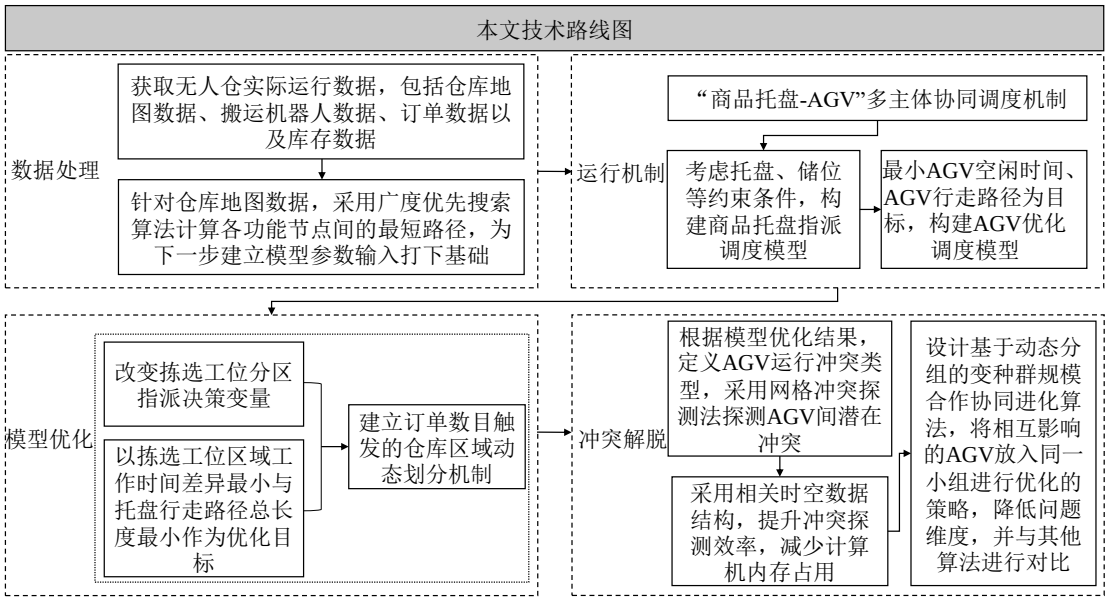


图 1-1 技术路线图

1.3 待解决问题

根据问题描述以及题目要求，本文以附件数据为基础，着重解决以下三个场景问题：

问题一：AGV 统筹调度的最佳策略问题

搬运机器人的统筹路径规划问题是最基本的问题，也是求解后续问题的基础。在模型中首先要解决托盘指派、托盘最优路径规划以及 AGV 指派问题，在既能尽快满足订单需求的前提下，最优 AGV 的路径以实现最佳策略的统筹调度。在求解 AGV 统筹调度的最佳策略时，本文考虑托盘调度问题与搬运机器人调度问题的耦合性，建立多约束受限、多主体协同、多资源耦合的两阶段优化调度模型，实现以最大化运行效率、最小化行走路径为目标的多目标集成导向的搬运机器人优化调度；采用自适应大邻域搜索与模拟退火算法求解所建立的两阶段优化模型。

问题二：任务均衡区域划分问题

依托于问题一中提出的 AGV 调度机制，在商品托盘调度模型中，加入拣选工位分区变量，以拣选工位区域工作时间差异最小与托盘行走路径总长度最小作为优化目标，对 AGV 进行优化调度的同时，对储位进行优化分区。采用 NS-SA 算法求解模型，得到 AGV 运行多维时空调度结果与分区结果。

问题三：多 AGV 避免碰撞和拥堵问题

基于 AGV 运行多维时空调度结果，定义了顶点冲突、对向冲突、对向死锁及死锁四种冲突场景，采用网格冲突探测法探测 AGV 间潜在冲突；同时，采用相关时空数据结构提升冲突探测效率，减少计算机内存占用。考虑 AGV 优化调度问题的复杂性，设计基于动态分组的变种群规模合作协同进化算法，采用将相互影响的 AGV 放入同一小组进行优化的策略，降低问题维度。将 APCCDG 与基于随机分组的合作协同进化算法、遗传算法进行对比。

2　模型假设与数据预处理

2.1　模型假设

为了便于问题研究，对题目中某些条件进行合理的假设：

（1）假设拣选工位节点可按优化分配结果拣选商品；

（2）假设 AGV 通过 1 个网格节点的时间为 1 个单位时间；

（3）假设所有 AGV 运行的速度恒定且相等；

（4）假设b的数目为 3，并且每个拣选工位的拣选工作台相互独立，即 AGV 可单独进出拣选工作台；

（5）假设 AGV 运行路径调度结果，可根据 AGV 途径网格长度得到。

2.2　仓库环境数据预处理

为方便对后续问题进行计算，首先根据附件中的坐标数据建立坐标系并分别计算储位节点到拣选工位节点、储位节点到空托盘回收节点、拣选工位节点到空托盘回收节点的最短路径，进而得到三个最短路径矩阵称为绿-蓝矩阵、绿-红矩阵和蓝-红矩阵，具体形式如表 1—表 3 所示。其中，最短路径计算方法采用广度优先搜索算法(Breadth First Search)，其可根据起始点的一度邻居，不断扩大到“二度邻居节点”、“三度邻居节点”.....直到找到终点节点为止，从而推算出最短路径[5]。具体过程分为如下六个步骤：

（1） 首先根据仓库地图坐标从某一节点出发，根据附件信息找出它的所有可通行邻居节点，并将该点的坐标 ID 及邻居节点的基本特性创建在一个类中;

（2） 为了编程实现最短路径的搜索过程，（1）所述的类中还需包含“当前节点是否被检查过”以及“当前节点前一节点是谁”两个属性；

（3） 选择队列作为数据结构，并先将起始节点的直接可通行邻居节点全部添加到队列中去，依次从队列头取出元素进行比较，若不是终点节点则将该点的直接可通行邻居节点添加到队尾。

（4） 跳过检查过的节点，避免循环重复；

（5） 重复上述四个步骤，直到队列为空，计算出两点之间的最短路径。

表 1　绿-蓝矩阵

	(7，0)	(8，0)	(9，0)	……	（31,14）	（31,15）	（31,16）
（5，5）	7	8	9	……	37	38	39
（6，5）	6	7	8	……	36	37	38
（7，5）	5	6	7	……	35	36	37
……	……	……	……	……	……	……	……
（22，16）	33	32	31	……	13	12	11
（23，16）	34	33	32	……	12	11	10

（24，16）	35	34	33	……	11	10	9

表 2　绿-红矩阵

	(1, 0）	（31,20）
（5，5）	9	43
（6，5）	10	42
（7，5）	11	41
……	……	……
（22，16）	39	15
（23，16）	40	14
（24，16）	41	13

表 3　蓝-红矩阵

	(1, 0）	（31,20）
（7，0）	8	44
（8，0）	9	43
（9，0）	10	42
……	……	……
（19，21）	39	15
（20，21）	40	14
（21，21）	41	13

3　问题一 AGV 统筹调度的最佳策略模型

3.1　两阶段协同调度机制概述

根据问题一中相关描述，AGV 的统筹调度问题，可以被理解为在商品托盘与订单需求满足的前提下，尽可能提高 AGV 的使用效率，减少 AGV 的空闲时间，缩短 AGV 的总行走距离，尽可能快速高效的完成商品托盘与订单需求的匹配任务。在此基础之上，提出 AGV 协同调度机制，即将 AGV 的调度分为两个阶段，在第一阶段，以托盘行走总距离最小为目标，完成商品托盘商品与订单需求的最优匹配，生成托盘调度任务集；在第二阶段，优化调度 AGV 运行，指派 AGV 完成托盘调度任务的同时，优化 AGV 运行时空网络，达到提高机器人运行效率、最小化机器人运行路径总长度的目标，调度机制如下图 3-1 所示。

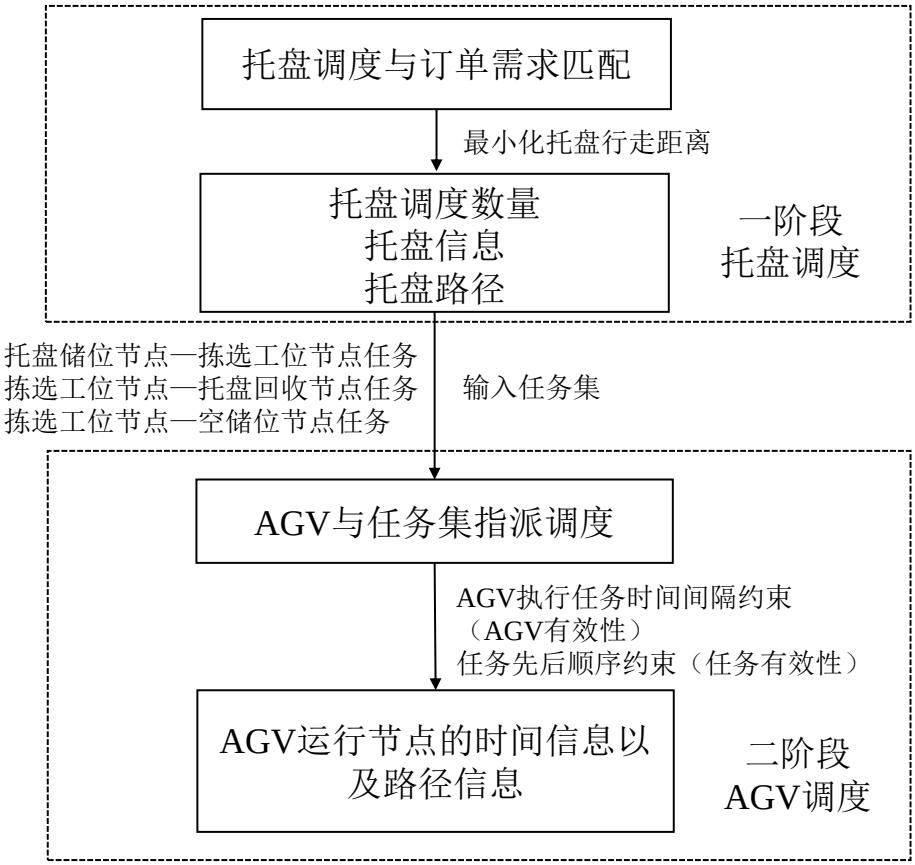


图 3-1 AGV 统筹调度机制图

3.2　一阶段优化调度：托盘指派与优化调度模型

在第一阶段，考虑托盘调度内商品数量限制、托盘出库与回库储位限制，建立以最小化托盘运行总路径长度为目标的托盘优化调度模型，模型建立逻辑如图 3-2 所示。

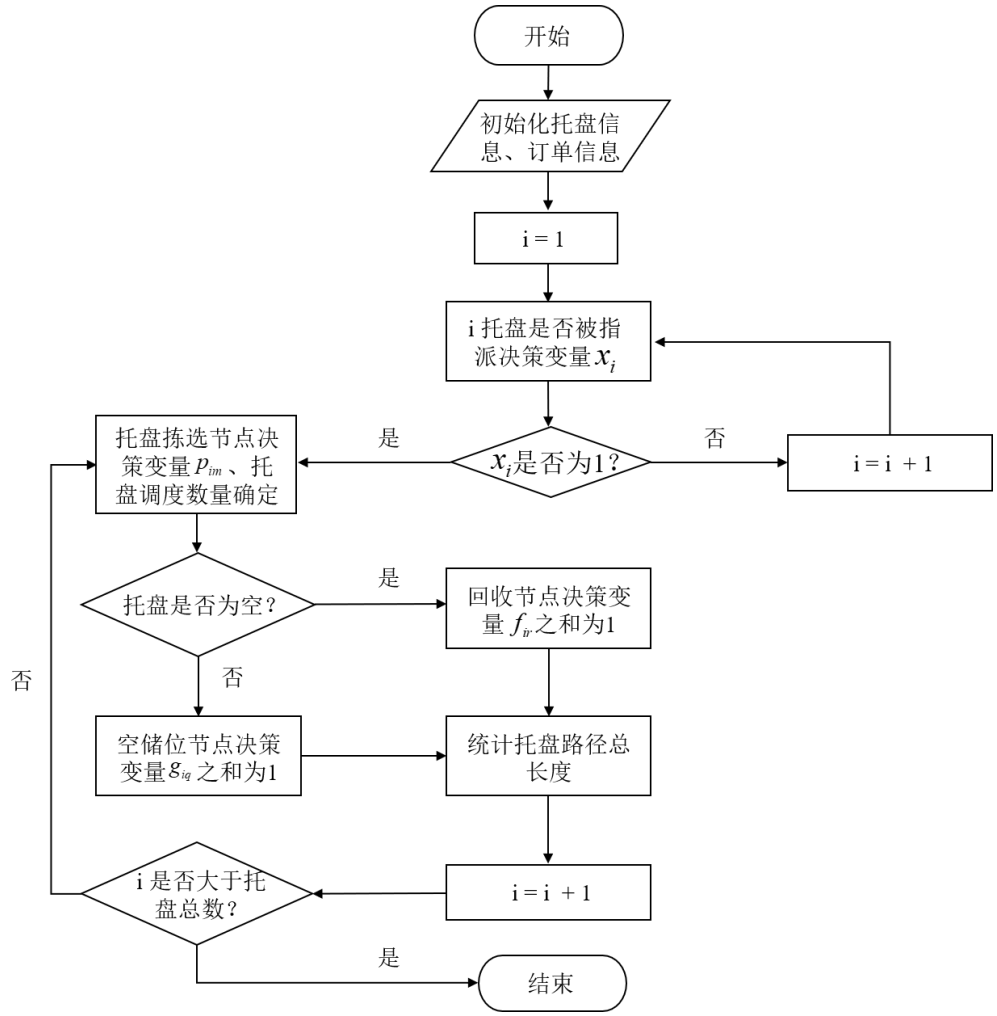


图 3-2 托盘调度模型建立流程

如表 3-1 所示，托盘指派与优化调度模型的符号变量相关定义。

表 3-1 模型的集合，参数，变量

集合	
$\mathcal{H}$	订单商品种类集合
$\mathcal{I}$	商品托盘集合
$\mathcal{Q}$	储位节点集合
$\mathcal{M}$	拣选工作台节点集合
$\mathcal{R}$	空盘回收节点集合
参数	
C_h	第 h 类商品的订单需求，$h \in \mathcal{H}$
Q_i	第 i 个商品托盘的初始储位，$Q_i \in \mathcal{Q}$
O_{ih}	第 i 个商品托盘中拥有第 h 类商品的数量，$i \in \mathcal{I}$，$h \in \mathcal{H}$
J_{iq}	第 i 个商品托盘初始位于第 q 个储位
D_{qm}	第 q 个储位至第 m 个拣选工作台的最短行走距离，$q \in \mathcal{Q}$，$m \in \mathcal{M}$
D_{mq}	第 m 个拣选工作台到至第 q 个储位的最短行走距离，$m \in \mathcal{M}$，$q \in \mathcal{Q}$
D_{mr}	第 m 个拣选工作台到至第 r 个空盘回收位的最短行走距离，$m \in \mathcal{M}$，$r \in \mathcal{R}$

M	无穷大的数值
变量	
x_i	如果商品托盘i被挑选去拣选工位则为1，否则为0，$i \in \mathcal{I}$
y_i	如果商品托盘i的商品全部被挑选则为1，否则为0，$i \in \mathcal{I}$
k_{ih}	商品托盘i在拣选工位应被挑选走的第h类商品的数量，$i \in \mathcal{I}$，$h \in \mathcal{H}$
p_{im}	商品托盘i选择去第m个拣选工作台分拣商品则为1，否则为0，$i \in \mathcal{I}, m \in \mathcal{M}$
g_{iq}	商品托盘i选择回到第q个储位则为1，否则为0，$i \in \mathcal{I}, q \in \mathcal{Q}$
f_{ir}	商品托盘i选择去第r个空盘回收位则为1，否则为0，$i \in \mathcal{I}, r \in \mathcal{R}$
d_{iqm}	商品托盘i从第q个储位去第m个拣选工作台的行走距离，$i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q}$
d_{imq}	商品托盘i第m个拣选工作台回到第q个储位的行走距离，$i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q}$
d_{imr}	商品托盘i从第m个拣选工作台到第r个空盘回收位的行走距离，$i \in \mathcal{I}, m \in \mathcal{M}, r \in \mathcal{R}$
z_q	第q个储位上的商品托盘被指派则为1，否则为0

目标函数：

$$\min \sum_{i \in \mathcal{I}} (d_{iqm} + d_{imq} + d_{imr}) \tag{3.1}$$

模型目标函数见上式，目标函数为最小商品托盘行走总距离。

约束条件：

（1）托盘内商品数量与订单需求匹配约束

$$0 \le k_{ih} \le P_{ih}, \forall i \in \mathcal{I}, h \in \mathcal{H} \tag{3.2}$$

$$\sum_{h \in \mathcal{H}} k_{ih} + M(1 - x_i) > 0, \forall i \in \mathcal{I} \tag{3.3}$$

$$\sum_{h \in \mathcal{H}} k_{ih} - \sum_{h \in \mathcal{H}} P_{ih} \times x_i \le 0, \forall i \in \mathcal{I} \tag{3.4}$$

$$\sum_{i \in \mathcal{I}} k_{ih} = C_h, \forall i \in \mathcal{I}, h \in \mathcal{H} \tag{3.5}$$

式（3.2）限制从商品托盘i中挑选出来的第h类商品数量k_{ih}，应小于该托盘实际拥有的第h类商品数量P_{ih}；式（3.3）限制若商品托盘i被挑选出前往拣选工作台，则从商品托盘i中挑选出来的商品数量总和应大于0；式（3.4）限制若商品托盘i没有被挑选出前往拣选工作台，则从商品托盘i中挑选出来的商品数量总和应等于0；式（3.5）限制从所有商品托盘中挑选出来的第h类商品数量应满足订单中对第h类商品数量的需求C_h。

（2）为指派托盘调度拣选工作台约束

$$\sum_{m\in\mathcal{M}} p_{im} \le 1, \forall i \in \mathcal{I} \tag{3.6}$$

$$\sum_{m\in\mathcal{M}} p_{im} - x_i = 0, \forall i \in \mathcal{I} \tag{3.7}$$

$$d_{iqm} + M(3 - x_i - p_{im} - J_{iq}) \ge D_{qm}, \forall i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q} \tag{3.8}$$

$$d_{iqm} - M(1 - x_i) \le 0, \forall i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q} \tag{3.9}$$

式（3.6）限制若商品托盘 i 至多前往一个拣选工作台；式（3.7）限制若商品托盘 i 被挑选出前往拣选工作台，则其一定会被指派前往一个拣选工作台；式（3.8）限制若商品托盘 i 从储位 q 被挑选出前往拣选工作台，并且商品托盘 i 前往拣选工作台 m，则托盘从储位 q 至拣选工作台 m 的行走路线长度 d_{iqm} 应大于从储位 q 至拣选工作台 m 的最短路径 D_{qm}；式（3.9）限制若商品托盘 i 没有被挑选出前往拣选工作台，则托盘从储位 q 至拣选工作台 m 的行走路线长度 d_{iqm} 应为 0。

（3）若托盘内还有商品，需要为托盘寻找空储位约束

$$\sum_{h\in\mathcal{H}} k_{ih} - \sum_{h\in\mathcal{H}} O_{ih} + M(2 - x_i - y_i) \ge 0, \forall i \in \mathcal{I} \tag{3.10}$$

$$x_i - y_i \ge 0, \forall i \in \mathcal{I} \tag{3.11}$$

$$\sum_{q\in\mathcal{Q}} g_{iq} \le 1, \forall i \in \mathcal{I} \tag{3.12}$$

$$y_i + \sum_{q\in\mathcal{Q}} g_{iq} \le 1, \forall i \in \mathcal{I} \tag{3.13}$$

$$x_i - \sum_{q\in\mathcal{Q}} g_{iq} \ge 0, \forall i \in \mathcal{I} \tag{3.14}$$

$$\sum_{q\in\mathcal{Q}} g_{iq} + \sum_{r\in\mathcal{R}} f_{ir} \le 1, \forall i \in \mathcal{I} \tag{3.15}$$

$$g_{iq} - z_q \le 0, \forall i \in \mathcal{I}, q \in \mathcal{Q} \tag{3.16}$$

$$d_{imq} - M(1 - x_i + y_i) \le 0, \forall i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q} \tag{3.17}$$

$$d_{imq} + M(4 - x_i + y_i - p_{im} - g_{iq} - z_q) \ge D_{mq}, \forall i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q} \tag{3.18}$$

$$z_q + J_{iq} - x_i \le 1, \forall i \in \mathcal{I}, q \in \mathcal{Q} \tag{3.19}$$

式（3.10）限制若商品托盘i中的商品全部被挑选走，则从商品托盘i中挑选出来的商品数量总和应等于商品托盘i中拥有的商品数量总和；式（3.11）限制若商品托盘i没有被挑选出前往拣选工位，则商品托盘i中必须有剩余商品；式（3.12）限制商品托盘i在被拣选完商品后，至多前往一个储位；式（3.13）限制若商品托盘i中的商品未被全部挑选出来，即商品托盘i中被拣选之后还有剩余商品，则为其分配一个储位；式（3.14）限制若商品托盘i没有被挑选出前往拣选工作台，则不需要为其分配储位；式（3.15）限制任意一个托盘被挑选完后只能前往储位或空盘回收位中的一个；式（3.16）限制为商品托盘i在拣选之后分配的储位仅能为空储位；式（3.17）限制若商品托盘i没有被挑选出前往拣选工作台，或商品托盘被挑选空，则托盘从拣选工作台m至空储位q的行走路线长度d_{imq}应为0；式（3.18）限制若商品托盘i被挑选出前往拣选工作台并且商品托盘i前往拣选工作台m，并且商品托盘没有被挑选空，则托盘从拣选工作台m至空储位q的行走路线长度d_{imq}应大于从拣选工作台m至储位q的最短路径D_{mq}；式（3.19）规定若商品托盘i被挑选走，则商品托盘i所在的储位q可被认为是空储位。

（4）若托盘内没有商品，需要为托盘寻找回收节点相关约束：

$$\sum_{r \in \mathcal{R}} f_{ir} \le 1, \forall i \in \mathcal{I} \tag{3.20}$$

$$x_i - \sum_{r \in \mathcal{R}} f_{ir} \ge 0, \forall i \in \mathcal{I} \tag{3.21}$$

$$\sum_{r \in \mathcal{R}} f_{ir} - M \times y_i \le 0, \forall i \in \mathcal{I} \tag{3.22}$$

$$d_{imr} + M(4 - x_i - y_i - p_{im} - f_{ir}) \ge D_{mr}, \forall i \in \mathcal{I}, m \in \mathcal{M}, r \in \mathcal{R} \tag{3.23}$$

$$d_{imr} - M(2 - x_i - y_i) \le 0, \forall i \in \mathcal{I}, m \in \mathcal{M}, r \in \mathcal{R} \tag{3.24}$$

$$d_{iqm} \ge 0, d_{imq} \ge 0, d_{imr} \ge 0 \forall i \in \mathcal{I}, m \in \mathcal{M}, q \in \mathcal{Q}, r \in \mathcal{R} \tag{3.25}$$

式（3.20）限制商品托盘i在被拣选完商品后，至多前往一个空盘回收位；式（3.21）限制若商品托盘i没有被挑选出前往拣选工作台，则不需要为其分配空盘回收位；中被拣选之后还有剩余商品，则为其分配一个储位；式（3.22）限制若商品托盘i中的商品被全部挑选出来，即商品托盘i中被拣选之后没有剩余商品，则为其分配一个空盘回收位；式（3.23）限制若商品托盘i被挑选出前往拣选工作台，并且商品托盘i前往拣选工作台m，并且商品托盘被挑选空，则托盘从拣选工作台m至空盘回收位r的行走路线长度d_{imr}应大于从拣选工作台m至空盘回收位r的最短路径D_{mr}；式（3.24）限制若商品托盘i没有被挑选出前往拣选工作台，或商品托盘没有挑选空，则托盘从拣选工作台m至空盘回收位r的行走路线长度d_{imr}应为0；式（3.25）限制托盘行走路线长度应为非负值。

3.3 二阶段优化调度：搬运机器人运行优化调度

在优化第一阶段得到商品托盘调度结果的基础之上，生成托盘调度任务集$\mathcal{E}$，其中包含有关托盘调度的三种任务，即托盘的出库任务、回库任务以及回收任务。第二阶段的搬运机器人调度问题可被描述如下：任务集$\mathcal{E}$中的每个任务均需要指派一辆搬运机器人去完成，并且同一辆搬运机器人连续完成两项任务须满足一定的时间间隔，在此背景下，优化调度搬运机器人的指派任务与任务时间，尽可能提高搬运机器人的使用效率，减少机器人的行走距离。在任务集中，由于搬运机器人在初期并未处于储位节点，因此加入从机器人的初始位置至需要调度托盘储位节点的行走路径任务。二阶段 AGV 调度模型构建流程如图 3-3 所示

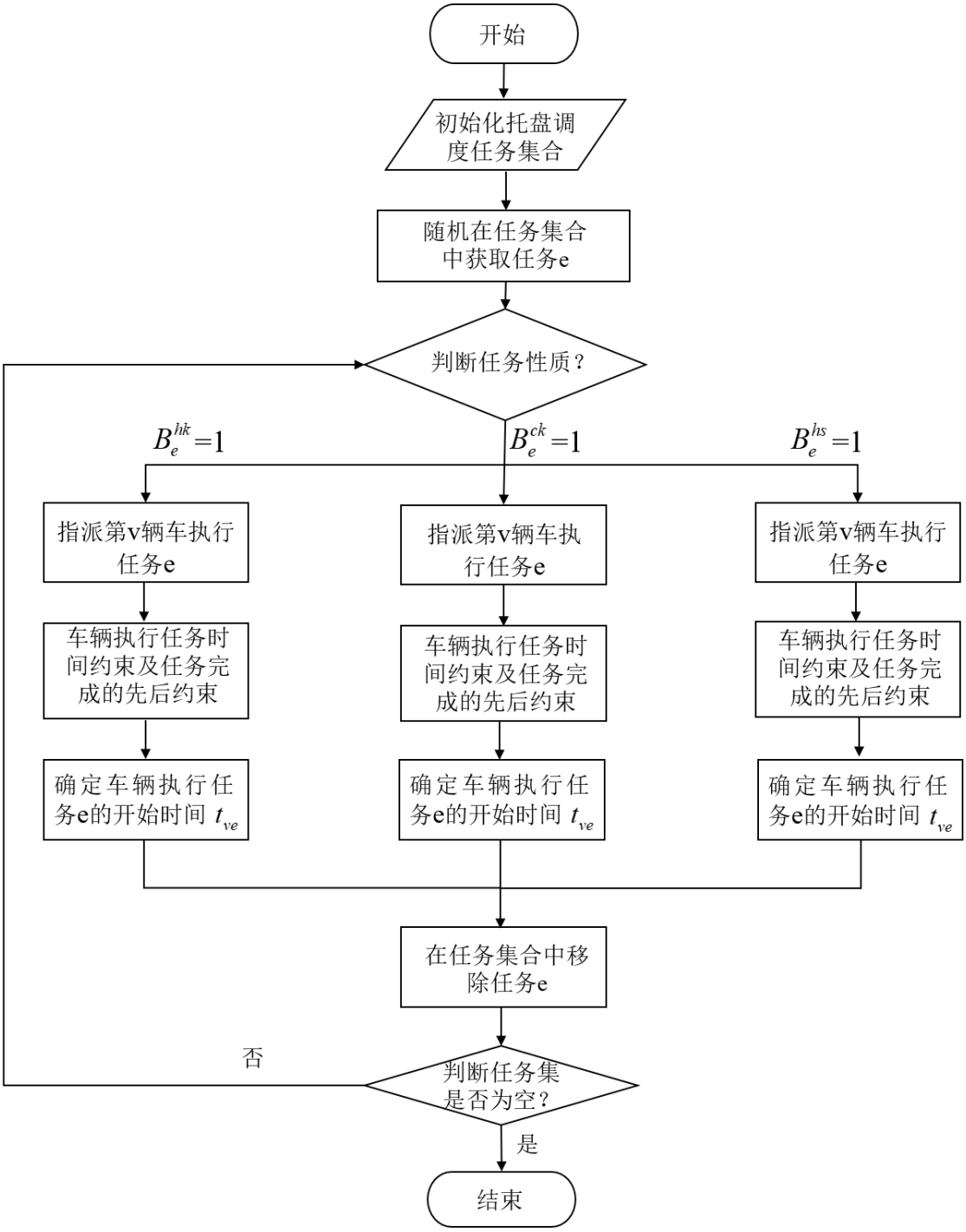


图 3-3 二阶段模型流程图

如表 3-2 所示，给出搬运机器人运行优化调度模型的符号变量相关定义。

表 3-2 模型的集合，参数，变量

集合	
$\mathcal{E}$	托盘调度任务集
$\mathcal{V}$	可被调度搬运机器人集合
参数	
B_e	第 e 项任务的任务性质，若 B_e^{ck}=1，则 e 项任务为出库任务；若 B_e^{hk}=1，则 e 项任务为回库任务；若 B_e^{hs}=1，则 e 项任务为回收任务，$e\in\mathcal{E}$
Num_e	第 e 项任务的 Pallets_ID，$e\in\mathcal{E}$
T_e	托盘在第 e 项任务中消耗时间，$e\in\mathcal{E}$
Chu_e	第 e 项任务的拣选工作台，若 B_e^{ck}=0，则 Chu_e 为 0，$e\in\mathcal{E}$
Jou_e	第 e 项任务的托盘初始储位，$e\in\mathcal{E}$
Hk_e	第 e 项任务的回库储位，若 B_e^{hk}=0，则 Hk_{eq} 为 0，$e\in\mathcal{E}$
Hs_e	第 e 项任务的回收位，若 B_e^{hs}=0，则 Hs_{er} 为 0，$e\in\mathcal{E}$
Dis_e	第 e 项任务中托盘的行走距离，$e\in\mathcal{E}$
$Initial_e$	第 e 项任务开始时的位置，$e\in\mathcal{E}$
End_e	第 e 项任务结束时的位置，$e\in\mathcal{E}$
$Dis_{Initial_e,End_e}$	搬运机器人从第 e 项任务开始时的位置行至第 e 项任务结束时的位置的最短距离，$e\in\mathcal{E}$
$Dis_{End_e,Initial_{e'}}$	搬运机器人从第 e 项任务结束时的位置行至第 e' 项任务开始时的位置的最短距离，$e,e'\in\mathcal{E}$
$Vloc$	搬运机器人的默认速度
M	无穷大的数值
α	0-1 之间的常数，目标权重值
变量	
s_{ve}	如果调度机器人 v 被指派执行第 e 项任务则为 1，否则为 0，$v\in\mathcal{V},e\in\mathcal{E}$
$a_{ee'}$	如果第 e 项任务在第 e' 项任务前面被执行则为 1，否则为 0，$e,e'\in\mathcal{E}$
t_{ve}	搬运车辆 v 执行第 e 项任务的开始时间，$v\in\mathcal{V},e\in\mathcal{E}$
d_{ve}	商品托盘 i 去第 m 个拣选工位的行走距离，$i\in\mathcal{I},m\in\mathcal{M}$

相关参数计算：

若第 e 项任务为出库任务，则第 e 项任务的任务消耗时间为托盘行走时间与拣选工作台工作时间之和。

$$T_e = t_0 + Dis_{Initial_e,End_e}/Vloc, \forall e\in\mathcal{E} \tag{3.26}$$

目标函数：

$$\min\left\{\max(t_{ve}+T_e)\right\} \tag{3.27}$$

模型目标函数见式（3.27），为最小化最后一项任务的完成时间，与问题一中要求搬运机器人调度目标为最大化搬运机器人使用效率以及最小化搬运机器人行走路径总长度的目标存在差异，下面给出证明，证明本阶段目标函数涵盖问题一中的优化目标，对本阶段目标函数的合理性进行分析。

证明 1：针对问题一中的最小化搬运机器人行走路径总长度目标，不妨假设对于任意一辆搬运机器人，该机器人在规划时间段内总共被指派完成 w 项任务，且 $w \geq 2$，最后一项任务即为任务 e_w，$e_w \in \mathcal{E}$，假设该机器人开始进行任务 e_w 的时间为 t_{e_w}，该机器人最后一项任务的开始时间应不小于执行完倒数第二项任务的时间，该机器人执行倒数第二项任务的开始时间应不小于执行完倒数第三项任务的时间，依次类推，则 $t_{e_w} \geq \sum_{i=1}^{w-1} T_{e_i} + \frac{\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}}{Vloc} + t_{e_1}$，其中，$T_{e_i}, Vloc$ 均为参数，$t_{e_1} \geq 0$，t_{e_w} 仅与 $\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}$ 和 t_{e_1} 相关，又因为 t_{e_1} 为机器人开始进行任务 e_1 的时间，上式可简化为 $t_{e_w} \geq \sum_{i=1}^{w-1} T_{e_i} + \frac{\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}}{Vloc}$，其中 $\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}$ 为搬运机器人为了执行任务行走路径总长度。最小化 t_{e_w}，即搬运机器人的最后一项任务开始时间，即可最小化搬运机器人为了执行任务而行走的路径，搬运机器人行走的总路径长度为第一阶段优化的托盘行走路径总长度与搬运机器人为了执行任务行走路径总长度之和，因此最小化所有搬运机器人中 t_{e_w} 的最大值，即可从整体上减少机器人行走路径总长度。

证明 2：针对问题一中的最大化搬运机器人使用效率的目标，可简化为最小化搬运机器人空闲时间，即搬运机器人在规划时间段内没有被分配任务的时间总和。不妨假设对于任意一辆搬运机器人，该机器人在规划时间段内总共被指派完成 w 项任务，且 $w \geq 2$，最后一项任务即为任务 e_w，$e_w \in \mathcal{E}$，假设该机器人开始进行任务 e_w 的时间为 t_{e_w}，该机器人最后一项任务的开始时间应等于执行完倒数第二项任务的时间与执行完倒数第二项任务后的机器人空闲时间之和，该机器人执行倒数第二项任务的开始时间应等于执行完倒数第三项任务的时间与执行完倒数第三项任务后的机器人空闲时间之和，依次类推，则 $t_{e_w} = \sum_{i=1}^{w-1} T_{e_i} + \frac{\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}}{Vloc} + t_{e_1} + \sum_{i=1}^{w-1} t_{empty_{ei}}$，其中，$t_{empty_{ei}}$ 为该机器人在执行完第 e_i 项任务后的空闲时间，由于 $T_{e_i}, Vloc$ 均为参数，根据证明 1，上式可简化为 $t_{e_w} \geq \sum_{i=1}^{w-1} T_{e_i} + \frac{\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}}{Vloc} + \sum_{i=1}^{w-1} t_{empty_{ei}}$，$t_{e_w}$ 仅与 $\sum_{i=2}^{w-1} Dis_{End_{i-1},Initial_i}$ 和 $\sum_{i=1}^{w-1} t_{empty_{ei}}$ 相关，因此最小化所有搬运机器人中 t_{e_w} 的最大值，即可从整体上减少搬运机器人总空闲时间，及提高搬运机器人使用效率的同时，减小搬运机器人行走路径总长度。

约束条件：

（1）任务指派约束

$$\sum_{v \in \mathcal{V}} s_{ve} = 1, \forall e \in \mathcal{E} \tag{3.28}$$

$$a_{ee'}+a_{e'e}=1,\forall e,e'\in\mathcal{E},e\neq e' \tag{3.29}$$

式（3.28）限制任意一个任务e有且仅能有一辆搬运车辆v执行；式（3.29）限制任意两个任务e和e'有且仅能有一个被执行的先后顺序；

（2）同一车辆调度时间间隔约束

$$\begin{gathered}t_{ve}-t_{ve'}+M(3-a_{ee'}-s_{ve}-s_{ve'})\geq T_e+Dis_{End_e,Initial_{e'}}/Vloc\\ \forall v\in\mathcal{V},\forall e,e'\in\mathcal{E},e\neq e'\end{gathered} \tag{3.30}$$

式（3.30）限制若两个任务e和e'被同一辆搬运车辆v执行，并且任务e的执行顺序在任务e'之前，则车辆v开始执行任务e'与开始执行任务e的时间间隔应大于托盘在第e项任务时间与搬运机器人从第e项任务结束时的位置行至第e'项任务开始时的位置的最短时间之和。

$$\begin{gathered}t_{v'e'}-t_{ve}+M(Initial_e-End_{e'})+M(Hk_e+Hs_e-Chu_{e'})\geq\frac{Dis_{Initial_{e'},End_{e'}}}{Vloc}\\ \forall v,v'\in\mathcal{V},\forall e,e'\in\mathcal{E},e\neq e'\end{gathered} \tag{3.31}$$

$$\begin{gathered}t_{v'e'}-t_{ve}+M(Initial_e-End_{e'})+M(Chu_e-Hk_{e'})\geq\frac{Dis_{Initial_{e'},End_{e'}}}{Vloc}\\ \forall v,v'\in\mathcal{V},\forall e,e'\in\mathcal{E},e\neq e'\end{gathered} \tag{3.32}$$

$$t_{ve}-M\times s_{ve}\leq 0,\forall v\in\mathcal{V},e\in\mathcal{E} \tag{3.33}$$

式（3.31）限制若任务e的任务性质为回库或回收，并且任务e'的任务性质为出库，并且任务e开始位于的拣选工作台节点与任务e'的拣选节点相同，则任务e'到达该拣选工作台的时间应在任务e开始执行后；式（3.32）限制若任务e的任务性质为出库，并且任务e'的任务性质为回库，并且任务e开始位于的储位节点与任务e'的储位节点相同，则任务e'到达该储位的时间应在任务e开始执行后；式（3.33）限制若任务e没有被指派至搬运车辆v执行，则t_{ve}为0。

（3）同一托盘调度时间间隔约束

$$B_e^{ck}+B_{e'}^{hs}-a_{ee'}\leq 1+M(Num_e-Num_{e'})^2,\forall e,e'\in\mathcal{E},e\neq e' \tag{3.34}$$

$$B_e^{ck} + B_{e'}^{hk} - a_{ee'} \le 1 + M(Num_e - Num_{e'})^2, \forall e, e' \in \mathcal{E}, e \ne e' \tag{3.35}$$

式（3.34）限制若两个任务e和e'属于同一个托盘，并且任务e的任务性质为出库，并且任务e'的任务性质为回收，则任务e的执行顺序必在任务e'之前；式（3.35）限制若两个任务e和e'属于同一个托盘，并且任务e的任务性质为出库，并且任务e'的任务性质为回库，则任务e的执行顺序必在任务e'之前。

（4）出库与回库任务先后约束

$$a_{ee'} + M\left(B_e^{ck} - B_{e'}^{hk}\right) + M(Jou_e - Hk_{e'}) \ge 1, \forall e, e' \in \mathcal{E}, e \ne e' \tag{3.36}$$

式（3.36）限制若任务e的任务性质为出库，并且任务e'的任务性质为回收，并且任务e的储位节点与任务e'的回库储位节点相同，则任务e的执行顺序必在任务e'之前。

3.4 求解算法

针对模型变量维数多，变量空间大的特点，针对本问题特点，设计邻域搜索与模拟退火相结合的启发式算法，分别对第一阶段托盘调度模型以及第二阶段的 AGV 调度模型进行求解。对于一阶段托盘调度问题与二阶段 AGV 调度问题，除了调度主体发生变化外，算法步骤以及相应逻辑并未发生改变，因此以第一阶段商品托盘调度过程为例，对算法进行介绍，并对算法步骤进行描述。

基于模拟退火框架，为了生成一个邻域解，我们不是简单地在别指派托盘集合中随机选择一个托盘i，而是使用一种类似于轮盘赌轮选择的方法，由于将每个托盘的行走路径总长度作为衡量其性能的关键指标，影响托盘i行走路径长度的因素有托盘i指派的拣选工作台、储位节点以及回收节点，影响托盘行走路径总长度的因素除上述因素外，还包括指派托盘的选择信息。因此，在邻域选择中过程中，首先记录了每个被指派托盘的行走路径长度，如表 3-3 所示；然后，我们使用基于路径长度的轮盘赌轮选择方法选择一个托盘；接下来，以这个托盘为主体进一步确定要改变哪个决策变量；最后，我们为相关的决策变量在可行域范围内随机选择一个离散化的值。邻域搜索算法伪代码如表 3-4 所示，模拟退火参数设置如表 3-5 所示，算法流程图如图 3-4 所示。

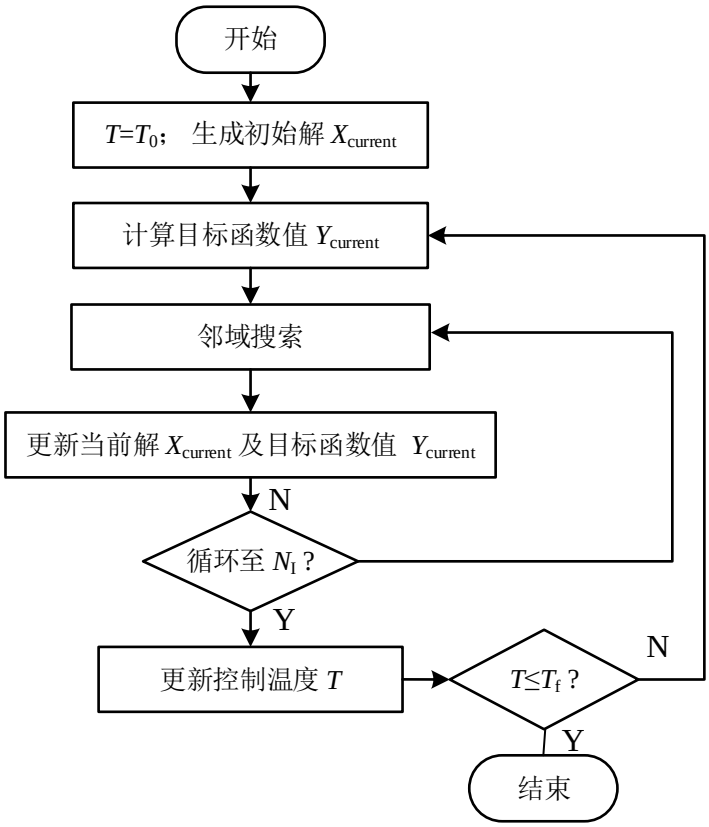


图 3-4 NS-SA 算法流程图

表 3-3 记录托盘行走路径长度示例

托盘行走路径长度	53	37	42	40
Pallet_ID	10008	10027	10047	10052

表 3-4 邻域搜索算法伪代码

Require: For each pallet we record its fitness value $Dist_i^{sum}$, distance from store station to picking station $Dist_i^{chu}$, distance from picking station to store station $Dist_i^{hk}$, distance from picking station to recycle station $Dist_i^{hs}$

1: Generate maximum cycles of neighborhood search φ;

2: The total number of fitness value $Dist^{sum} = \sum_{i \in \mathcal{I}} Dist_i^{sum}$;

3: $kk = 1$;

4: **while** $kk < \varphi$ **do**

5: sum=sum+ $Dist^{sum}$;

6: Generate random number, v= random(0,1);

7: $Poss_i = \frac{Dist_i^{sum}}{Dist^{sum}}$, $sum_{Poss} = \sum_{i \in \mathcal{I}} Poss_i$, $Poss_{di} = \frac{Poss_i}{sum_{Poss}}$

8: Cumulative summation $Poss_{di}$

9: Choose one pallet i in pallets set by selecting v falling in the Cumulative summation

10: $Poss_{di}$ interval

11: Generate random number, s= random(0,1);
12: **if** $i \le 0.5$
13: **then**
14: choose with equal probability among the picking station
15: **if** $Dist_i^{hk} > 0$
16: **then** choose with equal probability among the store station;
17: **else if** $Dist_i^{hs} > 0$
18: **then** choose with equal probability among the recycle station;
19: **end if**
20: **end if**
21: **else if** $i > 0.5$
22: **then**
23: choose randomly the pallet j except pallets have been selected from pallet set to
24: meeting order requirements that to replace pallet i;
25: assign pallet related information and calculate $Dist_i^{sum}, Dist_i^{chu}, Dist_i^{hk}, Dist_i^{hs}$
26: **end if**
27: $kk = kk + 1$
28: **end while**

表 3-5 NS-SA 算法参数设置

参数	取值
模拟退火初始温度	T_0
模拟退火最大循环次数 N_I	1200
局部搜索最大循环次数 φ	50
退化解的初始接受率 τ_0	0.4
温度降低系数 α	0.99

最终温度T_f	$T_0/10^7$

采用 Intel i7-9700KF8C8T 处理器，内存 16GB 计算机，基于 MATLAB R2016a 环境，输入上述算法参数以及订单数据、仓库信息数据进行仿真，得到 AGV 调度机制中第一阶段求解结果随迭代次数变化情况如图 3-5 所示；得到 AGV 调度机制中第二阶段求解结果随迭代次数变化情况如图 3-6 所示。

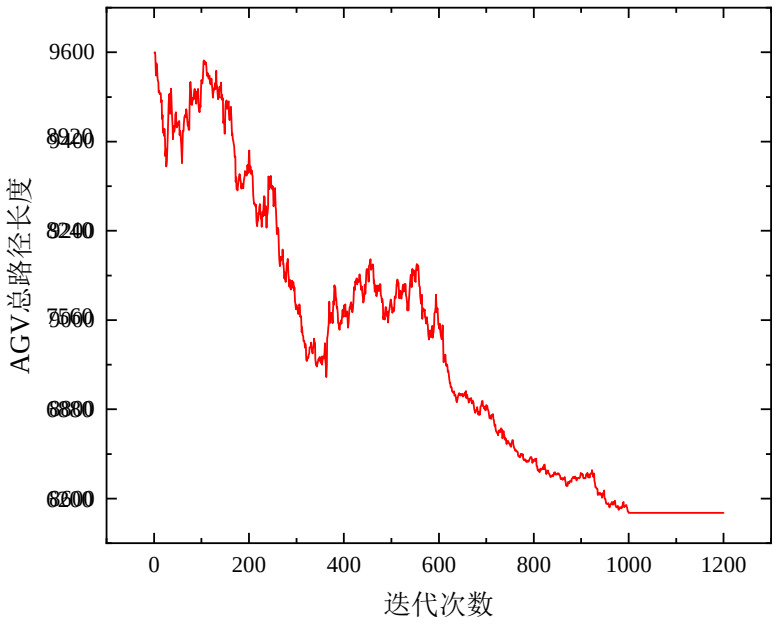


图 3-5 AGV 调度机制中第一阶段适应度变化图

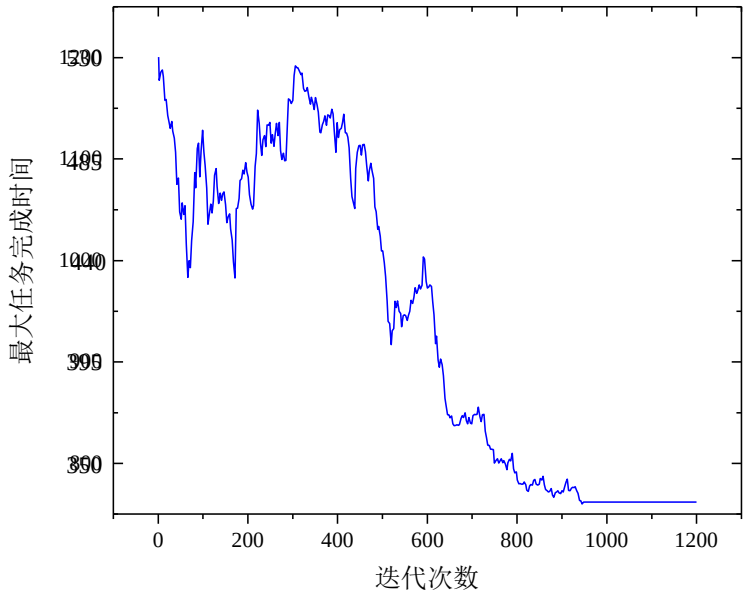


图 3-6 AGV 调度机制中第二阶段适应度变化图

如图所示，算法可在 1200 次迭代后收敛，根据 AGV 调度机制的求解结果，计算搬运机器人行走路径总长度为 6273 个单位长度，与初始可行解相比，优化了 34.7%，最大任务完成时间在第 347 个单位时间，与初始可行解相比，优化了 33.3%，优化效果显著，本文提出 AGV 调度机制与方法具有有效性。

4　问题二 任务均衡区域划分问题

根据问题 2 中的相关描述，在问题一的 AGV 调度机制基础上，在托盘调度模型中更改决策变量以及目标函数，最小化各拣选工位之间的工作量差距，即最小化为各拣选位指派托盘数量的差异。在问题一模型的基础之上，将决策变量 x_i 调整为 x_i^l，即如果商品托盘 i 指派的拣选工作台属于第 l 个拣选工位，则 $x_i^l=1$，否则为 0，其中 $l\in\mathcal{L}$，$\mathcal{L}$ 为拣选工位集合。

将 AGV 调度机制中第一阶段调度目标函数改为式 4.1。

$$\min\left\{\alpha\cdot\sum_{i\in\mathcal{I}}(d_{iqm}+d_{imq}+d_{imr})+(1-\alpha)\cdot\sum_{l=1}^{|\mathcal{L}|-1}\sum_{l'=l+1}^{|\mathcal{L}|}(\sum_i x_i^l-\sum_i x_i^{l'})\right\} \tag{4.1}$$

其中，α 为 0-1 间的常数，为目标函数权重系数，$\sum_{l=1}^{|\mathcal{L}|-1}\sum_{l'=l+1}^{|\mathcal{L}|}(\sum_i x_i^l-\sum_i x_i^{l'})$ 为任意两工作台间指派任务量的差值，目标函数意义为最小化托盘路线总长度以及任意两工作台间指派任务量间的差值，可以满足问题描述中合理均衡每个拣选工位工作量的目标，均衡每个拣选工位的任务。

根据问题二中的仓库题图动态分区需求，考虑订单顺序，假设订单量为动态分区触发条件，假设以 200 个订单为触发条件，即根据订单需求数据，首先优化订单需求数据中前 200 个订单，此时应用 AGV 调度机制，完成托盘指派以及 AGV 优化调度任务，更新托盘集合信息以及商品信息、各储位中托盘信息，同时根据商品托盘的优化调度结果，获得仓库优化分区结果；然后基于更新的托盘数据以及储位信息数据，考虑订单需求数据中第 201 个订单至第 400 个订单，应用 AGV 调度机制，根据商品托盘的优化调度结果，获得仓库优化分区结果。以调度前 200 个订单、第 201 个至第 400 个订单、第 401 个至最后一个订单为例，对 AGV 进行运行调度，输出各阶段托盘调度结果以及托盘初始储位，得到各种订单量下仓库地图划分结果如图 4-1 至图 4-3 所示。

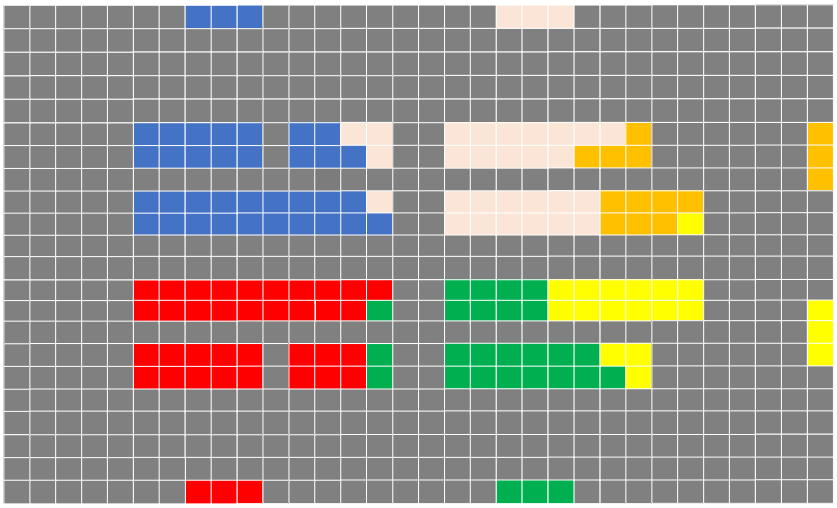


图 4-1 前 200 个订单调度下仓库分区

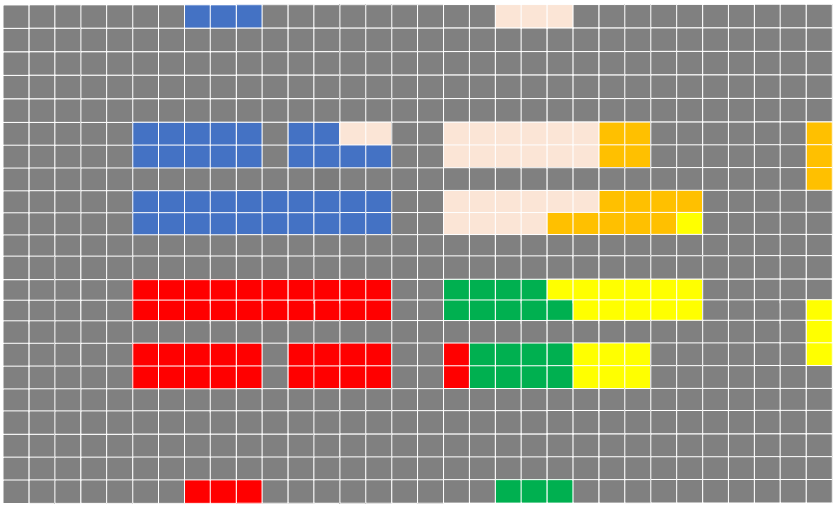


图 4-2 第 201 个至第 400 个订单调度下仓库分区

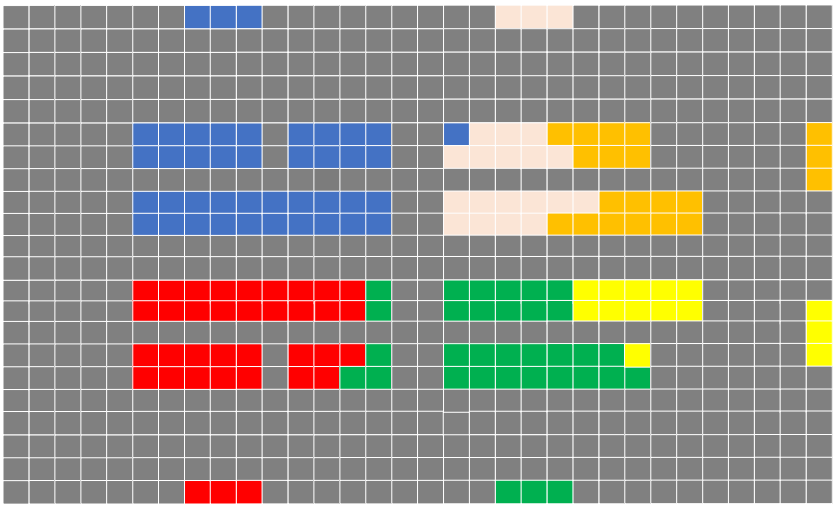


图 4-3 第 401 个至第 600 个订单调度下仓库分区

5 问题三 AGV 碰撞和拥堵问题

AGV 在接收到任务后，采用路径规划算法为机器人规划路径。在执行任务的过程中，AGV 从储位节点到拣选工位节点、从拣选工位节点到空托盘回收节点、从拣选工位节点到空闲的储位节点或其他突发情况不可避免的会导致 AGV 之间的路径冲突，导致碰撞和拥堵。因此，在仓库中需要设计一种避障算法为 AGV 选择出无冲突的路径，提高工作效率，保证安全作业。

5.1 AGV 冲突类型

由于仓库中货架较多，且受到仓库空间的限制，货架之间的通道较窄，尤其在第 8 行与第 15 行，两行货架之间只有一格间隙，这就导致在 AGV 取货过程中，可能会与其他 AGV 在此狭窄通道内相互阻挡。假设所有的 AGV 均以同一速度匀速运行，则根据仓库内的实际情况，可将仓库内 AGV 的冲突类型做出如下分类：

（1） **顶点冲突：**通过全局最优路径算法得到的最优路径，当两台 AGV 发生交叉，在同一时刻占据同一个网格时，此时两台 AGV 则认为会发生顶点冲突，如图 5-1（a）。在某个交叉路口处，AGV1 与 AGV2 沿着各自最优路径前进，在下一时刻两台 AGV 会同时占据同一位置，产生顶点冲突；

（2） **对向冲突：**对向冲突发生在两辆 AGV 在同一行或同一列上时。根据最优路径算法，当两台 AGV 在同一时刻沿着同一路线对向行驶时，会发生对向冲突，如图 5-1（b）。在同一行驶道路内，AGV1 和 AGV2 占用同样的路径往对方的方向驶去。此时，AGV1 和 AGV2 仍然有调整行驶方向进行避让的可能，因此，将产生对向冲突；

（3） **对向死锁：**与冲突（2）相对应，在同一行驶道路内，AGV1 和 AGV2 占用同样的路径往对方的方向驶去，如图 5-1（c）。但是，AGV1 和 AGV2 由于受到通行能力的限制，无法通过调整行驶方向进行避让，此时 AVG 产生死锁，因此，将产生对向死锁；

（4） **死锁：**当两辆 AGV 按照各自行驶路径运行时，在下一时刻，各自的网格都将被对方占据。如图 5-1（d），AGV1 需要前往储位节点 1 取货，AGV2 需要将空托盘存放到空储位节点 2 中，AGV1 和 AGV2 的下一时刻的网格点均被对方占据，此时，产生死锁。

5.2 AGV 冲突探测方法

在运行过程中，为了为 AGV 规划出无冲突的运行路径，避免 AGV 之间发生碰撞，需要能够在算法快速的识别 AGV 之间的冲突可能性及冲突类型，根据相应的冲突类型选择合适的冲突解脱策略。本文采用网格化方法对 AGV 进行冲突探测，同时，为了提高冲突探测的速度且占用更小的计算机内存，本文采用了相关时空数据（Relational Time-Space Data Structure, RTSDS）进行冲突探测。

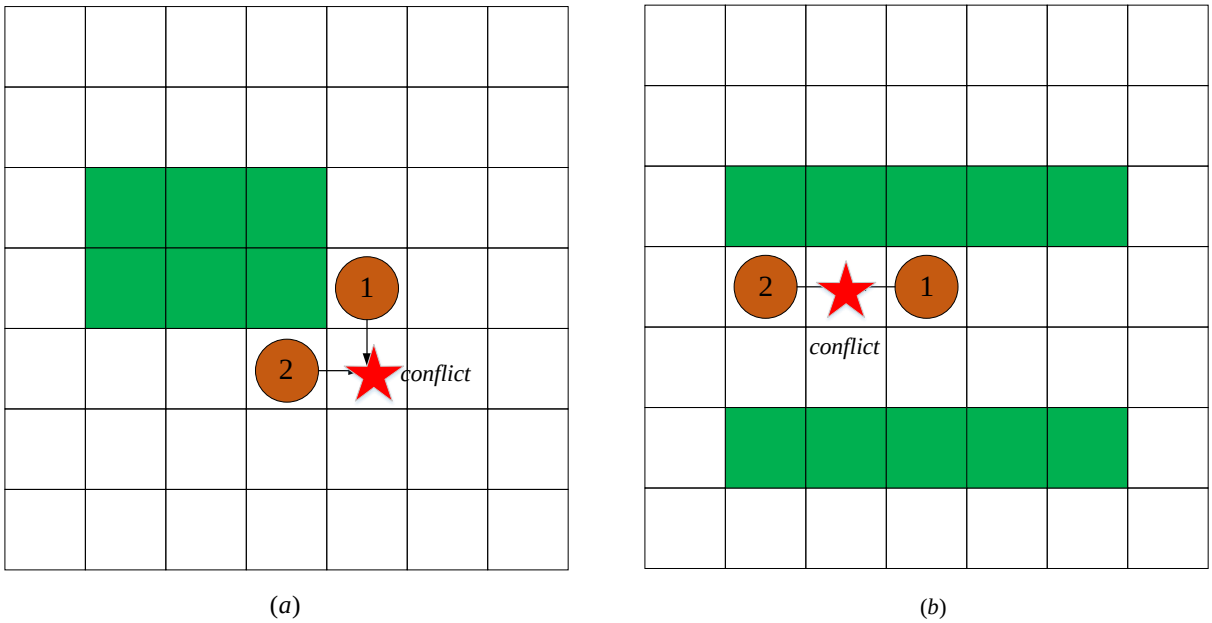
5.2.1 网格冲突探测

在实际的冲突探测场景中，部分 AGV 路径在时间和空间维度上有较大的间隔，不存在发生潜在冲突的可能性，为了提高对大量 AGV 路径的冲突探测效率，减少不必要

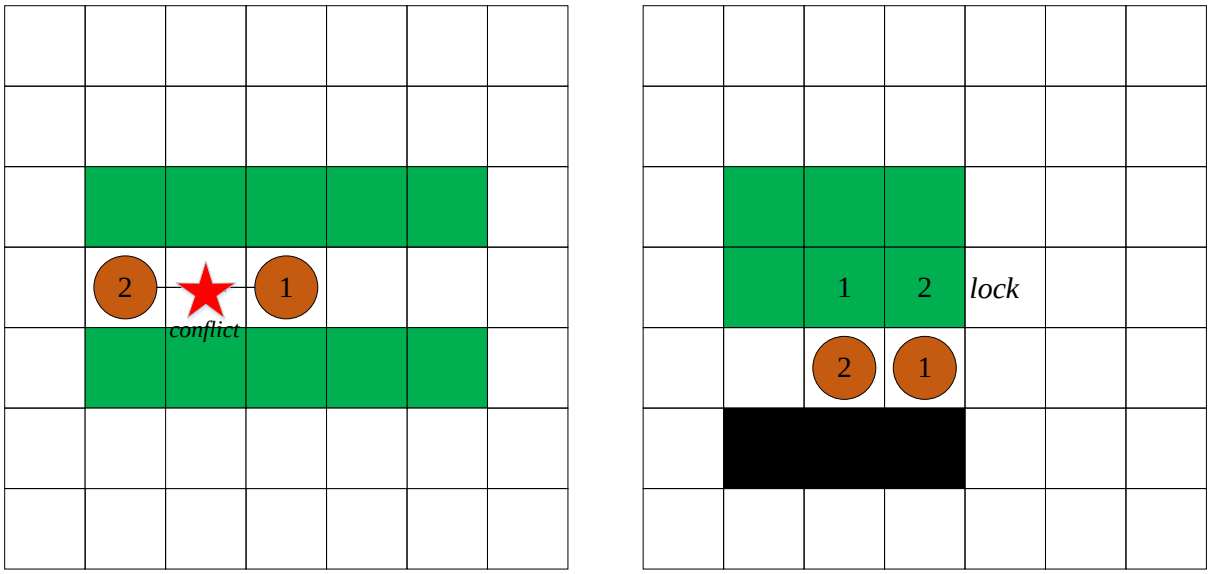
的计算过程，提出了一种基于三维网格的冲突探测算法。如图 5-2 所示，该方法首先利用三维时空网格（二维平面与时间维度）将所研究的仓库区域离散化。根据网格单位和离散的时间步长 Δt 规定网格中的每个单元的大小。识别每条路径的各个采样点所对应的网格单元，即对于路径 i 的第 j 个离散的采样点 $P_{i,j}$，根据其三维坐标 $(x_{i,j}, y_{i,j}, t_{i,j})$ 对应到相应的网格单元 $a_{m,n,l}$，其中 $m = x_{i,j}, n = y_{i,j}, l = t_{i,j}$。

定义一个网格单元 $a_{m,n,l}$ 的邻域由其自身以及其在空间维度上的周围 $3^2 - 1 = 8$ 个网格单元组成，记为：

$$A_{m,n,l} = \begin{bmatrix} a_{m-1,n-1,l} & a_{m,n-1,l} & a_{m+1,n-1,l} \\ a_{m-1,n,l} & a_{m,n,l} & a_{m+1,n,l} \\ a_{m-1,n+1,l} & a_{m,n+1,l} & a_{m+1,n+1,l} \end{bmatrix}_{3\times 3}$$



(a) (b)



(c) (d)

图 5-1 AGV 冲突类型图

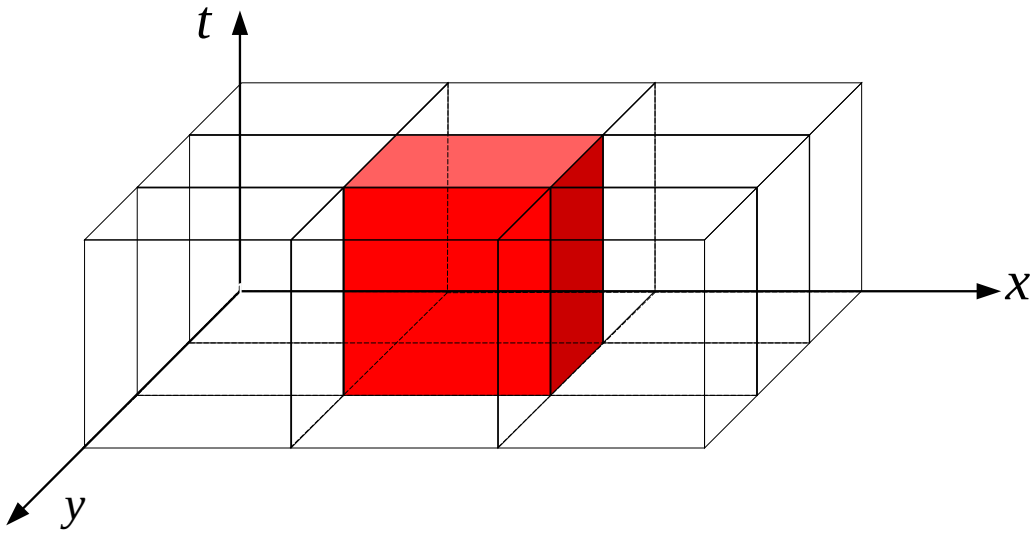


图 5-2 网格冲突探测

为了对路径的潜在冲突进行初步筛选，算法依次检查网格单元 $a_{m,n,l}$ 邻域中的 9 个网格单元，如果邻域中任意单元内存在其他 AGV 路径的采样点，则表明路径 i 可能存在潜在的路径冲突，并根据对应 AGV 下一时刻采样点的位置判断是否会发生冲突和冲突类型。否则，路径 i 的离散的采样点 $P_{i,j}$ 无冲突，不需要进一步判断。

5.2.2 相关时空数据结构

随着需要处理的 AGV 路径的增加，用于存储 AGV 路径采样点的计算机内存需求大幅增加，这对于计算机的性能提出了较高的要求。由于在最优路径规划中，为每一台 AGV 规划的最优路径具有一定的贪心效果，因此在实际运行过程中，所建立的网格的坐标总数远远大于被占用的坐标数。为了减少冲突探测过程需要的内存，提出了 RTSDS，通过创建两种不同的数据库用于管理信息。其中一个数据库用于存储所研究仓库的三维时空网格单元信息。另一个数据库用于存储路径的非时空信息，两个数据库的信息通过指针相互关联。

RTSDS 通过两个不同的数据库实现冲突探测。第一个数据库为基础时空数据结构（Base Time-Space Date Structure, BTSDS），每行代表仓库的每个网格单元，但每个记录仅有一列（通常占用 4 个字节）而不是 N 条路径对应的 N 列（$4\times N$ 个字节）。该列用于存储与第二个数据库关联的指针。

第二个数据库称为堆栈式路径信息（Stacked Trajectory Information, STI），用于存储所有路径使用的三维坐标。该数据库的特点是以堆栈的形式存储路径有关信息，因此数据库中不会存在空的记录。STI 的结构可以根据冲突探测和解脱算法的需要进行设计，但始终需要一列用于存储指向另一个 STI 位置的指针，如图 5-3。

对于 BTSDS 中的坐标，若其与 STI 关联的指针不为 0，则表示至少有一条路径占用了该坐标。该指针的值表示上一个占用该坐标的航迹信息在 STI 中的位置。在 STI 中前两列用于有关路径的非时空信息，第三列存储指向另一个 STI 位置的指针，表示上一个占用该坐标的路径在 STI 中对应的位置。若指针不为 0，则根据指针访问对应的 STI 位置上的路径信息。结束冲突判断后，继续检查当前 STI 位置的第四列，如果指针的值为 0，则表示之前没有路径占用该坐标，若不为 0 则重复上述过程。

RTSDS 主要的优点是构造 BS 所需的内存与所研究的仓库大小和网格单元的大小有关，因此所需内存不会随着所研究的航迹数量增加而提高，BS 的总内存为：

$$total\ Memory\ BS = X \cdot Y \cdot P \tag{5.1}$$

其中 P 为存储指向一条记录的指针所占用的内存，通常为 4 字节。STI 所需要的内存由下式计算得到：

$$total\ Memory\ STI = N \cdot L \cdot (B + P) \tag{5.2}$$

N 为处理的路径数量，L 为每条路径的平均时间步长数量。

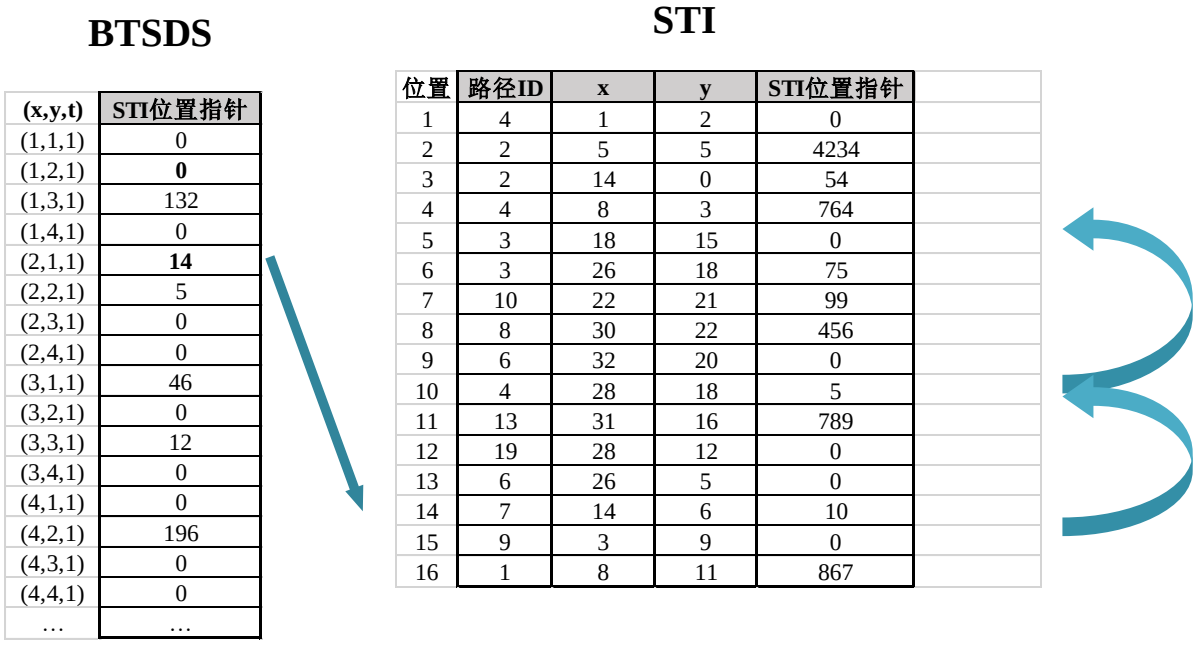


图 5-3 相关时空数据结构示意图

5.3 AGV 冲突解脱策略

5.3.1 冲突解脱方式

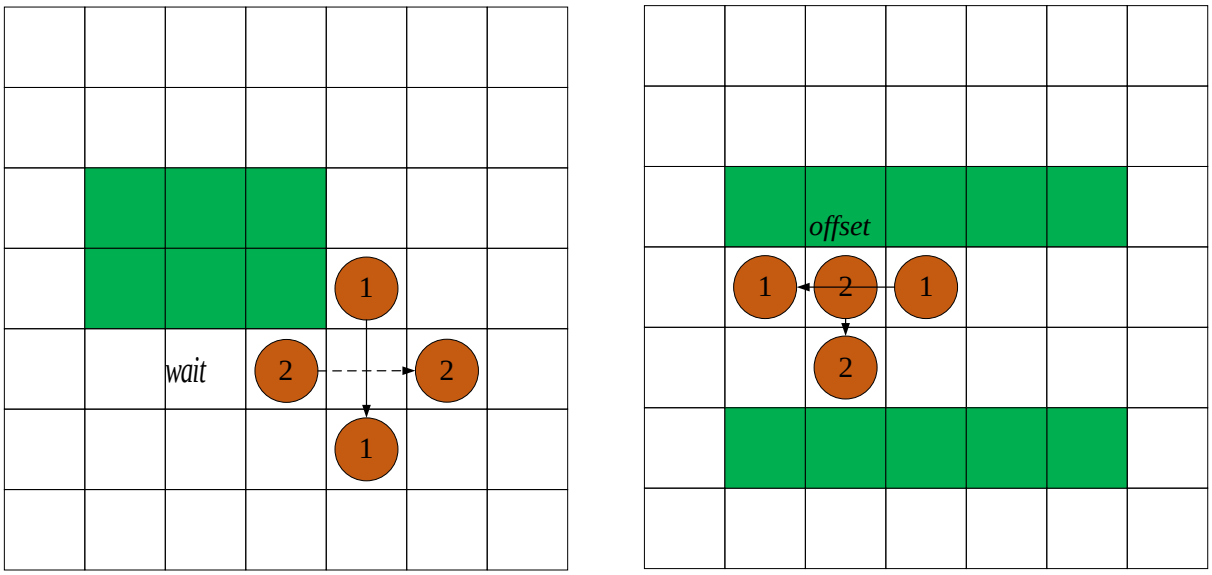
假设 AGV 能够在适当的位置等待，并且可以瞬间停止并匀速行驶。本文在对 AGV 的冲突进行处理时，针对不同的冲突类型，采用不同的冲突解脱方式：

（1） **顶点冲突——冲突点前等待：**对于顶点冲突，由于两台 AGV 在下一时刻需要同时占据同一网格，因此只需要其中一台 AGV 在当前网格中进行等待，等候另一辆通过即可解决冲突。等待的时间为 AGV 通过一个网格的时长 Δt，即 3.2.1 节中离散化网格中的网格的大小。如图 5-4（a）所示，当 AGV1 和 AGV2 发生顶点冲突时，令 AVG2 在冲突点前等待，等候 AGV1 通过再行通过。

（2） **对向冲突——路径偏移等待：**对于对向冲突，由于两辆车占用同一条路径而且对向行驶，所以采用等待策略会导致等待时间过长，降低 AGV 运行效率。因此对于此种冲突，采用路径偏移等待策略能更加有效的解决冲突。如图 5-4（b）所示，当 AGV1 和 AGV2 发生对向冲突时，令 AGV2 离开当前路径，任意前往周围某一网格，等候 AVG1 通过再返回原有路径通行。

（3） **死锁——起点等待：**对于死锁的情况，在两台 AGV 周围一般不存在可以调整路径的替代网格，而且对于对向死锁，采用冲突点前等待的策略可能会导致长时间的等待。因此，采用起点等待策略使得 AGV 在出发前提前预测死

锁，如果预测会出现死锁的情况，则在起点等待以避免死锁情况的出现。



(*a*) (*b*)

图 5-4 冲突解脱策略示意图

5.3.2 决策变量及约束条件

设 AGV 通过一个网格的时间为 1。根据上述冲突解脱策略，本文在时间维度，冲突解脱表现为与路径 v_i 相对应的起始时刻 δ_i 的调整以及在冲突点前等待的时间 γ_i；而在空间维度，表现为与路径 v_i 相对应的路径调整量 l_i。因此，本文的决策变量有三个：

$$\boldsymbol{\delta} := (\delta_1, \delta_2, \delta_3, \cdots, \delta_N)$$

$$\boldsymbol{\gamma} := (\gamma_1, \gamma_2, \gamma_3, \cdots, \gamma_N)$$

$$\boldsymbol{l} := (l_1, l_2, l_3, \cdots, l_N)$$

则本文的决策向量为 $\boldsymbol{u} := (\boldsymbol{\delta}, \boldsymbol{\gamma}, \boldsymbol{l})$。决策变量必须满足**延误约束**：

由于不能将 AGV 的起始时间推迟的太久，$\forall v_i \in V, i = 1, 2, \cdots, N$，其对应的路径修正延误（Total Path Amendment Delay，TPAD）必须保持在离散化的区间 $[0, T_H^r]$ 中，所有可能的延误集合 TPAD 定义如下：

$$\begin{aligned} TPAD_i &:= \{0, \Delta t_H, \ldots, T_H^r\} \\ TPAD_i &= \delta_i - \delta_i^{orig} + \gamma_i + 2 \times l_i \end{aligned} \tag{5.3}$$

其中，Δt_H 是 AGV 延误时隙，T_H^r 表示单台 AGV 最大允许延误，δ_i 为 v_i 预计开始工作时刻，δ_i^{orig} 为 v_i 的初始预计开始工作时刻，γ_i 为 v_i 冲突前等待点等待时间，l_i 为 v_i 路径调整量，由于调整一次路径需要花费二倍网格通行时间，因此路径调整延误时间为路径调整量的二倍。

5.3.3 目标函数

本文不仅需要尽可能多的解决 AGV 之间的冲突，还需要尽可能的使 AGV 遵循其初始最优路径及开始工作时刻。本文目标函数为：

$$fitness = \sum_{i=1}^{N} \frac{1 - TPAD_i / TPAD_{max}}{1 + CS} \tag{5.4}$$

其中，*CS* 为当前路径计划中的 AGV 冲突数量。

5.4 冲突解脱算法

本文采用了基于动态分组策略的变种群规模合作协同进化算法（Adaptive Population Size Cooperative Co-evolution Algorithm based on Dynamic Grouping，APCCDG）进行冲突解脱[6]。合作协同进化算法旨在将一个大规模的复杂问题通过相应的策略降低决策空间的维数，提高算法的求解效率，算法的具体流程如图 5-5。

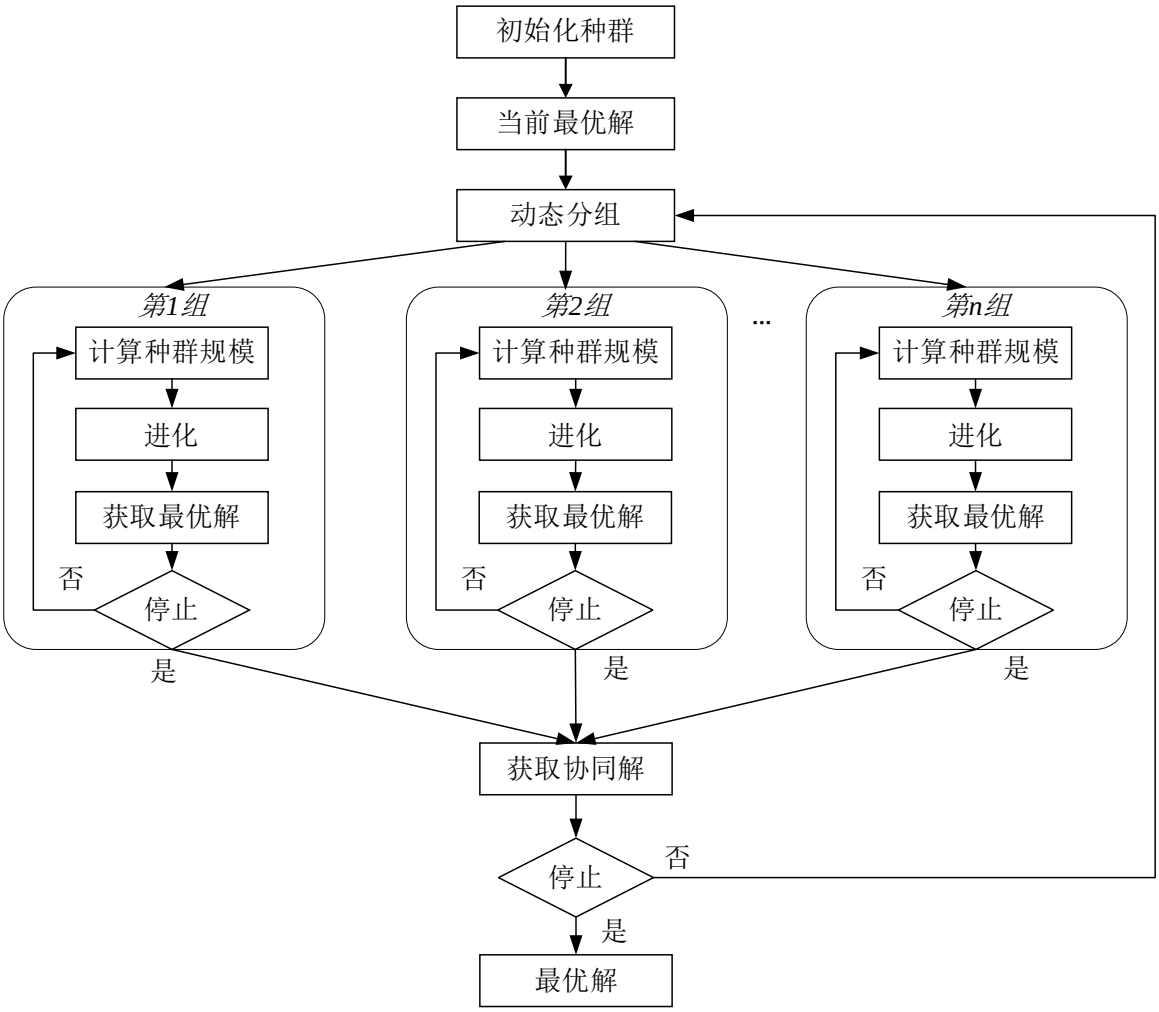


图 5-5 基于动态分组的变种群规模合作协同进化算法

算法伪代码如下：

Algorithm1: APCCDG

Input: Initial solution, *MAXgenerations* , *maxgenerations*

Output: APCCDG solution

// Main procedure

1:Initialize the initial solution to generate the initial population;

2:**for** $i=1:MAXgenerations$ **do**

3: Evaluate all the individuals in the population and compute the fitness;

// Cooperative co-evolution

4: Decompose the objective vector into m_i low subcomponents based on the dynamic grouping strategy;

5: **Set** $j=1$;

6: **while** $j \le m_i$ **do**
7: **for** $k = 1: maxgenerations$ **do**
8: Initialize the j^{th} subcomponent based on adaptive population size;
9: Select the $offspring^k$ based on the fitness;
10: Use the adaptive crossover and mutation strategy to obtain the $offspring^{k+1}$;
11: **end for**
12: $j = j + 1$;
13: **end while**
14:**end for**

5.4.1 动态分组策略

动态分组策略根据 AGV 路径之间的冲突关系对路径进行分组，定义一个矩阵 C 表示 AGV 路径之间的冲突关系：

$$C = \begin{pmatrix} C_{11} & \cdots & C_{1n} \\ \vdots & \vdots & \vdots \\ C_{n1} & \cdots & C_{nn} \end{pmatrix} \tag{5.5}$$

其中，

$$C_{ij} = \begin{cases} 1, if\ R_i\ and\ R_j\ conflict,\ i \neq j, \\ 0, otherwise \end{cases}, i, j = 1, 2, \cdots, n \tag{5.6}$$

那么，分组后各组内的 AGV 路径之间相互影响，而小组之间的 AGV 没有影响。在进行冲突解脱时，由于小组之间没有影响，因此可分别优化；但是当一次迭代完成后，由于路径发生变化，因此需要根据新的冲突关系重新进行分组。

动态分组伪代码如下：

Algorithm2: Dynamic Grouping Strategy
Input: $R = (r_1, r_2, r_3, \cdots, r_n)$
Output: Group set
// Main procedure
1: $group = \Phi$;
2:**Set** $i = 1$**;**
3:***Repeat***
4: $group_i$ = The first element in R;
5: $R = R \setminus group_i$;
6: Sequentially select $a \in group_i$ and $b \in R$
7: **if** $C_{ab} = 1$ **Then**
8: $group_i = group_i \cup b$;
9: $R = R \setminus b$;
10: **end if**
11: $group = group \cup group_i$;
12:i=i+1;
13:**Until** $R = \Phi$

5.4.2 种群初始化策略与选择策略

在对种群初始化时，采用随机改变 AGV 路径的决策变量产生初始种群的方式。首先确定需要随机扰动的 AGV，本文根据每一台 AGV 的冲突数量大小，优先选择冲突数量较大的 AGV 路径，根据冲突探测算法中获取的冲突类型，选择冲突类型最多的冲突对应的冲突解脱方式所对应的决策变量：调整起始时间、冲突点前等待、路径偏移等待。

本文采用锦标赛选择，从父代种群中随机选择两个父代，分别计算其适应度函数，选择适应度值较大的进行交叉操作，每次交叉操作产生两个子代。

5.4.3 自适应交叉策略

自适应交叉算子根据 AGV 计划路径的局部适应度确定，本文采用的局部适应度算子计算方式如下：

$$fitness_j^k = \frac{1 - TPAD_j^k / TPAD_{max}}{1 + CS_j^k} \tag{5.7}$$

式中，$fitness_j^k$ 为第 k 组第 j 台 AGV 计划路径的局部适应度，$TTAC_j^k$ 为第 k 组第 j 台 AGV 路径的调整成本，$TPAD_{max}$ 为允许的最大路径调整成本。在种群中选择两个亲本 a 和 b，比较两个亲本相同位置的 AGV 计划路径局部适应度，如果 $fitness_{a_j}^k < fitness_{b_j}^k$，那么交叉产生的两个子代同时保留父代 b 该位置基因；如果 $fitness_{a_j}^k > fitness_{b_j}^k$，那么交叉产生的两个子代同时保留父代 a 该位置基因；如果 $fitness_{a_j}^k = fitness_{b_j}^k$，那么由式（5.8）计算：

$$\begin{aligned} ca_j &= floor\left(\varepsilon a_j + (1-\varepsilon) b_j\right) \\ cb_j &= floor\left(\varepsilon b_j + (1-\varepsilon) a_j\right) \end{aligned} \tag{5.8}$$

式中，$floor$ 表示向下取整，ε 为线性重组系数。自适应交叉算子原理图如图 5-6 所示，交叉概率为 P_c。

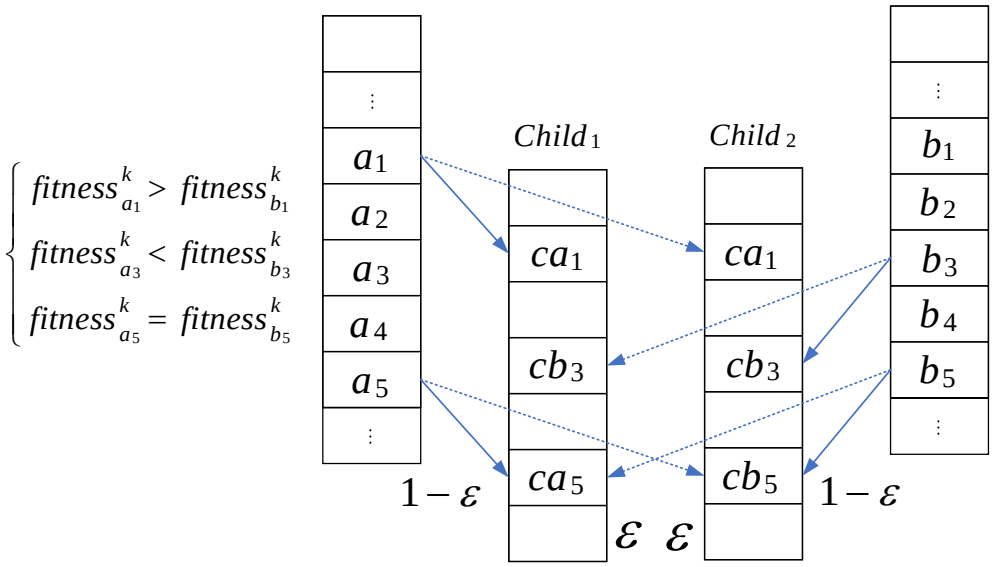


图 5-6 自适应交叉算子

5.4.4 自适应变异策略

当 AGV 计划路径的局部适应度值越高时，表明该 AGV 按当前计划路径会产生更

少的冲突数量和路径调整量。在本文采用的自适应变异算子中，当子种群 k 中的第 j 台 AGV 计划路径的局部适应度值小于 σ 时，发生变异的概率为 P_m，自适应变异算子如图 5-7 所示。

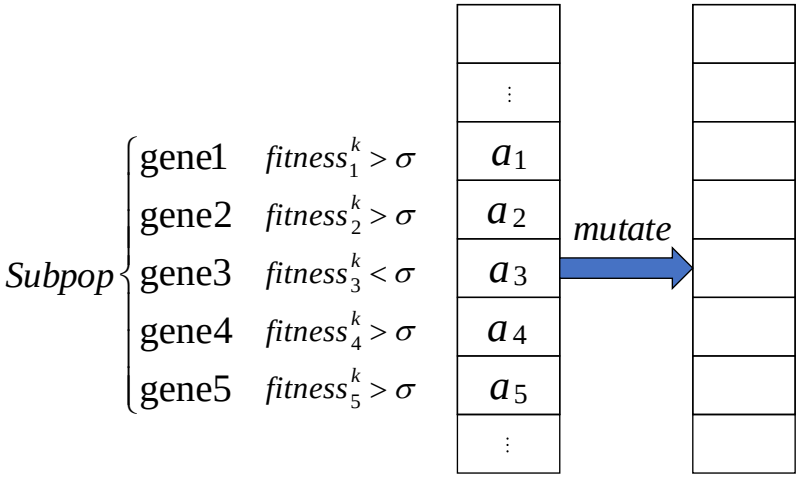


图 5-7 自适应变异算子

5.5 仿真实验结果及数据分析

首先通过两台 AGV 进行局部仿真实验。在仓库环境中，如图 5-8，红色圆圈代表 AGV1，黄色的圆圈代表 AGV2。其中，AGV1 正在执行从储位节点（5,5）取托盘送至拣选工位节点（7,0）；AGV2 正在执行从拣选工位节点（9,0）将空托盘运送到空托盘回收节点（1,0）。根据初始最优路径规划及两台 AGV 的开始工作时刻，两台 AGV 将在（7,1）节点发生顶点冲突。通过避障策略，AGV2 将在（8,1）节点处等待，等候 AGV1 通过后再行通过，AGV2 产生延误。两台 AGV 的冲突解脱后的路径如图 5-9 所示，证明了冲突解脱算法的有效性。

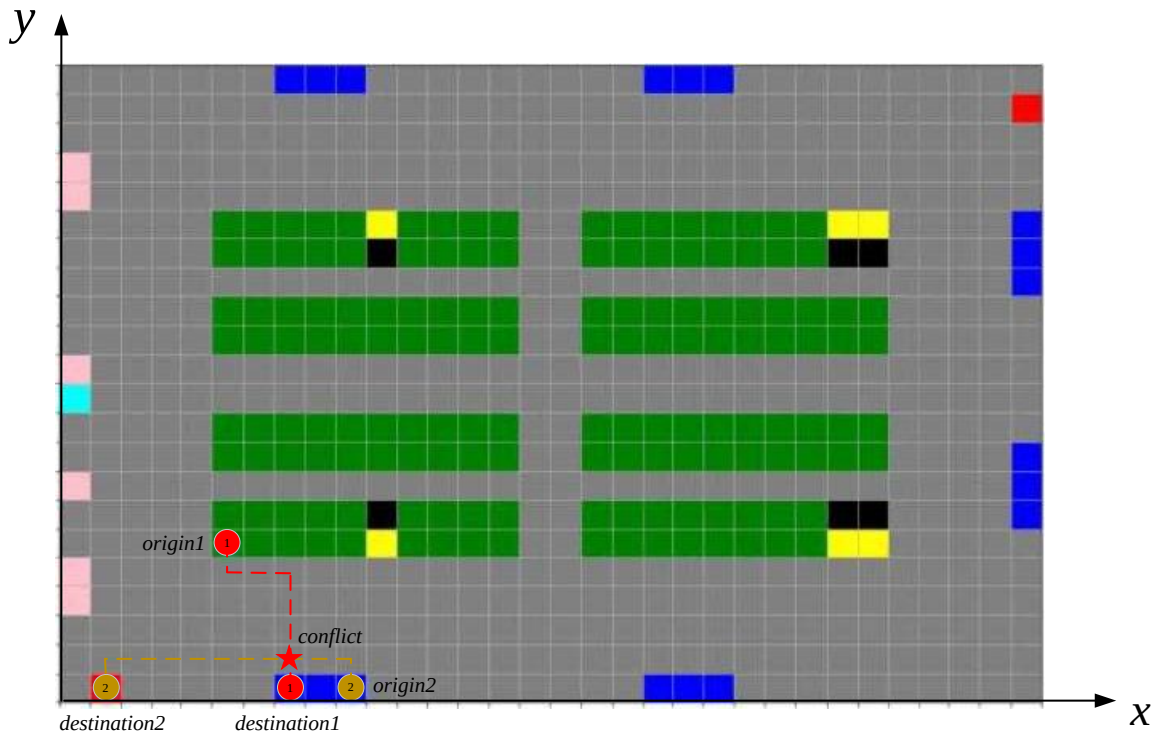


图 5-8 两台 AGV 局部冲突图

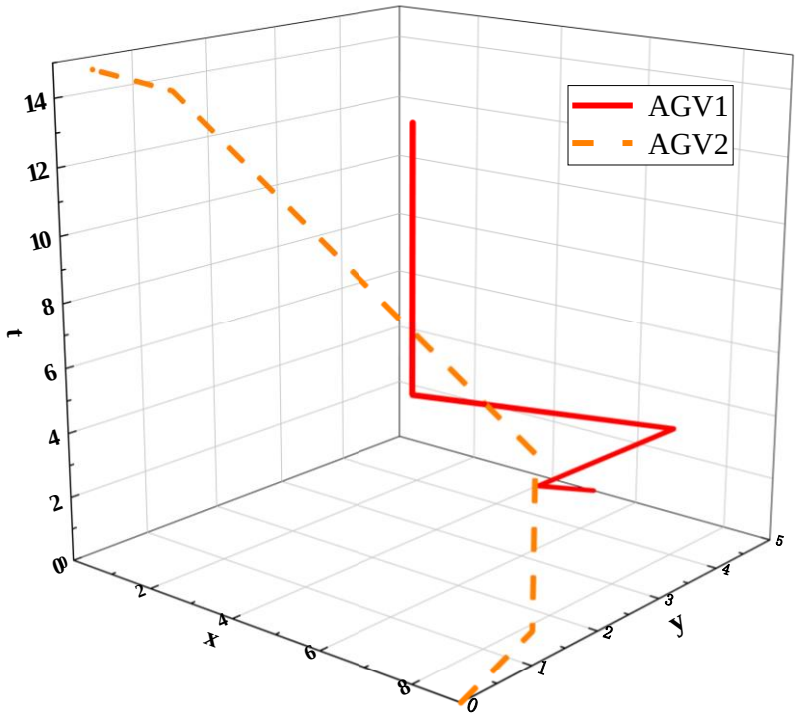


图 5-9 两台 AGV 路线图

针对前两个问题所提供的 AGV 最优路径，在此基础上对所有 AGV 路径进行无冲突规划。将所提算法与 CCRG，GA，SA 进行对比，算法参数如表 5-1 所示。

表 5-1 算法性能参数

参数符号	参数名（单位）	值
$TPAD_{max}$	最大允许延误（秒）	60
Δt	采样时隙（秒）	1
P_c	交叉概率	0.8
ε	线形重组系数	0.5
P_m	变异概率	0.1
σ	局部适应度期望值	0.5
population size	种群规模初值	100
generations	进化代数	100

其中，因为父代两个染色体的基因遗传概率相同，所以线形重组系数 ε 为 0.5。由式 5.7 可知，当前可行解中若存在冲突，那么 $fitness_j^k < 0.5$，由于在实际运行过程中，冲突是需要避免的，因此局部适应度期望值 σ 为 0.5 表示尽可能的选择无冲突的基因组合。

在本文中，由于采用了动态分组策略，在每一轮优化完成后都需要根据冲突关系对 AGV 进行重分组，因此，染色体的长度是不确定的。例如：双 AGV 冲突与多 AGV 冲突所对应的小组中其 AGV 数量不同，采用固定的种群规模会造成计算资源的浪费。自适应种群规模算子如下：

$$s_k = population\_size \times \frac{CS_k}{L_k} \times \frac{L}{CS} \tag{5.9}$$

其中，CS_k 为第 k 组组内冲突数量，L_k 为第 k 组 AGV 数量，CS 为冲突总数量，L 为 AGV 总数量。

在对三种算法进行对比时，其适应度变化曲线如图 5-10 所示。

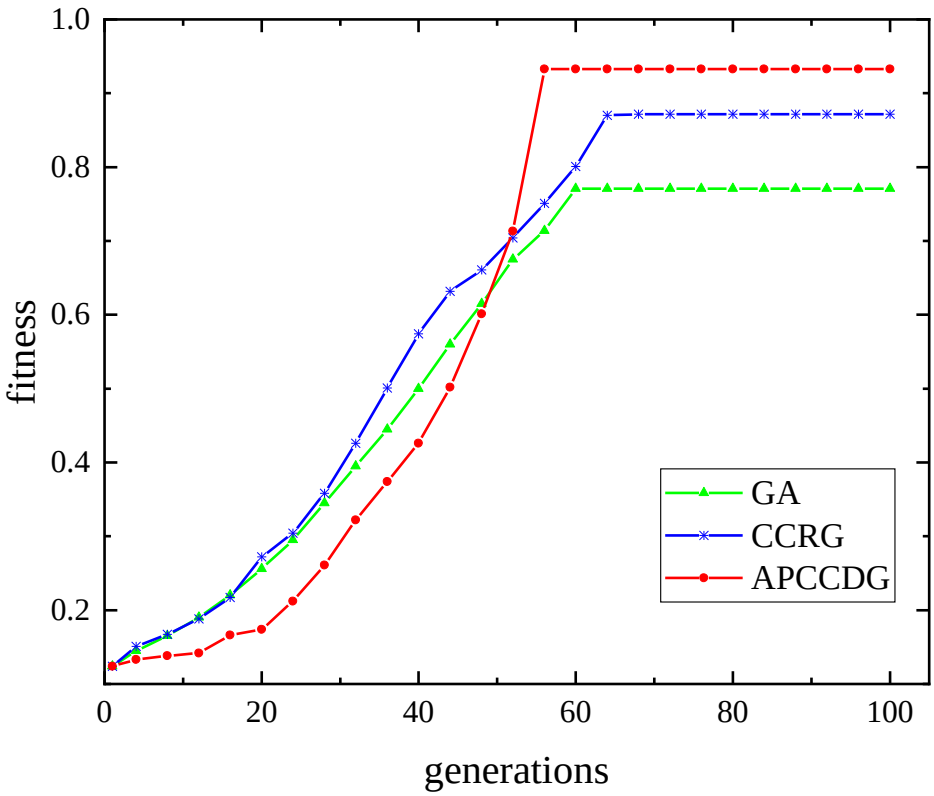


图 5-10 适应度变化趋势图

采用随机分组策略的 CCRG 与 GA 相比，其进化速率显著提高，因此引入分组策略，小组之间分别进化同时进行冲突解脱可以更快速的获取无冲突解。但是随机分组策略具有一定的随机性，采用了动态分组策略的 APCCDG，将相互影响的路径放入同一小组单独优化，小组之间通过协同策略进行选择，具有一定的启发性，因此其进化速度和适应度都要优于 CCRG。因此，采用分组策略可以加快进化速度，同时，具有启发性的动态分组策略更具有优势。

表 5-2 算法性能对比

指标	算法		
	GA	CCRG	APCCDG
冲突数量	3	0	**0**
AGV 工作延误（秒）	406	418	**393**
运行时间（秒）	102	96	**93**

表 5-2 给出了三种算法的优化结果与优化时间，结果均基于 15 次独立重复试验给出。由表可知，本文所提算法可以得到无冲突的解，同时其产生的冲突解脱延误也低于其他两种算法，成本优化了约 6.3%。

6 参考文献

[1] Vivaldini K C T, Galdames J P M, Bueno T S, et al. Robotic forklifts for intelligent warehouses: Routing, path planning, and auto-localization[C]//2010 IEEE International Conference on Industrial Technology. IEEE, 2010: 1463-1468.

[2] Kumar N V, Kumar C S. Development of collision free path planning algorithm for warehouse mobile robot[J]. Procedia computer science, 2018, 133: 456-463.

[3] 熊昕霞. 智能仓库中多移动机器人路径规划研究[D].浙江理工大学,2021.

[4] 申栋栋. 智能仓库多移动机器人拣货路径规划研究[D].北京交通大学,2019.

[5] 孙奇. AGV 系统路径规划技术研究[D].浙江大学,2012.

[6] 孙晓燕,巩敦卫.变种群规模合作型协同进化遗传算法及其在优化中的应用[J].控制与决策,2004(12):1437-1440.