# 2.3-4.30总结

整体分为单AGV路径规划，多AGV路径规划，死锁解除（调度与任务）三部分

# 一、单AGV路径规划

## 功能：

在接收到调度系统的任务后，为AGV寻找符合限制条件的最优路径，尽量避免与其他AGV发生冲突

## 输入：

### 地图信息

struct mapPath

{

double length; //path length

double weight;// weight factor

string type; //

bool block; //Whether the AGV can pass the path

};

以结构体数组形式存储路径信息

### AGV信息

struct AGVinfo {

float speed;

bool backup;//whether AGV can back up

};

### 任务起终点及途径点（货架和工作台）

string sourcePoint;

string destinationPoint;

string shelf;

string workStation;

需要在初始化路径规划类后，设置这四个点信息，起终点必须存在，货架及工作站可选；如果存在货架及工作站，需要分割任务，分段规划路径，最后连接成整段路径

## 输出

输出规划好的路径及所需总时间，路径以数组形式存储

## 核心函数

｛

Getmap（）：查询地图；（可选，如果没有地图传入可使用该函数读取地图）

Dividetask（）：任务拆分（可选，若只有起终点则不需要调用）；

Floyd（）:按floyed算法规划路径,并计算时间；

Ant（）：按蚁群算法规划路径,并计算时间；

Astar（）：按A\*算法规划路径,并计算时间；

Routecheck（）：检查时间窗冲突；

Linkpath（）:连接分段的路径；

｝

## 完成情况

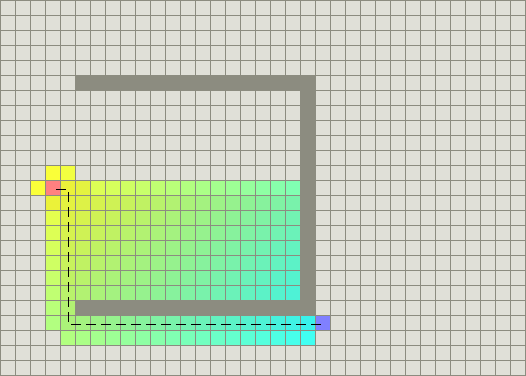
### 任务分割与路径连接

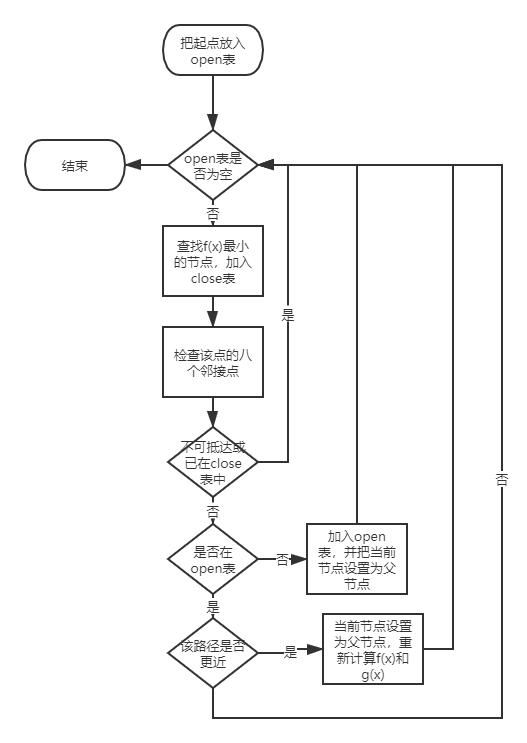
分割任务，分段规划路径，最后连接成整段路径。

作用：在有必经点的情况下，将任务分割为几个子任务，提高运算效率，防止路径过长造成的响应慢的问题

### A\*算法

改进的dijstra算法，f(n) = g(n) + h(n)，在dijstra g(n)基础上加入评估函数h(n)用来加速收敛，





用到open表和close表这两个数据结构，其中open表中存放当前考虑的点及其周边需要添加进来作为可能的路径上的点，close表存放所有不需要再次检查的节点。

1)把起点加入open表;

2)重复如下过程：

(1)遍历open表，查找f(x)最小的节点，把它作为当前要处理的节点，然后移到close表中；

(2)对当前节点的8个邻接点进行检查，如果他是不可抵达的或者已经在close表中，忽略它，否则做如下操作：

①如果它不在open表中，把它加入open表，并且把当前节点设置为其父节点；

②如果它已经在open表中，检查这条路径（即经由当前节点到达该节点）是否更近，如果更近，当前节点设置为其父节点，并重新计算它的g(x)值和f(x)值。如果open表是按照f(x)值进行排序的，则改变后可能需要重新排序。

### Floyed算法

一种动态规划算法，通过遍历地图，不断迭代找到最短路径,时间复杂度O(N3)，因此不适合计算大量数据，在地图较小的情况下表现较好。（一定可以找到最短路径）

for (int k = 1; k != NODE; ++k)

for (int i = 0; i != NODE; ++i)

for (int j = 0; j != NODE; ++j)

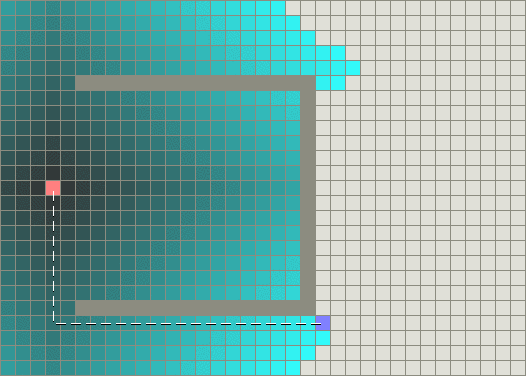
if (distmap[i][j].length > distmap[i][k].length + distmap[k][j].length)

{

distmap[i][j].length = distmap[i][k].length + distmap[k][j].length;

path[i][j]= k;//记录路径

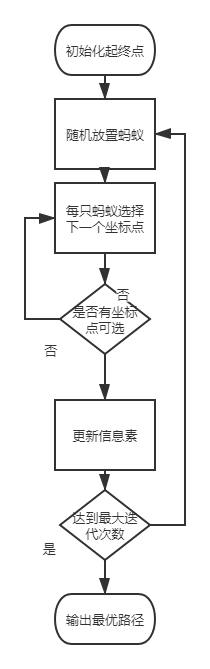
}

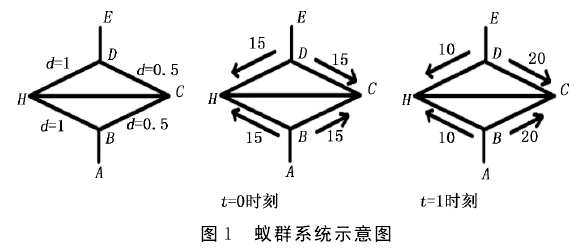


### 蚁群算法

蚁群觅食行为，在路径留下一定浓度的信息素，形成正反馈，

依然适用小规模地图，迭代次数过多容易陷入局部最优





### 遗传算法

优缺点同蚁群算法

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| algorithm | principle | **Advantage** | **Disadvantage** | remarks |
| Dijstra,Floyed | 从目标点开始依次探索所有路径 | 一定可以找到**最短路径**，容易实现 | 时间成本高 | 最短路径不代表最优路径 |
| BFS（最佳优先搜索） | 启发式算法，趋向离目标最近的节点 | 速度较Dijstra**快** | 不一定找到**最短路径** | 基于贪心策略 |
| A\* | f(n) = g(n) + h(n)  h(n)表示从结点n到目标点的启发式评估代价 | 速度**快**，可根据需要调整h(n) | 不一定找到**最短路径** | 常用来做游戏寻路策略 |
| Ant colony | 蚁群觅食行为，在路径留下一定浓度的信息素，形成正反馈 | 全局优化能力、本质上的**并行**性、易于用计算机实现 | 计算量大、易陷入**局部最优解** |  |
| genetic | 生物遗传学和适者生存，通过遗传和交叉变异进行迭代过程的搜索算法 | 易于与其他算法相结合，可进行**迭代**，适应栅格地图，与时间窗结合改进后可找到**最优路径** | 运算效率低，计算量大 | 实际应用中，最短路径的优先级不高 |

## 仍存在的问题

1. seer与Mir的地图只有点，没有路径，需要地图模块解析地图，生成坐标点和路径

# 多AGV路径规划

## 功能：

在已经有多辆AGV运行的场景下，避免新加入的AGV与正在运行的AGV发生冲突，同时找到AGV的最优路径（非最短路径）。通过时间窗算法，在路口和互斥区避让其他AGV

## 输入：

基本同单AGV路径规划，只是加入了时间窗邻接矩阵

### 地图信息

struct mapPath

{

double length; //path length

double weight;// weight factor

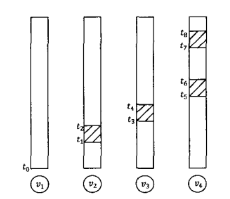
string type; //

bool block; //Whether the AGV can pass the path

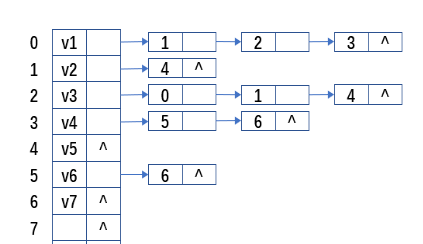
int\* timewindow; //Time period when the path is occupied

};

时间窗逻辑示意：



时间窗存储结构：



### AGV信息

struct AGVinfo {

float speed;

bool backup;//whether AGV can back up

};

### 任务起终点及途径点（货架和工作台）

string sourcePoint;

string destinationPoint;

string shelf;

string workStation;

## 输出：

同单AGV路径规划，输出规划好的路径及所需总时间，增加到达各节点的预计时间，便于调度系统计算时间窗

## 核心算法：

### 时间窗生成算法

已完成，根据路径长度及AGV速度、路口及互斥区等待时间，以邻接表存储节点占用时间情况

待确定：不需要考虑AGV变速情况（起步，停止与转弯）

1）默认AGV匀速行驶；

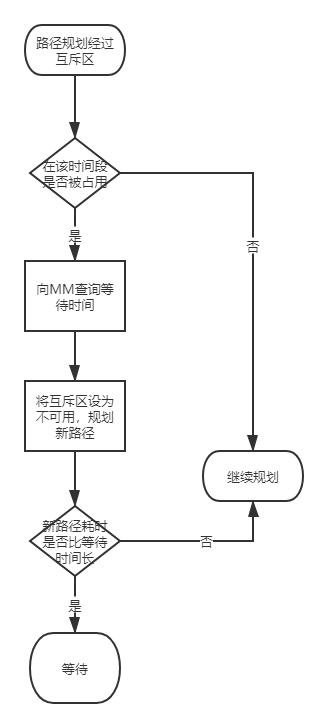
2）AGV在执行任务时，每个节点最多经过一次；

3）AGV在货架的等候时间和机台的工作时间暂不做计算；

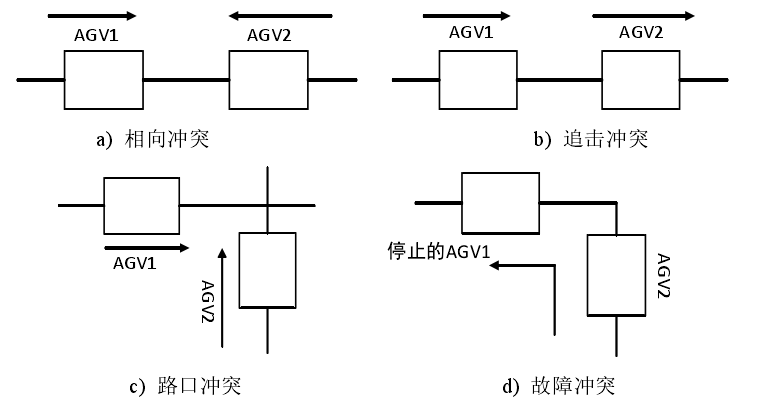
### AGV路径约束算法

已基本完成，包括：

AGV工作区域互斥逻辑，绕行或等待策略



AGV路口避让逻辑，



先到达路口的AGV通过，后方的等待

1）完成任务的路程花费和时间花费的最小化；

2）保证每个任务都有AGV执行（即在仍然有未分配任务时，没有空闲AGV）；

3）保证任务执行完成的总时间不晚于每个任务的截止时间；

4）AGV的电量约束，即任务的完成总路径应小于AGV当前状态下的最大行驶距离（暂时搁置，还不清楚AGV的电量与行驶距离关系）；

整体逻辑：

## 存在的问题

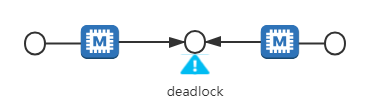
1. 时间窗更新问题，AGV在通过路径及完成任务后，时间窗该由调度系统来更新？
2. 效率问题，在加入时间之后，路径规划需要的迭代次数可能会过多，影响效率，还有陷入局部最优解的问题

# 死锁解除（调度与任务）

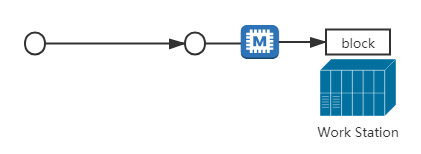
## 死锁场景

### 可能出现的死锁场景

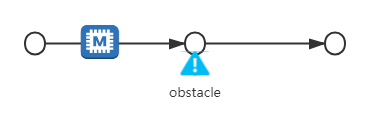
1. 两辆AGV相向冲突，在无向图场景下发生概率较大，有向图几乎不可能发生



1. 互斥区占用时间过长，例如工作站出现问题



1. 故障AGV堵塞路径或节点



### 死锁情况判定

1. A AGV的下一条路径被B占用，且B的下条路径被A占用
2. AGV等待时间过长（需要设定阈值，作为最终的判定条件）
3. 出现未知障碍物（算作异常状态，可以使用deadlock处理）

## 输入

int deadlockPoint;//deadlock point

string lockAGV1;//deaklock AGV1

string lockAGV2;//deadlock AGV2，可选

int AGV2info;//current position

int AGV1info;

mapPath \*map; //map array

地图格式与上相同

## 输出

新的路径，在多AGV情况下，按照避让顺序依次输出

int\* newPath1;//解除死锁的新路径

int\* newPath2;

## 避让逻辑

* 1. 双AGV避让

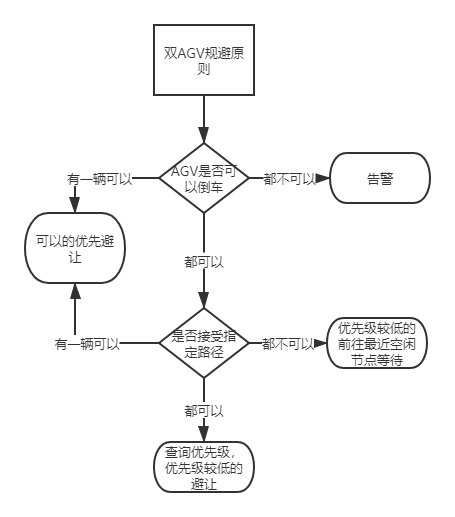
１）锁定该节点，在当前AGV离开该节点前后续AGV不可通过；

２）在锁定该节点的前提下，基于现有时间窗或拥堵为后续ＡＧＶ重新规划执行任务的路线；

３）计算当前AGV通过后一节点的新时刻，记作后一AGV的等待时间，基于此计算该AGV后续的各节点、路径的时间窗，判断与后续路径己知的时路径是否存在冲突，如存在，舍弃该路线；如不存在，暂时保留；

４）判断新生成的路线的时间成本与等待后的路线的时间成本；时间成本小的作为最终的最优路径；

５）生成任务路径命令，将重规划的路径派发给后一AGV。



* 1. 多AGV避让

1)确定冲突点;

2)遍历与冲突点连通的节点;

3)寻找所有在这些path上的AGV，加入冲突AGV表；

4)遍历冲突AGV表，计算每条path上的AGV优先级之和（可以避免优先级最小的AGV所在路径上AGV过多的情况），

5)循环进行如下操作：若AGV数量大于二，循环;若等于1，结束;

(1)寻找优先级最小的path；

(2)将该path上的AGV按到冲突点的距离（根据AGV当前位置计算）push避让AGV表；

(3)pop避让AGV表，避让AGV将冲突点设置为不可用，依次重新规划路径（这样可以做到整条路径上的AGV全部避让，减少循环次数）

## deadlock类核心功能

void resetmap(string lockpint);//重设地图

void newpath();//给出新路径

string unlock(int status);//判断应该避让的AGV

string predict(int AGV1position, float angle);//预测

### unlock(int status);与调度系统接口，

按优先级与AGV是否可以接受路径确定避让AGV（mir需要起终点，可以自主规划路径；Seer需要给定路径），

status分三种情况

status=1：两车相向冲突；

status=2：障碍物冲突；

status=3：等待时间过长

### void resetmap(string lockpint);重设地图，

将避让AGV周围的路径改写为双向路径（要解除死锁必须要倒车，因此单向路径无法使用）

### void newpath();退避路径的选择，

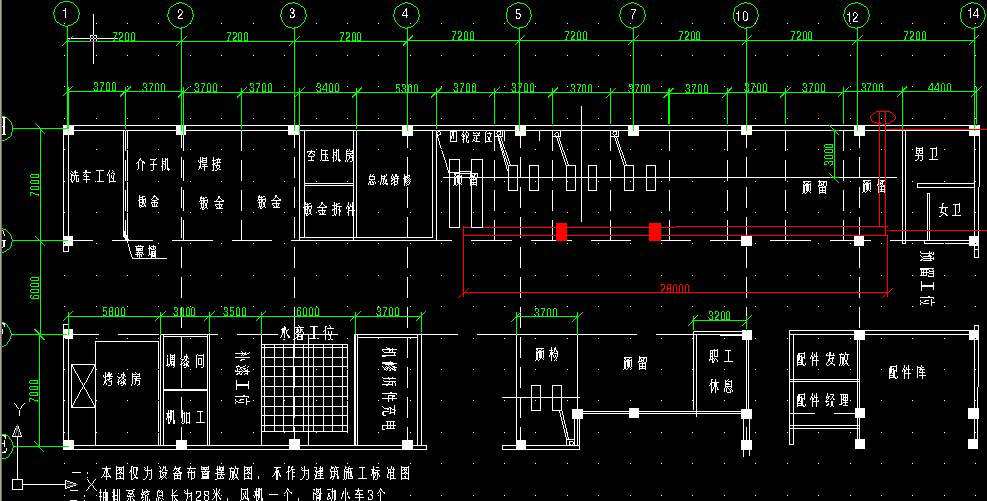
将死锁点设为不可用，规避死锁点，规划新的路径以解锁，调用路径规划类，传入修改后的地图，增加不可通行情况下的路径规划。解锁路径规划的算法拥堵系数的权重应尽量提高，以避免对正常运行的AGV产生影响

### string predict(AGVinfo AGV1info);预测节点

通过AGV当前位置与角度，遍历地图，查找其运动方向上的节点来预测AGV要经过的节点，以判断是否死锁及查找死锁点

* 1. 死锁场景模拟

1. 目前模拟方式为使用网络上的实际工厂地图，按照其布局规划合理路径，模拟以上三种死锁情况，向deadlock类传递两个虚拟AGV信息，测试是否能解除死锁及解除时间。
2. 在完善多AGV死锁逻辑后，完成多AGV死锁测试，目前测试五台车在三条相交路径随机分布的死锁



## 任务评价

作用：评价任务完成情况，给出AGV数量评判标准

1、解析法

将具体问题归纳整理成数学模型,通过对数学模型的求解,计算出AGV小车的数量。

Tr= Tw+ Tk+ Tm+ Tz。

式中，Tw为小车等待时间; Tk为空车行驶时间;Tm为满车行驶时间(满箱负载行驶时间及空箱负载行驶时间);Tz为物料转换时间(线旁空满箱切换及空箱卸载时间)。

其中为任务派发时间，为任务截至时间，为任务实际完成时间

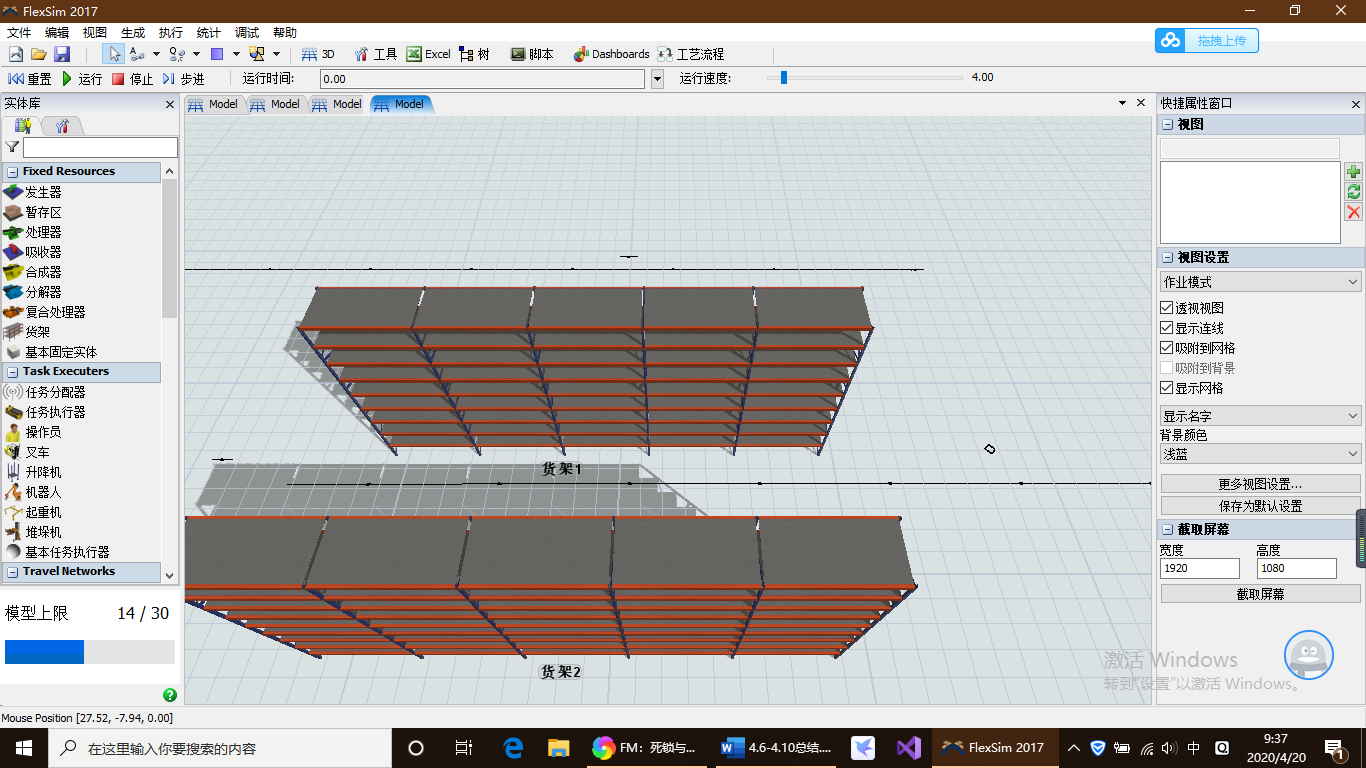
相对于任务组优先级系数、截止时间系数以及等待时间系数的任务执行顺序排序方法，从而保证任务执行的实时性和效率

在任务完成后，通过计算任务的完成效率，来确定当前的AGV数量是冗余还是较少，方便后续的调整。

2、仿真计算（限制太大，搁置）

(1)Flexsim软件，基于windows，用来模拟工作流及制造业的虚拟环境

<https://www.flexsim.com/>



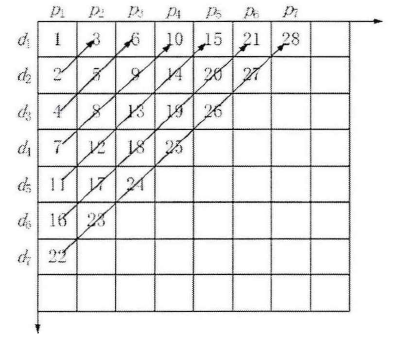
试用版有30个实体（包括货架、节点、路径）的限制，基本不可能用在实际运行的情况中，暂时搁置

(2)OpenTCS，之前有过介绍，开源，但是编程语言是JAVA，目前的算法全部使用C++

3、实际运行后根据工作效率进行调整

## 任务优先级

作用：用来处理任务过多时的AGV分配问题以及AGV在发生冲突时的避让顺序



i为初始优先级顺序，j为截至时间排序

发生冲突时，优先级低的AGV避让

优先级相同或极近时，遵循先来先服务（FCFS）策略

## 存在的问题

1. 不清楚AGV是否可以给出接下来要经过的坐标点，
2. 效率问题，加入动态优先级后会占用资源，需要确定更新时机，以避免不必要的浪费
3. AGV数量评价仍然没有定量的模型

# 工作计划安排

## 五月：

1. 完成调度系统，完善时间窗算法和拥堵系数，在回到公司后根据server表现进行优化，尽可能减少响应时间
2. 任务优先级与路径规划的融合
3. 确定地图的规划要求，尽量预防死锁和拥堵、冲突发生的可能

## 六月：

1. 测试功能，优化算法表现
2. 总结工作