

# 人工智能导论

## 第二次大作业实验报告

(2023 - 2024 学年度 春季学期)

实验名称 智能车运送货物

姓名	张章
学号	2023010916
院系	自动化系
教师	张长水
时间	2024/5

## 目录

## 1. 摘要

本实验旨在设计一种算法完成对多智能体的路径规划，使得各个智能体能够无冲突地完成将货物从一个仓库转移至另一个仓库的任务。在本实验中，作者设计了一种方法将该问题转换成一系列经典的多智能体路径规划问题，并使用基于冲突的搜索算法解决这一系列的多智能体规划问题。实验结果和项目仓库已经上传至 Githubb，请通过以下链接查看：

## 2. 相关工作

## 3. 定义实验任务

大作业的作业题目并没有对本次实验要完成的内容做出详细的指示，因此在介绍实验方法前，作者将先对实验任务进行定义。

本次实验中，我们将随机生成一幅地图，地图上分布着由等大的小正方形组成的障碍物以及仓库  $A$  和仓库  $B$ 。我们假设在初始状态下在仓库  $A$  存在着  $N$  件货物。 $M$  辆无人货车随机地分布在地图上的各处，它们在单位时间内能够向上下左右四个方向移动的距离固定，每辆货车最多只能装载一个货物。考虑到实际情况，我们认为  $M$  应当小于  $N$ 。因此，我们的任务可以定义为，规划所有小车的路线，使它们在两个仓库之间来回折返，运送货物，直到所有的货物都移动至  $B$  仓库为止。且这些货车的路线应当满足：1) 小车路线要能自主避开障碍物；2) 这些路线两两之间并无冲突，即两辆货车不会在同一时间占据同一个位置，也不会某一时刻彼此交换位置，如图 1 所示；3) 在满足前两个条件的前提下，路径花费的代价应该尽可能少，也就是所有小车移动距离的总和最小。

在实现对所有货车路径的规划之后，我们还需要使用可视化的方法展示。



图 1: 图中展示了两种冲突的可能，左侧表示两小车在同一时刻占据同一位置，右侧表示两小车在某一时刻互换位置

## 4. 实验目标

## 5. 实验方法

本次实验中，作者将本次实验的任务分解成一系列经典的多智能体路径规划（Multiple Agent Path Finding, 以下简称 MAPF）问题，然后使用基于冲突的搜索（Conflicts-Based Search, 以下简称 CBS）算法解决这些 MAPF 问题，最终得到整个问题的解。CBS 的底层寻路算法，作者使用了时空联合 A\* 算法。

## 5.1 全局算法

单个小车的状态和仓库 A 状态的组合有下述四种情况：1) 小车没货，仓库 A 有货 2) 小车有货，仓库 A 有货 3) 小车有货，仓库 A 没货 4) 小车没货，仓库 A 没货 上述四种情况对应小车的三种任务：1) 从当前位置去仓库 A 取货 2) 从当前位置去仓库 B 送货 3 与 4) 从当前位置前往所有小车的集合点 如果小车有货，就要去 B，如果没货再判断，如果此时所有小车记作  $\mathbf{c}$ ，在某一时刻  $t_0$ ，所有的小车的状态记作  $\mathbf{x}_n$ ，所有小车完成当前任务之后的状态记作  $\mathbf{x}_{n+1}$ （此时我们假设先到达目标位置的小车在到达之后暂停运动），可以发现，从  $\mathbf{x}_n$  到  $\mathbf{x}_{n+1}$  的过程是一个标准的 MAPF 问题，我们可以基于 CBS 算法，计算得到它的解  $S$ 。现在假设在上述解  $S$  中，小车  $c_i$  在  $t_1$  时刻率先到达了自己的目标位置，并更新自己的状态为  $x_n^{(i)'}$ ，上述解此时还没有被全部遍历完，已经遍历的部分解记作  $S'$ ，该小车更新后整体的状态变成  $\mathbf{x}'_n$ ，整体目标状态变成  $\mathbf{x}'_{n+1}$ ，可以看到这又是一个经典的 MAPF 问题。按照上述的思路，我们算法的过程可以这样描述：

---

**Algorithm 1** Get Global Solution

---

**Input:** game map matrix:  $\mathbf{M}$ ,

a list of all agents' states:  $\mathbf{A}$

**Output:** Global Solution  $\mathbf{G}$ , a two-dimension array of coordinates.

- 1: initialize Global Solution  $\mathbf{G}$
  - 2: **while** is not concluded **do**
  - 3:   caculate Partial Solution  $\mathbf{P}$
  - 4:   concatenate  $\mathbf{P}$  to  $\mathbf{M}$
  - 5:   update states of all agents.
  - 6: **end while**
  - 7: **return**  $\mathbf{G}$
- 

## 5.2 局部解算法——CBS 算法

在上面，我们已经将本次任务转化成一系列经典的 MAPF 问题，我们使用 CBS 算法来解决这个问题。

CBS 算法是一种两层算法，底层使用传统的寻路算法为单个智能体规划路径，顶层则通过维护一个冲突树来解决各个智能体路径之间的冲突。为了更详细的介绍 CBS 算法，我们先规定以下术语：*path* 表示单个智能体找到的路径，*solution* 为所有智能体路径的集合，*cost* 表示当前解的总成本，*conflict* 表示路径之间存在的冲突，*constraint* 表示由冲突产生的对单个智能体寻路的约束，由一个三元组  $(i, v, t)$  表示，即智能体  $i$  在  $t$  时刻不允许占据顶点  $v$  的位置。

我们先来介绍算法的顶层，顶层算法维护一个 *open* 表，其中每一个结点都包含一组 *constraint*，一个 *solution* 及其总成本 *cost*。算法刚开始时，我们不给智能体添加约束，生成一个 *solution* 并将结点放入 *open* 表中。此后，我们每一次都从 *open* 表中取出成本最低的结点，如果该结点的解没有冲突，那这个解正是我们要求的，如果有冲突，我们将通过该冲突生成两个约束，由此原来的结点就会分叉成两个子结点（见图 2），我们将子节点重新放入 *open* 表中，重复上述过程。

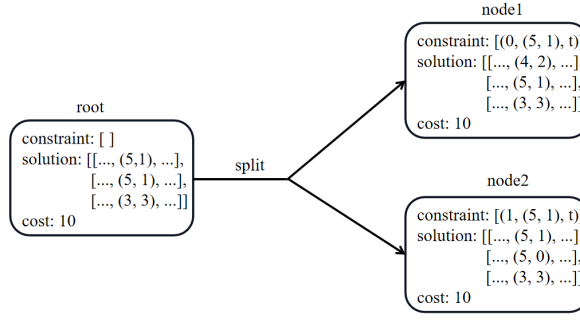


图 2: 根节点中智能体 0 和智能体 1 存在路径上的冲突, 由此产生了两个约束, 通过这两个约束重新生成结点 1 和结点 2

---

### Algorithm 2 Get Partial Solution

---

**Input:** game map matrix:  $\mathbf{M}$ ,  
a list of all agents' states:  $\mathbf{A}$   
**Output:** Partial Solution  $\mathbf{P}$ , a two-dimension array of coordinates.  
// definition of Node:  
// - class attributes: constraints, cost, solutionmayincludecollisions  
// - class methods: get first collision, split collision to constraints,  
initialize Partial Solution  $\mathbf{G}$   
2: initialize *openlist*  
initialize *rootnode*, push it in *openlist*  
4: **while** *openlist* is not empty **do**  
get least costly node  $n$  from *openlist*  
6: remove  $n$  from *openlist*  
**if** node  $n$  has no collision: **then**  
8: then the partial solution has already been founded  
**else**  
10: get constraints  $\mathbf{c}$  based on the first collision  
**for** constraint  $c$  in constraints  $\mathbf{c}$  **do**  
12: new constraints  $\leftarrow$  old constraints of node  $+ c$   
create new node based on new constraints  
14: push new node in *openlist*  
**end for**  
16: **end if**  
**end while**

---

### 5.3 CBS 的底层寻路——时空联合的 $A^*$ 算法

## 6. 实验结果展示

## 7. 回顾与展望

•

实验结果如图??所示

图 3: 图

图 4: 图

图 5: 图

---

**Algorithm 3** Space-Time A\* Path Finding

---

**Input:** Game map matrix: **M**,  
A single agent: *a*  
A constraints table **c**  
**Output:** A optimal path of agent *a*  
// definition of Node: (diffrent from Node in CBS algorithm)  
// - class attributes: current position, target, parent node, time, total cost  
// - class methods: find neighbours, return all available position for next time  
initialize *openlist*  
initialize *closedlist*  
3: initialize *rootnode*, push it in *openlist*  
  **while** *openlist* is not empty **do**  
    get least costly node *n* from *openlist*  
6:   remove *n* from *openlist*  
    get *neighbours* of node  
    **for** *neighbour* in *neighbours* **do**  
9:     **if** position of *neighbour* equals target of node **then**  
      path **p** has already been founded  
    **else**  
12:     add *neighbour* in *openlist*  
    **end if**  
  **end for**  
15: **end while**

---

图 6: 图

$$y = ax + b \tag{1}$$

$$\left\{ y = ax + b \tag{2}$$

$$y = ax + b$$

表 1: 表


1

2

```
def main():  
    print('Hello World!')
```