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　2024/5/30

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。项目仓库已经上传至Githuhb，请通过以下链接查看：https://github.com/SuoRuGithub/agents-deliver-goods/tree/main

2. 定义实验任务

大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，如图 1 所示。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，直到所有的货物都从仓库 A 移动至 B 仓库为止。且这些小车的路线应当满足：1）小车要能自主避开障碍物，也即路径与障碍物无冲突；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，也不会在某一时刻彼此交换位置，如图 2 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。

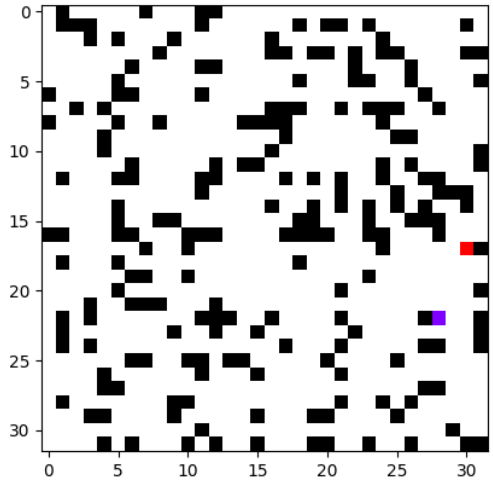

图 1: 一个地图的示例，图中黑色部分表示障碍物，紫色方格表示仓库 A，红色方格表示仓库 B

图 2: 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

3. 实验方法

本次实验中，作者将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，然后通过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，得到整个问题的解。CBS 是一种两层算法，底层寻找单个智能体的最佳路径，顶层负责处理路径之间的冲突。作者在 CBS 的底层使用了时空联合的 A*（Space-Time A*）算法进行单个智能体的寻路。

3.1 全局算法

经典的 MAPF 问题需要规划所有智能体的路径，使得它们能够无冲突地从起始位置到达目标位置。与之相比，本次实验中小车需要在两个仓库之间来回折返，终点位置也在不断变化，因此作者要先将本实验问题转化成一系列经典的 MAPF 问题。

小车是否载货和仓库 A 是否还有剩余货物的情况有以下几种组合：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。组合的情况决定了单个小车的状态：1）小车需要从当前位置去仓库 A 取货 2）与 3）小车需要从当前位置去仓库 B 送货 4）任务已经完成，无需移动。

现在我们将所有小车记作 $\mathbf{c}$，在某一时刻 t_n，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，而不会更新自己的状态），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 $\mathbf{S}$。现在假设在上述解 $\mathbf{S}$ 中，小车 c_i 在 t_{n+1} 时刻率先到达了自己的目标位置，并更新自己的状态为 $x_n^{(i)\prime}$，此时，上述解并没有被全部遍历完，我们将已经遍历的部分解记作 $\mathbf{S}'$，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。如此重复，当仓库 A 已经没有货物且所有小车都不再装载货物时，算法结束。将所有的部分解拼接在一起，我们就一个得到全局解 $\mathbf{G}$。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: **M**,
a list of all agents' states: **A**
Output: Global Solution **G**, a two-dimension array of coordinates.

```
initialize Global Solution G
while not every agent has concluded its task do
    caculate Partial Solution P with CBS algorithm
    concatenate P to G
    update states of all agents.
end while
return G
```

3.2 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 MAPF 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 *solution* 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 $(i, \mathbf{v}, \mathbf{t})$ 表示，即智能体 i 在 t 时刻不允许占据顶点 $\mathbf{v}$ 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，则说明 *solution* 已经被找到了，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），每个子节点在继承父节点已有的 *constraint* 的基础上加入了新的约束。我们将子节点重新放入 *open* 表中，重复上述过程，直到没有冲突为止。

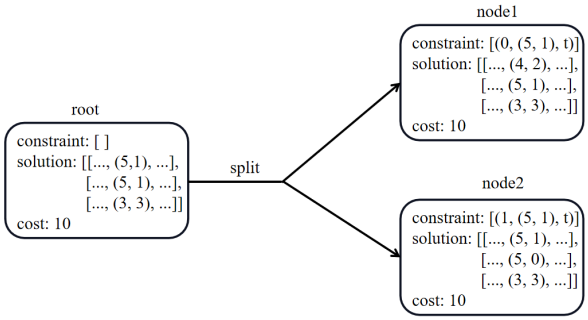

图 3: 根节点中智能体 0 和智能体 1 存在路径上的冲突，由此产生了两个约束，通过这两个约束重新生成结点 1 和结点 2

Algorithm 2 Get Partial Solution

Input: game map matrix: **M**,
a list of all agents' states: **A**
Output: Partial Solution **P**, a two-dimension array of coordinates.
// definition of Node:
// - class attributes: constraints, cost, solution(may include collisions)
// - class methods: get first collision, split collision to constraints,
initialize Partial Solution **G**
initialize *openlist*
initialize *rootnode*, push it in *openlist*
while *openlist* is not empty **do**
  get least costly node n from *openlist*
  remove n from *openlist*
  if node n has no collision: **then**
    the partial solution has already been founded
  else
    get constraints **c** based on the first collision
    for constraint c in constraints **c do**
      new constriants ← old constraints of node + c
      create new node based on new constriants
      push new node into *openlist*
    end for
  end if
end while

3.3 CBS 的底层寻路——时空联合的 *A** 算法

接下来我们介绍本次实验中 CBS 算法的底层部分，也就是为单个智能体寻找路径的算法。诸如 A* 算法和 Dijkstra 算法都能很好地完成寻路的任务，但是它们的缺陷在于它们处理的地图不能随着时间发生变化。在本次实验中，每个小车的移动需要考虑其他小车的位置，也就是地图上的障碍物会发生变化。因此，本次实验中 CBS 的底层寻路算法需要使用时空联合的 A* 算法而并非传统 A* 算法。

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 avilable position for next time
initialize *openlist*
initialize *closedlist*
initialize *rootnode*, push it in *openlist*
while *openlist* is not empty **do**
  get least costly node n from *openlist*
  remove n from *openlist*
  get *neighbours* of node
  for *neighbour* in *neighbours* **do**
    if position of *neighbour* equals target of node **then**
      path **p** has already been founded
    else
      add *neighbour* in *openlist*
    end if
  end for
end while

4. 实验结果展示

由于作者不会使用 LaTeX 插入动图，实验结果展示请看上传的作业文件中的/Demo 文件夹，或在项目的 Github 仓库：https://github.com/SuoRuGithub/agents-deliver-goods/tree/main 中查看。

5. 回顾与展望

本实验利用 CBS 算法，解决了货车运货的实际问题。不过，本次实验仍然存在着许多不足。

首先，实验代码的鲁棒性提升空间巨大。目前的程序只有在地图不太大，智能体和货物的数量不太多，地图障碍物并不是非常稠密的情况下才能找到解。作者使用了一些方法避免这些问题，但直到目前还没能够完全解决。

其次，实验中找到的解往往不是最优的。在项目的展示中也可以看到，货车的路径往往会包含一些不必要的步骤，这也许是因为作者在避免程序陷入类似“蚂蚁怪圈”的死循环时加入的随机性导致的。

此外，对于较大的搜索任务，本实验采用的方法较为耗时。

在前期调查的过程中，作者了解到 MAPF 问题解决方法主要有集中式规划的方法与分布式执行的方法。本实验中采用的 CBS 算法就是集中式规划方法中的一种情况，它仍然是以搜基础的。作者希望在经过未来的学习之后，能够使用基于强化学习的分布式执行的算法继续完善本次实验，解决上述提到的种种问题。