2024 华为软件精英挑战赛赛后复盘

西南赛区 重庆大学 CQU-42IsAllYouNeed
刘怡鹏 蒋佳宏

1 参赛初衷

软挑每年的赛题都很有意思，也很有价值，有很多同学参加。我们这次是第三年参加软挑，在一看到开始报名的消息时就报名啦。这次我们都大四保研了，相对压力小一些，时间比较充裕，因此希望能在本科阶段不留遗憾，冲个不错的名次。

2 解题思路及方案

今年的题目是关于智慧港口的，涉及到对机器人和轮船的购买和控制，以最大化收益。下面我们以赛段顺序回顾赛题和我们的解题思路及方案。我们将代码开源到了 https://github.com/MineQihang/CodeCraft2024-public，实现细节可以参考我们的代码，想讨论的地方欢迎提 issue，也欢迎对我们解决方案感兴趣的朋友给个 star。

2.1 初赛

在 200×200 的二维网格地图上，有陆地（空地和障碍）、海洋和泊位。机器人（10 个）捡取并运输货物，船（5 艘）将泊位上的货物运至虚拟点产生价值。机器人需要考虑捡哪个货物、送到哪个泊位以及其路径规划；船需要考虑去哪个泊位，如果没装满是去其他泊位还是直接回虚拟点。

2.1.1 机器人调度

我们机器人的基本逻辑是首先回到泊位（主要是方便使用预处理出来的泊位到其他点路径得到去货物路径），然后找货物，再找回到的泊位，以此往复。我们考虑以下几个部分：

1. 货物决策（当自己身上没有货物时要考虑去捡哪个货物）
2. 泊位决策（当自己拿到货时考虑要去哪个泊位）
3. 路径规划（去货物和泊位的路径或要执行的操作序列是什么）
4. 防撞避让（如何规避不同机器人之间的碰撞问题）

下面我们依次阐述。

货物决策。之前我们是在机器人到达泊位时再进行查找，但发现这样存在一个问题：由于机器人到达泊位时间有差异，机器人会去一个距离当前泊位很远的货物，而这个货物实际可以被分配给另一个距离他很近的机器人。因此，我们考虑在机器人选定泊位开始返回的路上就一直对每个泊位动态分配货物，机器人到达泊位放下货物后立即选定已经分配的货物前往即可。这里泊位与货物的二分图匹配我们本来准备使用匈牙利算法，但算了算时间复杂度感觉不对

劲，因此用了简单的贪心方法：我们计算每个泊位到货物的决策因子（即货物价值除以预估到达距离），然后对其进行从大到小排序，依次选定每个泊位与货物的匹配关系。注意预估到达距离中不包含拿到货物后的回泊位距离，测试发现加上这个效果会变差。需要注意的是，我们为每个泊位分配与前往它的机器人数量相等的货物，以避免两机器人同时到达造成的无分配货物问题。

泊位决策。拿到货物去最近的泊位即可。我们会在后期禁用一些泊位，使得机器人不前往它们，从而使货物集中在几个固定泊位，提高轮船运输效率。

路径规划。我们使用 BFS 预处理泊位到其他点的距离，在机器人寻路时因为都是以泊位作为目标或者出发点，所以可以沿距离下降从而找到路径（以泊位为出发点时，需要反向做，然后翻转路径移动操作）。

防撞避让。由于路径规划中没有考虑碰撞，我们需要在快发生碰撞时进行避让。我们使去泊位和从泊位出发的机器人的路径偏好不同，比如一种优先左右运动，一种优先上下运动，以降低碰撞可能。机器人相撞有两种情况，一种是去的方向相同，一种是去的方向不同。对于前者，只需让其中一个机器人原地不动即可。对于后者，执行随机避让。由于去货物的机器人的路径是固定的，它避让后会原路返回，可能产生时间浪费，我们优先让去泊位的机器人避让。多个机器人相遇时为其中一个设定最高优先级，其他机器人都为它让路，如果无可行让路方法，则转让优先级再次规划。

2.1.2 轮船调度

在初赛中，轮船无需考虑路径规划，因此对轮船的调度至关重要。具体来说，我们需要解决以下问题：

1. 在虚拟点时，去哪个泊位？
2. 在泊位时，要不要切换泊位？切换到哪个泊位？是否该去往虚拟点了？

针对第一个问题，我们会统计泊位效率（每帧泊位产生的货物价值），然后用运输时间乘以效率加上当前价值来预估到达时的泊位价值，选择预估泊位价值最大的前往即可。

针对第二个问题，我们直接在装满或最后时间离开泊位，前往虚拟点。

我们在使用以上策略时，会导致最后的时间被浪费，泊位剩余较多货物，大大降低分数。因此我们通过已知效率手动调度，每只船可以在两个或者一个泊位间往返运货，切换泊位的时间点由最后时刻进行逆推。最后泊位会逐步被禁用（机器人不再去放货），因为船不再有机会去那。需要注意的是，除非船的容量很小，不然泊位间轮船的调度是需要的。这一部分是我们初赛提分的关键部分，大幅降低了泊位剩余价值。

2.2 复赛

复赛有几大变化点：

1. 轮船与机器人不再直接生成在地图上，需要选手支付资金在特定的地点进行购买。
2. 轮船移动和泊靠需要我们自行调度。
3. 引入了新的地块，比如航行速度更慢的主航道等。
4. 机器人的碰撞惩罚变小。

我们没有对机器人策略做太多修改，只是修改了代码实现方式。主要修改

点在机器人和船的购买、船的调度上。

2.2.1 机器人与轮船的购买

我们针对每张地图手动确定机器人和轮船的数量，然后优先购买轮船，如果购买后还有钱可以买机器人则购买机器人。因为有很多购买点，因此我们需要确定从哪个购买点购买，我们对海和陆地都从购买点出发进行了分块，然后根据每个块中的购买比例（即块中机器人/船的数量除以预期购买数量）决定在哪个购买点购买。没有自动购买机制，因为我们发现机器人的数量对结果影响很小，不如把它作为超参数。

在复赛正式赛中，新增了一种价格为 5000 的可以载两件货的机器人。我们通过数值计算发现，买这个机器人不如买两个普通机器人，买这个反而会亏 1000。因此，我们直接不考虑这种机器人。

2.2.2 轮船的调度

我们需要解决以下问题：

1. 轮船在交货点时要去哪个泊位？
2. 轮船在泊位时什么时候离开去泊位或交货点？
3. 轮船如何寻路和避让？

针对第一个问题，我们考虑每个泊位预期价值与预计时间的比值，取最大的作为目标泊位。预期价值由已有货物价值和预期未来价值组成，未来价值通过过去的时间加权价值预估。预计时间由前往时间、装载时间和交货时间组成。

针对第二个问题，在前期，我们想尽快早的购买机器人，因此当去到的泊位装好货（泊位没货了）并且船上价值已经够买机器人时就会返回交货点交货，以尽快获得价值。如果船装满了显然是需要回到交货点的。在后期，如果泊位没货了则需要考虑进行泊位间转移，与前述的交货点到泊位一样，我们仍然通过考虑泊位预期价值与预计时间的比值以选择去往的泊位。

针对第三个问题，因为轮船只会从购买点或泊位出发，所以我们使用 Dijkstra 算法预处理所有购买点和泊位到其他点的距离和路径，然后在后续直接回溯即可。需要注意的是，由于船具有方向，到达每个点时（核心点与该点重合）的方向可能有所不同，所以执行 Dijkstra 算法时每个点都是点坐标+方向组成的。对于多船避让问题，我们会对每艘船后一个时刻的位置进行预测。检测到即将碰撞时会随机选择一个方向进行避让，避让后回到原来的点继续行驶。

由于之前我们一直考虑单船，几乎没有考虑多船的调度问题，新写的轮船调度和避让欠佳，在复赛正式赛中分数没上 30 万，比较遗憾。

2.3 决赛

决赛相对复赛有很大的变化，很多前述的算法都会存在超时的情况。决赛主要变化点如下：

1. 多队 PK，送至交货点的货物价值由机器人方、船方平分。
2. 贵重物品取货时需要调用大模型回答问题。
3. 地图大小变化，从 200×200 变为 800×800。

多队 PK 和地图大小变化是使得前述算法失效的主要原因。由于地图变大，程序运行时间不再够用，算法基本上重写了。而机器人和轮船的调度也因为 33

队同场抢夺有很大变化。

我们在练习赛中一直学习排名第一赛队的策略，基本上由多船策略转为珍贵货物送自己策略。多船或全船策略被称为“海贼王”，选手会购买大量轮船来抢夺泊位上的货物。多机器人或全机器人策略被称为“机甲王”，选手会大量购买机器人以抢夺陆地上的货物。珍贵货物送自己策略是机器人在场上游荡等待珍贵货物产生，并力争第一时间抢到珍贵货物，然后运送给有自己船的泊位。

为了保证统计的准确性，我们设计算法时要求不能发生任何跳帧。

我们需要解决以下几个问题：

1. 机器人与船的购买
2. 机器人的决策（决定去哪个地方拣货和游荡）
3. 机器人的寻路与避让（如何避开其他选手的机器人）
4. 船的决策（决定去哪个泊位待着，什么时候离开）
5. 船的寻路与避让（如何高效寻路和避开其他选手的船）
6. 大模型问答（如何回答问题，给出正确选项）

下面仅介绍我们最后提交的策略。

2.3.1 机器人与船的购买

我们针对每张地图手动确定机器人和轮船的数量，然后按照比例购买。具体来说，我们计算当前船数量与预期船购买数量的比值、当前机器人数量与预期购买机器人数量的比值。在钱数足够的情况下，如果前一个比值大于后一个比值则购买船，不然购买机器人。要是买船后还有机会买机器人也进行购买。

我们尝试了动态确定机器人与轮船的购买数量，但实现出来效果一直不是很好，因此放弃了这种方式。

2.3.2 机器人的决策

因为我们采取的是抢占珍贵货物的策略，所以需要找到最有可能产生珍贵货物且能被我们抢到的点。我们通过网格均匀选点然后从点开始 BFS 扩散的方式将地图划分为数百个块（patch）。对于每个块，我们会在每一帧计算它的潜在价值（通过块大小、块中我方机器人数量、对方机器人数量等进行估算），然后机器人会选择潜在价值最高（也可以说是最稀疏）的块前往。当发现高价值货物时则尽快前往即可。在前往稀疏块的过程中，可能身边会产生高价值货物，因此我们引入了一个信号机制：当高价值货物产生时，我们会通知附近的机器人前往。这样避免了错过货物的问题。

2.3.3 机器人的寻路与避让

分块之后，我们会预处理出所有块的连接图。当机器人选择前往稀疏块时，会进行从当前点到下一块中心的路径搜索。需要注意的是，我们不用搜索从当前点到目标块中心的路径，因为最优路径在珍贵货物匮乏的决赛中并不是必要的，反而会增加时间复杂度。而且，由于只需要前往下一块中心，而当前块和下一块是相邻的，所以得到这个路径是非常快速的。因为货物离机器人相对都是很近的，所以机器人到货物的路径也可以很快得到。

机器人会在每帧检测碰撞，检测到碰撞后，会随机选择原地停留、重新规划路径、随机避让或重复原命令。重新规划路径会将其他机器人视为障碍，然

后从当前点搜索到目标点的路径。随机避让就是随机选择一个方向前往，并不一定能避免碰撞。

2.3.4 船的决策

由于场上船过多，为尽可能抢占泊位，轮船平时装满才走。同时如果我们的船不是最近的非装载状态船只，则泊位上剩余的货物不被考虑。我们还是考虑预期价值与预计时间的比值，取最大的方案。预期价值包含自家机器人所带来的价值。预计时间由前往时间、等待时间、装载时间和交货时间组成。等待时间是指泊位上已有其他船而产生的等待时间。船只不会去靠泊区有更小 ID 船的泊位；己方不会出现两艘船都在一个泊位等待的情况。

船只决策可分为非实时决策和实时决策。非实时决策是我们初赛、复赛一直使用的，船仅在出发时刻决策，中途不会改道。而实时决策则会进行每帧的决策，船装满才走的条件也不会成立。决赛练习时我们发现实时决策的效果并不好，但正式赛时观察到第一名使用了实时决策，于是我们也在机器人和船未购买完成时使用实时决策以加快前期积累速度（实验发现全程实时决策分数仍然不高，可能实现细节有问题）。

2.3.5 船的寻路与避让

由于预处理的时间不足以对所有泊位跑 Dijkstra、轮船可能会中途改道，所以我们设计了一种时间上高效的截断 A*算法，大幅提升了寻路速度。我们先对所有购买点和泊位做 BFS（注意这里不用考虑方向），用它的结果作为之后进行 A*的代价函数。当船没有路径时，会做一个带步数中断的 A*，再掐掉 A*得到的路径的尾巴（避免船的卡顿问题，因为最后几步会考虑到达点的轮船方向），这样就得到船接下来十几步的路径了。这样依次做下去就能将船的寻路分配在不同的帧数，避免了跳帧。

船在每帧都会检测碰撞，当碰撞发生时，会随机选择原地停留或重新规划路径。重新规划路径会将其他船视为障碍，然后调用 A*算法，避免后续继续碰撞。

2.3.6 大模型问答

这部分我们做的不是很好，观察发现我们应该是所有队中回答速度较慢（练习赛回答时间 20 帧左右）、精度较低（练习赛正确率 92%）的队伍。赛后我们反思主要是没有设置 top_k 和没有维持连接造成的。同时，我们在 response 中解析不出答案选项时会重新请求，这也会导致作答时间过长。我们的 prompt 是“[问题]。答案是：”，测试发现效果还行。

练习赛后期分数随机性较强，前几策略基本相同。正式赛“元梦之星”队另辟蹊径，采取合作式的策略，反而取得了更好的成绩。尽管地图存在堵人收益，但没有人短时间实现。可能由于第一的误导作用（模仿超越其策略的时间不够），加上我们算法具有一定泛用性，在正式赛中有幸拿到了第二名。

3 比赛总结与收获

我们吸取了去年参赛的经验，在一开始就没有想实现论文级的最优算法，

而是选择改进基础算法，一步步优化。我们在设计算法时设置了超级多的参数，可以调节的东西实在是太多，这使得我们算法的上限相对较高（正式赛不至于会很无聊），但泛化性上会有所欠缺。

通过这次比赛，我们认为主要有以下几点收获：

1. 状态别设计太多，不然之后写状态转移真的会很多 bug。
2. 大模型确实需要多去调整参数和 prompt，好的参数和 prompt 真的能大幅提高速度和精度。
3. 团队成员之间的代码风格得统一，不然互相看不懂，很难 debug。
4. 尽量将算法封装好，重复写同样的东西之后不太好改。
5. 与赛题组和其他队伍的交流能使得自己对赛题有更好的认识。
6. 面对问题保持冷静，坚持才能取得胜利。

最后，没想到这次能拿下总决赛亚军。感谢赛题组老师们的题目设计，感谢西南赛区和决赛赛事组老师们的精心组织，参赛体验非常不错！