

Lab 1 实验报告 / 人工智能基础

马天开 / PB2100030

2024 年 5 月 3 日

Lab 1.1 A*

本部分实验代码从头独立完成、不依赖 TA 提供的参考代码、不参考其他同学代码。

在每个子项目 `./src` 目录下都配置了 `CMakeLists.txt` 文件，可以直接使用 `CMake` 进行构建。

值得注意的是, `output` 产物并不直接输出在 `./src/output` 下, 而是 `debug/` 或 `release/` 目录下, 请注意查看。

启发函数

使用的启发函数 $h(x)$ 为曼哈顿距离, 显然曼哈顿距离:

- `admissible`: 曼哈顿距离实际上是不考虑障碍物、补给点等条件下的最短路径, 加上这些条件后, 曼哈顿距离一定不会小于实际路径长度, 因此是可接受的启发函数。
- `consistent`: 在曼哈顿距离意义下, 每一步最多减少 1, 而路径长度也为 1, 因此是一致的。

算法的主要思路

在这个问题中, 我们需要把剩余补给的天数作为状态的一部分进行考虑, 以一个 `std::tuple<Pos, int>` 作为状态; 当然, 为方便描述起见, 我们把经过的步长 `step` 和到达这里的路径 `steps_str` 也作为状态的一部分。

接下来是维护最小数列的过程, 由于我们需要进行遍历查找操作, 且数据量并不大, 我们只维护一个 `std::vector` 并在 `pop` 前排序即可。

与一致代价搜索的比较

只需要将上述代码中 $h(x)$ 设置为 0 即可, 我们统计一共需要遍历多少个节点:

A*	一致代价搜索
[8, 27, 22, 49, 40, 163, 100, 118, 499, 303]	[12, 30, 53, 70, 125, 185, 117, 155, 501, ...]

在最后一个问题中, 一致代价搜索所耗时间远远超过 A* 算法。

Lab 1.2 $\alpha - \beta$ 剪枝

本部分实验代码从头独立完成、不依赖 TA 提供的参考代码、不参考其他同学代码。

在每个子项目 `./src` 目录下都配置了 `CMakeLists.txt` 文件，可以直接使用 CMake 进行构建。

值得注意的是，**output** 产物并不直接输出在 `./src/output` 下，而是 **debug/** 或 **release/** 目录下，请注意查看。

算法主要实现思路

参考教材 $\alpha - \beta$ 剪枝的伪代码，我们可以很容易地实现这个算法。（我们把教材中 MIN 和 MAX 节点算法合并为一个）

```
// ...
int value = is_red ? -inf : +inf;
for (const auto &board: next_boards(cur, is_red)) {
    int tmp = alpha_beta(board, depth + 1, !is_red, alpha, beta);
    if (is_red) {
        value = std::max(value, tmp);
        if (beta >= value) {
            return value;
        }
        alpha = std::max(alpha, value);
    } else {
        value = std::min(value, tmp);
        if (alpha <= value) {
            return value;
        }
        beta = std::min(beta, value);
    }
}
return value;
```

在此基础上我们加入一般的 MINMAX 算法中的阶段性条件、终止判定：

```
int alpha_beta(const Board &cur, int depth, bool is_red, int alpha, int beta) {
    if (depth == MAX_DEPTH || end_check(cur))
        return score(cur);
    // ...
}
```

我们在这里省略一些关于中国象棋基本框架的实现方法等细节。

实验结果

```
$ for file in *; do cat $file; done
R (1,8) (4,8)
C (6,5) (4,5)
N (1,8) (3,7)
P (5,6) (4,6)
R (1,1) (1,9)
N (1,6) (2,8)
C (2,0) (2,8)
P (4,6) (5,6)
R (1,7) (1,9)
C (4,8) (0,8)
```

容易注意到, 算法生成的下一盘棋局的顺序不同, 会直接影响剪枝的方向、速度以至于最终结果, 不同结果可能都是正确的。

效率分析

我们考虑 $\text{depth} = 2$ 的情况, 对比采用 $\alpha - \beta$ 前后, 遍历的棋盘数量:

$\alpha - \beta$	一般 MINMAX
[1260, 1360, 30, 1487, 673, 502, 82, 956, 748, 780]	[1395, 1436, 48, 1592, 749, 553, 130, 1031, 843, 888]

随着深度加深, $\alpha - \beta$ 剪枝的效果会更加明显。

评估函数的设计

关于评估函数, 主要由以下三部分构成:

- 棋力评估 (棋子位置)
- 行棋可能性评估 (取下一步可能吃到棋子的收益, 取最大值)
 - 在 Profiling 中, 此处评估函数占用了大部分时间
- 棋子价值评估 (棋子价值)