中国科学技术大学计算机学院
《人工智能基础》实验报告

2021.05.28

实验题目：Search 和 Multiagent

学生姓名：胡毅翔

学生学号：PB18000290

计算机实验教学中心制

2019 年 9 月

1 实验目的

1. 实现 BFS 算法和 A* 算法。
2. 实现 minimax 算法和 alpha-beta 算法。

2 实验环境

1.PC 一台。

2.Windows 10 操作系统。

3.Git Bash

4.Python 3.8.1

3 算法思路

3.1 BFS 算法

BFS（Breadth-first search）算法，即广度优先搜索算法。其边界选用的是 FIFO 队列。其完备性已在课堂中得到证明。在本次实验中，每一步的代价均相同，故满足最优性条件，一定返回最优解。设 b 为最大分支数，d 为目标节点的最小深度。算法的时间复杂度是 $O\left(b^d\right)$，空间复杂度是 $O\left(b^d\right)$。

算法的伪代码如下：

function BREADTH-FIRST-SEARCH(*problem*) **returns** a solution, or failure
  node ← a node with STATE = *problem*.INITIAL-STATE, PATH-COST = 0
  if *problem*.GOAL-TEST(*node*.STATE) **then return** SOLUTION(*node*)
  frontier ← a FIFO queue with *node* as the only element
  explored ← an empty set
  loop do
    if EMPTY?(*frontier*) **then return** failure
    node ← POP(*frontier*) /* chooses the shallowest node in *frontier* */
    add *node*.STATE to *explored*
    for each *action* **in** *problem*.ACTIONS(*node*.STATE) **do**
      child ← CHILD-NODE(*problem*, *node*, *action*)
      if *child*.STATE is not in *explored* or *frontier* **then**
        if *problem*.GOAL-TEST(*child*.STATE) **then return** SOLUTION(*child*)
        frontier ← INSERT(*child*, *frontier*)

本次实验中实现的 **Python** 代码如下：

```python
def myBreadthFirstSearch(problem):

    visited = {}
    frontier = util.Queue()

    frontier.push((problem.getStartState(), None))

    while not frontier.isEmpty():
        state, prev_state = frontier.pop()
```

```
        if problem.isGoalState(state):
            solution = [state]
            while prev_state != None:
                solution.append(prev_state)
                prev_state = visited[prev_state]
            return solution[::-1]
        if state not in visited:
            visited[state] = prev_state
            for next_state, step_cost in problem.getChildren(state):
                frontier.push((next_state, state))
    return []
```

3.2 A* 算法

A*（A-star）算法，即 A 星算法。该算法的评估函数为：

$$f(n) = g(n) + h(n)$$

其中 $f(n)$ 表示经过节点 n 的最低耗散的估计函数，$g(n)$ 表示到达节点 n 的耗散，$h(n)$ 为启发函数，表示从节点 n 到目标节点的最低耗散路径的耗散估计值。

可采纳的启发式函数须满足：

$$h(n) \leq h^*(n)$$

其中 $h^*(n)$ 表示从节点 n 到目标节点的最低耗散路径的实际耗散值。

A* 算法的最优性，完备性已在课堂上得到证明。其伪代码如下：

```
A* search {

closed list = [ ]
open list = [start node]

do {
        if open list is empty then {
                return no solution
        }
        n = heuristic best node
        if n == final node then {
                return path from start to goal node
        }
        foreach direct available node do{
                if current node not in open and not in closed list do {
                        add current node to open list and calculate heuristic
                        set n as his parent node
                }
                else{
                        check if path from star node to current node is
                        better;
                        if it is better calculate heuristics and transfer
                        current node from closed list to open list
                        set n as his parrent node
                }
        delete n from open list
        add n to closed list
} while (open list is not empty)

}
```

本次实验中实现的 **Python** 代码如下：

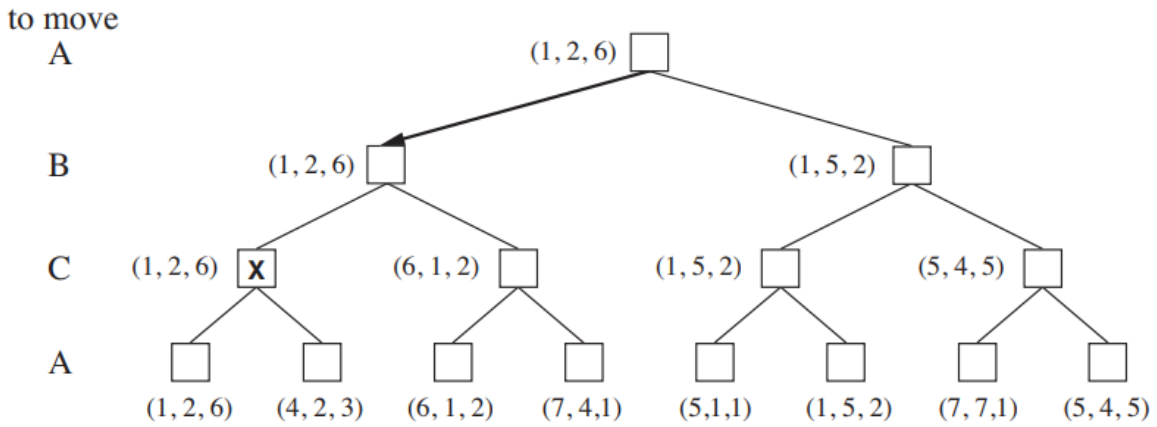
```
def myAStarSearch(problem, heuristic):

    visited = {}
    frontier = util.PriorityQueue()
    frontier.update((problem.getStartState(), None, 0),
                    heuristic(problem.getStartState()))

    while not frontier.isEmpty():
        state, prev_state, cost = frontier.pop()
        if problem.isGoalState(state):
            solution = [state]
            while prev_state != None:
                solution.append(prev_state)
                prev_state = visited[prev_state]
            return solution[::-1]
        if state not in visited:
            visited[state] = prev_state
            for next_state, step_cost in problem.getChildren(state):
                h_n = heuristic(next_state)
                frontier.update(
                    (next_state, state, cost+step_cost), cost+step_cost+h_n)
    return []
```

3.3 minimax 算法

minimax 算法，即极小极大值算法。该算法在假设对手也是用最优策略的条件下。能导致至少不必其他策略车的结果。换句话说，假设两个游戏者都按照最优策略进行，那么节点的极小极大值就是对应状态的效用值。



其算法的完备性，最优性已在课堂中得到证明。设 b 为最大分支数，m 为搜索树的最大深度。算法的时间复杂度是 $O(b^m)$，空间复杂度是 $O(bm)$。

算法的伪代码如下：

```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

function MAX-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s, a)))
  return v

function MIN-VALUE(state) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← ∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s, a)))
  return v
```

本次实验中实现的 **Python** 代码如下：

```python
class MyMinimaxAgent():

    def __init__(self, depth):
        self.depth = depth

    def minimax(self, state, depth):
        if state.isTerminated():
            return None, state.evaluateScore()

        best_state, best_score = None, - \
            float('inf') if state.isMe() else float('inf')

        for child in state.getChildren():
            if state.isMe():
                _, tmp_score = self.minimax(child, depth-1)
                if tmp_score > best_score:
                    best_score = tmp_score
                    best_state = child
            else:
                if child.isMe() and depth == 0:
                    tmp_score = child.evaluateScore()
                    if tmp_score < best_score:
                        best_score = tmp_score
                        best_state = child
                else:
                    _, tmp_score = self.minimax(child, depth)
                    if tmp_score < best_score:
                        best_score = tmp_score
                        best_state = child
        return best_state, best_score

    def getNextState(self, state):
        best_state, _ = self.minimax(state, self.depth)
        return best_state
```

3.4 alpha-beta 算法

alpha-beta 算法的思路与 minimax 算法类似，增加了 α，β 两个参数。α 表示到目前为止在路径上的任意点发现的 MAX 的最佳选择。β 则表示到目前为止在路径上的任意点发现的 MIN 的最佳选择。通过与这两个参数进行比较来判断后续的搜索是否必要，减少计算量。

这一剪枝操作不会改变 minimax 的最终结果。在最优情况下，可以把时间复杂度降至 $O\left(b^{m/2}\right)$，但在最坏情况下不会获得性能提升。

算法的伪代码如下：

function ALPHA-BETA-SEARCH(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the *action* in ACTIONS(*state*) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

---

function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow +\infty$
  for each a **in** ACTIONS(*state*) **do**
    $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

本次实验中实现的 **Python** 代码如下：

```python
class MyAlphaBetaAgent():

    def __init__(self, depth):
        self.depth = depth

    def alphabeta(self, state, depth, alpha, beta):
        if state.isTerminated():
            return None, state.evaluateScore()
        best_state, best_score = None, - \
            float('inf') if state.isMe() else float('inf')
        for child in state.getChildren():
            if state.isMe():
                _, tmp_score = self.alphabeta(child, depth-1, alpha, beta)
                if tmp_score > best_score:
                    best_score = tmp_score
                    best_state = child
```

```
                alpha = max(alpha, best_score)
                if best_score > beta:
                    return best_state, best_score
            else:
                if child.isMe() and depth == 0:
                    tmp_score = child.evaluateScore()
                else:
                    _, tmp_score = self.alphabeta(child, depth, alpha, beta)
                if tmp_score < best_score:
                    best_score = tmp_score
                    best_state = child
                beta = min(beta, best_score)
                if best_score < alpha:
                    return best_state, best_score
        return best_state, best_score

    def getNextState(self, state):
        best_state, _ = self.alphabeta(
            state, self.depth, float("-inf"), float("inf"))
        return best_state
```

4 实验结果

4.1 BFS 算法

```
Question q2
===========
*** PASS: test_cases\q2\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q2\graph_bfs_vs_dfs.test
***     solution:               ['1:A->G']
***     expanded_states:        ['A', 'B']
*** PASS: test_cases\q2\graph_infinite.test
***     solution:               ['0:A->B', '1:B->C', '1:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q2\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']
*** PASS: test_cases\q2\pacman_1.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         269

### Question q2: 4/4 ###


Finished at 11:45:28

Provisional grades
==================
Question q2: 4/4
------------------
Total: 4/4

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of
importlib; see the module's documentation for alternative uses
  import imp
Starting on 5-28 at 11:45:32
```

4.2 A* 算法

```
Question q3
===========
*** PASS: test_cases\q3\astar_0.test
***     solution:               ['Right', 'Down', 'Down']
***     expanded_states:        ['A', 'B', 'D', 'C', 'G']
*** PASS: test_cases\q3\astar_1_graph_heuristic.test
***     solution:               ['0', '0', '2']
***     expanded_states:        ['S', 'A', 'D', 'C']
*** PASS: test_cases\q3\astar_2_manhattan.test
***     pacman layout:          mediumMaze
***     solution length: 68
***     nodes expanded:         221
*** PASS: test_cases\q3\astar_3_goalAtDequeue.test
***     solution:               ['1:A->B', '0:B->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C']
*** PASS: test_cases\q3\graph_backtrack.test
***     solution:               ['1:A->C', '0:C->G']
***     expanded_states:        ['A', 'B', 'C', 'D']
*** PASS: test_cases\q3\graph_manypaths.test
***     solution:               ['1:A->C', '0:C->D', '1:D->F', '0:F->G']
***     expanded_states:        ['A', 'B1', 'C', 'B2', 'D', 'E1', 'F', 'E2']

### Question q3: 4/4 ###


Finished at 11:45:32

Provisional grades
==================
Question q3: 4/4
------------------
Total: 4/4

Your grades are NOT yet registered.  To register your grades, make sure
to follow your instructor's guidelines to receive credit on your project.

[SearchAgent] using function astar and heuristic manhattanHeuristic
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 221
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
autograder.py:17: DeprecationWarning: the imp module is deprecated in favour of
importlib; see the module's documentation for alternative uses
  import imp
Starting on 5-28 at 11:45:37
```

4.3 minimax 算法

```
Question q2
===========

*** PASS: test_cases\q2\0-eval-function-lose-states-1.test
*** PASS: test_cases\q2\0-eval-function-lose-states-2.test
*** PASS: test_cases\q2\0-eval-function-win-states-1.test
*** PASS: test_cases\q2\0-eval-function-win-states-2.test
*** PASS: test_cases\q2\0-lecture-6-tree.test
*** PASS: test_cases\q2\0-small-tree.test
*** PASS: test_cases\q2\1-1-minmax.test
*** PASS: test_cases\q2\1-2-minmax.test
*** PASS: test_cases\q2\1-3-minmax.test
*** PASS: test_cases\q2\1-4-minmax.test
*** PASS: test_cases\q2\1-5-minmax.test
*** PASS: test_cases\q2\1-6-minmax.test
*** PASS: test_cases\q2\1-7-minmax.test
*** PASS: test_cases\q2\1-8-minmax.test
*** PASS: test_cases\q2\2-1a-vary-depth.test
*** PASS: test_cases\q2\2-1b-vary-depth.test
*** PASS: test_cases\q2\2-2a-vary-depth.test
*** PASS: test_cases\q2\2-2b-vary-depth.test
*** PASS: test_cases\q2\2-3a-vary-depth.test
*** PASS: test_cases\q2\2-3b-vary-depth.test
*** PASS: test_cases\q2\2-4a-vary-depth.test
*** PASS: test_cases\q2\2-4b-vary-depth.test
*** PASS: test_cases\q2\2-one-ghost-3level.test
*** PASS: test_cases\q2\3-one-ghost-4level.test
*** PASS: test_cases\q2\4-two-ghosts-3level.test
*** PASS: test_cases\q2\5-two-ghosts-4level.test
*** PASS: test_cases\q2\6-tied-root.test
*** PASS: test_cases\q2\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q2\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q2\7-2c-check-depth-two-ghosts.test
*** Running MinimaxAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running MinimaxAgent on smallClassic after 1 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q2\8-pacman-game.test

### Question q2: 5/5 ###


Finished at 11:45:38

Provisional grades
==================
Question q2: 5/5
------------------
Total: 5/5
```

4.4 alpha-beta 算法

```
Question q3
===========

*** PASS: test_cases\q3\0-eval-function-lose-states-1.test
*** PASS: test_cases\q3\0-eval-function-lose-states-2.test
*** PASS: test_cases\q3\0-eval-function-win-states-1.test
*** PASS: test_cases\q3\0-eval-function-win-states-2.test
*** PASS: test_cases\q3\0-lecture-6-tree.test
*** PASS: test_cases\q3\0-small-tree.test
*** PASS: test_cases\q3\1-1-minmax.test
*** PASS: test_cases\q3\1-2-minmax.test
*** PASS: test_cases\q3\1-3-minmax.test
*** PASS: test_cases\q3\1-4-minmax.test
*** PASS: test_cases\q3\1-5-minmax.test
*** PASS: test_cases\q3\1-6-minmax.test
*** PASS: test_cases\q3\1-7-minmax.test
*** PASS: test_cases\q3\1-8-minmax.test
*** PASS: test_cases\q3\2-1a-vary-depth.test
*** PASS: test_cases\q3\2-1b-vary-depth.test
*** PASS: test_cases\q3\2-2a-vary-depth.test
*** PASS: test_cases\q3\2-2b-vary-depth.test
*** PASS: test_cases\q3\2-3a-vary-depth.test
*** PASS: test_cases\q3\2-3b-vary-depth.test
*** PASS: test_cases\q3\2-4a-vary-depth.test
*** PASS: test_cases\q3\2-4b-vary-depth.test
*** PASS: test_cases\q3\2-one-ghost-3level.test
*** PASS: test_cases\q3\3-one-ghost-4level.test
*** PASS: test_cases\q3\4-two-ghosts-3level.test
*** PASS: test_cases\q3\5-two-ghosts-4level.test
*** PASS: test_cases\q3\6-tied-root.test
*** PASS: test_cases\q3\7-1a-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1b-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-1c-check-depth-one-ghost.test
*** PASS: test_cases\q3\7-2a-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2b-check-depth-two-ghosts.test
*** PASS: test_cases\q3\7-2c-check-depth-two-ghosts.test
*** Running AlphaBetaAgent on smallClassic 1 time(s).
Pacman died! Score: 84
Average Score: 84.0
Scores:        84.0
Win Rate:      0/1 (0.00)
Record:        Loss
*** Finished running AlphaBetaAgent on smallClassic after 0 seconds.
*** Won 0 out of 1 games. Average score: 84.000000 ***
*** PASS: test_cases\q3\8-pacman-game.test

### Question q3: 5/5 ###


Finished at 11:45:40

Provisional grades
==================
Question q3: 5/5
------------------
Total: 5/5
```

5 总结

本次实验实现了 BFS，A*,minimax，alpha-beta 四种算法，加深了对课堂所学知识的理解。通过吃豆人游戏的方式，也增加了同学们对课程实验的兴趣。