Homework assignment#2

2020.12.09

(A) Pseudo codes documentation

請解釋每行程式碼或重要部分

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to "temperature"

  current ← MAKE-NODE(problem.INITIAL-STATE)
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← next.VALUE – current.VALUE
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/T)
```

Figure 4.5 The simulated annealing algorithm, a version of stochastic hill climbing where some downhill moves are allowed. Downhill moves are accepted readily early in the annealing schedule and then less often as time goes on. The *schedule* input determines the value of the temperature T as a function of time.

Simulated Annealing

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new_population ← empty set
    for i = 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Figure 4.8 A genetic algorithm. The algorithm is the same as the one diagrammed in Figure 4.6, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.

Genetic Algorithm

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then return [action | plan]
  return failure

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

Figure 4.11 An algorithm for searching AND–OR graphs generated by nondeterministic environments. It returns a conditional plan that reaches a goal state in all circumstances. (The notation $[x \mid l]$ refers to the list formed by adding object x to the front of list l.)

AND-OR Search

```
function LRTA*-AGENT(s') returns an action
  inputs: s', a percept that identifies the current state
  persistent: result, a table, indexed by state and action, initially empty
              H, a table of cost estimates indexed by state, initially empty
              s, a, the previous state and action, initially null

  if GOAL-TEST(s') then return stop
  if s' is a new state (not in H) then H[s'] ← h(s')
  if s is not null
      result[s, a] ← s'
      H[s] ← min_{b ∈ ACTIONS(s)} LRTA*-COST(s, b, result[s, b], H)
  a ← an action b in ACTIONS(s') that minimizes LRTA*-COST(s', b, result[s', b], H)
  s ← s'
  return a

function LRTA*-COST(s, a, s', H) returns a cost estimate
  if s' is undefined then return h(s)
  else return c(s, a, s') + H[s']
```

Figure 4.24 LRTA*-AGENT selects an action according to the values of neighboring states, which are updated as the agent moves about the state space.

LRTA* -Agent

(B) Exercises

4.1 Give the name of the algorithm that results from each of the following special cases:

a. Local beam search with $k = 1$.

b. Local beam search with one initial state and no limit on the number of states retained.

c. Simulated annealing with $T = 0$ at all times (and omitting the termination test).

d. Simulated annealing with $T = \infty$ at all times.

e. Genetic algorithm with population size $N = 1$.

根據下面的特殊情況給出演算法的名稱

4.6 Explain precisely how to modify the AND-OR-GRAPH-SEARCH algorithm to generate a cyclic plan if no acyclic plan exists. You will need to deal with three issues: labeling the plan steps so that a cyclic plan can point back to an earlier part of the plan, modifying OR-SEARCH so that it continues to look for acyclic plans after finding a cyclic plan, and augmenting the plan representation to indicate whether a plan is cyclic. Show how your algorithm works on (a) the slippery vacuum world, and (b) the slippery, erratic vacuum world. You might wish to use a computer implementation to check your results.

詳細說明如果不存在無循環規劃，如何修改AND-OR-GRAPH-SEARCH 演算法來產生一個有循環規劃。

```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
  OR-SEARCH(problem.INITIAL-STATE, problem, [])

---

function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
  if problem.GOAL-TEST(state) then return the empty plan
  if state is on path then return loop
  cyclic − plan ← None
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
      if plan ≠ failure then
          if plan is acyclic then return [action | plan]
          cyclic − plan ← [action | plan]
  if cyclic − plan ≠ None then return cyclic − plan
  return failure

---

function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
  loopy ← True
  for each s_i in states do
      plan_i ← OR-SEARCH(s_i, problem, path)
      if plan_i = failure then return failure
      if plan_i ≠ loop then loopy ← False
  if not loopy then
     return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n−1} then plan_{n−1} else plan_n]
  return failure
```

4.7 In Section 4.4.1 we introduced belief states to solve sensorless search problems. A sequence of actions solves a sensorless problem if it maps every physical state in the initial belief state b to a goal state. Suppose the agent knows $h^*(s)$, the true optimal cost of solving the physical state s in the fully observable problem, for every state s in b. Find an admissible heuristic $h(b)$ for the sensorless problem in terms of these costs, and prove its admissibilty. Comment on the accuracy of this heuristic on the sensorless vacuum problem of Figure 4.14. How well does A* perform?

假設agent知道$h^*(s)$，即 b 中的每個狀態 s，在fully observable problem，解決實體狀態 s 的真正最佳化成本。對每個狀態S，求sensorless problem的admissible heuristic $h(b)$，並證明其為admissible。對圖4.14 sensorless vacuum problem，這種heuristic的準確性，作出評論。A*的執行效果如何？

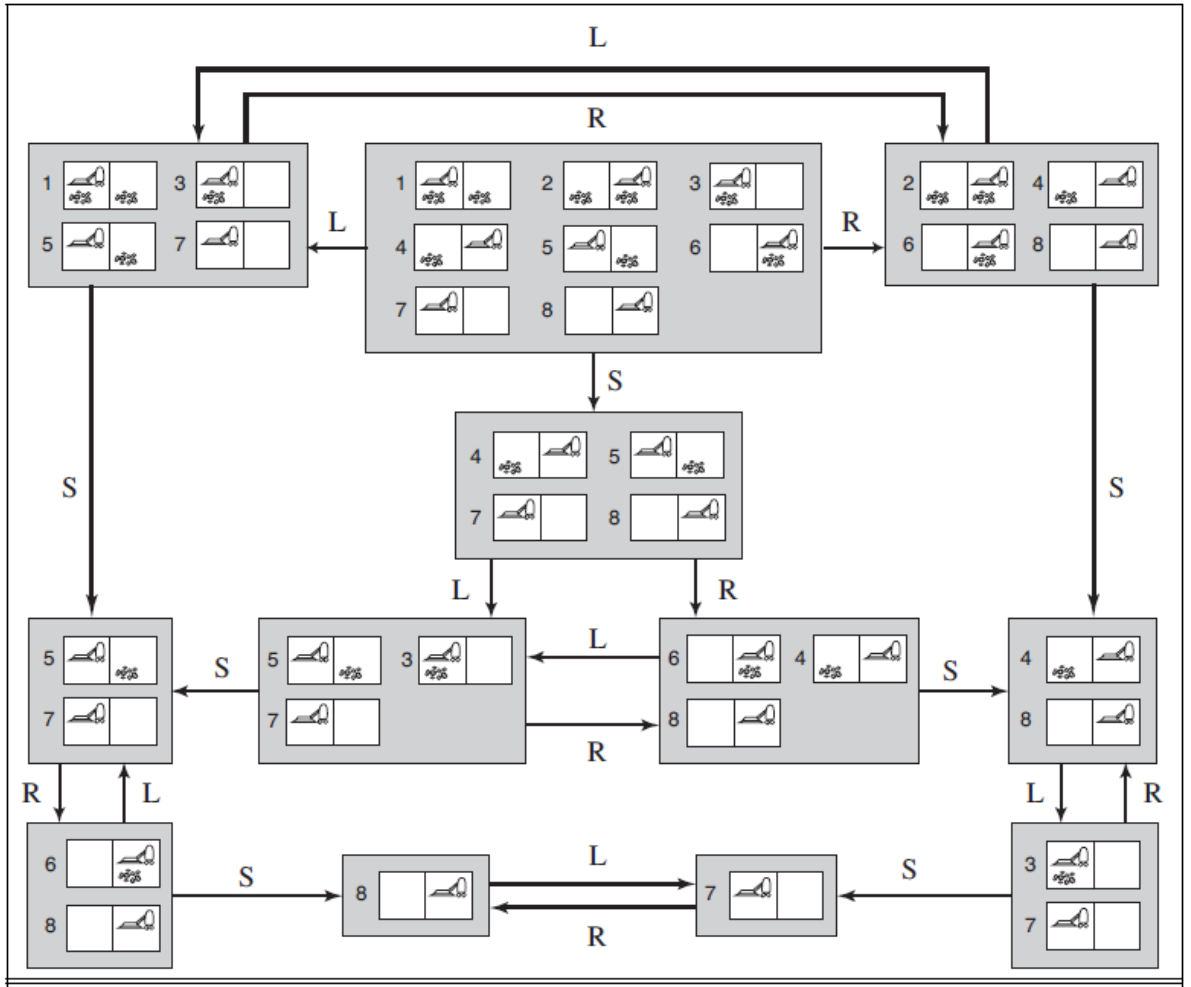


Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.

4.8 This exercise explores subset–superset relations between belief states in sensorless or partially observable environments.

a. Prove that if an action sequence is a solution for a belief state b, it is also a solution for any subset of b. Can anything be said about supersets of b?

b. Explain in detail how to modify graph search for sensorless problems to take advantage of your answers in (a).

c. Explain in detail how to modify AND–OR search for partially observable problems, beyond the modifications you describe in (b).

a. 證明如果一個action sequence是一個belief state b 的一個解，則它也是任何b 子集的一個解。關於supersets b 可以表示些什麼？
b. 詳細說明如何利用答案a.的優點，修改sensorless problems的graph search。
c. 詳細說明在你在b.所描述的修改之外，如何修改partially observable problems的AND-OR search

4.9 On page 139 it was assumed that a given action would have the same cost when executed in any physical state within a given belief state. (This leads to a belief-state search problem with well-defined step costs.) Now consider what happens when the assumption does not hold. Does the notion of optimality still make sense in this context, or does it require modification? Consider also various possible definitions of the "cost" of executing an action in a belief state; for example, we could use the *minimum* of the physical costs; or the *maximum*; or a cost *interval* with the lower bound being the minimum cost and the upper bound being the maximum; or just keep the set of all possible costs for that action. For each of these, explore whether A* (with modifications if necessary) can return optimal solutions.

假設在一個給定的belief-state中執行任何的實體狀態，則一個給定的行動將具有相同的成本。(這導致了一種具有明確步驟costs的belief-state search problem)。現在考慮當假設不成立時會發生什麼。最佳化的概念在此背景下是否還有意義，還是需要作修改？還要考慮在belief-state中執行一個行動，對於「cost」的各種可能的定義，例如，我們可以使用最小的，或最大的實體成本；或是在最低成本的下限和最高成本的上限的成本區間，或只是保留該行動所有可能的費用集。對於這裡的每一種，探討A*(如果有必要則作修改)是否可以返回最佳解。

4.10 Consider the sensorless version of the erratic vacuum world. Draw the belief-state space reachable from the initial belief state $\{1, 2, 3, 4, 5, 6, 7, 8\}$, and explain why the problem is unsolvable.

考慮sensorless版本的erratic vacuum world畫出可從最初的belief-state{1, 2, 3, 4, 5, 6, 7, 8}到達的belief-state space，並解釋為什麼這個問題是無法解的。

4.12 Suppose that an agent is in a 3×3 maze environment like the one shown in Figure 4.19. The agent knows that its initial location is (1,1), that the goal is at (3,3), and that the actions *Up*, *Down*, *Left*, *Right* have their usual effects unless blocked by a wall. The agent does *not* know where the internal walls are. In any given state, the agent perceives the set of legal actions; it can also tell whether the state is one it has visited before.

a. Explain how this online search problem can be viewed as an offline search in belief-state space, where the initial belief state includes all possible environment configurations. How large is the initial belief state? How large is the space of belief states?

b. How many distinct percepts are possible in the initial state?

c. Describe the first few branches of a contingency plan for this problem. How large (roughly) is the complete plan?

Notice that this contingency plan is a solution for *every possible environment* fitting the given description. Therefore, interleaving of search and execution is not strictly necessary even in unknown environments.

假設一個agent在一個如圖4.19 所示的3x3 大小的迷宮裡。agent知道它的初始位置是(3,3)，目標位置是(1, 1)，四種行動*Up*(上)，*Down*(下)，*Left*(左)，*Right*(右) 通常可以發揮效果，除非遇到有牆阻礙。agent不知道迷宮內部的牆在哪些地方。在任何給定的狀態，agent知道合法行動集；它也知道一個狀態是已經存取過的狀態還是新狀態。

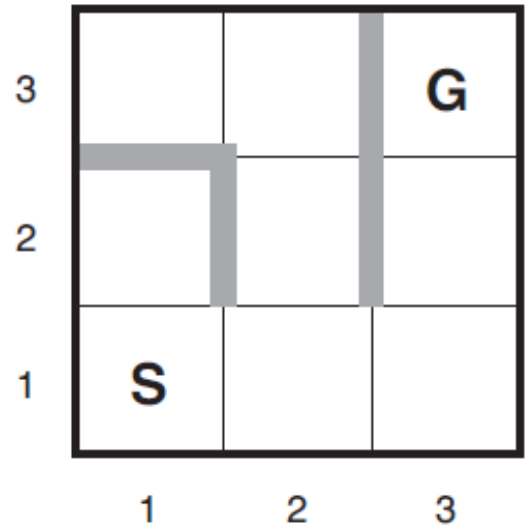
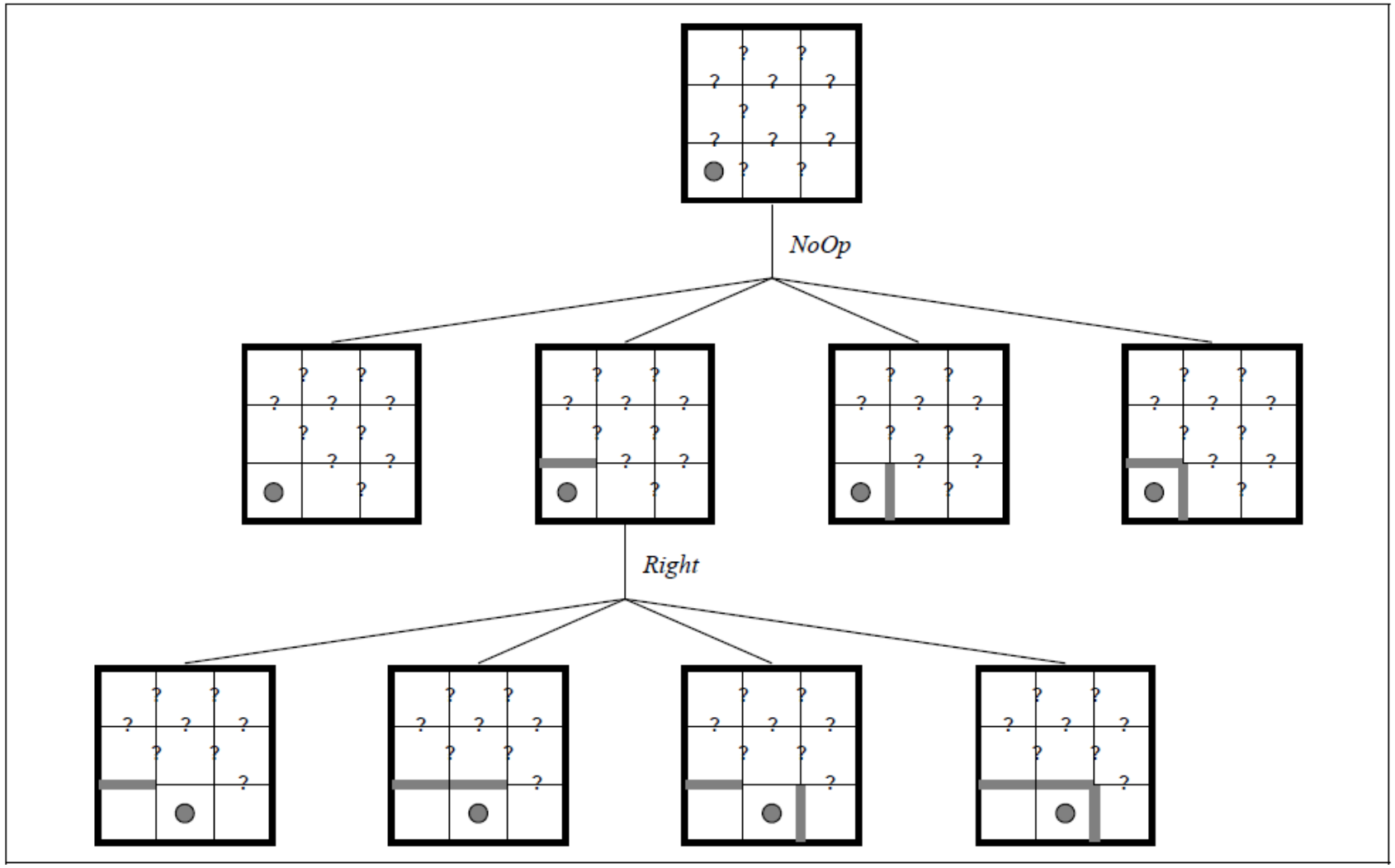


Figure 4.19 A simple maze problem. The agent starts at *S* and must reach *G* but knows nothing of the environment.

a. 解譯這個online search problem如何可以視為在belief-state space中的offline search problem，初始的belief-state包括所有可能的環境佈局。初始belief-state有多大？belief-state空間有多大？
b. 在初始狀態可能有多少個不同的感知資訊？
c. 描述這個問題的偶發性規畫的前幾個分支。完整的規畫(大約)有多大？

注意這個偶發性規畫是符合給定描述的每個可能環境的解。因此，即使在未知環境下搜尋和執行的交叉也不是嚴格必要的了。


NoOp
Right

4.14 Like DFS, online DFS is incomplete for reversible state spaces with infinite paths. For example, suppose that states are points on the infinite two-dimensional grid and actions are unit vectors $(1,0)$, $(0,1)$, $(-1,0)$, $(0,-1)$, tried in that order. Show that online DFS starting at $(0,0)$ will not reach $(1,-1)$. Suppose the agent can observe, in addition to its current state, all successor states and the actions that would lead to them. Write an algorithm that is complete even for bidirected state spaces with infinite paths. What states does it visit in reaching $(1,-1)$?

對於無限路徑可逆狀態空間的online DFS就像DFS一樣是incomplete。例如，假設各狀態是在無限二維網格的點且actions為單位向量(1, 0)，(0, 1)，(-1, 0)，(0，-1)，以這個順序作嘗試。證明online DFS，從(0, 0)開始將不會到達(1, -1)。假設agent是可以觀察的，則除了其目前的狀態，將導致到所有繼承的狀態和行動。撰寫一個演算法，即使是在無限路徑的雙向狀態空間，它仍是complete的。在到達(1, -1)時它存取哪些狀態？