Homework assignment#2 (Chap4)

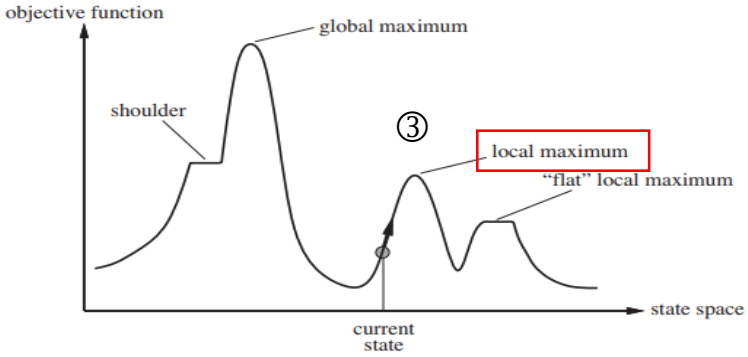
106971001 林上人

(A) Pseudo codes documentation

Pages: 14

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

① *current* ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
② *neighbor* ← a highest-valued successor of *current*
if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
current ← *neighbor*



爬山演算法 - 尋找局部最優

爬山演算法是最基本的局部搜索演算法，首先①確定當前節點之後，②持續檢查 successor，取出值最高的做比較，若比當前節點值低就回傳當前節點，否則就用值高的 successor 替換當前節點，以此方式不斷向值高的地方走，缺點就如上圖所示，雖然 current state 會一直向上走直到③local maximum 處，找到局部最優，但是其實在另一個地方還存在更高的 global maximum。

Pages: 28

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

①  current ← MAKE-NODE(problem.INITIAL-STATE)
   for t = 1 to ∞ do
②      T ← schedule(t)      [T is a decreasing function of t]
       if T = 0 then return current
③      next ← a randomly selected successor of current
       ΔE ← next.VALUE – current.VALUE
       if ΔE > 0 then current ← next
       else current ← next only with probability ΔE/T
```

Simulated-Annealing 演算法 – 可以下山的登山演算法

開頭①確定初始的當前節點後，我們用②schedule 函數取得一個 T 值，這個 T 值會隨時間下降，當 T 降至 0 時就回傳當前的節點，而③處開始即是 Simulated-Annealing 實作可以下山的部分，首先取得 successor 的方式不再直接挑選值高的，而是以隨機的方式選取，若值比當前節點高則替換當前節點（上山），若值比當前節點低則以$e^{(\Delta E/T)}$的機率決定是否替換當前節點(下山)，因爲 T 會隨時間下降，所以時間往後下山的機會就會持續降低，若 T 下降的足夠緩慢，找到 global optimum 的機率就越趨近 1，但整個運作的時間就越久。

Pages: 43

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
① 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
②     x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
④     if (small random probability) then child ← MUTATE(child)
      add child to new_population
    population ← new_population
⑤ 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
③ 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))
```

基因演算法

基因演算法流程首先①會輸入一個 initial population，然後計算裡面個體的合適度，②從合適度在一定百分比以上的個體中，隨機選出兩個進行繁殖產生 child，繁殖的方式如③，在總基因長度 n 中隨機選擇一個點 c 作為切割點，並由 x 的基因 1~c 和 y 的基因 c~n 組合成 child，而產生的 child④都會有一定的機率進行變異，並把 child 加入 population 集合之中，然後重複整個過程直到達成終止條件為止，此處終止條件⑤是設定為當出現某個個體的合適度已經足夠合適或是繁衍的時間已經足夠久就停止，並回傳 population 之中合適度最高的個體。

Pages: 48

```
OR-SEARCH(state, problem, path)
1  if problem.GOAL-TEST(state)
2        return the empty plan
3  if state is on path return failure
4  for each action in problem.ACTIONS(state)
5        plan = AND-SEARCH(RESULTS(state, action), problem, [state|path])
6        if plan ≠ failure
7              return [action|plan]
8  return failure
```

①

②

```
AND-SEARCH(state, problem, path)
1  for each Si in states
2        plani = OR-SEARCH(Si, problem, path)
3        if plani == failure
4              return failure
5  return [if s1 then plan1 elseif... elseif sn−1 then plann−1 else plann]
```

③

AND-OR 搜尋樹 – 求不確定性問題的可能解

OR 節點選擇一個動作之後進行分支，因此某一分支後繼狀態有解即可，而 AND 節點則是由一個動作被選擇之後產生的所有狀態，因此必須處理所有的後繼狀態，才能找出解，因此在 OR-SEARCH 中①，檢查到達的是不是 Goal，然後檢查 state 是否存在先前的路徑中，這部分是為了解決循環的情況，因為產生循環，所以如果有解從之前的路徑搜尋中就可以找到解，因此就把循環產生的部分回傳 failure，然後②如前面所述，選擇一個動作之後到達 AND 節點，對這個 AND 節點做 AND-SEARCH 運算，③則表示 AND-SEARCH 中必須對所有產生的狀態都有策略規劃才能找到所有的解，並以 if-than-else 的方式回傳規劃。

Pages: 77

```
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, s') + H[s']
```

① ② ③ ⑤ ④

s —a→ s'

new state, assign h(s') to H[s']

old state, update H[s]

LRTA*

本演算法初始建構元素，①處使用 result 建構環境轉移的地圖情形，H 則記錄每個狀態的估計代價，首先取得下一個狀態時還是先對他做 GOAL-TEST，之後②若新狀態未曾出現過在 H 中就先用 A*的 h(s')估計值代表，然後③使用 result 表紀錄狀態轉移的情況，並且使用 LRTA*-COST 更新 H 表中 H(s)的值為下一個動作的最小代價估計值，LRTA*-COST 做的事情如④所示，若新狀態是未曾出現過的就先以 h(s)代表，否則就更新 H 表的值為從狀態 S 轉移到 S'的成本加上 S'的估計值，最後⑤則選出 COST 最小的 action，並更新當前狀態 S 為 S'。

(B) Exercises

1. 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.

```
function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
    if problem.GOAL-TEST(state) then return the empty plan
①   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
②           if plan is acyclic then return [action | plan]
③           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 si in states do
        plan i ← OR-SEARCH(si, problem, path)
        if plan i = failure then return failure
        if plan i = loop then loopy ← False
    if not loopy then
        return [if s1 then plan1 else if s2 then plan 2 else . . . if sn-1 then plan n-1 else plan n]
    return failure
```

Deal with three issues:

① labeling the plan steps so that a cyclic plan can point back to an earlier part of the plan : 當發生路徑循環的時候就回傳 loop 表示要回到此狀態沿著路徑最近一次出現的時候

②③modifying 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 : 可以在 plan 中檢查是否存在 loop，進而讓 plan 持續搜尋非循環路徑規劃

(a) 因為加入回傳 loop，可以讓 slippery vacuum world 發生動作失敗時，持續嘗試直到動作成功

(b) labeling the plan steps 回傳 loop 主要針對 slippery world, And-Or-Search 則針對 erratic world

2. 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?

因為 h*(s)是 true optimal cost，而我們需要找 admissible heuristic h(b)，而根據 admissible 的定義 h(b)必須不高估，也就是須小於等於實際成本所以可以定義一個 belief-state 的 heuristic 為所有在這個 belief-state 裡面的 physical state 之中 h*(s)最大的值 → $\mathrm{h(belief\ state)} = \max h^*(s)$, s = belief state 中的 physical state

在 Figure 4.14 使用此估計法時，initial belief state 和 belief state{4,5,7,8}、{3,5,7}、{4,6,8}的 h 值會比實際上到達 goal state 的距離少 1，因此 A*在到達 goal state 之前也會先拓展 belief state{4,5,7,8}、{3,5,7}、{4,6,8}

3. 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. 當一個動作序列為一個 belief states *b* 的解時，他會解決 *b* 所包含的所有狀態，所以 *b* 的子集合同樣能解，但是 supersets 就不行，因為 supersets 可能會包含不存在於 *b* 的狀態。

b. On expansion of a node, do not add to the frontier any child belief state which is a superset of a previously explored belief state.

c. If you keep a record of previously solved belief states, add a check to the start of ORsearch to check whether the belief state passed in is a subset of a previously solved belief state, returning the previous solution in case it is.

4. 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 slate; 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.

Consider a very simple example: an initial belief state $\{S_1, S_2\}$, actions a and b both leading to goal state G from either initial state

$c(S_1, a, G) = 3$; $c(S_2, a, G) = 5$;

$c(S_1, b, G) = 2$; $c(S_2, b, G) = 6$.

In this case, the solution [a] costs 3 or 5, the solution [b] costs 2 or 6. Neither is “optimal” in any obvious sense. In some cases, there *will* be an optimal solution.

Let us consider just the deterministic case. For this case, we can think of the cost of a plan as a mapping from each initial physical state to the actual cost of executing the plan. In the example above, the cost for [a] is $\{S_1{:}3, S_2{:}5\}$ and the cost for [b] is $\{S_1{:}2, S_2{:}6\}$.

We can say that plan p_1 *weakly dominates* p_2 if, for each initial state, the cost for p_1 is no higher than the cost for p_2. If a plan p weakly dominates all others, it is optimal. Notice that this definition reduces to ordinary optimality in the observable case where every belief state is a singleton.

As the preceding example shows, however, a problem may have no optimal solution in this sense. A perhaps acceptable version of A* would be one that returns any solution that is not dominated by another.

In particular, if we define the cost of a plan in belief-state space as the minimum cost of any physical realization, we violate Bellman’s principle. Modifying and extending the previous example, suppose that a and b reach S_3 from S_1 and S_4 from S_2, and then reach G from there:

$c(S_1, a, S_3) = 6$; $c(S_2, a, S_4) = 2$;

$c(S_1, b, S_3) = 6$; $c(S_2, b, S_4) = 1$;

$c(S_3, a, G) = 2$; $c(S_4, a, G) = 2$;

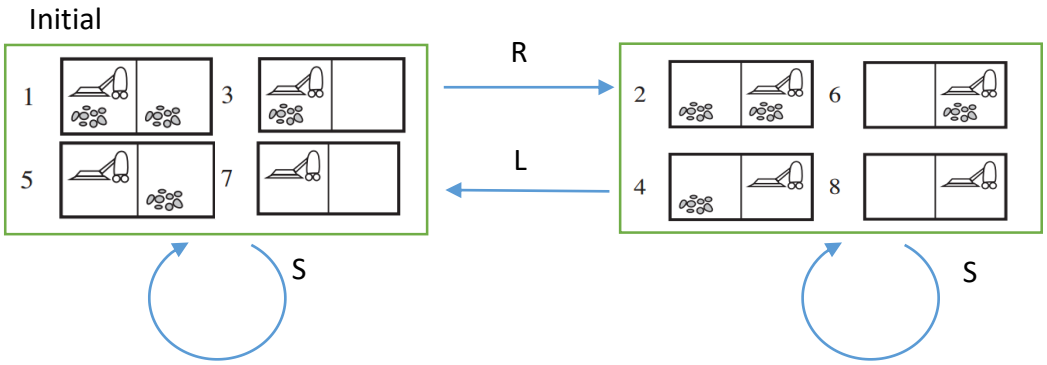
$c(S_3, b, G) = 1$; $c(S_4, b, G) = 9$;

In the belief state $\{S_3, S_4\}$, the minimum cost of [a] is $\min\{2, 2\} = 2$ and the minimum cost of [b] is $\min\{1, 9\} = 1$, so the optimal plan is [b]. In the initial belief state $\{S_1, S_2\}$, the four possible plans have the following costs:

[a, a] : $\min\{8, 4\} = 4$; [a, b] : $\min\{7, 11\} = 7$; [b, a] : $\min\{8, 3\} = 3$; [b, b] : $\min\{7, 10\} = 7$.

Hence, the optimal plan in $\{S_1, S_2\}$ is [b, a], which does *not* choose b in $\{S_3, S_4\}$ even though that is the optimal plan at that point. This counterintuitive behavior is a direct consequence of choosing the minimum of the possible path costs as the performance measure.

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



No solution is possible because no path leads to a belief state all of whose elements satisfy the goal.

6. Suppose that an agent is in a 3 x 3 maze environment like the one shown in Figure 4.19. The agent knows that its initial location is (3,3), that the goal is at (1,1), 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 or is a new state.
(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 slates?
(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.

a. Online search is equivalent to offline search in belief-state space where each action in a belief-state can have multiple successor belief-states: one for each percept the agent could observe after the action. A successor belief-state is constructed by taking the previous belief-state, itself a set of states, replacing each state in this belief-state by the successor state under the action, and removing all successor states which are inconsistent with the percept.
The initial belief state has 1024 states in it, as we know whether two edges have walls or not but nothing more.
There are $2^{2^{12}}$ possible belief states, one for each set of environment configurations.

b. Assuming the external walls are known, there are two internal walls and hence 4 possible percepts.

c. From each belief state, the agent chooses a single action which can lead to up to 8 belief states. Given the possibility of having to retrace its steps at a dead end, the agent can explore the entire maze in no more than 18 steps, so the complete plan (expressed as a tree) has no more than 8^{18} nodes. On the other hand, there are just 3^{12} reachable belief states.

7. 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)?

Infinite two-dimensional grid, so online DFS starting at (0, 0) will always apply vectors (1, 0) and will not reach (1, -1)

Since we can observe successor states, we always know how to backtrack from to a previous state. This means we can adapt iterative deepening search to solve this problem. The only difference is backtracking must be explicit, following the action which the agent can see leads to the previous state.

The algorithm expands the following nodes:

Depth 1: (0,0), (1,0), (0,0), (0,1), (0,0), (-1,0), (0,0), (0, -1), (0,0)

Depth 2: (0,0), (1,0), (2,0), (1,0), (1,1), (1,0), (1, -1)