2018년 4월 17일 화요일 오후 8:19

Depth-First Search
Breadth-First Search
Uniform-Cost Search

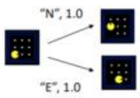
Optimal vs. complete planning
Planning vs. replanning

Search Problems

용어!

What are the states?
How many states?
What are the actions?
How many successors from the start state?
What should the costs be?

- A **search problem** consists of:
  - A state space
  - A successor function (with actions, costs)
  - A start state and a goal test
- A **solution** is a sequence of actions (a plan) which transforms the start state to a goal state



example

- State space:
  - Cities
- Successor function:
  - Roads: Go to adjacent city with cost = distance
- Start state:
  - Arad
- Goal test:
  - Is state == Bucharest?
- Solution?

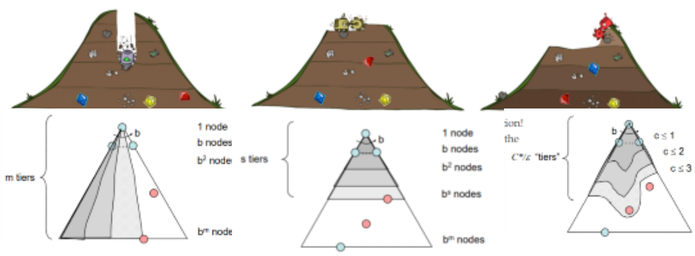
Problem이 무엇인지에 따라
달라지는 것을 볼 수 있다.

A search state keeps only the details needed for planning (abstraction)

- Problem: Pathing
  - States: (x,y) location
  - Actions: NSEW
  - Successor: update location only
  - Goal test: is (x,y)=END
- Problem: Eat-All-Dots
  - States: {(x,y), dot booleans}
  - Actions: NSEW
  - Successor: update location and possibly a dot boolean
  - Goal test: dots all false

Search Trees and State Space Graphs

Depth-First Search | Breadth-First Search | Uniform Cost Search



No, optimal | Only if costs are all 1 | optimal

, takes time $O(b^m)$ | $O(b^s)$ | $O(b^{C^*/\varepsilon})$

space does the fringe take? $O(bm)$ | $O(b^s)$ | $O(b^{C^*/\varepsilon})$

LIFO stack | *FIFO queue* | *priority queue* *cheapest node first*

```
function TREE-SEARCH(problem, fringe) return a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    for child-node in EXPAND(STATE[node], problem) do
      fringe ← INSERT(child-node, fringe)
    end
  end
```

Tree A*

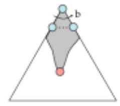
Combining UCS and Greedy

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

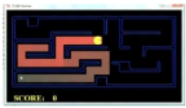
Uniform-cost | Greedy

Admissible Heuristics

$0 \leq h(n) \leq h^*(n)$



VS



No information about goal location
->단점, 방향성x

greedy

UCS and A*

never expand a state twice

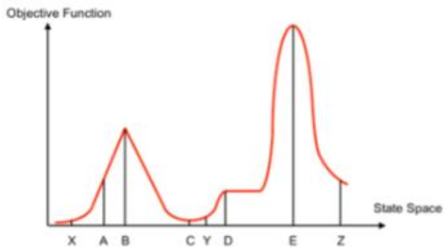
A* Graph Search

Consistency of Heuristics

$h(A) - h(C) \leq \text{cost}(A \text{ to } C)$

노드 확장 순서,
결과, 남아있는 노드

```
function GRAPH-SEARCH(problem, fringe) return a solution, or failure
  closed ← an empty set
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST(problem, STATE[node]) then return node
    if STATE[node] is not in closed then
      add STATE[node] to closed
      for child-node in EXPAND(STATE[node], problem) do
        fringe ← INSERT(child-node, fringe)
      end
  end
```



QUIZ >

Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

풀기

Constraint Satisfaction Problems

Variables: WA, NT, Q, NSW, V, SA, T

Domains: D = {red, green, blue}

Constraints: adjacent regions must have different colors

n-queens

Variables: Q_k

Domains: $\{1, 2, 3, \ldots N\}$

Constraints:

Backtracking Search

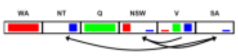
백트레킹:더 이상 전진불가

Idea 1: One variable at a time
Idea 2: Check constraints as you go

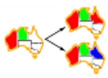
Forward checking

-> efficient

Arc Consistency in a CSP



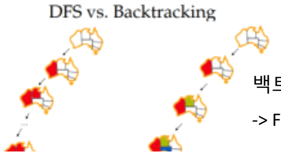
Ordering: Minimum Remaining Values, Least Constraining Value



Filtering
Structure – turns out trees are easy?

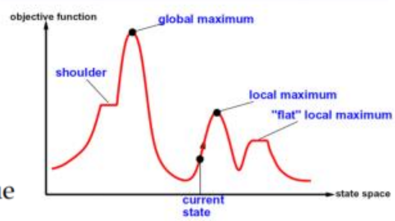
BFS 아주 비효율

DFS vs. Backtracking



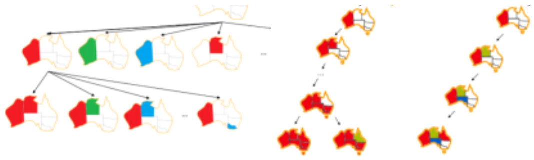
백트레킹에서 진화
-> Filtering : 포워드 체킹

Hill Climbing Diagram



상대가 반드시
최선의 선택을 할 것임

Expectimax 출현



백트레킹에서 진화

-> Filtering : 포워드 체킹

상대가 반드시
최선의 선택을 할 것임

Expectimax 출현

Worst-Case vs. Average Case

Minimax Values

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

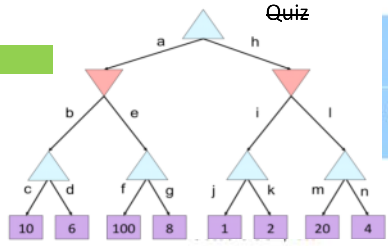
```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

Just like (exhaustive) DFS
Time: O(b^m)
Space: O(bm)

~~Depth지정에 따라~~
~~결과 다름~~

Depth Matters

~~Quiz~~



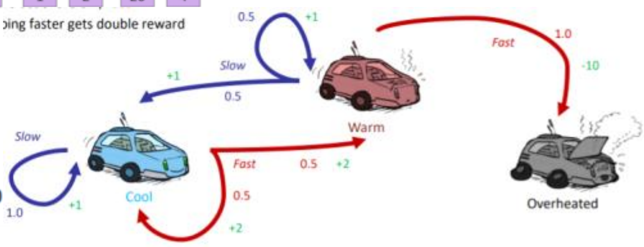
	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

Alpha-Beta Pruning

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

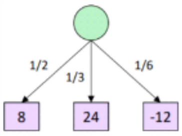
ɔing faster gets double reward



Expectimax

```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

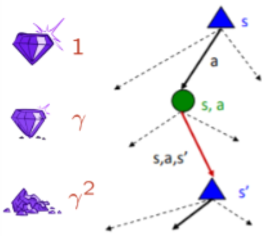


v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

An MDP is defined by:

- A set of states s ∈ S
- A set of actions a ∈ A
- A transition function T(s, a, s')
  - Probability that a from s leads to s', i.e., P(s' | s, a)
  - Also called the model or the dynamics
- A reward function R(s, a, s')
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state

policy $\pi^*: S \rightarrow A$



s is a *state*

(s, a) is a *q-state*

(s,a,s') is a *transition*

Given:

10				1
a	b	c	d	e

- Actions: East, West, and Exit (only available in exit states a, e)
- Transitions: deterministic

Quiz 1: For γ = 1, what is the optimal policy?

10				1

Quiz 2: For γ = 0.1, what is the optimal policy?

10				1

Quiz 3: For which γ are West and East equally good when in state d?

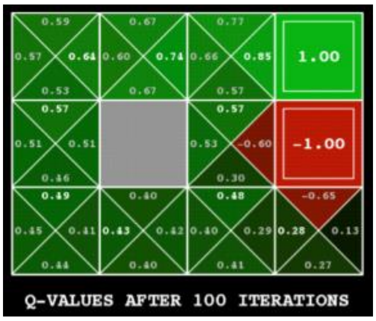
Markov decision processes:

- Set of states S
- Start state s_0
- Set of actions A
- Transitions P(s'|s,a) (or T(s,a,s'))
- Rewards R(s,a,s') (and discount γ)

$$V^*(s) = \max_a Q^*(s,a)$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V^*(s')\right]$$



Value Iteration

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$$

V_2: (empty)

V_1: S: 1 / F: .5*2+.5*2=2

V_0: 0 0 0

현실(don't know T or R)

A set of states s ∈ S
A set of actions (per state) A
A model T(s,a,s')
A reward function R(s,a,s')

ll looking for a policy π(s)

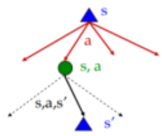
Value iteration 단점 -> Policy iteration
결과는 같지만 과정은 다른,

$$V_0^\pi(s) = 0$$

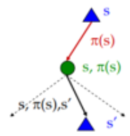
$$V_{k+1}^\pi(s) \leftarrow \sum_{s'} T(s,\pi(s),s')\left[R(s,\pi(s),s') + \gamma V_k^\pi(s')\right]$$

Efficiency: O(S^2) per iteration vs O(S^2A)

Do the optimal action



Do what π says to do



Model-Based Learning Model-Free Learning

Model-Free (temporal difference) Learning

Sample of V(s): $sample = R(s,\pi(s),s') + \gamma V^\pi(s')$

Update to V(s): $V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + (\alpha)sample$

Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

$$V^\pi(s) \leftarrow (1-\alpha)V^\pi(s) + \alpha\left[R(s,\pi(s),s') + \gamma V^\pi(s')\right]$$

Q-Value Iteration

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')\right]$$

Policy Iteration
0단계: policy를 임의로 정한다.
1단계: 현재의 policy에 대해 (not optimal) utility를 계산
2단계: 계산한 utility를 이용하여 policy를 개선
개선한 policy를 현재의 policy로 한다.
3단계: 1~2단계를 policy가 수렴할 때까지 반복한다.

상태집합 S ∋ s
시작상태 (start state) 종료상태 (terminal state)
행동집합 A ∋ a
전이함수(transition function) T(s, a, s') = P(s' | s, a)
현재 상태 s에서 행동 a를 했을 때, 다음상태가 s'이 될 확률
보상함수(reward function) R(s, a, s')

Utility 보상의 합
reward sequence가 주어졌을 때 구함

$$\text{Utility} = r_0 + r_1 + r_2 + r_3 + \cdots + r_{end}$$

$$\text{utility} = \sum_{t=0}^{t=end} r_t$$

$$\text{Utility} = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \gamma^3 r_3 + \cdots + \gamma^{end} r_{end}$$