ôn cuối kỳ

note: với n nhỏ (n:so lần chơi) ⇒? phương sai lớn làm cho khả năng ước tính sai lớn hơn (

với n lớn -⇒ phương sai nhỏ vì chọn nhiều ng thì trung bình sẽ gần xác xuất ước tính hơn là chọn n nhỏ

⇒ xác xuất ước tính đúng lớn hơn

R(tau),tổng điểm thưởng theo bộ trọng số tìm được khi tuân theo chiến lượt theta(0-), nếu dươnglàm tăng xác xuất theo hướng có lợi

▼ các thuật toán search

- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test

- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans

Heuristic là các kỹ thuật dựa trên kinh nghiệm là :

Một hàm ước tính mức độ gần của một trạng thái với một mục tiêu,Được thiết kế cho một vấn đề tìm kiếm cụ thể
Ví dụ: Khoảng cách Manhattan, Khoảng cách Euclid để vỗ nhẹ

search : DFS,bfs

ucs search : vấn đề : chạy chậm

Chiến lược: mở rộng chi phí đường dẫn thấp nhất
Ưu điểm: UCS đã hoàn thiện và tối ưu!
Nhược điểm: Khám phá các tùy chọn theo mọi “hướng”, Không có thông tin về vị trí mục tiêu

greedy: nhanh nhưng ko đảm bảo mở rộng nút có vẻ gần với mục tiêu nhất

Chiến lược: mở rộng một nút mà bạn cho là gần nhất với trạng thái mục tiêu
:Heuristic: ước tính khoảng cách đến mục tiêu gần nhất cho mỗi trạng thái

Một trường hợp phổ biến:Best-first đưa bạn đến thẳng mục tiêu (sai)
Trường hợp xấu nhất: giống như một DFS được định hướng sai

ucs+greedy = A* :Chiến lược: mở rộng nút với tổng chi phí đường dẫn và chi phí ước tính thấp nhất.

Thuật toán A* có những ưu điểm quan trọng như tính tối ưu và tính đầy đủ. Tuy nhiên,
nếu hàm heuristic không tốt, thuật toán sẽ phải xem xét nhiều trạng thái và có yêu cầu bộ nhớ
trong trường hợp xấu nhất là O(bm). Yêu cầu bộ nhớ theo hàm mũ như vậy làm giảm khả năng
sử dụng A*. Để giải quyết vấn đề này có thể sử dụng phiên bản của A* được gọi là A* sâu
dần (Iterative Deepening A*) có yêu cầu bộ nhớ tỷ lệ tuyến tính với độ sâu lời giải.
Phương pháp: Nguyên tắc của A* sâu dần là lặp lại việc tìm kiếm theo chiều sâu trên
các cây tìm kiếm con có giá trị hàm f(n) không lớn hơn các ngưỡng 0, α, 2α, 3α, .v.v

Admissible Heuristics

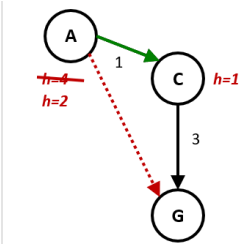
Inadmissible (pessimistic) (bi quan) phá vỡ sự lạc quan bằng cách bẫy những kế hoạch tốt ở ngoài rìa
Admissible (lạc quan) làm chậm các kế hoạch tồi nhưng không bao giờ vượt quá chi phí thực

A heuristic h is admissible (optimistic) if: o ≤ h(n) ≤ h*(n)

where h*(n) chi phí thực cho mục tiêu gần nhất

Tính nhất quán của Heuristics (heuristic is consistent)
Khả năng chấp nhận: chi phí phỏng đoán ≤ chi phí thực tế để đạt được mục tiêu
h (A) ≤ chi phí thực tế từ A đến G
Tính nhất quán: chi phí “cung” heuristic ≤ chi phí thực tế cho mỗi cung
h (A) - h (C) ≤ chi phí (A đến C)

Hệ quả của tính nhất quán:
Giá trị f dọc theo một đường dẫn không bao giờ giảm
h (A) ≤ chi phí (A đến C) + h (C)
Tìm kiếm đồ thị A * là tối ưu



Optimality of A* Graph Search(Tính tối ưu của Tìm kiếm Đồ thị A *)

Proof:
Main idea: Show nodes are popped with non-decreasing f-scores for n'
popped after n : f(n') ≥ f(n) is this enough for optimality?
Sketch:
assume: f(n') ≥ f(n), for all edges (n,a,n') and all actions a is this true?
proof: A* never expands nodes with the cost f(n)>C*
proof by induction(1) always pop the lowest f-score from the fringe, (2) all new nodes have larger (or equal) scores, (3) add them to the fringe, (4) repeat

Bằng chứng:
Ý chính: Các nút hiển thị được xuất hiện với điểm số f không giảm cho n 'xuất hiện sau n: f (n') ≥ f (n) điều này có đủ tối ưu không?
Phác thảo:
giả sử: f (n ') ≥ f (n), với mọi cạnh (n, a, n') và mọi hành động a điều này có đúng không?
bằng chứng: A * không bao giờ mở rộng các nút với chi phí f (n)> C *
bằng chứng bằng quy nạp (1) luôn bật điểm f thấp nhất từ rìa, (2) tất cả các nút mới có điểm lớn hơn (hoặc bằng), (3) thêm chúng vào rìa, (4) lặp lại

Optimality - sự tối ưu trên grap search and tree search

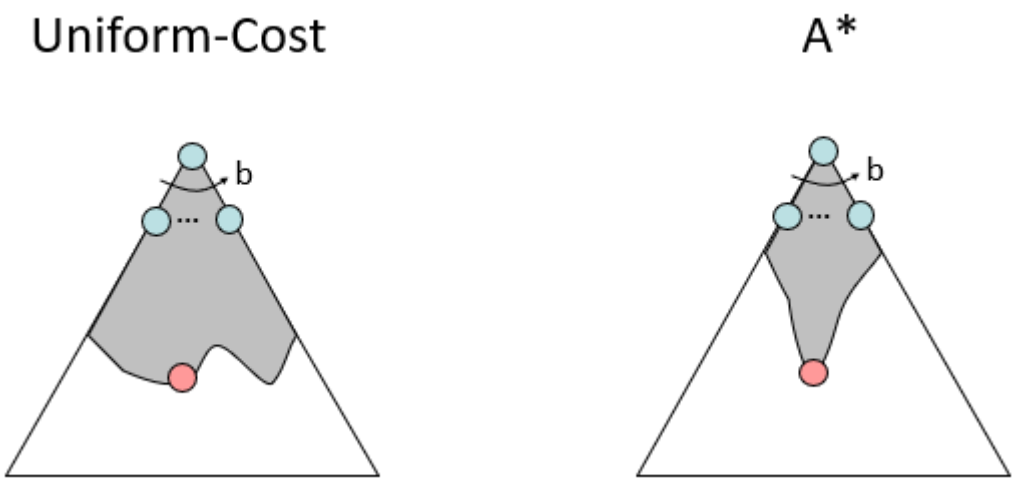
Tìm kiếm cây:
A * là tối ưu nếu heuristic có thể chấp nhận được
UCS là một trường hợp đặc biệt (h = 0)

Tìm kiếm đồ thị:
A * tối ưu nếu heuristic nhất quán (**heuristic is consistent**)

UCS tối ưu (h = 0 là nhất quán)

Tính nhất quán ngụ ý khả năng được chấp nhận

Nhìn chung, hầu hết các phương pháp phỏng đoán tự nhiên có thể chấp nhận được có xu hướng nhất quán, đặc biệt nếu từ các vấn đề thoải mái



▼ search → CSP bài toán thỏa mãn ràng buộc:

CSPs are specialized for identification problems(CSP được chuyên dụng cho các vấn đề nhận dạng)

Các vấn đề ràng buộc về sự hài lòng (CSP):
Một tập hợp con đặc biệt của các vấn đề tìm kiếm
Trạng thái được xác định bởi các biến Xi với các giá trị từ miền D (đôi khi D phụ thuộc vào i)
Kiểm tra mục tiêu là một tập hợp các ràng buộc C chỉ định các tổ hợp giá trị được phép cho các tập con của các biến

`Các thuật toán CSP có mục đích chung sử dụng cấu trúc đồ thị để tăng tốc độ tìm kiếm.`

Solving CSPs

Các trạng thái được xác định bởi các giá trị được chỉ định cho đến nay (chỉ định một phần)
Trạng thái ban đầu: nhiệm vụ trống, {}
Hàm kế thừa: gán giá trị cho một biến chưa được gán

Kiểm tra mục tiêu: bài tập hiện tại đã hoàn thành và đáp ứng tất cả các ràng buộc

Chúng tôi sẽ bắt đầu với cách tiếp cận đơn giản, ngây thơ, sau đó cải thiện nó

> 💡 is BFS is antive search ??

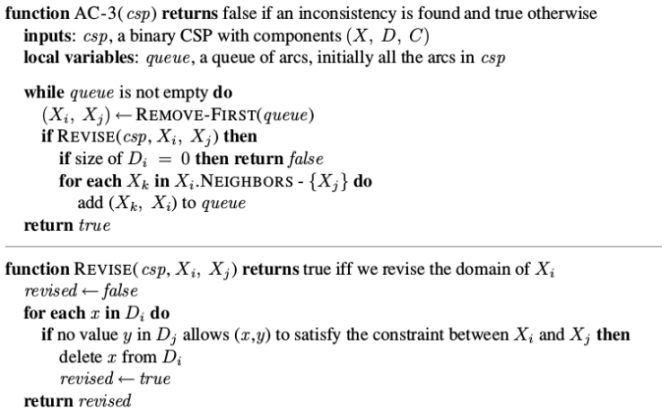
Varieties of CSPs - Discrete Variables(biến rời rạc) Continuous variables (biến liên tục)

Unary constraints (Ràng buộc đơn nguyên) TP.HCM ≠ blue

Binary constraints (Ràng buộc nhị phân) TP.HCM == tp.cantho

Tùy chọn (ràng buộc mềm)**soft constraints**:
Ví dụ: màu đỏ tốt hơn màu xanh lá cây
Thường có thể biểu thị bằng chi phí cho mỗi lần chuyển nhượng biến đổi
Đưa ra các vấn đề tối ưu hóa bị hạn chế

AC-3



```
function AC-3(csp) returns false if an inconsistency is found and true otherwise
  inputs: csp, a binary CSP with components (X, D, C)
  local variables: queue, a queue of arcs, initially all the arcs in csp

  while queue is not empty do
    (X_i, X_j) ← REMOVE-FIRST(queue)
    if REVISE(csp, X_i, X_j) then
      if size of D_i = 0 then return false
      for each X_k in X_i.NEIGHBORS - {X_j} do
        add (X_k, X_i) to queue
  return true
```

```
function REVISE(csp, X_i, X_j) returns true iff we revise the domain of X_i
  revised ← false
  for each x in D_i do
    if no value y in D_j allows (x,y) to satisfy the constraint between X_i and X_j then
      delete x from D_i
      revised ← true
  return revised
```

backtracking for CSP search

backtracking : quay lui

Bản chất của giải thuật RescusiveBacktracking là phép duyệt theo chiều sâu (DFS)có thêm
bước kiểm tra sự thỏa mãn của các ràng buộc ở mỗi bước

tại sao ko dùng bfs vì vì trong csP tất cả lời giải nằm trong tập đáy nên nếu dùng

kiểm tra ràng buộc trên đường đi và mỗi laafn chỉ sử lý 1 bến

những kỹ thuật nâng cao hiệu xuất (cải tiến) của Backtraking : (`Chúng ta có thể phát hiện sớm lỗi không thể tránh khỏi không?)`

Khi một số biến được gán giá trị, miền giá trị của các biến còn lại cũng sẽ bị co hẹp lại do tập các ràng buộc chi phối.

ordering:

Nguyên tắc 1: Lựa chọn biến mà miền giá trị hợp lệ còn lại là ít nhất (biến có ít lựa
chọn nhất nên được chọn trước để làm giảm độ phức tạp của cây tìm kiếm) - **Minimum Remaining Values**

Nguyên tắc 2: Lựa chọn biến tham gia vào nhiều ràng buộc nhất (gán cho biến khó
thỏa mãn nhất)

Nguyên tắc chọn thứ tự giá trị gán cho biến:lựa chọn giá trị cho biến đang xét sao cho nó tạo ra ít ràng buộc đến các biến còn lại nhất (Ordering: Least Constraining Value)

1 filtering (Nguyên tắc chọn biến tiếp theo):

f**orward tracking:** kiếm tra trước xóa bớt điểm gây ko thỏa mãn ràng buộc ; không phát hiện được phần ko có kết quả) Kiểm tra tiến (kiểm tra trước – forward checking)
Trong nguyên tắc chọn thứ tự giá trị gán cho một biến, chúng ta cần phải kiểm tra xem
giá trị định gán sẽ tác động thế nào đối với các biến chưa gán thông qua các ràng buộc.
Việc hạn xác định tác động trước như vậy gọi là forward checking.
Forward checking còn
có tác dụng hạn chế không gian tìm kiếm (hạn chế miền giá trị cho các biến còn lại khi
biến hiện tại được gán một giá trị cụ thể)

AC3: mỗi lần gán giá trị kiểm tra tất cả các cung trong bộ dữ liệu có gây ra vi phạm ràng buộc hay không (làn truyền ràng buộc)

ordering : minimun MRV gán biến cho có số giá trị hợp lệ còn lại là ít nhất

chọn biến ít gây ra ảnh hưởng đến ràng buộc ,Cho một lựa chọn biến, hãy chọn giá trị hạn chế nhất (tức là giá trị loại trừ ít giá trị nhất trong các biến còn lại) chọn biến mà khi chọn nó thì miến giá trị của các biến chưa được

gán giá trị bị ràng buộc là ít nhất ⇒ `tạo ra nhiều sự lựa chọn hơn để tìm ra kết quả nhanh hơn.`

Hãy nhớ: Xóa từ đuôi vì nếu xóa ràng buộc từ trên xuống thì các ràng buộc sẽ bị ảnh hưởng nên mõi lầm xóa từ tren xuống phải kiểm tra lại từ đầu để kiếm tra xem nó có vi phạm ràng buộc với nút đã tô hay không nên sẽ tốn nhiều lần kiểm tra!

khai thác cấu trúc đồ thị: catset condition

Giả sử một đồ thị gồm n biến có thể được chia thành các bài toán con chỉ gồm c biến:
Chi phí giải pháp trong trường hợp xấu nhất là O ((n / c) (d^c)), tuyến tính theo n
Ví dụ: n = 80, d = 2, c = 20
2^80 = 4 tỷ năm ở 10 triệu nút / giây
(4) (2*20) = 0,4 giây ở 10 triệu nút / giây

Định lý: nếu đồ thị ràng buộc không có vòng lặp, CSP có thể được giải quyết trong thời gian O (n*d^2)
So sánh với các CSP chung, trong đó thời gian trong trường hợp xấu nhất là O (d^n)

K-Consistency

Tăng mức độ nhất quán

1-Nhất quán (Node Consistency): Mỗi miền của một nút có một giá trị đáp ứng các ràng buộc đơn nguyên của nút đó

2-Nhất quán (Arc Consistency): Đối với mỗi cặp nút, bất kỳ sự phân công nhất quán nào cho nút này đều có thể được mở rộng cho nút kia

K-Tính nhất quán: Với mỗi k nút, bất kỳ phép gán nhất quán nào cho k-1 đều có thể được mở rộng đến nút thứ k.

Cao hơn k thì đắt hơn để tính toán

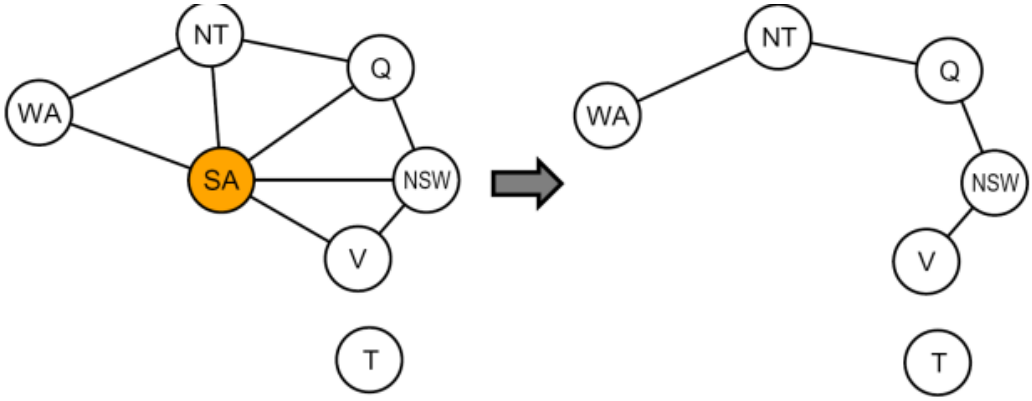
Tree-structured CSPs solve

Yêu cầu 1: Sau khi chuyền ngược, tất cả các cung từ lá đến lá đều nhất quán
Chứng minh: Mỗi X → Y được tạo ra nhất quán tại một điểm và miền của Y không thể giảm sau đó (vì con cái của Y.
đã được xử lý trước Y)
Yêu cầu 2: Nếu các vòng cung từ gốc đến lá nhất quán, chuyển tiếp sẽ không

quay lại
Bằng chứng: Cảm ứng về vị trí
Tại sao thuật toán này không hoạt động với các chu trình trong biểu đồ ràng buộc?
Lưu ý: chúng ta sẽ thấy lại ý tưởng cơ bản này với lưới Bayes

Nearly tree-structured CSPs

(Bạn cần biết trường hợp **k = 2: tính nhất quán của cung**)



Cutset size c gives runtime **O((d^c)(n-c)d^2)** very fast for small c

Sau khi có được nearby tree structed tree Solve the residual CSPs (tree structured)

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$

Local Search

Tìm kiếm địa phương: cải thiện một tùy chọn duy nhất cho đến khi bạn không thể làm cho nó tốt hơn
Chức năng kế thừa mới: thay đổi cục bộ
Nói chung là nhanh hơn và hiệu quả hơn nhiều về bộ nhớ (nhưng không đầy đủ và không tối ưu)
Khá khó tránh khỏi khi trạng thái là "chính bạn"

Hill Climbing

Bắt đầu ở bất cứ đâu→ Lặp lại: chuyển sang trạng thái lân cận tốt nhất
Nếu không có hàng xóm nào tốt hơn hiện tại, hãy bỏ

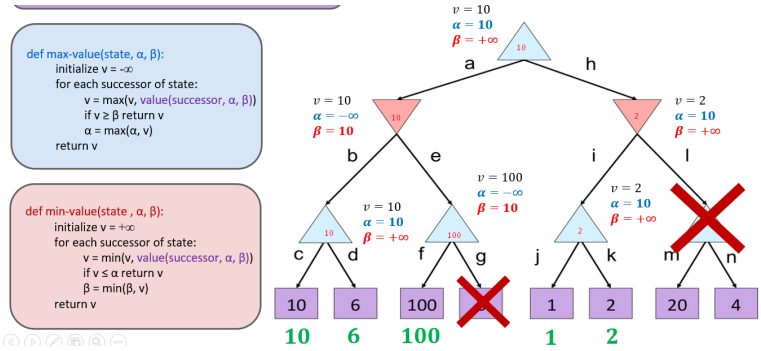
▼ **csp → adversarial search tìm kiếm đối kháng**

minimax:là một tam giác hướng lên và mình chọn hành động cực đại hóa điểm thưởng hành động , địch luôn cực tiểu hóa điểm

bản chất là:vét cạn cây trò chơi = DFS ⇒ lâu

⇒ kỹ thuật cãi thiện hiệu xuất

1 cắt tỉa anphabeta:

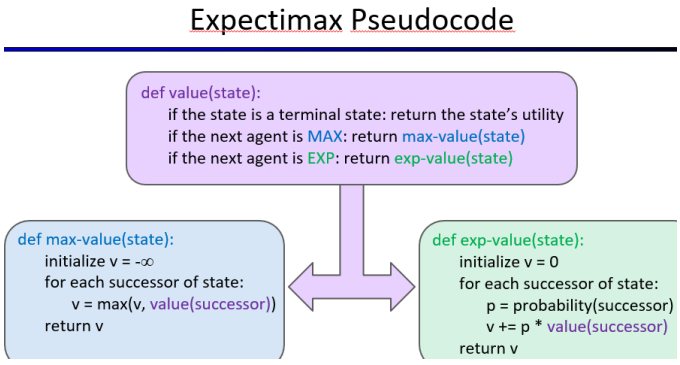


2 sử dụng hàm lượng giá : evaluation fintion

Lưu ý: tính sai của các hàm eval càng ít và càng đi sâu thì tìm kiếm càng ít!

▼ **ADV→ expextimax:**

tam giác hướng lên (mminhf có nhiều hành đôgn) chọn hành dodocngj cực đại hóa điểm thưởng với địch: ko nhất thiết cực tiểu hóa ddierme thưởng → chọn theo phân phối xác xuất ⇒ chọn hành động có giá trị kì vọng lớn nhất



EXpecimax → MDP markow pecision process

What **Utilities** to Use?

Đối với trường hợp xấu nhất suy luận minimax, quy mô chức năng đầu cuối không quan trọng
Chúng tôi chỉ muốn các tiểu bang tốt hơn có đánh giá cao hơn (thực hiện đúng

thứ tự)
Chúng tôi gọi đây là sự vô cảm đối với các phép biến đổi đơn điệu

Các tiện ích đến từ đâu?→ Trong một trò chơi, có thể đơn giản (+ 1 / -1)

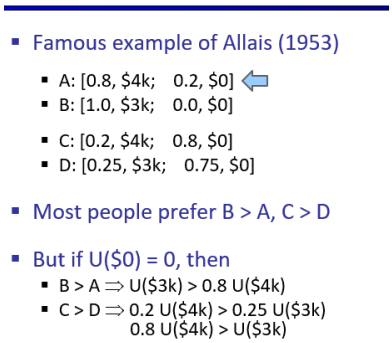
Đối với suy luận expectimax trường hợp trung bình, chúng ta cần các cường độ để có ý nghĩa

Micromorts: khả năng tử vong một phần triệu, hữu ích cho việc thanh toán để giảm rủi ro sản phẩm, v.v.

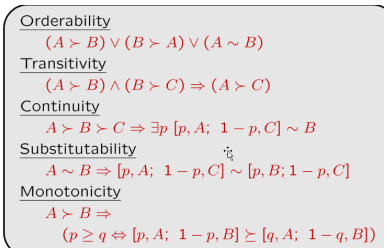
QALYs: số năm sống được điều chỉnh chất lượng, hữu ích cho các quyết định y tế liên quan đến rủi ro đáng kể

Lưu ý: hành vi là bất biến theo chuyển đổi tuyến tính tích cực

Chỉ với các giải thưởng xác định (không có lựa chọn xổ số), chỉ có thể xác định tiện ích thứ tự, tức là tổng thứ tự trên các giải thưởng



Tiên đề về tính hợp lý



mô hình MDP (Markov Decision Processes)

Non-Deterministic Search -One way to solve them is with expectimax search

"Markov" thường có nghĩa là cho trạng thái **hiện tại, tương lai và quá khứ là độc lập →kết quả hành động chỉ phụ thuộc vào trạng thái hiện tại**

π* là một chính sách tối ưu là một chính sách tối đa hóa tiện ích mong đợi nếu được tuân thủ

Expectimax không tính toán toàn bộ chính sách. Expectimax chỉ tính toán hành động **cho một trạng thái duy nhất**

cho biết :khi làm hành động a thì sác xuất qua được s' là bao nhiêu (P(S',s,a))

có những bài toán: P(S',s,a)

policy evaluation:O(S^2) per iteration - đã có ,theo policy

V^π(s)*: lợi ích(giá trị) đạt được mong đợi có được bằng cách bắt đầu từ trạng thái *s* và tuân theo chính sách *π
***V^π*: hàm giá trị trạng thái cho chính sách *π*.(state-value)**

Q^π(s,π(s)) =*V^π(s)*

$$V^{\pi}(s) = Q^{\pi}(s,\pi(s)) = \sum_{s'} P(s'|s,\pi(s))[R(s,\pi(s),s') + \gamma V^{\pi}(s')]$$

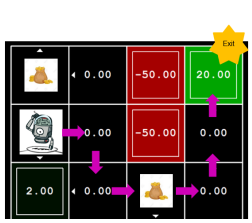
***Q^π (s, a)*: tiện ích mong đợi có được bằng cách bắt đầu từ trạng thái *s*, thực hiện hành động *a*, và sau đó tuân theo chính sách *π*.**

***Q ^ π*: hàm giá trị hành động cho chính sách *π*.(action-value)**

tính ra *V^π(s)* ,Q^π(s,a) ⇒ tính ra valueiteration V*(s) hàm giá trị trạng thái tối ưu Q*() -hành động tối ưu

Discounting reward we recive

Trạng thái hấp thụ: đảm bảo rằng đối với mọi chính sách, **cuối cùng sẽ đạt được trạng thái cuối**



Living Reward = -0.3, Noise = 0.2

$$U = (-0.3) + (0.9) \times (-0.3) + (0.9)^2 \times (-0.3 + 0.5) + (0.9)^3 \times (-0.3) + (0.9)^4 \times (-0.3) + (0.9)^5 \times (-0.3) + (0.9)^6 \times 20 = 9.628$$

Phương trình Bellman: Điều kiện cần thiết để đạt được sự tối ưu trong các bài toán tối ưu hóa được xây dựng dưới dạng Lập trình động

Value Iteration : O(S^2*A) per iteration V^∗(s) không theo policy -Evaluate an optimal policy

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

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

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

Bellman equations

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

$$V_t^*(s) \leftarrow \max_{a \in Actions(s)} \sum_{s'} P(s'|s,a)\,[R(s,a,s') + \gamma V_{t-1}^*(s')]$$

π^∗ (s) =argmax(Q^∗ (s,a)) = **hành động tối ưu** trong các hành động
V^∗ (s) =max(Q^∗ (s,a)) : **giá trị điểm thưởng mong đợi** bắt đầu tại s và tuân theo hoạt động tối ưu( )

Q^∗ (s,a) =**giá trị điểm thưởng mong đợi** bắt đầu khi thực hiện hành động a= {a1,a2,a3,a4..} từ trạng thái s và tuân theo hành động a (**action-value**)

Vấn đề :"max" ở mỗi trạng thái hiếm khi thay đổi
Chính sách thường hội tụ rất lâu trước khi các giá trị

Policy iteration ⇒ π^∗(s)

step 1 rand dom a policy π_i(s) tùy ý cho s ∈ S.→ 2.For fixed current policy π_i, find values with policy evaluation → 3.For fixed values, get a better policy using policy extraction

Both are dynamic programs for solving MDPs

Reforcementlearning : tính ra chiến lượt tối ưu nhưng không có MDP:

T(s,a,s') = transitions :xác xuất trạng thái s theo hành động a thành s' trong tổng số trường hợp bắt **đầu từ s mà đi theo hướng a**

R(s,a,s') = reward tổng điểm thưởng trung bình nhận được của th **đầu từ s mà đi theo hướng a**

Direct Evaluation:

- Goal: Compute $V^{\pi}(s)$ for each state s under given π
- Idea: Average together observed sample values
  1. Act according to π for several episodes
  2. Afterward, for every state s and every time t that s is visited: determine the rewards $r_{t+1}, r_{t+2}, \dots, r_T$ subsequently received in the episode.
  3. Sample for s at time t = sum of discounted future rewards.
     $$sample = G_t(s) = r_{t+1} + \gamma R_{t+2}(s')$$
  4. Average those samples.

Good: doesn't require any knowledge of T, R,easy to understand, computes the correct average values, using just sample transitions

Bad :Nó lãng phí thông tin về các kết nối trạng thái , Mỗi trạng thái phải được học riêng biệt, Vì vậy, phải mất một thời gian dài để học hỏi

giảm learning rate (alpha) có thể cung cấp mức trung bình hội tụ

Lưu ý:
Bạn phải khám phá đủ,Cuối cùng bạn phải đưa ra tỷ lệ học tập đủ nhỏ
… Nhưng không giảm quá nhanh
Về cơ bản, trong giới hạn, việc bạn chọn hành động như thế nào không quan trọng (!)

Temporal Difference Learning

1. Initialize each $V^{\pi}(s)$ with some value.
2. Observe experience (s, π (s), r, s').
3. Use observation in rough estimate of long-term reward $V^{\pi}(s)$
   $$sample = r + \gamma V^{\pi}(s')$$
4. Update $V^{\pi}(s)$ by moving values slightly toward estimate:
   $$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha\big(sample - V^{\pi}(s)\big)$$

$$V^{\pi}(s) \leftarrow (1-\alpha)V^{\pi}(s) + \alpha \cdot sample$$
$$V^{\pi}(B) \leftarrow (1-\alpha)V^{\pi}(B) + \alpha(r + \gamma V^{\pi}(C))$$
$$V^{\pi}(B) \leftarrow V^{\pi}(B) + \alpha(r + \gamma V^{\pi}(C) - V^{\pi}(B))$$

problem:

Tuy nhiên, nếu chúng tôi muốn chuyển các giá trị thành một chính sách (mới), chúng tôi sẽ bị chìm:

Note: This is the simplest Monte Carlo method

→ tính V^π(s) bằng cách: **monte carlo policy evaluation** tính ra đc Vpi(s) or Qpi(s,a) dùng n ván chơi phát sinh ra :

vấn đề đơn giản mà chậm; chơi hết n ván chơi mới cập nhập

→ đẻ cải thiện sau này có dùng temporal diferent learning → tính ra Vpi(s) or Qpi(s,a) hiệu quả hơn

muốn đi tìm chiến lượt tối ưu

→ thuật toán Q-learning or SARSA

→ giúp tìm ra Q*(s,a) ,V*(s)

Learn Q(s,a) values as you go

1. Initialize $Q(s,a) = 0$ for each s,a pair
2. Select an action an observe an experience (s,a,r,s')
3. Consider your old estimate: $Q(s,a)$
4. Use observation in rough estimate of $Q(s,a)$
$$sample = r + \gamma \max_{a'} Q(s',a')$$
5. Update $Q(s,a)$ by moving values slightly toward estimate
$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big)$$
$$= (1-\alpha)Q(s,a) + \alpha \cdot sample$$

Q-learning hội tụ với chính sách tối ưu - ngay cả khi bạn đang hành động kém hiệu quả!

Sarsa-with

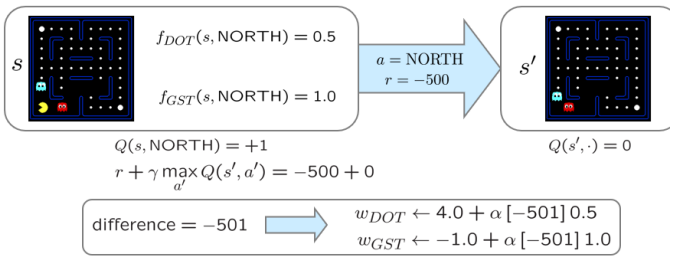
1. Initialize $Q(s,a) = 0$ for each s, a pair, $t = 0$
2. Set $\pi_b(s) = \arg\max_{a'} Q(s,a)$ with prob. $1-\varepsilon$, else random.
3. Sample action a from policy π_b an observe an experience (s,a,r,s')
4. Loop
   1. Sample action a' from policy π_b an observe an experience (s',a',r'',s'')
   2. Consider your old estimate: $Q(s,a)$
   3. Use observation in rough estimate of $Q(s,a)$
   $$sample = r + \gamma Q(s',a')$$
   4. Update $Q(s,a)$ by moving values slightly toward estimate
   $$Q(s,a) \leftarrow Q(s,a) + \alpha\big(sample - Q(s,a)\big) = Q(s,a) + \alpha\big(r + \gamma Q(s',a') - Q(s,a)\big)$$
   $$\pi_b(s) = \arg\max_{a'} Q(s,a) \text{ with prob. } 1-\varepsilon, \text{else random.}$$
   5. $s \leftarrow s'; a \leftarrow a'$

Deep RL → Deep Q network

mà nếu số trạng thái quá lớn không hoạt động nỗi : vì bản chất chúng là 1 bảng vì số lượng trạng thái quá nhiều bảng chứa không nỗi

sau đó phát triển ra Deep RL → Deep Q network không phải bảng nữa mà là 1 mạng noron,input : đầu vào là trạng thai , hành động → ouput giá trị ước lượng trạng thái đầu ra là gì :

$$Q(s,a) = 4.0 f_{DOT}(s,a) - 1.0 f_{GST}(s,a)$$



$$Q(s,a) = 3.0 f_{DOT}(s,a) - 3.0 f_{GST}(s,a)$$

$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i) \right)^2$$

Approximate q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s,a) \right] f_m(s,a)$$

"target" "prediction"

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$

$$= f_m(s,a) \text{ when linear approximation}$$

policy gradien → cx có mang noron input : trạng thái s out put : trong trạng thái S xác xuất hành động a là bao nhiêu

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$

$$= f_m(s,a) \text{ when linear approximation}$$