AI stuff for exam

COURSE 2:

Hanoi Towers :

Choosing a representation for a state

Compact enough to be easy to store and perform changes on it

A list of towers where pieces are placed, in the order of piece size, with the number of towers added at the beginning

$$(3, 3, 3, 3, 3, 1, 1, 2, 2)$$

$$(n, t_1, t_2, \ldots, t_m),\ 1 \le t_i \le n$$

A representation for a state must be compact, expressive, include all data, not contain ambiguities

Special states

Initial state (3, 1, 1, 1, 1, 1, 1, 1, 1)

State Initialize (int n, int m)
{
    Return $(n, \underbrace{1, 1, 1, \ldots, 1}_{m})$;
}

There can be more than one initial state. There has to be at least one initial state.

Special states

Final state(s) (n, n, n, n, n, n, n, n, n)

```
Boolean IsFinal (State s)
{
```

If s = $\underbrace{(k, k, k, \dots, k)}_{m+1}$ then return true;

else return false;

```
}
```

There can be more than one final state. There has to be at least one final state.

Transitions

Only one possible way to change current state: move one piece to another tower.

$(n, t_{11}, t_{12},\dots,t_{1m}) \rightarrow (n, t_{21}, t_{22},\dots,t_{2m})$, where $t_{1i} = t_{2i}$ for all $1 \le i \le m$, except exactly one i = k

State Transition (State s, piece, tower)

Valid transitions

1. No smaller piece is placed atop piece k
2. No smaller piece ends up below piece k

1. $t_{1i} \mp t_{1k}$, for all $1 \le i < k$
2. $t_{2i} \mp t_{2k}$, for all $1 \le i < k$

Boolean Validate (State s, piece, tower)

Implement transitions and validation separately, it’s good practice!

Search strategy

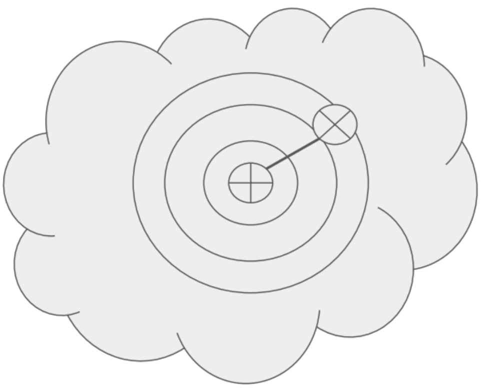
```
Void strategy(State s)
{
    While (!isFinal(s))
        {
            Choose piece, tower;
            If (Validate (s, piece, tower))
                s = Transition(s, piece, tower);
        }
}
```

Uninformed search strategies:

1. random
2. bfs and uniform cost
3. dfs and iterative deepening
4. bkt
5. bidirectional

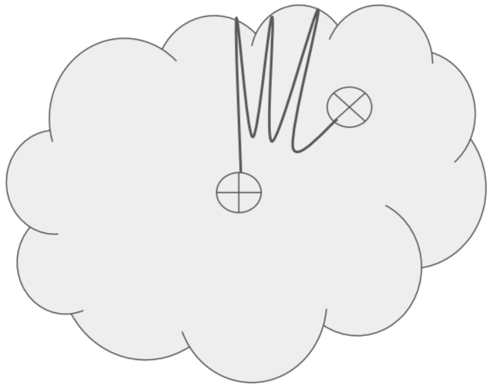
Breadth First Search and Uniform Cost

- Visit states in the order of distance (number of transitions) from the initial state.
- Explores all immediate neighbors (accessible states) until no more neighbors or final state found.
- Has to memorise each generated state: very costly.
- Might visit the same state multiple times.
- Finds shortest path (optimum solution)
- Uniform cost: if options have different costs, explore cheaper paths first



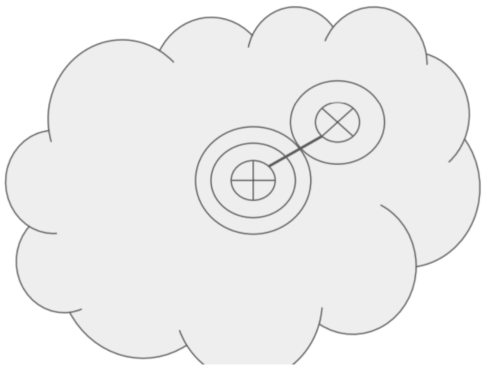
Depth First Search and Iterative Deepening Search

- Visit one immediate neighbor of the current state until final state is found, return to previous unexplored neighbor if no more neighbours available for current state.
- Has to memorise each generated state: very costly.
- Might visit the same state multiple times.
- Might not finish (if loops are present in the problem space)
- IDS: Explore only up to an increasing depth (distance from initial state) - no more infinite paths



Bidirectional Search

- Starts exploring from both the initial and final state simultaneously
- Has to memorise each generated state, but they should not be that many
- Finds shortest path (optimum solution)
- What are the reverse transitions?



Comparison

Complexity of the solution is given by the problem, NOT by the way in which you solve it

Criterion	BFS	DFS	IDS	Bidirectional	BKT	Random
ETC	N^O	N^A	N^O	N^O	N^A	N^A
Optimum	Yes	No	Yes	Yes	No	No
All	Yes	No	Yes	Yes	Yes	No

N = average number of accessible states
O = length of the optimum solution
A = length of the average solution

COURSE 3:

Gasirea unui drum intr-un graf - problema NP-completa - Pentru aceasta trebuie

1. descrierea unui model
2. identificarea starilor speciale si a spatiului problemei
3. descrierea si validarea tranzitiilor
4. strategie de cautare

Exista 2 tipuri de strategii de cautare:
neinformate: se exploreaza complet spatiul problemei si se va gasi sigur o solutie
informate : exista reguli EURISTICE

Diferenta intre regula de deductie si euristici consta in faptul ca regula de ded. poate fi demonstrata in timp ce euristica nu poate, motiv pentru care pot fi contrazise.

In euristica exista o proprietate: se vor plasa satrile finale in extreme

h: S -> [min, max], h(IS) = min/max, h(FS) = max/min

An admissible heuristic never overestimates the distance between a state and the goal.

Pentru turnurile Hanoi : exista un nmar de piese care trebuie asezate pe un anumit turn. Daca piesa de start este 1, iar scopul este n, atunci trebie sa gasim toate starile posibile.

$$(m, \underbrace{1, 1, \dots, 1}_{m})$$

$$(n, m, m, \dots, m)$$

$$m, n \longrightarrow \frac{m}{m+2}$$

Ne dorim o euristica care sa intoarca un numar cat mai apropriat de numarul de stari din problema.
h2(IS) = m -> m+ m+1+ m+2+ …+ m+n
h2(FS) = m*n

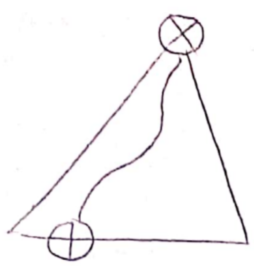
La fiecare pas, euristica trebuie sa calculeze m stari

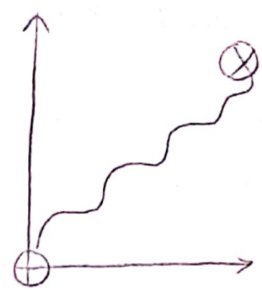
Exista 3 moduri de a reprezenta o solutie:

- path in a graph

- path in a space -> ne spune despre complexitatea problemei si este folosit mai mult pentru strategii

- heuristic function -> o proiectie intr-un grafic; ne prezinta avantajele acelei euristici





Strategii informale

1)
2) Greedy -> alege mere cea mai buna mutare prin evaluarea tuturor starilor disponibile accesibile din starea curenta. Apoi se va selecta cea mai apropriata stare neexplorata de cea dorita.

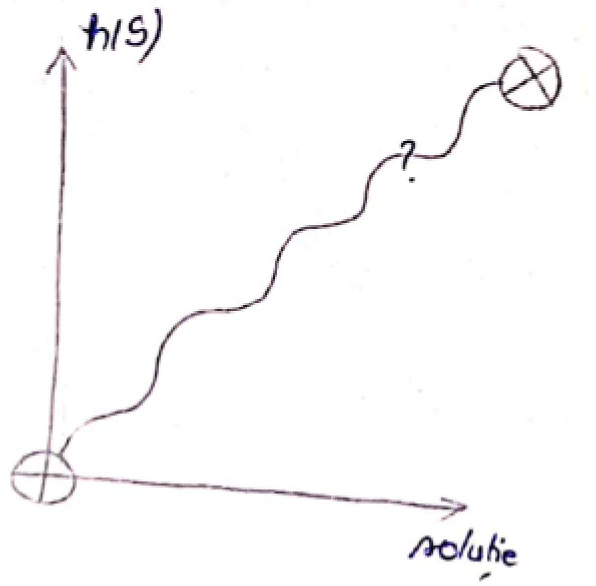
```
while isFinal(CS){
    for i = 1 to n {
        for j = 1 to m {
            if valid(CS,(i,j))
                state.add(transition(CS,i,j)
        }
    }
}
```

Best case: nu garanteaza solutia optima, insa este mult mai rapid decat DFS.
Worst case: aceeasi complexitate cu DFS si face alegeri gresite

3) Hillclimbing -> din starile accesibile din starea curenta vom alege o stare cel putin la fel de apropiata de starea finala :
   h(new_state) >= h(curr_state)

   Hillclimbing nu spune ce decizie luam ci cum sa fie filtrate.

   De multe ori, hillclimbing merge mai repede decat Greedy.
   Reprezentarea unei solutii in hillclimbing arata ca un grafic in scari



Avantajul consta in faptul ca este cea mai rapida strategie rezonabila. Nici Greedy, nici Hillclimbing nu ofera garantia ca produc solutia cea mai optima (poate fi optima doar din perspectiva acelei euristici)

Hillclimbing este cea mai rapida strategie neinformata cu mai multe variante, in functie de cum selectam starile dupa filtrare.
Dezavantaje:
euristica poate duce in stari de optim local (toate starile din cea curenta sunt mai rele, caz in care alg se blocheaza)

Aceste stari nu sunt usor de prevazut

3) A*

Ne dorim toate solutiile posibile, deci trebuie sa aratam ca am parcurs exhaustiv spatiul problemei (strategii informate; BFS este singurul care returneaza sol. corecta)

IDDFS -> exploram prin DFS pana la adancimea maxima 1

A* exploreaza d(S) + h(S), unde h(S) este distanta estimata pana la starea finala.
La IDDFS era dat numarul de pasi de la starea initiala pana la cea curenta.

La A* nu trebuie sa ne oprim la starea finala.

A* Algorithm

```
Current_state = Initial_state;
Best_score = maximum of h;
Sorted_queue = {All neighbours of Current_state, with computed h+1 scores, marked as
unexplored};

While the score of the first unexplored state S in Sorted_queue is lower than Best_Score
     If S is a final state, Best_score = score of S;
     Mark S as explored;
     Add to Sorted_queue all its neighbours with computed h+d score and mark them
     unexplored;
     If a duplicate state appears in Sorted_queue, keep only the occurrence with lowest
     score;
     Sort Sorted_queue;

The optimal path is recovered from the last state updating Best_score and looking for
explored neighbours with lowest score up to the initial state.
```

Diferenta intre A* si Dijkstra

Dijkstra exploreaza cat mai aproape de starea initiala(concentric)

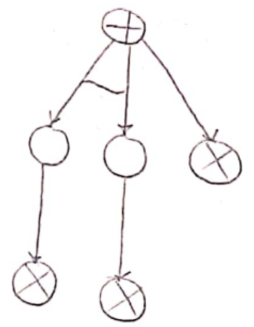
BFS/ IDDFS exploreaza la distanta egala fata de starea initiala in timp ce la A* impinge solutiile explorate catre starea finala

Optimizarea lui A* -> euristica sa nu fie doar admisibila ci si persistenta.

Simplified Memory Bounded A* -> se bazeaza pe prunning pentru elimanarea starilor care costul prea mare decat a fost estimat

Spațiile de probleme pot fi:
- nedeterministe → nu știm în ce stare vom ajunge și cu ce probabilitate
- deterministe → știm stările posibile și probabilitățile.

Arbori AND-OR



Spații parțial observabile: solitaire: știm ce carte urmează dar nu și ce urmează după ea.

Spații care mai pot fi cunoscute: avem foarte multe butoane și nu știm ce face niciunul.
Sau ce opțiuni vom avea disponibile după apăsarea unuia / după o tranziție.

COURSE 5:

Game initialisation pt X si O(cred)

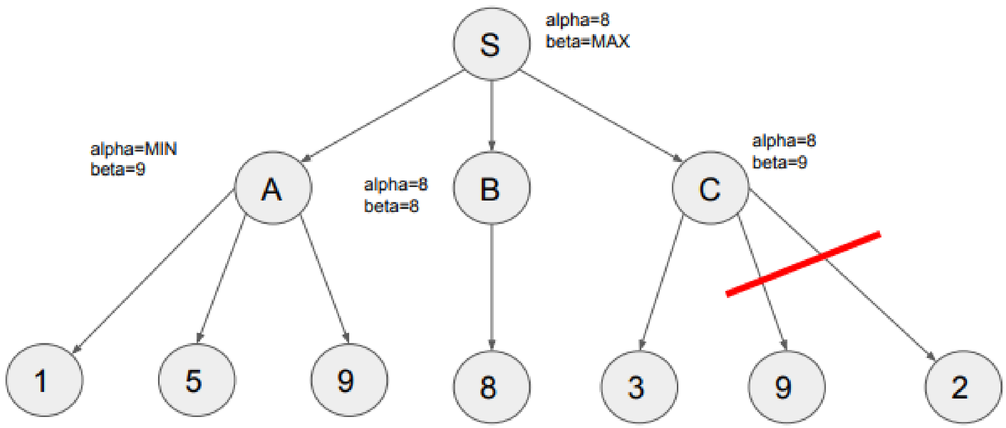
```
State initialisation (int first_player) {
        return (first_player, 0, 0, 0, 0, 0, 0, 0, 0, 0) ;
}
```

Select which player is moving first and send selection as parameter for initialisation.

Game ending

```
int IsFinal (State S) {
if((S[1]==S[2]==S[3])||(S[4]==S[5]==S[6])||(S[7]==S[8]==S[9])
||((S[1]==S[4]==S[7])||(S[2]==S[5]==S[8])||(S[3]==S[6]==S[9])
||(S[1]==S[5]==S[9])||(S[3]==S[5]==S[7])) return 3-S[0];
        else return -1;
        }
```

MINI-MAX cu Alpha-Beta pruning:



Minimul in A este 1-> S va avea maximul 1

Minimul in B este 8-> S va avea maximul 8(8>1)

Prima valoare in C este 3 si cum C alege un minim atunci valoarea in C va fi cel mult 3 care este mai mic decat 8 deci S nu va schimba maximul orice element ar fi in C deci ultimele 2 ramuri ale lui C se pot taia.

Dominant Strategy:

Player B / Player A	Silent	Confess
Silent	-2 / -2	0 / -8
Confess	-8 / 0	-5 / -5

We ignore the payoffs for Player B, we try to see this from Player A's perspective

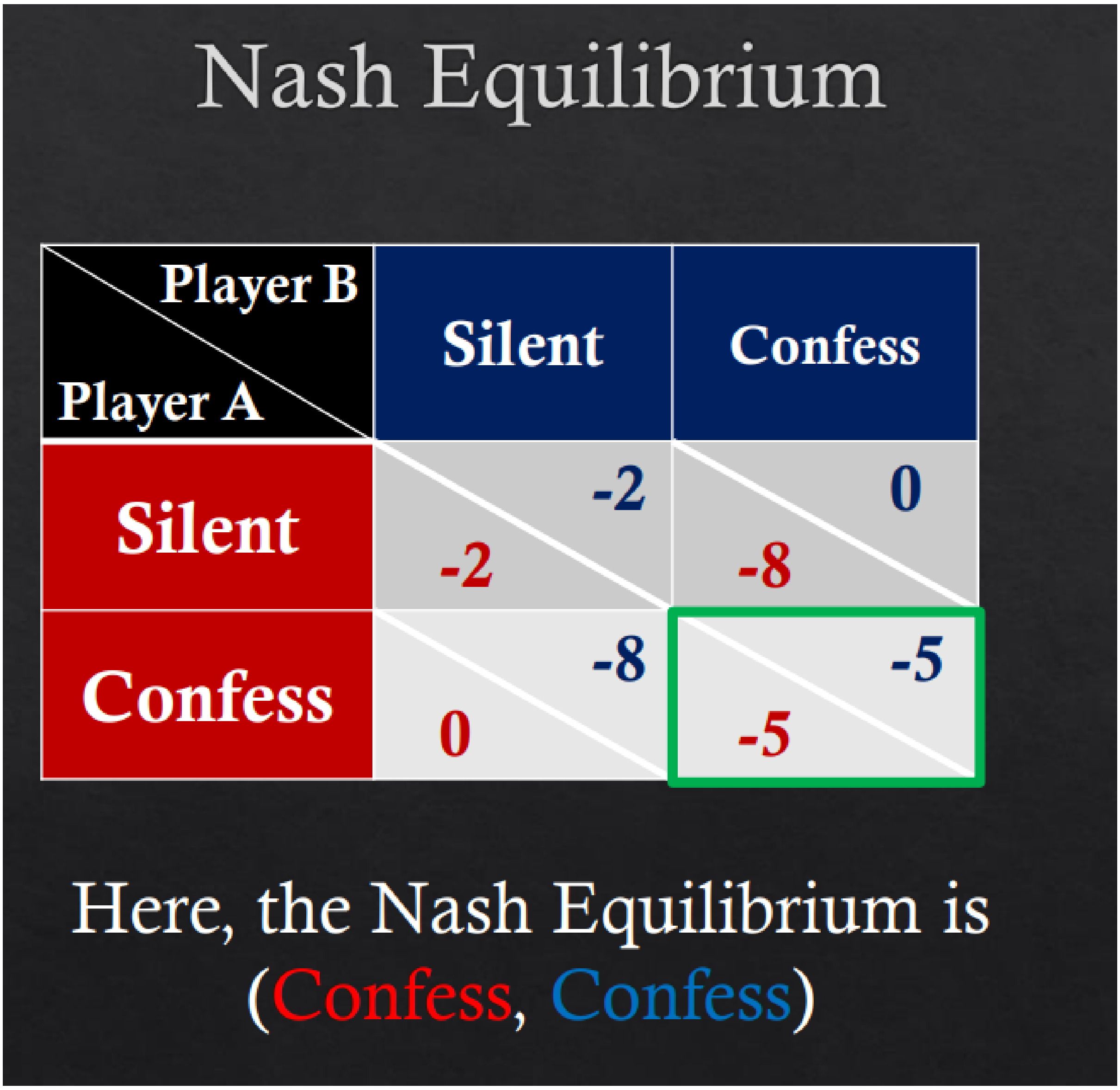
Player B / Player A	Silent	Confess
Silent	-2	-8
Confess	0	-5

Player B / Player A	Silent	Confess
Silent	-2	-8
Confess	0	-5

Confess it's always at least as good as the Accept option

Confess este clar superior lui Silent pentru jucatorul A deci linia Silent poate fi ignorata/stearsa.

Echilibru Nash:



Dominant Strategy:

Player B / Player A	Left	Center	Right
Up	3	4 >	3
Middle	1	3 >	2
Down	9	8 >	-1

Player B / Player A	Left	Center
Up	3	4
Middle	1	3
Down	9	8

Echilibru Nash pentru tabel mult mai mare

	V	W	X	Y	Z
A	9, 9	7, 1	5, 6	3, 4	1, 1
B	7, 8	5, 2	3, 6	1, 4	3, 3
C	5, 6	3, 3	1, 8	9, 7	1, 5
D	3, 9	1, 9	9, 4	7, 9	5, 9
E	1, 2	9, 8	7, 7	5, 6	3, 7

Se verifica pe rand, pentru fiecare jucator, care sunt valorile maxime in functie de ce ar alege celalalt jucator. In zonele in care ambele valori sunt maxime acelea vor fi echilibre NASH

Markov Decision Process:
La Markov se stie recompensa.

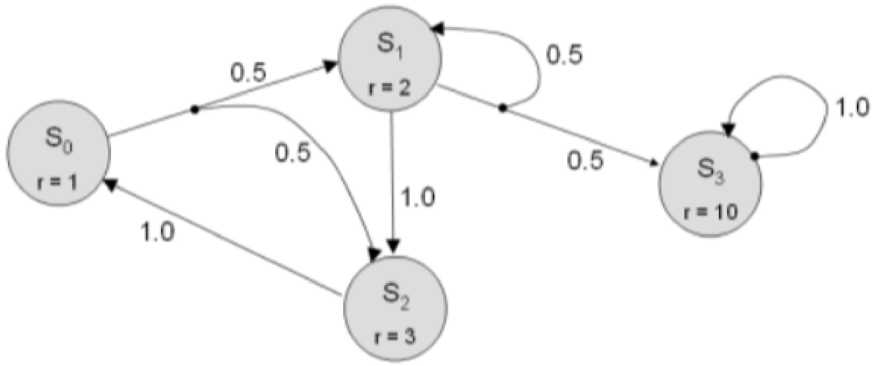
$$U_h([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

U->utilitatea(la inceput este 0 sau va fi specificata de problema)
R(s)->recompensa(se va specifica in problema)
Gamma->factorul de discount

ECUATIA BELLMAN:

$$U(s) = R(s) + \gamma max_{a \in A(s)} \sum_{s'} P(s'|s, a) U(s')$$

✕ Considerăm următorul proces de decizie Markov. Aplicați algoritmul Value Iteration. Factorul de discount este 1. Există o singură acțiune disponibilă pentru fiecare stare, cu excepția stării S1 care are două acțiuni. Valorile inițiale ale utilităților sunt 0. Care din afirmațiile de mai jos sunt adevărate? Calculul (C2) îl atașați formularului. 0/5



Iteratia 1:

$U(s_1)$:

inspre-S2=r1+gamma(P(s1->s2)*$U(s_2)$=2+1(1*0)=2

inspre-S3/S1=r1+gamma(P(s1->s3)*$U(s_3)$+P(s1->s1)*$U(s_1)$)

=2+1(0.5*0+0.5*0)=2

$U(s_1)$=max dintre(2 si 2)=2

$U(s_2)$:

e doar inspre-S0=>$U(s_2)$=r2+gamma(P(s2->s0)*$U(s_0)$)=3+1*0=3

$U(s_0)$:

e doar inspre-S1/S2

=>$U(s_0)$= r0 + gamma(P(s0->s1) * Us1 + P(s0->s2)*Us2)

=1+1(0.5*0+0.5*0)=1

$U(s_3)$:

e doar inspre S3 =>$U(s_3)$=r3+gamma(P(s3->s3)*Us3)=10+1(1*0)=0

Iteratia 2

$U(s_1)$:

inspre-S2=r1+gamma(P(s1->s2)*Us2=2+1(1*3)=2+3=5

inspre-S3/S1=r1+gamma(P(s1->s3)*Us3+P(s1->s1)*Us1)=2+1(0.5*10+0.5*2)=2+6=8

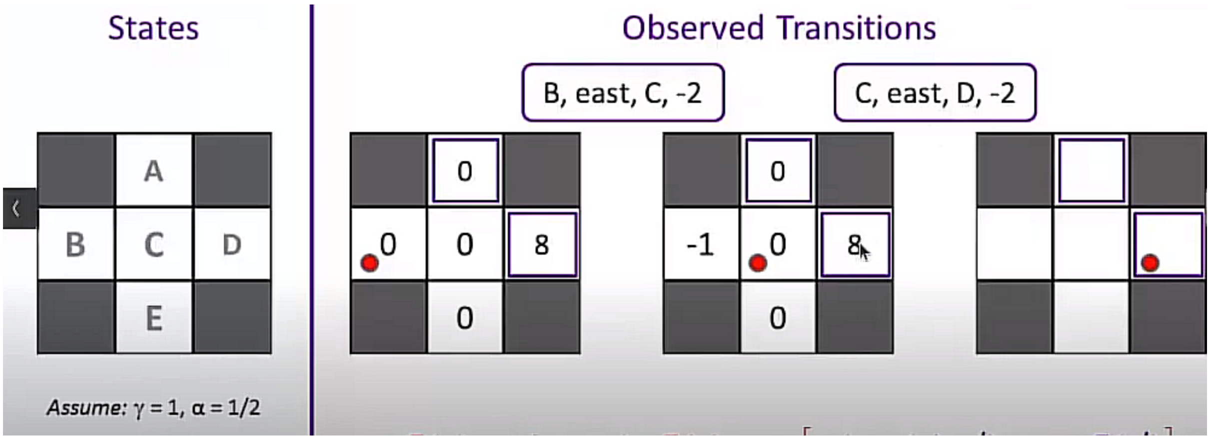
$U(s_1)$=max(5,8)=8

Temporal Difference Learning

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma U^{\pi}(s') - U^{\pi}(s))$$

alfa->rata de invatare

gama->factorul de discount



U(B)=U(B)+alfa*(R(b)+gamma*U(C)-U(B))=0+0.5(-2+1*0-0)=-1

U(C)=U(C)+alfa*(R(c)+gamma*U(D)-U(C))=0+0.5(-2+1*8-0)=3

Q-learning

$$Q(s,a) = Q(s,a) + \alpha(R(s) + \gamma max_{a'} Q(s',a') - Q(s,a))$$

Pacman is in an unknown MDP where there are three states [A, B, C] and two actions [Stop, Go]. We are given the following samples generated from taking actions in the unknown MDP. For the following problems, assume $\gamma = 1$ and $\alpha = 0.5$.

(a) We run Q-learning on the following samples:

s	a	s'	r
A	Go	B	2
C	Stop	A	0
B	Stop	A	-2
B	Go	C	-6
C	Go	A	2
A	Go	A	-2

What are the estimates for the following Q-values as obtained by Q-learning? All Q-values are initialized to 0.

(i) $Q(C, Stop) =$ ___0.5___

(ii) $Q(C, Go) =$ ___1.5___

For this, we only need to consider the following three samples.

$$Q(A, Go) \leftarrow (1-\alpha)Q(A, Go) + \alpha(r + \gamma \max_a Q(B, a)) = 0.5(0) + 0.5(2) = 1$$

$$Q(C, Stop) \leftarrow (1-\alpha)Q(C, Stop) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(1) = 0.5$$

$$Q(C, Go) \leftarrow (1-\alpha)Q(C, Go) + \alpha(r + \gamma \max_a Q(A, a)) = 0.5(0) + 0.5(3) = 1.5$$

Inferenta probabilista:

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

$$P(cavity \vee toothache) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064 = 0.28$$

Marginalizare:
Se face suma la probabilitatile pentru fiecare variabila posibila

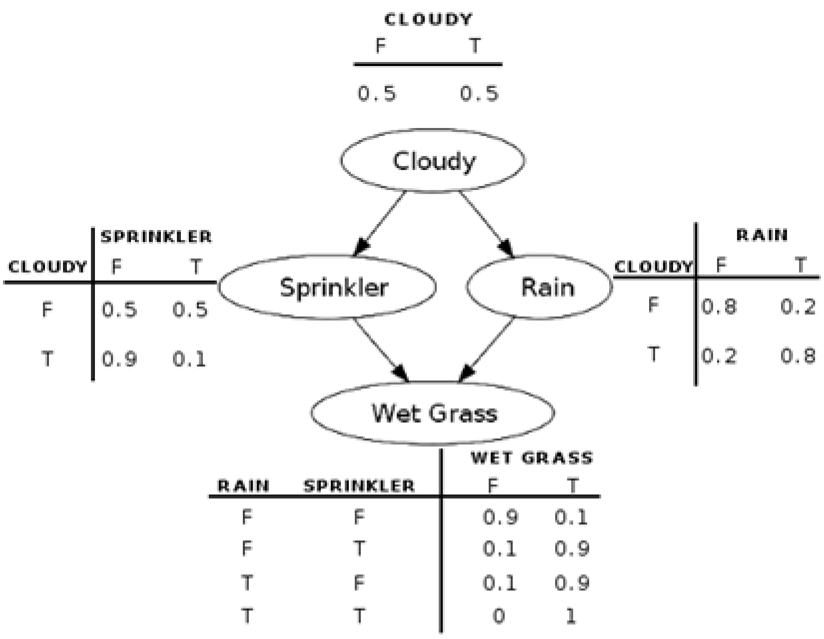
$$\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z} \in \mathbf{Z}} \mathbf{P}(\mathbf{Y}, \mathbf{z})$$

Exemplu: $\mathbf{P}(Cavity) = \sum_{z \in \{Catch, Toothache\}} \mathbf{P}(Cavity, \mathbf{z})$

$P(cavity) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$

	toothache		¬toothache	
	catch	¬catch	catch	¬catch
cavity	0.108	0.012	0.072	0.008
¬cavity	0.016	0.064	0.144	0.576

✓ Fie rețeaua bayesiană din figură. Calculați probabilitățile marginale ale nodului "Wet Grass".



Wet Grass(T)=0.1+0.9+0.9+1+0.5=3.4(0.5 de la cloudy deoarece trebuie folosite toate elementele si nu avem cloudy cand calculam probabilitatile)

Wet Grass(F)=0.9+0.1+0.1+0+0.5=1.6

alfa(3.4;1.6)=(0.68,0.32)

3.4+1.6=5=>Pt true:3.4/5=0.68, Pt fals 1-0.68=0.32

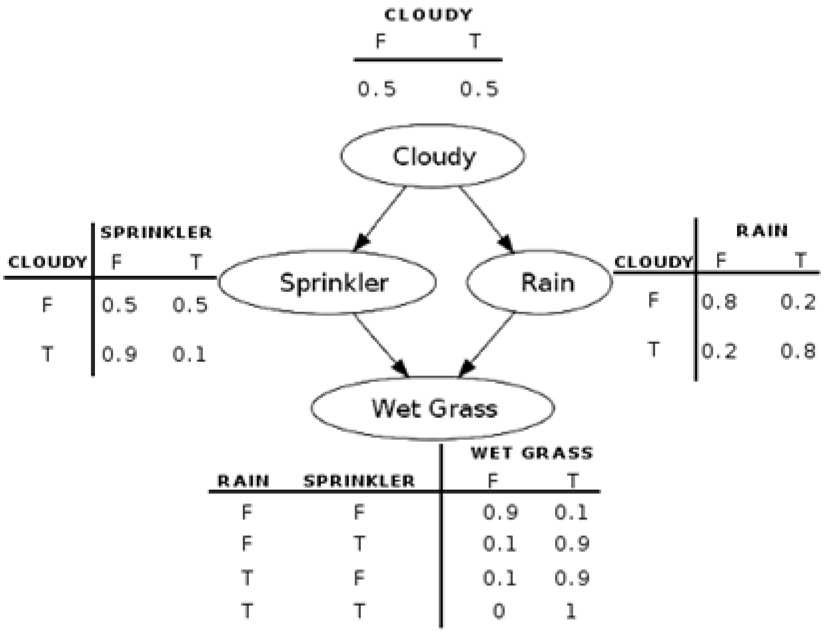
Condiționare $\mathbf{P}(\mathbf{Y}) = \sum_{\mathbf{z}} \mathbf{P}(\mathbf{Y}|\mathbf{z})P(\mathbf{z})$

$$P(cavity|toothache) = \frac{P(cavity \wedge toothache)}{P(toothache)} = \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064} = 0.6$$

$$P(\neg cavity|toothache) = \frac{P(\neg cavity \wedge toothache)}{P(toothache)} = \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064} = 0.4$$

Inferenta prin enumerare:

✓ Fie rețeaua bayesiană din figură. Folosind metoda inferenței prin enumerare, calculați probabilitatea de a ploua (rain = T) dacă cerul este înnorat (cloudy = T) și iarba nu este udă (wet grass = F).



CLOUDY F	CLOUDY T
0.5	0.5

CLOUDY	SPRINKLER F	SPRINKLER T
F	0.5	0.5
T	0.9	0.1

CLOUDY	RAIN F	RAIN T
F	0.8	0.2
T	0.2	0.8

RAIN	SPRINKLER	WET GRASS F	WET GRASS T
F	F	0.9	0.1
F	T	0.1	0.9
T	F	0.1	0.9
T	T	0	1

P(R|C,W) trebuie calculat pentru R TRUE și FALSE

$$P(R_T|C_T,W_F) = \alpha \cdot \sum_{S\in(T,F)} P(C_T,R_T,S,W_F)$$

Deoarece S este o variabilă anonimă (adică nu apare în cerință, dar există în rețea) trebuie făcută sumă de probabilități pentru fiecare stare a lui S.

$$P(R_T|C_T,W_F) = \alpha \cdot \sum_{S\in(T,F)} P(C_T)\cdot P(R_T|C_T)\cdot P(S|C_T)\cdot P(W_F|S,R_T)$$

$$= \alpha\, P(C_T)\cdot P(R_T|C_T)\cdot \sum_{S\in(T,F)} P(S|C_T)\cdot P(W_F|S,R_T)$$

Deoarece P(C) și P(R|C) vor avea valori constante, adică nu depind de variabila anonimă S, ele pot fi scoase înaintea sumei

$$=\alpha\, P(C_T)\cdot P(R_T|C_T)\cdot[P(S_T|C_T)\cdot P(W_F|S_T,R_T)+P(S_F|C_T)\cdot P(W_F|S_F,R_T)]$$

$$=\alpha\, 0{,}5\cdot 0{,}8\cdot[0{,}1\cdot 0+0{,}9\cdot 0{,}1] = 0{,}036$$

Acum calculăm pentru R FALSE

$$P(R_F|C_T,W_F) = \alpha\cdot\sum_{S\in(T,F)} P(C_T,R_F,S,W)$$

$$=\alpha\cdot P(C_T)\cdot P(R_F|C_T)\cdot[P(S_T|C_T)\cdot P(W_F|S_T,R_F)+P(S_F|C_T)\cdot P(W_F|S_F,R_F)]$$

$$=\alpha\, 0{,}5\cdot 0{,}2\cdot[0{,}1\cdot 0{,}1+0{,}9\cdot 0{,}9]=0{,}082$$

$$\alpha(0{,}036;0{,}082)\approx\left(\frac{0{,}036}{0{,}036+0{,}082};\frac{0{,}082}{0{,}036+0{,}082}\right)\approx\left(\frac{0{,}036}{0{,}118};\frac{0{,}082}{0{,}118}\right)\approx$$

$$\approx(0{,}31;0{,}69)$$

Sau poți să faci doar pt una și pt cealaltă scazi din 1

$$P(R_F|C_T,W_F)=\frac{0{,}082}{0{,}118}=0{,}69 \Rightarrow P(R_T|C_T,W_T)=1-0{,}69=0{,}31$$