ÚVOD DO AI

Agent = něco co reaguje na prostředí

→ racionální agent volí nejlepší akci aby max. utility

$$\text{akce} = \operatorname{argmax}_a \mathbb{E}[\text{utility}(o, a)]$$

observation ↲ ↳ akce

• <u>Druhy prostředí</u>:

- fully / partially observable
- deterministické / stochastické – další stav závisí jen na aktuálním + akci
- episodic / sequential – episody jsou nezávislé
  ↳ akce z aktuální ep. neovlivní tu další
- static / dynamic
  ↳ nemění se zatímco agent rozmýšlí akci
- diskrétní / spojité
- single agent / multi agent
  ↳ co-operative × competetive

• <u>Druhy agentů</u>

• <u>Reflex agent</u>: chování se neliší podle cíle

simple: $o \rightarrow a$ ... observation → akce

→ má <u>transition model</u> → pamatuje si nějaký vnitřní <u>stav</u>

↳ $(s_t, a_t, o_{t+1}) \rightarrow s_{t+1}$, minulý stav + akce + percept → nový stav

$s_{t+1} \rightarrow a_{t+1}$

• <u>Goal based agent</u> – flexibilnější, má cíl

$(s_t, a_t, o_{t+1}) \rightarrow s_{t+1}$

$(s_{t+1}, \text{Goal}) \rightarrow a_{t+1}$

• <u>Reprezentace stavu</u>

• atomické – nedělitelný, nemá vnitřní strukturu → prohledávání stav. prost.

• factored – stav = reálný vektor → CSP, plánování

• structured – stav = množina objektů a relacemi mezi nimi ≈ graf → first-order logic

• Problem solving agent

- atomická reprezentace stavů
  cíl = množina stavů
  akce = transitions mezi stavy ~ hrany

→ máme transition model
(state, akce) → state

→ chceme najít posloupnost akcí, co mě dostane do cílového stavu

⇒ SEARCH

- předpoklady: prostředí = fully observable, diskrétní, statické, deterministické

- abstrakce prostředí - real world prostředí je moc komplexní
  • validní ... řešení abstrakce dokážeme převést na řešení původního prob.
  • useful ... vyřešit abstrakci má být jednodušší

Strategie

- vždy máme nějakou frontu
- vybíráme z ní uzel podle nějaké strategie
  ↳ expandujeme ho = sousedy (ne nutně všechny) dáme do fronty

repeat

• Tree search
  - nepamatuju si navštívené, do fronty dávám vše
  - z grafu generuje nějaký až exponenciálně velký strom

• Graph search
  - pamatuje si navštívené vrcholy

Implementace

- DFS, BFS, Dijkstra (fancy BFS)
- iterative deepening - pamatuju si jen cestu k aktuálnímu vrcholu
  ↳ postupně zvětšuju search radius

• Dijkstra - expanduje uzel co je nejblíž ke startu - $g(n)$
  ↳ aktualizuje $g(n)$ sousedů tohoto vrcholu

• Best first search - mám heuristiku h, co udává odhadovanou vzdálenost do cíle
  ↳ vybírá vrchol s co nejmenší h

• A^* ⇒ vybírá min $f(n) := g(n) + h(n)$

• Heuristiky

- aby heuristiky fungovaly a dalo se o A* dokázat, že funguje, něco chceme

↗ je optimistická

Def: Heuristika $h: V \to \mathbb{R}$ je

1) přípustná $\equiv \forall n: 0 \leq h(n) \leq$ délka nejkratší cesty z n do cíle

2) monotónní $\equiv$ pro $\forall$ souseda n' uzlu n platí

$$h(n) \leq c(n, n') + h(n') \quad \ldots \quad \Delta\text{-nerovnost}$$

Monotónní heuristika je přípustná

Dk: nechť $n_1, n_2, \ldots, n_k$ je optimální cesta z n_1 do cíle n_k

$$\forall i: \quad h(n_i) - h(n_{i+1}) \leq c(n_i, n_{i+1})$$

$$\Rightarrow h(n_1) = h(n_1) - h(n_2) + h(n_2) - h(n_3) + \ldots + h(n_{k-1}) - h(n_k) + \overbrace{h(n_k)}^{0}$$

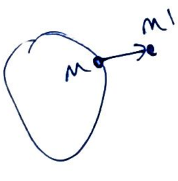
$$\leq \underbrace{c(n_1, n_2) + c(n_2, n_3) + \ldots + c(n_{k-1}, n_k)}_{\text{délka nejkratší cesty}}$$

Pro monotónní heuristiku jsou hodnoty $f(n)$ neklesající na každé cestě

↳ $f(n) = h(n) + g(n)$ $\quad\quad g(n') = g(n) + c(n, n')$

→ expandoval jsem n, následník n', ukážu $f(n') \geq f(n)$

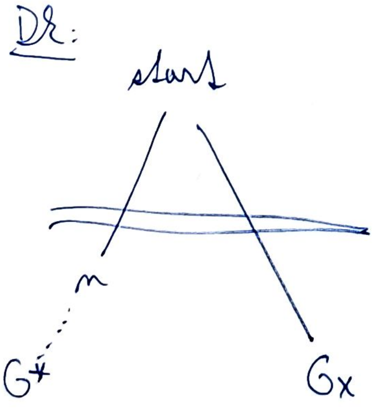
$$f(n') = g(n') + h(n') = g(n) + c(n, n') + h(n') \geq g(n) + h(n) = f(n)$$



Věta: Pokud je $h(n)$ přípustná, pak je tree search verze A* optimální.

↳ neboli: první expandovaný cíl je optimální

Dk:



Pro spor nechť algoritmus expanduje jako první cíl G_x, co není optimální

→ nechť je optimální G^* s cenou C^*

$$\Rightarrow f(G_x) = g(G_x) + h(G_x) = g(G_x) > C^*$$

→ nechť n je vrchol z fronty na optimální cestě

↳ pak $f(n) = g(n) + h(n) \overset{*}{\leq} C^*$

* h je přípustná

⇒ dohromady $f(n) < f(G_x)$ ⇒ n se expanduje před G_x

Věta: Pokud je $h(n)$ přípustná, tak je Graph search verze A^* optimální.

př:

– možný problém G-S: najdu nějakou cestu do vrcholu n, a už nenajdu lepší, protože jsem n už navštívil

→ ale pro monotónní jsou hodnoty $f(n)$ neklesající a A^* expanduje uzel s nejmenším $f(n)$

⇒ mezi všemi otevřenými cestami do n vždy vybereme tu nejkratší

Kombinování heuristik

Tvrzení: Pokud jsou h_1, h_2 přípustné / monotónní, tak

i) $\alpha h_1 + (1-\alpha) h_2$, $\alpha \in [0,1]$

ii) $\max\{h_1, h_2\}$

jsou taky přípustné

afinní kombinace $\leq$ max ⇒ max je pro A^* lepší

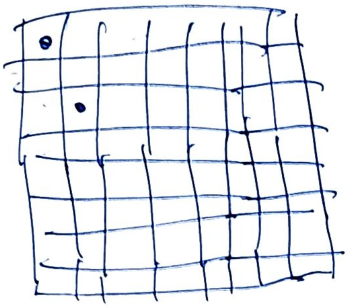
↳ říkáme, že max dominuje afin.

A^* prozkoumá všechny vrcholy t.ž. $f(n) < C^*$, tedy $h(n) < C^* - g(n)$

⇒ pokud A^* expanduje vrchol s max. tak ho musí exp. i s afin.

• <u>Constraint Satisfaction Problems</u> – CSP

– 8-Queen problem



proměnné: $r_1, \dots r_8 \in [8]$

r_i = pozice královny v i-tém sloupci

⇒ domény: $\{1, 2, \dots 8\}$

⇒ podmínky: královny se nesmějí ohrožovat

↳ $\forall i \neq j: r(i) \neq r(j)$ & $|i-j| \neq |r(i) - r(j)|$

<u>Obecně</u>:

- konečná množina proměnných
- domény = konečné množiny hodnot pro ∀ proměnnou
- konečná množina podmínek

  ↳ podmínka ∀ relace na podmnožině proměnných

  ↳ arita podmínky = počet ovlivněných proměnných

<u>Příklad</u>:

• Sudoku: $\forall\, i,j \in [9]: x_{ij} \in \{1, 2, \dots 9\}$

podmínky: ∀ řádek, ∀ sloupec, ∀ ▦ : all-different

• Barvení vrcholů grafu k barvami

proměnné: vrcholy G

domény: $[k]$

podmínky: $\forall\, uv \in E: C(u) \neq C(v)$

<u>Řešení</u>

• <u>Backtracking</u> = tree/graph search s DFS strategií

---

1. přiřaď hodnotu zvolené (dosud neohodnocené) proměnné
2. zkontroluj podmínky na již ohodnocených proměnných

→ pokud jsou splněné → další proměnná

→ jinak zkus jinou hodnotu

→ pokud žádná hodnota nefunguje → vrať se k minulé proměnné

• **Forward checking**

→ navíc si „proškrtávám" hodnoty, které jsou zakázané

→ když dám někam královnu, tak si proškrtám ohrožená políčka

⇒ proškrtané hodnoty vůbec nezkouším

• **Arc-Consistency**

→ povolíme pouze binární podmínky

⇒ $\forall$ podmínka ~ arc v grafu podmínek

Def: Arc (V_i, V_j) je arc-consistent $\equiv \forall x \in D_i \ \exists y \in D_j$ t.ž. (x,y) splní všechny podmínky

to že (V_i, V_j) je konz. neznamená, že (V_j, V_i) je

Def: CSP je arc-consistent $\equiv \forall$ arc je consistent v obou směrech

Příklad:

$A = \{1, 2, 3\}$

$B = \{1, 2, 3\}$

$C = \{1, 2, 3\}$

$A > B$

$B = C$

	Agenda	Arcs
1	$A > B$	$A > B$
2	$B < A$	$B < A$
3	$B = C$ ✓	$B = C$
4	$C = B$	$C = B$
5	$A > B$	
6	$B = C$	

Algoritmus AC-3

1. udělej z $\forall$ bin. podmínky dva Arcs: $A = B \Rightarrow A = B$ & $B = A$
2. přidej všechny Arcs do Agendy
3. Opakuj, dokud nebude Agenda prázdná
   - vem Arc (V_i, V_j) z agendy
   - pro $\forall$ hodnotu V_i musí být nějaká hodnota V_j
   - odstraň nekonzistentní hodnoty z V_i
   - pokud se doména V_i změnila, přidej do agendy všechny (V_k, V_i)

↳ protože rychle zúžíme prostor toho ještě víc

- <u>Maintaining arc consistency</u> (look ahead)
  1. udělej problém Arc-consistent
  2. backtracking
     – vždy, když proměnné přiřadíme hodnotu ⇒ obnov konzistenci

- <u>Silnější konzistence</u>
  – arc-consistency je lokální

sudoku:

		6
3	9	
2	1	8

$X_{1,1} = \{4,7\}$ $\quad X_{1,1} \neq X_{1,2}$

$X_{1,2} = \{4,7\}$ $\quad \neq \; \neq$ $\quad$ → je to arc-consistent

$X_{2,3} = \{4,7,5\}$ $\quad X_{2,3}$

↳ 5 můžeme odstranit

⇒ lze rozšířit k-consistency

↳ pro konzistentní přiřazení každých $k-1$ proměnných vyhodnotíme konzistentní hodnotu v každé další (k-té) proměnné

<u>Věta</u>: Pokud je CSP i-consistent pro $\forall i = 1, \dots n$ a máme právě n proměnných, tak ho lze vyřešit bez backtrackingu.

<u>Dk</u>: DFS vždy najde hodnotu konzistentní s ohodnocením předchozích proměnných

↳ bohužel, časová komplexita k-consistence je exponenciální v k

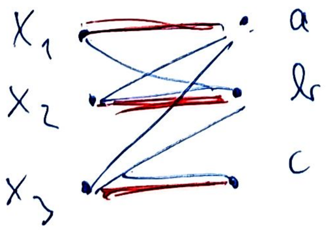
- <u>Global constraints</u>
  – lepší je používat global constraints na podproblémy s konkrétní strukturou

⇒ all-different$(X_1, X_2, \dots X_k)$ $\quad$ ↓ dají se vyřešit efektivně

$X_1 = \{a, b\}$

$X_2 = \{a, b\}$

$X_3 = \{c, d\}$



⇒ maximální párování

→ pokud hrana není v žádném max. párování

⇒ odstraň hodnotu

- Variable ordering
  - jak vybrat pořadí vybíraných proměnných?
  - ⇒ fail-first principle ... vezmu tu, co nejspíš povede k fail
    - dom heuristic : nejmenší doména
    - deg heuristic : proměnná zúčastněná v nejvíce podmínkách
- Value ordering
  - ⇒ succeed-first principle → nějaké heuristiky

SAT - solvery

→ 8-Queens šlo by bez namodelovat pomocí CNF – booleovské proměnné, logické podmínky v CNF

Algoritmus DPLL – splnitelnost CNF formulí

1, jednotková propagace X_1: $X_1 \wedge (X_1 \vee X_2) \wedge (X_2 \vee \neg X_3) \wedge (X_3 \vee \neg X_1 \vee X_2)$
$\rightsquigarrow (X_2 \vee \neg X_3) \wedge (X_3 \vee X_2)$

2, čistý výskyt X_2: $\rightsquigarrow$ konec ✓

→ pokud není čistý výskyt ⇒ branching

Vylepšení
- analýza konfliktů - klauzule je nutné rozdělit do nezávislých podúloh
- variable (value) ordering
- random restarts
- clever indexing – efektivně identifikujeme jednotkové klauzule
  - ↳ watched literals
    - → v ∀ klauzule sledujeme 2 literály
    - → dokud oba neohodnotím, tak určitě není jednotková
      - ↳ pokud nějaký ohodnotím ⇒ vyberu další
- clause learning – ve větvi zjistím, že nějaká kombinace hodnot dohromady nefunguje ⇒ udělám novou klauzuli, co ji zakáže

• Knowledge-based agents – Wampus

– nějaký formální jazyk

TELL ... přidání do knowledge-base

ASK ... dotaz = inference

– je dané políčko bezpečné?

↳ pokud je bezpečné v každém modelu KB ⇒ ANO

↳ jinak: IDK

Wampus svět: breeze okolo děr, stench okolo wumpuse

$P_{i,j}$ = pit

$W_{i,j}$ = Wamp

$B_{i,j}$ = breeze

$S_{i,j}$ = stench

$\neg P_{1,1}$ → začínám tam

$\neg W_{1,1}$

→ pozorování

$\neg B_{1,1}$

$B_{2,1}$

→ model světa

$B_{x,y} \Leftrightarrow (P_{x,y+1} \vee P_{x,y-1} \vee P_{x+1,y} \vee P_{x-1,y})$

$S_{x,y} \Leftrightarrow (W_{x,y+1} \vee W_{x,y-1} \vee W_{x+1,y} \vee W_{x-1,y})$

KB = knowledge base

→ je jenom 1 wamp

→ alespoň 1: $W_{1,1} \vee W_{1,2} \vee \dots \vee W_{n,n}$

– max. 1: pro $\forall x_1, x_2, y_1, y_2$: $\neg W_{x_1 y_1} \vee \neg W_{x_2, y_2}$

→ množina modelů sentence α je $M(\alpha)$

→ $KB \models \alpha$ ... $M(KB) \subseteq M(\alpha)$ → α je důsledek KB

👁 $KB \models \alpha \Leftrightarrow KB \wedge \neg \alpha$ je nesplnitelná

→ rezoluce: rezoluční pravidlo: $\dfrac{X \vee Y \qquad Z \vee \neg Y}{X \vee Z}$

⇒ pokud jsem odvodil □, pak $KB \models \alpha$

→ pokud nelze vygenerovat další klauzule, pak $KB \not\models \alpha$

- Hornovské klauzule = nejvýše 1 pozitivní literál

↳ vlastně implikace : $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$ → prolog

→ existuje lineární algoritmus ⇒ LI-rezoluce

$$\frac{A \;\wedge\; A \Rightarrow B}{B}$$

1) Backward chaining - rozdělí query na sub-queries

- goal-driven metoda

$P \Rightarrow Q$ ASK: platí Q ?

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$ → musím splnit P

$A \wedge P \Rightarrow L$ ⇒ musím splnit $L \wedge M$

$A \wedge B \Rightarrow L$ ⇒ musím splnit L

A → $A \wedge B \Rightarrow L$ ✓   platí

B ⇒ musím splnit M

→ $B \wedge L \Rightarrow M$ ✓

2) Forward chaining

→ pro ∀ klauzuli si pamatuju # nesplněných předpokladů

↳ snižuju, když odvodím nový fakt (všechny před. splněné)

Fakta: A, B, L, M, P Q

$A \wedge P \Rightarrow L$ data-driven

$A \wedge B \Rightarrow L$

$B \wedge L \Rightarrow M$

$L \wedge M \Rightarrow P$

$P \Rightarrow Q$

Reasoning děláme pomocí logické inference

! Tento přístup lze uplatnit jen na statické prostředí

"Automatické" plánování

→ co když se prostředí mění v čase?

Fluent = proměnná anotovaná časem

↳ $L^t_{x,y}$ = agent je v čase t na pozici (x,y)

Observational model

→ spojuje observation s modelem světa

$$L^t_{x,y} \Rightarrow (Breeze^t \Leftrightarrow B_{x,y})$$

↳ říká, zda v čase t cítím breeze

$$Safe^t_{x,y} \Leftrightarrow (\neg Pit_{x,y} \wedge \neg(W_{x,y} \wedge WumpusAlive^t))$$

Transition model

→ popisuje jak akce ovlivňují svět

- effect axioms – říkají, co akce změní

$$L^t_{x,y} \wedge North^t \wedge Forward^t \Rightarrow L^{t+1}_{x,y+1} \wedge \neg L^t_{x,y}$$

- frame axioms – říkají, co se nemění

↳ axiomy rámce

$$Forward^t \Rightarrow (HaveArrow^{t+1} \Leftrightarrow HaveArrow^t)$$

→ je jich potřeba strašně moc ⇒ neefektivní

- successor-state axioms

→ pro každý fluent F definujeme pravdivost F^{t+1} pomocí fluentů a akcí v čase t

$$F^{t+1} \Leftrightarrow (AkceCoZpůsobíF)^t \vee (F^t \wedge \neg(AkceCoZpůsobíNotF)^t)$$

$$HaveArrow \Leftrightarrow PickUpArrow^t \vee (HaveArrow^t \wedge \neg Shoot^t)$$

- precondition axioms – kdy lze akci použít: $Shoot^t \Rightarrow HaveArrow^t$
- action-exclusion axioms: $\forall t, \forall i \neq j: \neg(A^t_i \wedge A^t_j)$

↳ nemůžeme povolit současně některé akce – neměly by rušit své efekty/podmínky

- Hybridní plánování
  - logikou odvodíme pravdy o světě
    ↳ na základě nich se rozhoduje, co bude dělat
  - pro plánování trasy použijeme A^*

- SAT Plan
  → problém zakódujeme do SATu
    - $Init^0$ ... počáteční stav světa
    - $Transition^1, \dots Transition^t$ → axiomy popisující akce – successor-state, precondition, exclusion
    - $Goal^t$ ... chceme, aby cíl platil v čase t.
      ↳ ($HaveGold^t \wedge ClimbedOut^t$)

  → předhodíme to SAT solveru
    - pokud najde model ⇒ existuje řešení, model ho kóduje
    - pokud model není, tak neexistuje plán délky t

  ⇒ SAT Plan tohle zkouší pro $t = 1, \dots T$

- Automatické plánování
  - plán lze najít prohledáváním (A^*), ale mám hodně stavů
    ↳ potřeboval bych hodně dobrou heuristiku – jaké akce jsou kdy dobré ...
  → ve výrokové logice potřebuju na každou blbost formuli
    ⇒ chtělo by to logiku 1. řádu ⇒ schémata axiomů

⇒ Situační kalkulus

stavy popisujeme pomocí predikátů: at(Robot, Location)
  - pokud se pravdivost té relace mění v různých situacích
    ⇒ (fluents): at(Robot, Location, s) ↳ konkrétní situace
  - pokud se nemění (rigidní predikáty): connected(loc1, loc2)

→ pro ∀ akci possibility axiom: $\varphi(s) \Rightarrow Possible(s, a)$, φ je formule
→ pro ∀ fluent successor-state axiom: říká, zda platí v dalším stavu

$Poss(a, s) \Rightarrow F(s') \iff$ (a způsobí F) $\vee$ ($F(s) \wedge$ F is not made False by a)

→ plánování v situačním kalkulu

↳ $(\exists s)$ Goal(s)

↳ $(\exists s)$: HaveGold(s) $\wedge$ ClimbedOut(a, s)

- <u>Klasické plánování</u>

stav = vektor proměnných ... factored representation

akční schémata popisují jak agent může měnit svět

⇒ problém: stavů je hodně i pro malé problémy

→ plánování ale lze snadno přes prohledávání

→ stav je vždy množina nějakých atomů

- fluents - mění hodnotu ve stavech
- rigid atoms - nemění se

<u>Konvence</u>: Closed world assumption = atom $a \notin s$ ⇒ a neplatí v s

<u>Def</u>: Stav s splňuje cíl $g \equiv g^+ \subseteq s \wedge g^- \cap s = \emptyset$

g^+: pozitivní atomy

g^-: atomy, co jsou negativní v g

<u>Akční schéma (operátory)</u>

load (car, box, location)

preconditions:

empty (car)
at (car, location)
at (box, location)

effects:

not empty (car)
not at (box, place)
at (box, car)

⇒ operátor má jméno, parametry, předpoklady a efekty

⇒ akce je instance operátoru - za proměnné dosadíme konstanty

Domain model = popis operátorů

Planning problem obsahuje

- Domain model
- počáteční stav, jaké objekty (konstanty) ve světě existují
- cíl

Plán je sekvence akcí

→ kdy lze jakou akci použít?

Def: Akce a je aplikovatelná na stav s

$$\equiv precond^+(a) \subseteq s \quad \wedge \quad precond^-(a) \cap s = \emptyset$$

Výsledek aplikování akce a na stav s je

$$\gamma(s, a) := (s \setminus effects^-(a)) \cup effects^+(a)$$

Akce a je relevantní pro cíl $g \equiv$

i) akce přispěje cíli: $g \cap effects(a) \neq \emptyset$

ii) efekty akce nejsou v konfliktu s cílem:

$$g^- \cap effects^+(a) = \emptyset$$
$$g^+ \cap effects^-(a) = \emptyset$$

Regresní množina pro cíl g a relevantní akci a je

$$\gamma^{-1}(g, a) := (g \setminus effects(a)) \cup preconditions(a)$$

Plán $\pi = \langle a_1, a_2, \dots a_k \rangle$ řeší problém P $\equiv \gamma^*(s_0, \pi)$ splní cíl

- Forward-search – upravuje stav
  - začnu se stavem $s = s_0$, zkouším aplikovat akce, jejichž předpoklady jsou splněny, vždy udělám $s \leftarrow \gamma(s, a)$
- Backward-search – upravuje cíl
  - začne se zadaným cílem a zkouší aplikovat akce, pro které je jasně definovaný $\gamma^{-1}(g, a)$ ⇒ pak udělá $g \leftarrow \gamma^{-1}(g, a)$

→ menší branching factor, ale je těžší vymyslet heuristiky

• Plánovací heuristiky

- chceme přípustnou heuristiku = dolní odhad na # akcí do cíle
⇒ vyřešíme relaxaci toho problému

1, ignoruju předpoklady akcí ⇒ pak to je problém pokrytí množiny
2, ignoruju negativní efekty akcí
  ↳ nejmenší množina akcí, co vyrobí podmínky cíle

• Hierarchické plánování - rozloží problém na pod-problémy

Příklad: Hanojské věže

objekty: tyče, disky, stůl
predikáty: menší(d1, d2)
  top(tyč, disk)
  bellow(d1, d2)
                  → statické
init: at(1, *), menší(...), top(1, nejmenší disk)
  bellow(...), bellow(největší disk, stůl)

goal: top(1, stůl), top(2, stůl)
akce: move(from, to, disk, cílový disk, pod)

předpoklady: top(from, disk)
  bellow(disk, pod)
  top(to, cílový disk)
  menší(disk, cílový disk)

efekty: top(from, pod), not top(from, disk)
  top(to, disk), not top(to, cíl)
  bellow(disk, cíl), not bellow(disk, pod)

• Plan-Space Planning

open-goal = atom, co ještě neumím „zavřít"
  ⇒ najdu akci, co ho splní - její předpoklady - nové open goals
kauzální vazba = jedna akce vyrábí předpoklady pro jinou akci
⇒ víme, že některé akce se musí udělat před jinou akcí
⇒ Start = akce bez předpokladů, nastaví počáteční stav | cílový stav = akce s předpoklady cíle, → nic nedělá

Probabilistic reasoning

- prostředí může být částečně pozorovatelné / nedeterm.
- logický agent
  - může pracovat s belief states = množina možných stavů světa
    - ⇒ contingency plans – handle every possible eventuality
      - ↳ velké, komplexní,
    - → vše je buď true / false
- pravděpodobnostní agent – stupeň důvěry $\in [0,1]$
  - → možné světy $\omega \in \Omega$, $0 \leq P(\omega) \leq 1$ → diskrétní pstní prostor

$$\sum_{\omega \in \Omega} P(\omega) = 1$$

  - → jevy (events)

$$P(A) = \sum_{\omega \in A} P(\omega) \longrightarrow \text{píšeme } P(A \cap B) =: P(A, B)$$

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

→ product rule: $P(A \cap B) = P(A \mid B) \cdot P(B)$

→ marginalizace:

$$\mathbb{P}(Y) = \sum_{z \in Z} \mathbb{P}(Y, z) = \sum_{z \in Z} \mathbb{P}(Y \mid z) \cdot P(z)$$

budu psát $\mathbb{P}$ když vektor

Y je proměnná → vlastně děláme vektor pro všechny možné hodnoty Y

↳ jako kdybych to počítal v numpy

→ normalizace

$$\mathbb{P}(Y \mid E = e) = \mathbb{P}(Y, E = e) / P(E = e) = \alpha \cdot \mathbb{P}(Y, E = e)$$

↳ náhodná proměnná pro evidence, e je co jsem viděl

→ vím, že na konci má vyjít suma toho vektoru 1

$$\sum_{y \in Im(Y)} P(Y = y \mid E = e) = 1$$

⇒ to $P(E = e)$ ignoruju a na konci provedu normalizaci

→ hidden random vars

$$\mathbb{P}(Y \mid E = e) = \alpha \mathbb{P}(Y, E = e) = \alpha \sum_h \mathbb{P}(Y, E = e, H = h)$$

→ často mi to usnadní výpočet

→ nezávislost

$X \perp Y \Rightarrow P(X|Y) = P(X)$, $P(X,Y) = P(X)P(Y)$

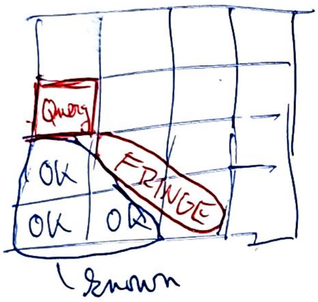
$X \perp_C Y \Rightarrow P(X|Y,C) = P(X|C)$

$\mathbb{P}(X,Y|C) = \mathbb{P}(X|C)\cdot\mathbb{P}(Y|C)$

velká tabulka

malé tabulky (o dimenzi menší)

Wumpus:



$P_{i,j}$ = pit at (i,j) – pro $\forall i,j$

$B_{i,j}$ = breeze at (i,j) – pro navštívené políčka

$known = \neg p_{1,1} \wedge \neg p_{1,2} \wedge \neg p_{2,1}$

$b = \neg b_{1,1} \wedge b_{1,2} \wedge b_{2,1}$

$\Rightarrow \mathbb{P}[P_{1,3} \mid known, b] = ?$

→ přímočaře: $\mathbb{P} = \alpha \cdot \sum_{unknown} \mathbb{P}(P_{1,3} \mid unknown, known, b)$

↳ $P_{i,j}$ kromě $P_{1,3}$ a known $\Rightarrow 2^{12}$ členů součtu

$\Rightarrow$ unknown = Querry $\cup$ Fringe ... Pits

$\mathbb{P} = \alpha \sum_{unknown} \mathbb{P}(P_{1,3}, unknown, known, b)$ Bayes $P(A|B) = \frac{P(B|A)\cdot P(A)}{P(B)}$

$= \alpha \sum_{unknown} P(b \mid P_{1,3}, known, unknown)\cdot\mathbb{P}(P_{1,3}, known, unknown)$

$= \alpha \cdot \sum_{fringe}\sum_{other} \mathbb{P}(b \mid P_{1,3}, k, fringe, other)\cdot\mathbb{P}(P_{1,3}, k, fringe, other)$

→ nezávislost $b \perp other$

$= \alpha \sum_{fringe}\sum_{other} \mathbb{P}(b \mid P_{1,3}, k, fringe)\cdot\mathbb{P}(P_{1,3}, k, f, o)$

$= \alpha \sum_{fringe} \mathbb{P}(b \mid P_{1,3}, k, f) \sum_{other} \mathbb{P}(P_{1,3}, k, f, o)$

→ jámy jsou na sobě nezávislé

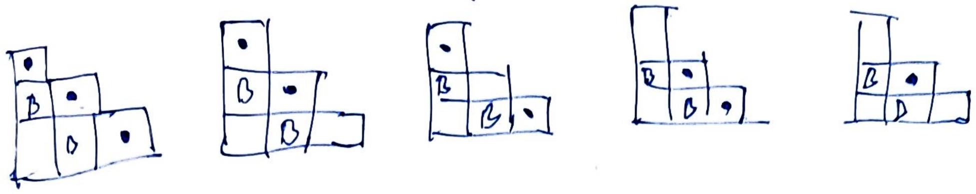
$=$ ——"—— $\cdot \sum_{other} \mathbb{P}(P_{1,3})\cdot P(k)\cdot P(f)\cdot P(o)$

$= \alpha\cdot P(k)\cdot\mathbb{P}(P_{1,3})\cdot\sum_{fringe} P(b \mid P_{1,3}, k, f)\cdot P(f) \sum_{other} P(o) = 1$

$= \alpha\cdot\mathbb{P}(P_{1,3})\cdot\sum_{fringe} \mathbb{P}(b \mid P_{1,3}, k, f)\cdot P(f)$

→ $|fringe| = 2 \Rightarrow 2^2$ možností

· $p = 0.2$ ... prob of pit



$$\mathbb{P}(P_{1,3} \mid known, b) = \alpha \langle 0.2(0.04 + 0.2 \cdot 0.8 + 0.8 \cdot 0.2),\ 0.8(0.04 + 0.2 \cdot 0.8) \rangle$$
$$= \langle 0.31,\ 0.69 \rangle$$

je tam — není tam

<u>Bayesovo pravidlo</u>: $P(Y|X) = \dfrac{P(X|Y) \cdot P(Y)}{P(X)} = \alpha \cdot P(X|Y) \cdot P(Y)$

<u>Naivní Bayesovský model</u>

$P(\text{Příčina} \mid \text{Důsledek}) = P(\text{Důsledek} \mid \text{Příčina}) \cdot P(\text{Příčina}) / P(\text{Důsledek})$

→ $P(\text{Cause}, \text{Effect}_1, \dots \text{Effect}_n) =$

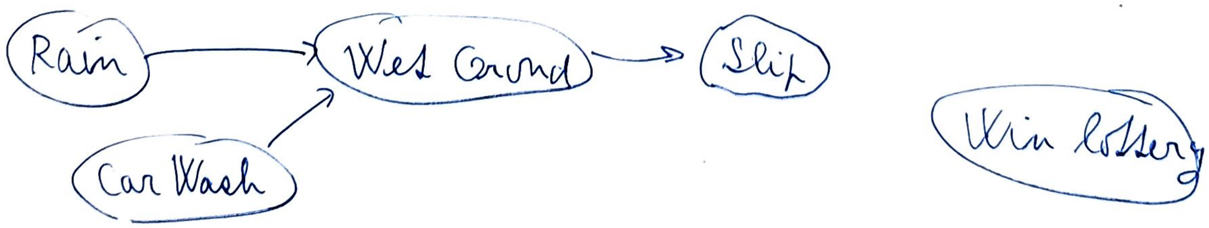
$= P(\text{Cause}) \cdot P(E_1, \dots E_n \mid \text{Cause}) =$

$= P(\text{Cause}) \cdot \prod_i P(E_i \mid \text{Cause})$

↳ předpokládáme nezávislost efektů za podmínky příčiny

<u>Bayesovské sítě</u>

- reprezentace vztahů podmíněné nezávislosti mezi proměnnými
  - uzly ~ proměnné
  - předchůdci =: parents
  - → každý uzel X má pstní distribuci $P(X \mid Parents(X))$

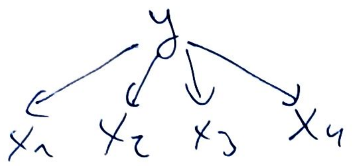


$P(\text{Rain}, \text{Wet}, \text{Car}, \text{Slip}, \text{Lot}) = P(R) \cdot P(C) \cdot P(W \mid R, C) \cdot P(S \mid W) \cdot P(L)$

→ full joint probability distribution

$\Rightarrow P(X_1, X_2, \dots X_n) = \prod_i P(X_i \mid Parents(X_i))$

Příklad



$$P(y, X_1, X_2, X_3, X_4) = P(y) \cdot \prod_i p(X_i | y)$$

↳ naive Bayes

Konstrukce Bayesovské sítě

→ dostaneme množinu náhodných proměnných $X_1, \dots X_n$

↳ chceme vyrobit B.S. aby $P(X_1, \dots X_n) = \prod_i P(X_i \mid Parents(X_i))$

⇒ uspořádáme proměnné jako $X_1, \dots X_n$

↳ ideálně aby příčiny byly před důsledky

→ Hrany vyrobíme takto:

X_1 nebude mít předka

X_i ... z množiny $\{X_1, \dots X_{i-1}\}$ vybereme nejmenší podmnožinu aby $P(X_i | Parents(X_i)) = P(X_i | \{X_1, \dots X_{i-1}\})$

→ proč to funguje?

$$P(X_1, \dots X_n) = P(X_n | X_1, \dots X_{n-1}) \cdot P(X_1, \dots X_{n-1})$$
$$= P(X_n | X_1, \dots X_{n-1}) \cdot P(X_{n-1} | X_1 \dots X_{n-2}) \cdot P(X_1, \dots X_{n-2})$$
$$= \prod_i P(X_i | X_1, \dots X_{i-1}) = \prod_i P(X_i | Parents(X_i))$$

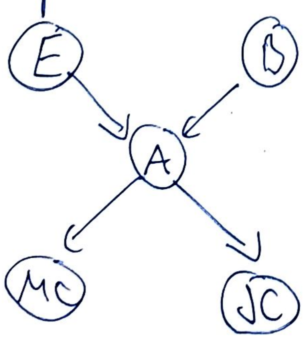
Příklad

- alarm proti zemětřesení, může zazvonit i při loupeži

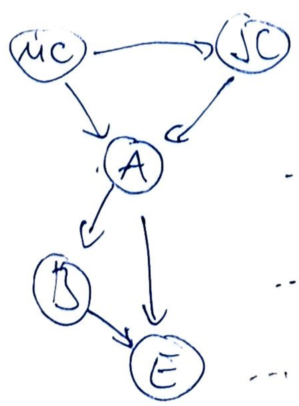
↳ John volá když slyší alarm, ale občas si to splete s telefonem

↳ Mary ——"—— , ale občas ho neslyší

→ přirozená B.S.



Pořadí: MC, JC, A, B, E



... souvisí spolu nepřímo $P(MC | JC) \neq P(MC | \overline{JC})$

.. $P(A | MC) \neq P(A | \overline{MC})$, podobně JC

... $P(B | A) = P(B | A, JC)$

... $P(E | A, B) \neq P(E | A, \overline{B}) = 1$

Inference v Bayesově síti

→ skryté proměnné

$$P(b|j,m) = \alpha \cdot P(b,j,m) = \alpha \cdot \sum_e \sum_a \underbrace{P(b,j,m,e,a)}$$

→ ale počítat ten $\prod$ je neefektivní $\quad \prod_i P(X_i, Parents(X_i))$

→ tady když jsem vyrobil schválně blbou síť, tak asi neznám třeba $P(A|JC, MC)$

→ v té rozumnější síti bych dostal

$$P(b|j,m) = \alpha \sum_e \sum_a P(e)P(b)P(a|e,b)\cdot P(m|a)\cdot P(j|a)$$

$$= \alpha P(b)\cdot \sum_e P(e) \sum_a P(a|e,b)\cdot P(m|a)\cdot P(j|a)$$

→ některé věci budu počítat vícekrát

→ třeba $P(m|a)$ je stejné pro všechny e v vnější sumě

→ má to stromovou strukturu

⇒ dynamické programování

• **Eliminace proměnných**

$$\mathbb{P}(B|j,m) = \alpha\, \mathbb{P}(B)\cdot \sum_e P(e) \sum_a \mathbb{P}(a|e,B)\cdot P(m|a)\cdot P(j|a)$$

$$= \alpha\, f_1(B)\cdot \sum_e f_2(E)\cdot \sum_a f_3(A,B,E)\cdot f_4(A)\, f_5(A)$$

→ m, j jsou konstanty

→ pro ∀ uzel mám v síti tabulku podmíněné pst: CPT

⇒ $f_1, \dots f_5$ = faktory, vyhodnocuju to zprava doleva

$f_4(A)\cdot f_5(A)$ ... vynásobím po složkách A = True a A = False

$f_3(A,B,E)\cdot f'(A)$ → vznikne nová tabulka velikosti $f(A,B,E)$

↳ ∀ entry vlevo vynásobím pstí A vpravo

→ složitější situace

A	B	$f_1(A,B)$
T	T	0.3
T	F	0.7
F	T	0.9
F	F	0.1

*

B	C	$f_2(B,C)$
T	T	0.2
T	F	0.8
F	T	0.6
F	F	0.4

=

A	B	C	$f'(A,B,C)$
T	T	T	0.3 · 0.2
T	T	F	0.3 · 0.8
T	F	T	0.7 · 0.6
T	F	F	0.7 · 0.4
⋮			⋮

$\sum_a f(A,B,E)$

→ prostě sečtu tabulky pro všechny hodnoty A a dostanu $f(B,E)$

→ <u>Monte-Carlo metody</u>

→ samplujeme, odhad toho

→ vzorek z instance náhodných proměnných

→ jak to generovat?

↳ sítě topologicky uspořádáme a projdeme

⇒ můžeme tak počítat tzv podmíněné pstní distribuce

→ ale my navíc máme nějaký evidence e

• <u>Rejection sampling</u>: vzorky nekonzistentní s e ignoruju

$$\Rightarrow P(X|e) = \frac{\#\text{ kde platí } X \text{ i } e}{\#\text{ kde platí } e}$$

• <u>Likelyhood weighting</u>

→ generujeme jen vzorky konzistentní s e

⇒ hodnoty proměnných co známe z evidence zafixuju

→ ale tohle by nefungovalo

↳ chci odhadovat $P(X|e) = \frac{P(X,e)}{P(e)}$

↳ vlastně bych určoval $P(e) = 1$

⇒ když vygeneruju sample z, tak mu přiřadím váhu

$$w(z,e) = \prod_i P(e_i | Parents(e_i))$$

$$\Rightarrow P(X|e) \approx \alpha\, (\#\text{ kde platí } X \text{ i } e) \cdot w(X,e)$$

Decision making

- **Transition model** → definuje $P(X_t \mid X_{0:t-1})$

  → Markov assumption: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$

  → pokud jsou pro všechny t psti stejné → statický model

- **Sensor (observation) model** → $P(E_t \mid X_{0:t}, E_{1:t-1})$

  → Markov assumption: $P(E_t \mid X_{0:t}, E_{1:t-1}) = P(E_t \mid X_t)$

→ lze to modelovat pomocí Bayesovské sítě

$$
\begin{tikzcd}
X_0 \arrow[r] & X_1 \arrow[r] \arrow[d] & X_2 \arrow[r] \arrow[d] & \dots \\
 & E_1 & E_2 &
\end{tikzcd}
$$

$$P(X_{0:t}) = P(X_0) \prod_{t=1}^{T} P(X_t \mid X_{t-1})$$

$$P(X_{0:t}, E_{1:t}) = P(X_0) \prod_{t=1}^{T} P(X_t \mid X_{t-1}) \cdot P(E_t \mid X_t)$$

- **Inference tasks**

1) **Filtering**: Where am i now?

$$P(X_t \mid e_{1:t}) =: f_{1:t}$$

$$f_{1:0} = P(X_0)$$

$$f_{1:t+1} = P(X_{t+1} \mid e_{1:t+1}) = P(X_{t+1} \mid e_{1:t}, e_{t+1})$$

$$= \alpha \cdot P(e_{t+1} \mid X_{t+1}, e_{1:t}) \cdot P(X_{t+1} \mid e_{1:t}) \quad \text{... Bayes rule}$$

$$= \alpha P(e_{t+1} \mid X_{t+1}) \cdot P(X_{t+1} \mid e_{1:t}) \quad \text{... Markov assumption}$$

$$= \alpha P(e_{t+1} \mid X_{t+1}) \cdot \sum_{x_t} P(x_t \mid e_{1:t}) \cdot P(X_{t+1} \mid e_{1:t}, x_t)$$

$$= \alpha P(e_{t+1} \mid X_{t+1}) \cdot \sum_{x_t} f_{1:t} \cdot P(X_{t+1} \mid x_t) \quad \text{... Markov assumption}$$

2) **Prediction**:

$$P(X_{t+k} \mid e_{1:t}) = ?$$

$$P(X_t \mid e_{1:t}) = f_{1:t}, \quad P(X_{t+k+1} \mid e_{1:t}) = \sum_{x_{t+k}} \underline{P(x_{t+k} \mid e_{1:t})} \cdot P(X_{t+k+1} \mid x_{t+k})$$

3) Smoothing → where was I in the past?

$P(X_k|e_{1:t})$, $0 \le k < t$.

→ $P(X_k|e_{1:t}) = P(X_k|e_{1:k}, e_{k+1:t})$ ← Bayes Rule

$= \alpha P(e_{k+1:t}|X_k, e_{1:k}) \cdot P(X_k, e_{1:k})$

$= \alpha \underbrace{P(e_{k+1:t}|X_k)}_{b_{k+1:t}} \cdot f_{1:k}$ -- Markov assumption

→ $b_{k+1:t} = P(e_{k+1:t}|X_k) = \sum_{x_{k+1}} P(x_{k+1}|X_k) \cdot P(e_{k+1:t}|X_k, x_{k+1})$

$= \sum_{x_{k+1}} P(x_{k+1}|X_k) \cdot P(e_{k+1:t}|x_{k+1})$ ... Markov

$= \sum_{x_{k+1}} P(x_{k+1}|X_k) \cdot P(e_{k+1}, e_{k+2:t}|x_{k+1})$

$= \sum_{x_{k+1}} P(x_{k+1}|X_k) \cdot P(e_{k+1}|x_{k+1}) \cdot P(e_{k+2:t}|x_{k+1})$

↳ nezávislost → je vidět z B. s.

$= \sum_{x_{k+1}} P(x_{k+1}|X_k) \cdot P(e_{k+1}|x_{k+1}) \cdot b_{k+2:t}$

→ $b_{t+1:t} = P(e_{t+1:t}|X_t) = P(\text{nic}|X_t) = 1$

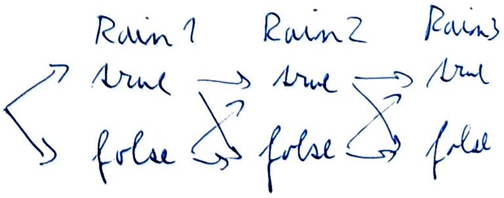
↳ base-case

4) Most likely explanation

→ chci najít sekvenci stavů která nejspíš vygenerovala pozorování

$\operatorname{argmax}_{x_{1:t}} P(x_{1:t}|e_{1:t})$

→ každá sekvence $x_{1:t}$ je cesta grafem



$\max_{x_{1:t}} P(x_1, \dots x_t, X_{t+1}|e_{1:t+1}) =$

$= \alpha \cdot P(e_{t+1}|X_{t+1}) \cdot \max_{x_t} \{P(X_{t+1}|x_t) \cdot \max_{x_{1:t-1}} P(x_1, \dots x_t|e_{1:t})\}$

• **Hidden markov models**

→ pokud pro náš model platí Markovské předpoklady, tak to je HMM &

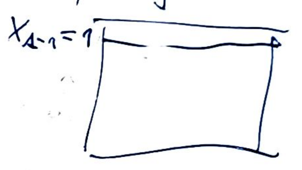
$$\begin{array}{ccccccccc} X_0 & \to & X_1 & \to & X_2 & \to & X_3 & \to & \dots \\ & & \downarrow & & \downarrow & & \downarrow & & \\ & & e_1 & & e_2 & & e_3 & & \end{array}$$

→ jediná proměnná X, kterou nevidíme
→ pozorujeme jedinou proměnnou E

↳ tento jednoduchý model lze reprezentovat maticemi

• **Transition model** je matice $\in \mathbb{R}^{S \times S}$, kde $Im(X) = \{1, \dots S\}$

$$T_{(i,j)} = P(X_t = j \mid X_{t-1} = i)$$



• **Sensor model**

$$O_{t(i,i)} = P(E_t = e_t \mid X_t = i)$$ → diagonální matice

→ filtering a smoothing, respektive
- f = forward message propagation
- b = backward ——"——

se dají implementovat pomocí maticového násobení

→ například lokalizace robota podle čtení ze senzorů
↳ mám evidence $e_1, \dots e_n$ → chci X_t

• **Dynamické Bayesovské sítě**
- reprezentují fct. v čase
- spojení jsou replikovaná mezi time slices
- každá proměnná má rodiče buď ve stejném / předchozím timeslice
  ↳ markov assumption

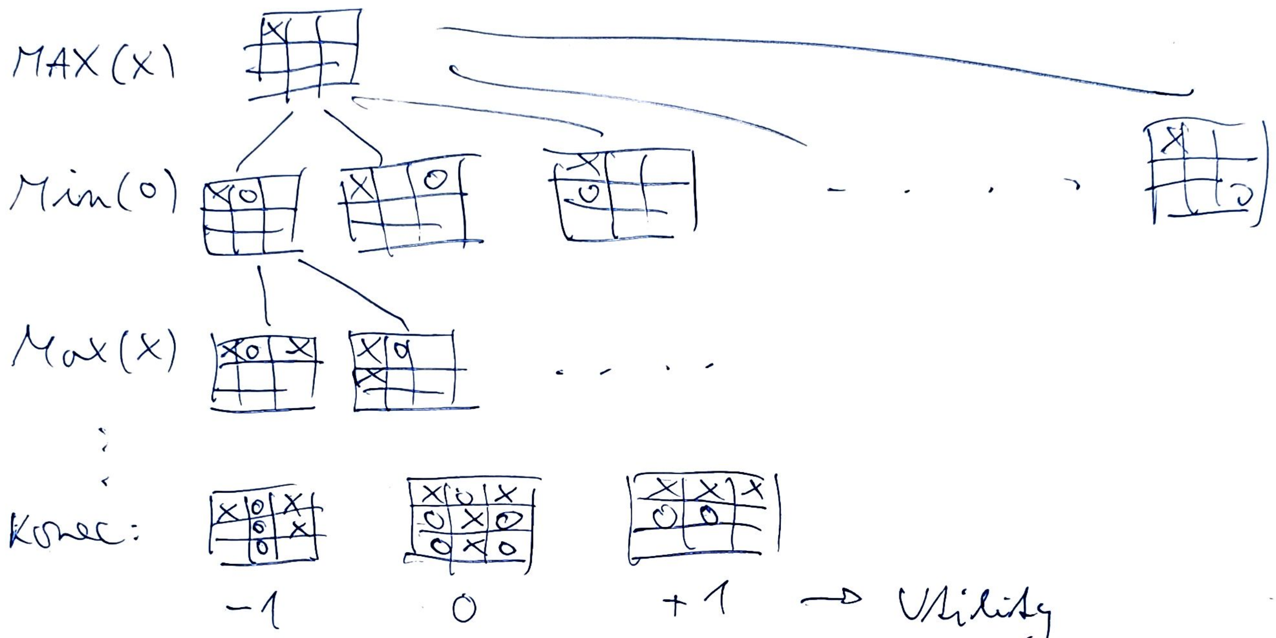
→ HMM je special case DBS, ale ∀ DBS lze reprezentovat jako HMM
↳ ale je to exponenciálně náročnější

Hry a multi-agentní systémy

- nejběžnější hry: deterministické, dva hráči, střídání tahů, zero-sum, plně pozorovatelné

  ↳ Šachy, Go, Piškvorky...

• **Minimax**



Def: Pro stav n definujeme MinimaxValue(n) jako

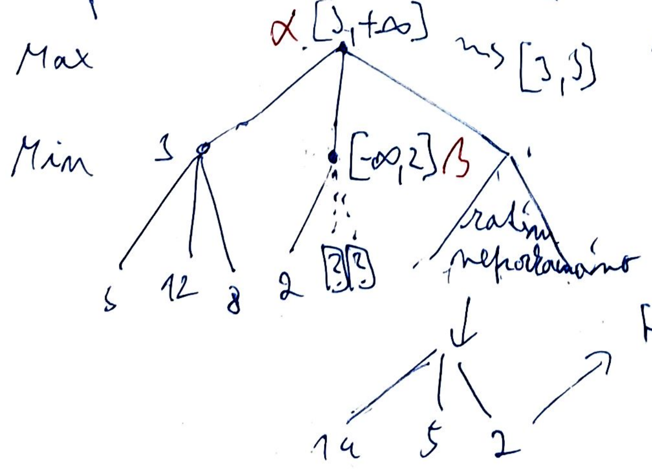
a) utility(n) ... když n je terminální stav

b) $\max_{s \in potomci(n)}$ MinimaxValue(s) ... Max je na tahu

c) $\min_{s \in potomci(n)}$ MinimaxValue(s) ... Min je na tahu

→ v kořeni je min / max, podle toho co je cíl hry

• **Alfa-Beta prořezávání**



α = nejlepší hodnota pro MAX

β = nejlepší pro MIN

↳ max nečeká nadále nic většího

pořadí vyhodnocování potomků je důležité

↳ chci procházet potomky co nejrychleji dostat

↳ mají nejmenší větvící faktor

↳ heuristika

- Evaluation function
  - reálně ten strom celý nikdy nevygeneruju
  - → učíznu prohledávání v nějaké hloubce a aproximuju utility pomocí evaluation fction = heuristika
  - → najdu nějaké rozumé features (šachy)
    - ↳ # pěšáků
    - ↳ # figur celkem
    - ↳ is queen alive
    - ↳ is šach / garde
  - → každé přiřadím váhu → EVAL je nějaká lin. kombinace

- Stochastické hry
  - random element - házení kostkou
  - → počítám Expected minimax Value = EMMV(n)

1) Utility(n) ... n je terminální
2) $\sum_{s \in \text{potomci}(n)} P(s) \cdot EMMV(s)$ ... n je random-uzel
3) $\max_s EMMV(s)$ ... Max hraje
4) $\min_s EMMV(s)$ ... Min hraje

• Single - move games

- všichni hrajou paralelně a mají jen 1 tah

→ payoff function dává hráči utilitu podle toho jak hráli všichni

→ Two - finger Morra

hráči Odd a Even ukážou 1 nebo 2 prsty

příklad: $\left.\begin{array}{l} O=1 \\ E=1 \end{array}\right\} \begin{array}{l} U(O)=-2 \\ U(E)=2 \end{array}$

$\left.\begin{array}{l} O=2 \\ E=1 \end{array}\right\} \begin{array}{l} U(O)=3 \\ U(E)=-3 \end{array}$

→ policy = strategie
- pure - deterministická
- mixed - randomizovaná

→ prisoners dilema

Alice a Bob chyceni, můžou testify, jdou do vězení

	A:T	A:¬T
B:T	A=-5 B=-5	A=-10 B=0
B:¬T	A=0 B=-10	A=-1 B=-1

Racionální strategie je Testify - je to dominant pure strat.

Def: Strategie s pro hráče p dominuje strategii s' ≡ výsledek s je pro p lepší než výsledek s' pro každou možnou volbu strategií ostatních hráčů

Def: Nash equilibrium: žádný hráč si nepomůže, pokud změní strategii, jestliže ji ostatní nezmění

Def: Výsledek hry je Pareto dominated jiným výsledkem, jestliže by ho všichni hráči preferovali.

Def: Outcome je Pareto optimal, pokud jej nic nedominuje.

→ prisoners dilemma vzniká, protože má dominující strategie v Nash eq. (Testify, Testify), která je Pareto dominated výsledkem (refuse, refuse)

• Maximin technique – morra nemá pure strategii (optimální)

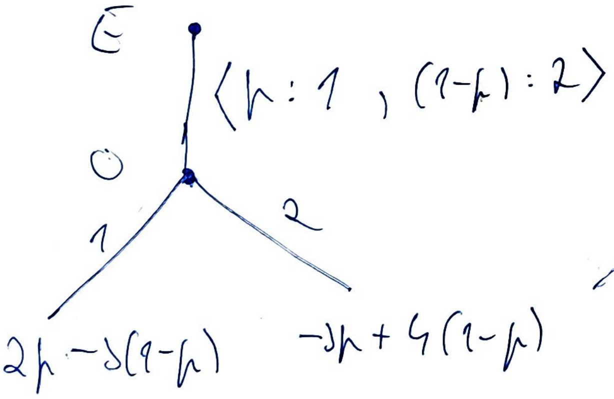
→ budeme se tvářit, že to je turn-taking game

⇒ druhý hráč má výhodu

→ první hráč bude mít mixed strategii

↳ druhý might as well mít pure

→ pst 1 prstu $= p$



→ pro jakou hodnotu p je expected utility pro E nejlepší?

↳ $p = 7/12$

→ podobně pro O

⇒ optimální strategie pro oba hráče je $[7/12 : 1,\ 5/12 : 2]$

- Repeated games
  - bráchi vždy řeší stejný problém a mají historii odpovědí ostatních bráchů
  - payoff se sčítá
  - strategie pro repeated prisoners dilemma
    - hrajou 100 her
      → vlastně záleží jen na té poslední hře
        ⇒ racionální je pořád testify ⇒ všichni 500 let vězení
    - pst, že bude další hra = 0.99
      ⇒ očekávaný počet her je pořád 100, ale not sure
        - perpetual punishment: refuse, dokud druhý brách nedá testify
          → pak vždy testify
        - tit-for-tat
          - začnu s refuse a pak se vždy opičím po protivníkovi
            ↳ tohle funguje překvapivě dobře

Teorie her analyzuje chování agentů

Inverzní teorie her se snaží definovat prostředí, aby celkový payoff pro všechny agenty byl co nejlepší

↳ Mechanism design

↳ příklad = aukce

• Mechanism design

Aukce funguje tak, že každý Bidder má privátní value v_i pro daný předmět, a ten předmět má common value (ta nikdo nezná)

→ všichni udělají bid — různé důvody

↳ nejvyšší bid dostane item

Good mechanism

- maximalizuje expected revenue pro prodejce
- maximalizuje global utility — vyhraje ten, kdo si itemu váží nejvíce

1) Ascending-bid auction

- začne s minimem bid_{min}

→ pokud to někdo je ochotný zaplatit → požádám $b_{min} + d$

→ pokračuju, dokud už nikdo nedá víc

- simple dominant strategy = bid iff price $\leq v_i$
- odradí ostatní, pokud tam je Elon Musk

2) descending-bid auction

- začne vysoko a snižuju, dokud to někdo nekoupí
- rychlé → prodej kytek v Holandsku

3) sealed-bid auction

- všichni pošlou bid v obálce → nejvyšší vyhraje
- nemá simple dominant strategy

4) sealed-bid second-price auction

- jako sealed-bid, ale vyhrávající zaplatí druhou nejvyšší cenu

⇒ dominantní strategie = bid v_i

• Tragedy of commons

- common goals: pollution

→ země bude zaplatit -10 za akci pro zlepšení

→ nebo -5 na zdraví občanů a -1 pro okolní země

⇒ dominantní je nic nedělat

⇒ alkoví horší pro všechny

→ tohle řeší nějaký Vickrey-Clarke-Groves mechanism

Decision trees

- typ machine learning: vezme vektor a vrátí jednu hodnotu

→ je to strom, ve vnitřních vrcholech jsou testy, které čnějí podle odpovědi → hrany ~ odpovědi.

→ listy = return values

Hypotesis space = množina stromů, které jsou konzistentní se vstupními daty → chceme ten nejjednodušší
↳ Okhamova břitva

Jak vyrobit strom?

→ rozděl a panuj

→ zvolíme nejdůležitější atribut a ten dáme do kořene

atributy: vybírám restauraci → hungry?, Type rest.?, Day of week?

odpovědi: jít / nejít

→ máme množinu examplů ex.

↳ hungry bude asi hodně : Not hungry → NO
Hungry → spíš Yes

↳ day of week: asi půl na půl, moc info mi to neříká

→ formálně to používá statistickou entropii

→ když mám nejdůležitější atribut, tak examples rozdělím podle hodnot toho atributu

Hungry Not hungry

→ teď se rekurzivně zavolám na tyhle dvě hromádky dat, ale ignoruju atribut Hungry

→ až bude pořád už nebudu mít žádný zbývající atribut nebo všechny examples spadnou do stejné hromádky → list

• Logická klasifikace

– lze to zapsat formálně

→ atributy jsou predikáty

¬ Type(X_1, French)

¬ Hungry(X_1) ∧ Fri/Sat(X_1) ∧ ...

– rozhodnutí jsou taky predikáty

Př: WillWait (r) ⟺ Hungry(r) ∨ (¬Hungry(r) ∧ Type(r, French)) ∨ ...

– Hypotéza je ta správná formulka pro zadanou otázku

→ věříme, že existuje nějaká hypotéza konzistentní se všemi příklady

jak být inconsistent?

• false negative – říkám NE ale je ANO
• false positive – říkám ANO ale je NE

→ jak získat hypotézu?

1) Current-best-Hypothesis

– pamatuju si jen 1 hypotézu a upravuju ji postupně examply
– když example je
  • konzistentní s hypotézou → nic neměním
  • false negative – hypotéza je moc přísná → generalizace
    ⇒ odeberu podmínky (∧), nebo přidám možnosti (∨)
  • false positive ⇒ specializuju: přidám (∧), nebo odeberu (∨)

2) Least commitment search

→ nemám jen 1 hypotézu, ale všechny, co jsou konzistentní s zatím viděnými examples

" Version space

→ jak efektivně reprezentovat version space?

→ mám boundary sets

- G-set = most general boundary

  ↳ na začátku tam dám jen True = vše projde

  → pro ∀ nový example:

  - false positive pro $h \in G$

    ⇒ nahradím h v G za všechny immediate specializace h

  - false negative pro $h_i \in G$

    ⇒ odeberu h_i z G – je málo obecná

- S-set = most specific boundary

  ↳ na začátku false

  nový example

  - false positive ⇒ odeberu h z S
  - false negative ⇒ nahradím h v S za generalizaci h

→ všechno „mezi" G a S je konzistentní se všemi examples

↳ musím mít nějaké uspořádání (ne lineární) na tom hypothesis space

↳ abych mohl říct co je immediate specializace / generalizace

- Statistické učení

- bodové odhady — metoda momentů
  — most likely explanation
  ↳ předpokládám nezávislost a stejnou distribuci
  ↳ napíšu formuli pro pst v závislosti na parametru

$$L(X_1, \dots X_n, \theta) = P(X_1 = x_1, \dots X_n = x_n ; \theta)$$

$$= \prod_i p_i(x_i) = \prod_i f_i(x_i)$$

diskrétní ↳ spojité

⇒ zlogaritmuju to

$$l(\overline{X}, \theta) := \log L(\overline{X}, \theta)$$

→ najdu maximum pomocí derivace podle θ

- Expectation maximalization alg.

⇒ máme: nějakou hidden variable, která ovlivňuje data, ale neznáme její hodnotu

1, předstírám, že znám parametr pstního modelu pro tu proměnnou

2, inferuju expected hodnoty skrytých proměnných, abych „doplnil" data — mám Bayesovskou síť

3, podívám se, jestli to sedí s modelem

↳ je to most-likely explanation?

⇒ updatnu parametr (pomocí pst pro více proměnných)

Decision theory

$R_a(s)$ = reward za akci a ve stavu

↳ v deterministickém prostředí

→ teď nedeterministické, partially observable

$R(a)$ → náhodná veličina udávající efekt akce a

Def: Expected utility akce a, jestliže jsem zatím dostal evidence o prostředí e (třeba ze senzorů) je

↳ $e = e_1, e_2, \dots e_s$

$$EU[a|e] := \sum_s P[R(a) = s \mid a, e] \cdot U(s)$$

↳ utility stavu

Maximum EU principle: akce = $\arg\max_a EU(a|e)$

Utility theory

→ než nějak obecně vyjádřit Utility stavu, je pro agenta lehčí říct, jestli se mu víc líbí stav A nebo stav B

↳ tournament selection

→ chceme vyrobit utility tak, aby

$$U(A) < U(B) \equiv A < B$$
$$U(A) = U(B) \equiv A \sim B$$

→ nejlepší stav S_{max} ⇒ $U = 1$
nejhorší stav S_{min} ⇒ $U = 0$ $\Big\}$ $U \in [0,1]$

↳ kam zařadit S ?

→ zeptáme se agenta, jestli dá přednost S nebo loterii s pstí p mezi S_{max} a S_{min} : $\langle p : S_{max}, (1-p) : S_{min} \rangle$

→ binárním vyhledáváním najdu takové p, že pro agenta $S \sim$ loterie(p)

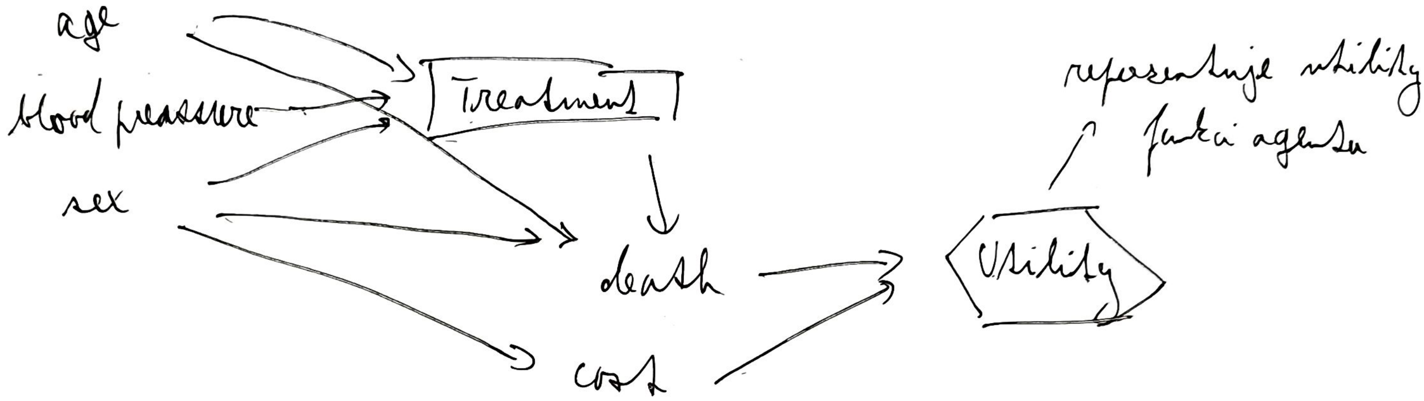
$$\Rightarrow U(S) := p$$

Decision networks

→ Bayesovské sítě, do kterých kromě uzlů pro náhodné proměnné dáme i decision nodes

↳ decision maker sám zvolí nějakou akci

→ utility nodes → reprezentují utility pro agenta



Evaluace:

1. nastavím hodnoty proměnných co znám z evidence
2. pro ∀ možnou hodnotu decision uzlu
    a) nastav hodnotu uzlu
    b) spočítej postupně psti rodičů / předků utility nodu
        ↳ normální bayesovská inference
    c) urči utility pro tuto akci
3) vyber akci s nejlepší utility

- Sequential decision problems

- agent se musí platně rozhodovat v nedeterministickém prostředí

→ Saturn: rozbitý robot, p NORTH, p WEST, p SOUTH, p EAST

↳ prostředí je nebezpečné, cena za jednotlivá políčka

↳ vím, kde jsem a chci dojít do cíle za co nejmenší cenu

Def: Markovský rozhodovací proces

$$P_a(s, s'), \quad R(s)$$

$\Rightarrow$ celková odměna $R(s_0, s_1, \dots) = \sum_{i=0}^{\infty} \gamma^i R(s_i)$

→ hledáme policy $\pi : S \to A$

→ expected utility $U^\pi(s) := \mathbb{E}[R^\pi \mid s_0 = s]$, $a_i \sim \pi(s_i)$

Bellmanova rovnice:

$$U^\pi(s) = \mathbb{E}_{a, s_1}\left[r_0 + \sum_{i=1}^{\infty} \gamma^i R^a(s_i) \mid s_0 = s\right]$$

$$= \mathbb{E}_{a, s_1}\left[r_0 + \gamma U^\pi(s_1) \mid s_0 = s\right]$$

$$\Rightarrow U^\pi(s) \approx r_0 + \gamma \cdot \max_a \mathbb{E}_{s'}\left[U^\pi(s_1)\right]$$

$$U(s) = r_0 + \gamma \cdot \max_a \sum_{s'} P_a(s, s') \cdot U(s')$$

↳ for truly optimal utility / policy

↳ to stačí jako akční volič

$$\pi^*(s) = \arg\max_a \sum_{s'} P_a(s, s') \cdot U^{\pi^*}(s')$$

• Value iteration

→ máme rovnice pro každý stav, jejichž řešení je optimální matice U

→ ale ty rovnice nejsou lineární :(

1. U inicializujeme náhodně
2. iterativně pomocí utility sousedů updatujeme jejich utility

$$U(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P_a(s, s') \cdot U(s')$$

→ to dělám, dokud se to nějak rozumně mění

→ pokud je maximální změna hodnoty v utilitě $< \varepsilon \Rightarrow$ konec

• Policy iteration

je možné mít optimal policy i když U není optimální

1. U opět náhodně, π náhodně
2. iterativně

$U \leftarrow$ vyhodnotit policy (dole)

→ pokud pro nějaký stav $s \in S$ existuje akce $a \in A$ t.ž.

$EU(s|a) > EU(s|\pi(s))$, tak $\pi(s) \leftarrow a$

→ pokud se policy nezměnila → konec

Vyhodnotit policy (π): vrátí U^{π_i}

$$U^{\pi_i}(s) = R(s) + \gamma \cdot \sum_{s'} P(s' | s, \pi_i(s)) \cdot U^{\pi_i}(s')$$

→ tohle se dá vyřešit přesně pomocí Gaussovky

→ nebo pro velké prostory aproximace pomocí value iteration

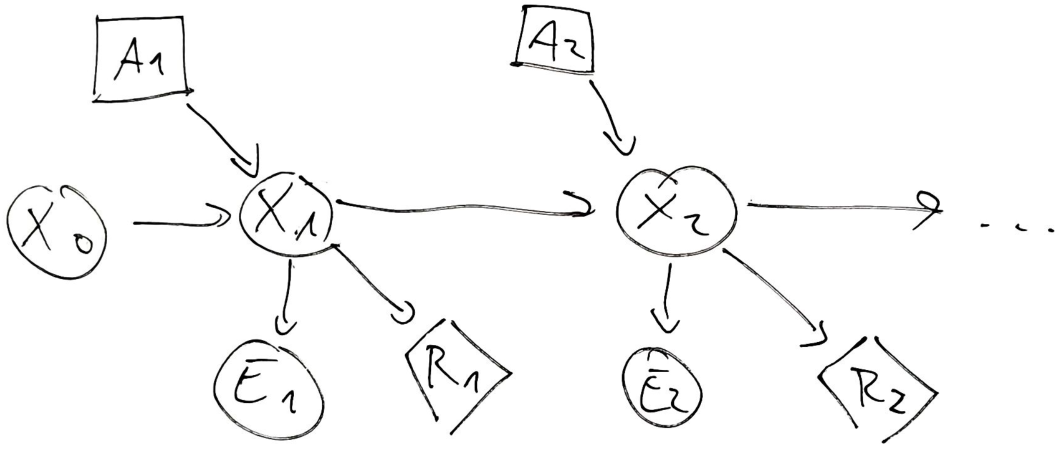
• **Partially observable MDP**

- předtím fully observable

→ teď navíc máme sensor model $P(s|e)$

→ nevíme, kde jsme → belief states = pstní distribuce přes všechny možné stavy

→ rozšiřuje se to do dynamické decision network



jak velká má síť být? je tam diskontní faktor γ, takže budoucnost není tak důležitá

→ rozhodnutí dělám tak, že přes filtering odhadnu kde jsem a zkouším nějaké posloupnosti akcí, vyberu tu nejlepší