

Základy umělé inteligence

Studijní materiál ke státnicím (S1)

Jakub Klesa

Pokryté předměty: NAIL120 (Úvod do umělé inteligence)

Obsah

Úvod	3
1 Řešení úloh prohledáváním (DFS, BFS, uniform-cost, A*, heuristiky)	3
1.1 Formulace úlohy	3
1.2 Strom versus graf prohledávání	4
1.3 Neinformované prohledávání	5
1.4 Informované (heuristické) prohledávání	5
1.5 Řešený příklad SZZ	7
2 Splňování podmínek – CSP (hranová konzistence AC-3, algoritmus MAC)	8
2.1 Definice CSP	8
2.2 Modelování (jak úlohu zformulovat jako CSP)	9
2.3 Řešení prohledáváním s navracením (backtracking)	9
2.4 Konzistenční techniky: hranová konzistence	9
2.5 Algoritmus AC-3	10
2.6 Inference v průběhu prohledávání: FC, look-ahead a MAC	11
2.7 Heuristiky pořadí proměnných a hodnot	12
2.8 Řešené příklady SZZ	12
3 Logické uvažování (DPLL, dopředné a zpětné řetězení, rezoluce)	15
3.1 Model, splnitelnost, důsledek	15
3.2 Konjunktivní a disjunktivní normální forma	16
3.3 Algoritmus DPLL	16
3.4 Řešený příklad SZZ: DPLL a perfektní párování	17
3.5 Rezoluce	19
3.6 Hornovy klauzule, dopředné a zpětné řetězení	20
4 Pravděpodobnostní uvažování (Bayesovské sítě, exaktní a aproximační odvozování)	22
4.1 Základy teorie pravděpodobnosti	22
4.2 Nezávislost a podmíněná nezávislost	23
4.3 Bayesovské sítě: definice, konstrukce, vztah ke sdružené distribuci	24
4.4 Exaktní odvozování: enumerace	25
4.5 Exaktní odvozování: eliminace proměnných	25
4.6 Aproximační odvozování: metody Monte Carlo	27
4.7 Řešený příklad SZZ	28

5	Reprezentace znalostí (situační kalkulus, Markovovy modely, HMM)	29
5.1	Situační kalkulus a problém rámce	30
5.2	Uvažování v čase: Markovovy předpoklady	30
5.3	Čtyři základní inferenční úlohy	31
5.4	Filtrace a dopředná rovnice	32
5.5	Vyhlažování a zpětná rovnice	32
5.6	Nejpravděpodobnější průchod: Viterbi	33
5.7	Skryté Markovovy modely (HMM) a maticová formulace	34
5.8	HMM versus dynamické Bayesovské sítě (DBN)	34
5.9	Řešený příklad SZZ	34
6	Automatické plánování (dopředné a zpětné plánování)	36
6.1	Stavy, atomy a cíle	36
6.2	Operátory a akce	37
6.3	Použitelnost akce, výsledek a relevance	37
6.4	Plánovací doména, problém a plán	38
6.5	Dopředné a zpětné plánování	38
6.6	Plánovací heuristiky	39
6.7	Řešený příklad SZZ	39
7	Markovovy rozhodovací procesy (Bellmanova rovnice, iterace hodnot a strategií, POMDP)	41
7.1	Rozhodování za nejistoty a princip MEU	41
7.2	Markovův rozhodovací proces	42
7.3	Bellmanova rovnice	43
7.4	Iterace hodnot (value iteration)	44
7.5	Iterace strategií (policy iteration)	44
7.6	Částečně pozorovatelný MDP (POMDP)	45
7.7	Řešený příklad SZZ	46
8	Hry a teorie her (minimax, alfa-beta, Nashovo ekvilibrium, aukce)	47
8.1	Herní strom a optimální strategie	47
8.2	Minimax	48
8.3	Alfa-beta prořezávání	49
8.4	Nedokonale rozhodnutí: ohodnocovací funkce a ořezání hloubky	50
8.5	Jednotahové (simultánní) hry	50
8.6	Směšené strategie a maximinová technika	51
8.7	Opakované hry	51
8.8	Mechanism design a aukce	51
8.9	Řešené příklady SZZ	52
9	Strojové učení – přehled (druhy učení, rozhodovací stromy, regrese, SVM, zpětnovazební učení)	54
9.1	Druhy učení	54
9.2	Rozhodovací stromy	55
9.3	Lineární modely: regrese a klasifikace	57
9.4	Support vector machines (SVM)	58
9.5	Bayesovské učení a EM algoritmus	59
9.6	Zpětnovazební učení (reinforcement learning)	61
9.7	Shrnutí sekce	63
	Shrnutí	63

Úvod

Tento materiál pokrývá okruh **S1 – Základy umělé inteligence** (předmět NAIL120 *Úvod do umělé inteligence*). Výklad stojí na slidech přednášejícího **Romana Bartáka** (stránka kurzu https://ktiml.mff.cuni.cz/~bartak/ui_intro/); kurz oficiálně staví na učebnici Russella a Norviga *Artificial Intelligence: A Modern Approach* (AIMA), z níž pochází i standardní pseudokód algoritmů a jednotné značení. Okruh je v podstatě „celá umělá inteligence v kostce“: prochází hlavní paradigmaty – prohledávání, splňování podmínek, logické a pravděpodobnostní uvažování, uvažování v čase, plánování, rozhodování za nejistoty, hry a strojové učení.

Role důkazů. Požadavky okruhu jmenují pojmy, algoritmy a jejich vlastnosti, ne formální důkazy. Těžiště je ve formalizaci úlohy, v simulaci algoritmu na konkrétním vstupu, ve výpočtu pravděpodobnosti či užitku a v protipříkladech. Věty a vlastnosti proto uvádíme se zněním, intuicí a krátkým *odvozením* tam, kde pomáhá pochopení – například optimalita A^* slouží jako *argument* (proč přípustná heuristika zaručí optimální řešení).

Do výkladu je zařazeno všech třináct reálných otázek SZZ k okruhu z let 2022–2026 jako řešené příklady; jejich originální zadání (a diagramy oříznuté z PDF) jsou i v samostatném dokumentu *Příklady otázek*. Použité zdroje jsou uvedeny na konci.

Značení. Stav značíme s , akci a , množinu stavů S , množinu akcí použitelných ve stavu s jako $A(s)$. Cena (krok) akce je $c(s, a, s')$, cesta z kořene do uzlu n má cenu $g(n)$, heuristika $h(n)$ odhaduje cenu z n do cíle, hodnoticí funkce $f(n) = g(n) + h(n)$. Náhodné veličiny píšeme velkými písmeny (X, E), jejich hodnoty malými (x, e); $\mathbf{P}(X)$ je *rozdělení* (vektor), $P(x)$ konkrétní pravděpodobnost. Sdružené rozdělení $\mathbf{P}(X, Y)$, podmíněné $\mathbf{P}(X | Y)$; α je normalizační konstanta. V uvažování v čase je X_t skrytý stav v čase t , E_t pozorování, $X_{a..b}$ posloupnost $X_a, \dots, X_b$. V rozhodování je $P(s' | s, a)$ přechodový model, $R(s)$ odměna, $\gamma \in [0, 1]$ diskontní faktor, $U(s)$ užitek stavu, $\pi(s)$ strategie. Logaritmus log bez základu je $\log_2$ (entropie v bitech).

1 Řešení úloh prohledáváním (DFS, BFS, uniform-cost, A^* , heuristiky)

Mnoho úloh umělé inteligence lze zformulovat jako hledání *posloupnosti akcí*, která agenta převede z výchozího stavu do stavu cílového. Takový *problémově orientovaný agent* pracuje s *atomickou* reprezentací stavů (stav je nedělitelný „černý box“), cíl popisuje množinou cílových stavů a akce popisují přechody mezi stavy. Řešení se hledá *prohledáváním*. Předpokládáme přitom prostředí *plně pozorovatelné, diskrétní, statické a deterministické*: agent zná svůj stav i výsledek každé akce, takže může celou posloupnost naplánovat dopředu a pak ji slepě provést.

1.1 Formulace úlohy

Definice – Dobře definovaná úloha

Úloha prohledávání je dána čtveřicí:

- **počáteční stav** s_0 ;
- **přechodový model** (a množina akcí): pro stav s je $A(s)$ množina použitelných akcí a $\text{RESULT}(s, a)$ je stav po provedení akce a ve stavu s ;
- **test cílovosti**, který rozhodne, zda je daný stav cílový;
- **cena cesty**: nezáporná *cena akce* (kroku) $c(s, a, s')$; cena cesty je součet cen jejích kroků.

Stavový prostor (graf, jehož vrcholy jsou stavy a hrany akce) je tím zadán *implicitně* – nevypisujeme ho celý, jen ho generujeme po krocích. *Řešením* je posloupnost akcí z s_0 do nějakého cílového stavu; *optimální řešení* má nejmenší cenu cesty.

💡 Intuice. Klíčem k rozumné formulaci je *abstrakce*: ze stavu světa ponecháme jen to, co je pro úlohu podstatné. Abstrakce musí být *platná* (každé abstraktní řešení lze rozvinout na řešení v detailnějším světě) a *užitečná* (jednotlivé akce se provádějí snáz než původní úloha). Bez abstrakce by agenta reálný svět okamžitě zahltil.

Příklad — 8-puzzle jako úloha prohledávání

V hlavolamu 8-puzzle je 3×3 mřížka s osmi očíslovanými kostkami a jedním volným místem; tahem posuneme kostku na sousední volné místo. **Stav** = rozmístění kostek (a volného místa); **počáteční stav** = zadané rozmístění; **akce** = posun volného místa nahoru/dolů/vlevo/vpravo (ekvivalentně posun sousední kostky), **cena** každého kroku = 1; **test cílovosti** = shoda se zadaným cílovým rozmístěním. Stavový prostor má $9!/2 = 181\,440$ dosažitelných stavů (poloviny permutací jsou nedosažitelné kvůli paritě). Plné řešení této úlohy je níže (SZZ jaro 2024).

1.2 Strom versus graf prohledávání

Jádro každého prohledávacího algoritmu je stejné: udržujeme *hranici* (frontier, otevřený seznam) ještě nerozvinutých uzlů, opakovaně z ní podle nějaké *strategie* vybereme uzel, *rozvineme* ho (vygenerujeme jeho následníky a přidáme je na hranici) a končíme, jakmile narazíme na cílový uzel (nebo je hranice prázdná).

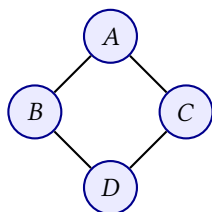
Algoritmus — Obecné prohledávání (strom / graf)

```
GRAPH-SEARCH(úloha):
  frontier ← { uzel s počátečním stavem  $s_0$  }
  explored ← ∅
  while frontier ≠ ∅:
    n ← vyber uzel z frontier
    if n.stav je cílový: return cesta k n
    přidej n.stav do explored
    for každý následník  $n'$  uzlu n:
      if  $n'.stav \notin explored$  (a není už na frontier): přidej  $n'$  na frontier
```

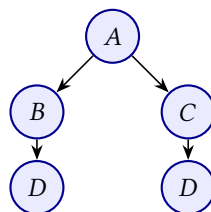
▷ *closed list; jen u grafového prohledávání*

▷ *strategie: FIFO / LIFO / min g / min f*

Stromové prohledávání vynechá množinu explored: tatáž konfigurace se tak může objevit ve více uzlech a algoritmus *opakovaně rozvíjí* už navštívené stavy (prochází redundantní cesty). **Grafové prohledávání** si pomocí explored (closed list) pamatuje rozvinuté stavy a každý stav rozvine *nejvýš jednou*; strom prohledávání pak obsahuje každou konfiguraci nanejvýš jednou. Cenou za úsporu času je paměť na uložení explored. (Pseudokód výše testuje cílovost *při výběru* uzlu z hranice; u BFS lze testovat už *při generování* následníka, což je rychlejší, ale u uniform-cost a A^* by to vedlo k neoptimálnímu řešení – tam musí být test až při výběru.)



stavový graf



strom (stromové prohl.)

Stav D je dosažitelný dvěma cestami ($A-B-D$ a $A-C-D$). *Stromové* prohledávání proto D rozvine **dvakrát** (pravý strom má dva uzly D); *grafové* prohledávání drží closed list a rozvine D jen jednou. Redundance roste exponenciálně s hloubkou, takže u grafů (cykly, více cest) je grafové prohledávání zásadní.

1.3 Neinformované prohledávání

Neinformované (slepé) strategie nemají o úloze žádnou informaci nad rámec její definice; liší se jen tím, který uzel z hranice vyberou.

Definice – BFS, DFS, uniform-cost

- **Prohledávání do šířky** (BFS): hranice je fronta *FIFO*, rozvíjí se vždy *nejmělčí* nerozvinutý uzel.
- **Prohledávání do hloubky** (DFS): hranice je zásobník *LIFO*, rozvíjí se vždy *nejhlubší* nerozvinutý uzel.
- **Uniform-cost** (Dijkstra): rozvíjí se uzel s *nejnižší cenou cesty* $g(n)$; test cílovosti se provádí až při výběru uzlu (ne při generování), jinak hrozí neoptimální řešení.

 **Intuice.** BFS je úplné (najde řešení, je-li větvení konečné) a je *optimální*, pokud cena roste nezáporně s hloubkou (typicky jednotkové ceny). Platí za to ale paměť: musí držet celou hranici, časová i prostorová složitost je $O(b^d)$ (b = faktor větvení, d = hloubka cíle). *Paměť je u BFS větší problém než čas.*

DFS sám o sobě *není* optimální a ve stromovém prohledávání ani úplný (může se zacyklit nebo spadnout do nekonečné větve). Jeho přednost je paměť: s *backtrackingem* (generuje se vždy jen jeden následník) drží jen jednu větev, tedy $O(m)$ paměti (m = maximální hloubka); časově $O(b^m)$, kde m může být $\gg d$.

Uniform-cost je rozšíření BFS na libovolné nezáporné ceny – expanduje stavy v pořadí rostoucí ceny cesty, je úplné i optimální. Je to přesně Dijkstrův algoritmus.

 **Poznámka.** Srovnání hlavních strategií (faktor větvení b , hloubka cíle d , max. hloubka m , nejmenší cena kroku ε , cena optima C^*):

	výběr uzlu	úplné?	optimální?	čas / paměť
BFS	nejmělčí (FIFO)	ano [†]	ano [‡]	$O(b^d) / O(b^d)$
DFS	nejhlubší (LIFO)	ne [§]	ne	$O(b^m) / O(bm)$
Uniform-cost	nejmenší $g(n)$	ano ^{†*}	ano	$O(b^{1+\lceil C^*/\varepsilon \rceil})$

[†] při konečném větvení; ^{*} navíc je-li cena každého kroku aspoň $\varepsilon > 0$ (jinak hrozí nekonečná cesta s nulovými cenami); [‡] je-li cena neklesající funkcí hloubky; [§] úplné je grafové DFS v konečném prostoru, stromové ne.

1.4 Informované (heuristické) prohledávání

Informované prohledávání využívá doménovou znalost ve formě *heuristiky*. Obecné schéma je *best-first*: rozvíjí se uzel s nejmenší hodnotou *hodnoticí funkce* $f(n)$ (uniform-cost je speciální případ s $f = g$).

Definice – Heuristika, hladové prohledávání, A^*

Heuristická funkce $h(n) \geq 0$ odhaduje cenu nejlevnější cesty z (stavu) uzlu n do cíle; v cílovém stavu je $h(n) = 0$.

- **Hladové best-first:** $f(n) = h(n)$ – rozvíjí uzel zdánlivě nejbliž cíli. Není optimální ani (ve stromové variantě) úplné.
- **A^* :** $f(n) = g(n) + h(n)$ – odhad ceny nejlevnějšího řešení *vedoucího skrz* n . Jako uniform-cost, jen místo g používá $g + h$. Za vhodných podmínek na h (připustnost, resp. konzistence – definujeme vzápětí) je úplné a optimální.

Definice – Přípustná a konzistentní heuristika

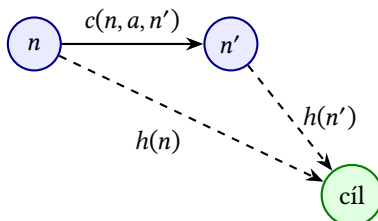
- Heuristika je **přípustná** (admissible), pokud nikdy *nenadhodnotí*: pro každý uzel n je

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

kde $h^*(n)$ je skutečná cena nejlevnější cesty z n do cíle. (Optimistický odhad.)

- Heuristika je **konzistentní** (monotónní), pokud pro každý uzel n a jeho následníka n' akcí a platí *trojúhelníková nerovnost*

$$h(n) \leq c(n, a, n') + h(n').$$



Konzistence jako trojúhelníková nerovnost: odhad z n není větší než cena kroku do souseda n' plus odhad z n' , tj. $h(n) \leq c(n, a, n') + h(n')$. Důsledkem je, že hodnoty $f(n) = g(n) + h(n)$ *neklesají* podél žádné cesty.

★ Lemma – Konzistence implikuje přípustnost

Každá konzistentní heuristika je přípustná.

Důkaz (nepovinný – pro pochopení vztahu obou pojmů). Buď $n_1, n_2, \dots, n_k$ optimální cesta z n_1 do cíle n_k (tedy $h(n_k) = 0$). Z konzistence pro každý krok $h(n_i) - h(n_{i+1}) \leq c(n_i, a_i, n_{i+1})$. Sečtením přes $i = 1, \dots, k-1$ se levá strana teleskopicky sečte na $h(n_1) - h(n_k) = h(n_1)$, pravá na cenu celé (optimální) cesty $h^*(n_1)$. Tedy $h(n_1) \leq h^*(n_1)$, což je přípustnost. $\square$

★ Věta – Optimalita A^*

Je-li h **přípustná**, je A^* ve *stromovém* prohledávání optimální. Je-li h **konzistentní**, je A^* v *grafovém* prohledávání optimální.

💡 Intuice. Strom + přípustnost. Předpokládejme, že na hranici je suboptimální cíl G_2 (cena $g(G_2) > C^*$, kde C^* je cena optimálního řešení) a zároveň uzel n ležící na optimální cestě. Pak

$$f(G_2) = g(G_2) + \underbrace{h(G_2)}_{=0} = g(G_2) > C^* \geq g(n) + h(n) = f(n),$$

kde poslední nerovnost plyne z přípustnosti ($f(n)$ je dolní odhad ceny optima skrz n). A^* tedy rozvine n *dříve* než G_2 a přes n najde optimální řešení – suboptimální cíl se nikdy nevybere první.

Graf + konzistence. U grafového prohledávání hrozí, že stav navštívíme podruhé lepší cestou, ale closed list ho už nepustí. Konzistence to vyloučí: hodnoty f jsou podél každé cesty *neklesající*, takže když A^* vybírá uzel s nejmenším f k rozvinutí, žádná pozdější cesta k témuž stavu nemůže být kratší. První rozvinutý cílový uzel je proto optimální. (Bez konzistence by stačilo přidat účetnictví: při nalezení lepší cesty stav z closed listu nechat „obživnout“.)

📌 Poznámka. [Dominance heuristik] Pro dvě přípustné heuristiky řekneme, že h_2 *dominuje* h_1 , pokud $h_2(n) \geq h_1(n)$ pro všechny n . Pak A^* s h_2 nikdy nerozvine víc uzlů než s h_1 (rozvíjejí se uzly s $f(n) < C^*$, tj. $h(n) < C^* - g(n)$; vyšší h ořeže víc). Vyšší *přípustná* heuristika je tedy *lepší* – pokud ji umíme spočítat dostatečně levně.

1.5 Řešený příklad SZZ

Následující reálná otázka SZZ spojuje všechny tři dovednosti tohoto okruhu: *zformalizovat* úlohu jako prohledávání, *zvolit* a popsat optimální algoritmus a *rozebrat* vlastnosti navržené heuristiky.

Příklad — SZZ jaro 2024 č. 32: Informované prohledávání (A^* , heuristika, 8-puzzle)

(Reálná otázka SZZ: jaro 2024, č. 32 – Informované prohledávání (A^* , heuristika, 8-puzzle))

Zadání.

Pro 8-puzzle na obrázku níže (vlevo počáteční, vpravo cílový stav): (1) formalizujte úlohu jako informované prohledávání (stav, přechodový model, test cílovosti); (2) navrhnete optimální informovaný algoritmus a popište jeho průběh; (3) navrhnete heuristiku, určete její vlastnosti a rozhodněte, zda s algoritmem z (2) zaručuje optimální řešení.

Řešení.

(1) Formalizace. Stav je rozmístění osmi kostek a volného místa v mřížce 3×3 . Počáteční stav je zadané rozmístění. Přechodový model: akce posune volné místo na jedno ze sousedních polí (tj. sousední kostku na volné místo), čímž vznikne nové rozmístění; cena každé akce je 1 (chceme nejmenší počet tahů). Test cílovosti porovná rozmístění s cílovým.

(2) Algoritmus. Optimální informovaný algoritmus je A^* s $f(n) = g(n) + h(n)$, kde $g(n)$ je počet dosud provedených tahů. Hranice je prioritní fronta podle f ; opakovaně vybíráme uzel s nejmenším f , testujeme cílovost (při výběru), a rozvíjíme. S přípustnou (resp. konzistentní) heuristikou A^* vrátí řešení s nejmenším počtem tahů.

(3) Heuristika a její vlastnosti. Dvě klasické přípustné heuristiky:

- h_1 = počet kostek na špatném místě (vyjma volného pole). Pro náš počáteční stav jsou správně jen kostky 3 a 7, ostatních šest je mimo: $h_1 = 6$.
- h_2 = Manhattanská (taxikářská) vzdálenost = součet vzdáleností kostek od jejich cílových pozic. Sečteno přes kostky 1 až 8: $1 + 1 + 0 + 2 + 2 + 1 + 0 + 2 = 9$.

Obě jsou přípustné: žádný tah neopraví víc než jednu kostku ani neposune kostku o víc než jedno pole, takže ani jedna heuristika skutečnou cenu nenadhodnotí. Obě jsou navíc konzistentní (každý tah posune jednu kostku o 1, takže h klesne nejvýš o 1 = cena kroku, což je trojúhelníková nerovnost). h_2 dominuje h_1 (vždy $h_2 \geq h_1$), je tedy lepší.

Záruka optimality. Ano. Protože jsou heuristiky konzistentní, je A^* optimální v grafové i stromové variantě (u 8-puzzle se pro vyhnutí cyklům typicky volí grafová verze s closed listem; ta vyžaduje právě konzistenci). Otázka přitom nežadá nalézt konkrétní řešení – stačí úlohu zformalizovat, zvolit optimální algoritmus a rozebrat vlastnosti heuristiky.

i Poznámka. [Řešitelnost instance – nad rámec zadání] Řešitelnost konkrétní instance 8-puzzle se řídí paritou počtu inverzí (dvojic kostek v obráceném pořadí, volné pole se nepočítá): stav je řešitelný do standardního cíle právě tehdy, má-li sudý počet inverzí. Počáteční stav výše má 11 inverzí (liché), cíl 0 (sudé) – parita se neshoduje, takže tento konkrétní stav **není** do daného cíle řešitelný (ověřeno i vyčerpávajícím prohledáním: ze startu je dosažitelných všech $9!/2 = 181\,440$ stavů jeho paritní třídy a cíl mezi nimi není). A^* by tedy po prohledání konečného prostoru správně ohlásil *neúspěch* – slouží tak i jako rozhodovací procedura řešitelnosti. Tvrzení typu „optimum má N tahů“ by tu proto bylo chybné.

2	8	3
1	6	4
7		5

počáteční stav

1	2	3
4	5	6
7	8	

cílový stav

8-puzzle: počáteční stav $\begin{bmatrix} 2 & 8 & 3 \\ 1 & 6 & 4 \\ 7 & \cdot & 5 \end{bmatrix}$ a cílový stav $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & \cdot \end{bmatrix}$ (tečka = volné pole). Hodnoty heuristik v počátečním stavu: $h_1 = 6$ (špatně umístěné kostky 2, 8, 1, 6, 4, 5), $h_2 = 9$ (Manhattan, oba odhady přípustné i konzistentní).

V příklady/ je u této otázky původní oříznutý obrázek z PDF a oficiální náčrt řešení.

2 Splňování podmínek – CSP (hranová konzistence AC-3, algoritmus MAC)

V předchozí sekci jsme stav chápali *atomicky* – jako nedělitelný „černý box“, o jehož vnitřní stavbě prohledávací algoritmus nic nevěděl. *Splňování podmínek* (Constraint Satisfaction Problem, CSP) je naopak *faktorovaná* reprezentace: stav je rozložen na množinu *proměnných*, každá s vlastní *doménou* hodnot, a řešení je takové přiřazení hodnot proměnným, které splňuje zadané *podmínky*. Tato struktura má dvě výhody. Za prvé řadu úloh (N-dam, Sudoku, rozvrhování, obarvení mapy) lze popsat v jednotném formalismu a řešit obecným *řešičem*. Za druhé – a to je klíčové – nám podmínky umožní *aktivně prořezávat* domény ještě před tím, než cokoliv přiřadíme: hodnotu, která nemůže být součástí žádného řešení, vyřadíme z domény (*inference*), čímž dramaticky zmenšíme prohledávaný prostor.

2.1 Definice CSP

Definice – Problém splňování podmínek (CSP)

CSP je dán trojicí (X, D, C) :

- **proměnné** $X = \{X_1, \dots, X_n\}$ – popisují hledané rysy stavu světa (např. pozice dam na šachovnici);
- **domény** $D = \{D_1, \dots, D_n\}$ – D_i je *konečná* množina možných hodnot proměnné X_i (různé proměnné mohou mít různé domény);
- **podmínky** $C = \{C_1, \dots, C_m\}$ – každá podmínka je *relace* nad nějakou podmnožinou proměnných, tj. podmnožina kartézského součinu jejich domén. Podmínku lze zadat *extenzionálně* (výčtem n -tic, které ji splňují) nebo *formulí* (např. $row_A \neq row_B$). Počet proměnných v podmínce je její *arita*.

Ohodnocení je přiřazení hodnot proměnným; je *úplné*, má-li hodnotu každá proměnná, a *konzistentní*, splňuje-li všechny podmínky. (*Přípustným*) *řešením* CSP je úplné a konzistentní ohodnocení.

Definice – Binární CSP

CSP je **binární**, pokud má každá podmínka aritu nejvýše 2 (váže nanejvýš dvě proměnné). Binární CSP přirozeně reprezentujeme *sítí podmínek* (constraint network): graf, jehož vrcholy jsou proměnné a každá binární podmínka mezi X_i a X_j je *hrana* (oblouk, „arc“) (X_i, X_j) .

 **Intuice.** Na binární CSP se omezuje hlavně proto, že na něm krásně funguje pojem hranové konzistence (a že každý CSP lze na binární převést). Síť podmínek dává geometrickou představu: techniky inference si pak představíme jako *šíření* omezení po hranách sítě – zúžím-li doménu jedné proměnné, „informace“ o tom poteče po hranách k sousedům, kteří mohou zúžit svoje domény, a tak dál.

i Poznámka. [Vstup a výstup] Pohledem rozhodovacího problému: *vstupem* CSP je trojice (X, D, C) , *výstupem* je buď konkrétní řešení (úplné konzistentní ohodnocení), nebo informace, že žádné neexistuje. Rozhodnout, zda obecný CSP řešení má, je **NP-úplný** problém (proto se vyplatí inference, která prostor prořezává).

2.2 Modelování (jak úlohu zformulovat jako CSP)

Prvním krokem je vždy *modelování* – volba proměnných, domén a podmínek. Stejnou úlohu lze obvykle namodelovat víc způsoby a na volbě modelu hodně záleží.

Příklad – N dam jako CSP

Hledáme rozmístění N dam na šachovnici $N \times N$ tak, aby se žádné dvě neohrožovaly (nesmí být ve stejném řádku, sloupci ani diagonále). **Klíčové pozorování modelu:** každou dámu předem přiřadíme do *vlastního sloupce*, takže hledáme jen *řádek*. *Proměnné:* $r_1, \dots, r_N$ (řádek dámy v i -tém sloupci). *Domény:* $D_i = \{1, \dots, N\}$. *Podmínky:* pro každé $i \neq j$

$$r_i \neq r_j \quad (\text{různé řádky}) \quad \text{a} \quad |i - j| \neq |r_i - r_j| \quad (\text{různé diagonály}).$$

Sloupce jsou ošetřeny už samotnou volbou modelu (jedna dáma na sloupec). Prostor se tím zmenší z $(N \cdot N)^N$ (libovolné políčko každé dámy) na N^N , navíc je to binární CSP.

2.3 Řešení prohledáváním s navracením (backtracking)

Nejjednodušší způsob řešení CSP je *prohledávání s navracením* (backtracking). Je to hloubkové prohledávání nad částečnými ohodnoceními: vybereme dosud nepřirazenou proměnnou, zkusíme jí přiřadit hodnotu, ověříme podmínky vůči už přiřazeným proměnným, a je-li ohodnocení konzistentní, pokračujeme na další proměnnou. Narazíme-li na spor (žádná hodnota nejde přiřadit), *navrátíme se* (backtrack) k předchozí proměnné a zkusíme jinou hodnotu.

Algoritmus – Prohledávání s navracením (BACKTRACK)

```
BACKTRACK(csp, ohodnocení):
  if ohodnocení je úplné: return ohodnocení
  X ← vyber nepřirazenou proměnnou
  for každou hodnotu v z DX v nějakém pořadí:
    if v je konzistentní s ohodnocením:
      přidej {X = v} do ohodnocení
      inference ← INFERENCE(csp, X, v)
      if inference ≠ selhání:
        výsledek ← BACKTRACK(csp, ohodnocení)
        if výsledek ≠ selhání: return výsledek
      vrať zpět {X = v} a změny inference
  return selhání
```

> heuristika pořadí: dom / deg
 > heuristika pořadí hodnot
 > vůči už přiřazeným proměnným
 > forward checking / udržení AC; volitelné
 > backtrack

💡 Intuice. Holý backtracking je často *neefektivní*: spor odhalí až ve chvíli, kdy se ke sporné proměnné dostane a zkusí jí přiřadit hodnotu, i když už dávno bylo „jasné“, že volba dál nahoře nemůže vést k řešení. Proto ho posílíme *inferencí* ve volání INFERENCE – po každém přiřazení proaktivně vyřadíme z domén nepřirazených proměnných hodnoty, které jsou s nově přiřazenou hodnotou v rozporu. Tím spor odhalíme dříve (případně už při přiřazení) a ořežeme celé podstromy.

2.4 Konzistenční techniky: hranová konzistence

Místo pouhé *kontroly* podmínek při přiřazování je můžeme využít *aktivně* k prořezávání domén. Motivační příklad: máme $A \in \{3, \dots, 7\}$, $B \in \{1, \dots, 5\}$ a podmínku $A < B$. Hodnota $A = 7$ nemůže být v žádném

řešení, protože v doméně B není žádná hodnota větší než 7; podobně $A \in \{5, 6, 7\}$ a $B \in \{1, 2, 3\}$ jsou neudržitelné. Zúžením domén dostaneme $A \in \{3, 4\}$, $B \in \{4, 5\}$ – a to vše *bez jediného přiřazení*. Tato úvaha vede k pojmu hranové konzistence.

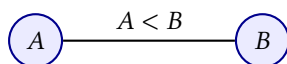
Definice – Hranová konzistence (arc consistency)

Mějme binární CSP. Hrana (oblouk) (X_i, X_j) je **hranově konzistentní**, pokud pro *každou* hodnotu $x \in D_i$ existuje hodnota $y \in D_j$ taková, že dvojice (x, y) splňuje (binární) podmínku mezi X_i a X_j . Hodnotě y se říká *opora* (support) hodnoty x . CSP je **hranově konzistentní**, je-li hranově konzistentní každá jeho hrana v *obou* směrech.

i Poznámka. [Směrovost] Hranová konzistence je *směrová*: konzistence (X_i, X_j) nezaručuje konzistenci (X_j, X_i) . V příkladu výše je po zúžení $A \in \{3, 4\}$, $B \in \{4, 5\}$ hrana (A, B) konzistentní (pro každé a existuje větší b), ale i (B, A) musíme ověřit zvlášť (pro každé b existuje menší a). Proto „CSP je AC“ znamená konzistenci *všech hran v obou směrech*.

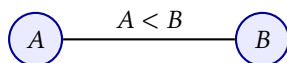
před AC: žádná hrana není konzistentní

$$D_A = \{3, 4, 5, 6, 7\} \quad D_B = \{1, 2, 3, 4, 5\}$$



po AC: (A, B) i (B, A) konzistentní

$$D_A = \{3, 4\} \quad D_B = \{4, 5\}$$



Síť podmínek (constraint graph) pro jedinou binární podmínku $A < B$. Uzly jsou proměnné, hrana je podmínka. *Před* hranovou konzistencí (nahore) má A hodnoty bez opory (5, 6, 7 – v D_B není nic většího) a B má hodnoty bez opory (1, 2, 3). *Po* prořezání (dole) zůstane $D_A = \{3, 4\}$, $D_B = \{4, 5\}$. Pozor: hranová konzistence *nezaručuje*, že každá zbylá kombinace je řešení – např. $A = 4, B = 4$ podmínku $A < B$ neplní.

2.5 Algoritmus AC-3

Pro *nastolení* hranové konzistence v celé síti slouží algoritmus **AC-3**. Drží frontu hran, které je třeba zkontrolovat (zpočátku všechny hrany). Z fronty vždy vybere hranu (X_i, X_j) a zavolá podproceduru $\text{REVIZE}(X_i, X_j)$, která z domény D_i vyřadí všechny hodnoty bez opory v D_j . Pokud se D_i *zúžila*, mohly tím přijít o oporu hodnoty v doménách *sousedů* proměnné X_i (těch, kteří na X_i závisí podmínkou). Proto do fronty znovu zařadíme všechny hrany *vedoucí do* X_i , tj. (X_k, X_i) pro každého souseda $X_k \neq X_j$. Algoritmus končí, když je fronta prázdná (síť je hranově konzistentní), nebo když nějaká doména zůstane prázdná (CSP nemá řešení).

Algoritmus – Hranová konzistence: AC-3

```
AC-3(csp):
    fronta ← všechny hrany  $(X_i, X_j)$  sítě podmínek
    while fronta  $\neq \emptyset$ :
         $(X_i, X_j)$  ← odeber hranu z fronty
        if REVIZE(csp,  $X_i, X_j$ ):
            if  $|D_i| = 0$ : return false
            for každého souseda  $X_k$  proměnné  $X_i$  kromě  $X_j$ :
                přidej hranu  $(X_k, X_i)$  do fronty
```

▷ vrátí **false** při nekonzistenci, jinak **true**

▷ doména D_i se zúžila

▷ prázdná doména $\Rightarrow$ bez řešení

▷ hrany vedoucí DO změněné X_i

```

return true
REVIZE(csp,  $X_i$ ,  $X_j$ ):
    revidováno  $\leftarrow$  false
    for každou hodnotu  $x \in D_i$ :
        if žádné  $y \in D_j$  nesplní s  $x$  podmínku mezi  $X_i, X_j$ :
            vyřaď  $x$  z  $D_i$ ; revidováno  $\leftarrow$  true
    return revidováno

```

▷ vrátí **true**, právě když zúžila D_i

▷ x nemá oporu

i Poznámka. [Složitost a varianty] Časová složitost AC-3 je $O(ed^3)$, kde e je počet podmínek a d velikost největší domény: každou z $O(e)$ hran můžeme do fronty vložit až d -krát a jedna REVIZE stojí $O(d^2)$. Není to optimální – zapamatováním výsledků kontrol (algoritmy AC-4, AC-3.1, AC-2001) lze dosáhnout $O(ed^2)$. Znalost *sémantiky* podmínky kontrolu dále zrychlí (pro $X < Y$ stačí porovnat krajní hodnoty).

★ Věta – Co hranová konzistence (ne)zaručuje

Hranová konzistence je *lokální* vlastnost: zaručuje konzistenci po dvojicích, ne globálně. Proto:

- Hranově konzistentní CSP **nemusí být splnitelný** (může mít prázdnou množinu řešení).
- Pokud nějaká doména po AC-3 *zbude prázdná*, CSP **řešení nemá** (to je jediná jistá implikace o (ne)existenci řešení, kterou AC-3 dává).
- CSP, který *není* hranově konzistentní, **může** mít řešení (nekonzistence se týká jednotlivých hodnot, ne nutně celého problému).

💡 Intuice. Proč hranová konzistence nestačí? Zachycuje jen interakci *dvou* proměnných najednou. Klasický protipříklad: tři proměnné X_1, X_2, X_3 s doménami $\{0, 1\}$ a podmínkami $X_i \neq X_j$ pro každou dvojici. Každá hrana je hranově konzistentní – pro 0 je oporou 1 a naopak – a přesto problém *nemá řešení*: tři proměnné nelze obarvit dvěma hodnotami tak, aby byly po dvojicích různé. AC-3 tu žádnou hodnotu nevyřadí. Globální spor odhalí až prohledávání (nebo silnější, ale dražší k -konzistence).

i Poznámka. [k -konzistence – jen pro kontext] Hranová konzistence je speciální případ obecnější k -konzistence (AC = 2-konzistence; cestová/path konzistence = 3-konzistence): pro každé konzistentní ohodnocení $k - 1$ proměnných existuje hodnota k -té proměnné, která ho rozšíří. Je-li CSP i -konzistentní pro všechna i , lze ho vyřešit *bez navracení*. Vyšší konzistence ale stojí čas *exponenciálně* v k , takže se v praxi místo ní používají *globální podmínky* (např. *all_different* s efektivním prořezáváním přes párování v bipartitním grafu). Požadavky okruhu jmenují jen hranovou konzistenci a AC-3; k -konzistence tu slouží jako kontext.

2.6 Inference v průběhu prohledávání: FC, look-ahead a MAC

Konzistenční techniky teprve naplno vyniknou *uvnitř* prohledávání, jako obsah procedury INFERENCE. Po každém přiřazení proměnné z domén nepřirazených proměnných odstraníme hodnoty, které se s přiřazením neslučují. Liší se jen *rozsah* prořezávání.

📖 Definice – Forward checking, look-ahead / MAC

- **Kontrola dopředu** (forward checking, FC): po přiřazení $X = v$ projdeme jen podmínky obsahující *právě přiřazenou* proměnnou X a z domén jejich dosud nepřirazených sousedů vyřadíme hodnoty neslučitelné s v . Selže-li tím nějaká doména na prázdnou, hned se navrátíme.
- **Pohled dopředu** (look-ahead) / **udržení hranové konzistence** (*Maintaining Arc Consistency*, MAC): po přiřazení $X = v$ obnovíme *plnou hranovou konzistenci* – spustíme AC-3 (s frontou inicializovanou hranami (X_k, X) od nepřirazených sousedů X_k k X) a necháme prořezání šířit po síti, dokud se mění nějaká doména. Je to silnější technika než FC: kontroluje *všechny* podmínky

zasažené šířením, ne jen ty s X , a tak vyřadí víc nekonzistentních hodnot.

💡 Intuice. Rozdíl FC vs. MAC: *forward checking* se dívá jen „o krok dál“ (sousedé přiřazené proměnné), *look-ahead/MAC* nechá efekt zúžení *propagovat dál do sítě* – zúžím-li doménu souseda, znovu zkontroluji i jeho sousedy atd. MAC tedy odhalí spor dřív a prořeže víc, za cenu vyšší práce na uzlu. Bartákovo srovnání na N-damách: holý backtracking potřeboval 19 pokusů, forward checking 3, a look-ahead jen 2.

📌 Poznámka. [MAC = AC-3 inkrementálně] MAC je přesně AC-3 spouštěné *inkrementálně* během prohledávání. Na začátku problém *jednou* zhranověkonzistentníme (AC-3 na vstupu). Po každém přiřazení $X = v$ je doména X jednoprvková a do fronty AC-3 vložíme jen hrany (X_k, X) od sousedů; AC-3 pak prořeže a šíří. Při navracení se vyřazené hodnoty vrátí zpět. Volání INFERENCE v pseudokódu BACKTRACK je tedy „spuštění AC-3 z hran k právě přiřazené proměnné“.

2.7 Heuristiky pořadí proměnných a hodnot

Heuristiky pořadí požadavky přímo nejmenují, ale patří k efektivnímu řešiči a stojí za to je znát. Backtracking instancuje proměnné a zkouší hodnoty v *nějakém* pořadí; dobré pořadí dramaticky zkrátí prohledávání. Vede nás princip „spor odhalit co nejdřív“:

- **Pořadí proměnných – princip „fail-first“:** ber jako další tu proměnnou, jejíž přiřazení nejspíš povede ke sporu (a tedy spor odhalíme brzy, blízko kořene).
 - **dom** (minimum remaining values): proměnná s *nejmenší doménou* jako první (nejméně možností $\Rightarrow$ nejdřív „dojdou“ hodnoty).
 - **deg** (degree): proměnná v *nejvíce podmínkách* jako první (typicky rozhodne při rovnosti **dom**).
- **Pořadí hodnot – princip „succeed-first“:** naopak zkus nejdřív hodnotu, která nejspíš *patří do řešení*. Heuristika *least constraining value*: hodnota, která *nejméně omezuje* ostatní proměnné (ponechá nejvíc flexibility). Hledání skutečně nejlepší hodnoty bývá drahé, takže se používají doménově specifické heuristiky.

💡 Intuice. Proč u proměnných „fail-first“ (nejhůř první) a u hodnot „succeed-first“ (nejlíp první)? Proměnnou musíme *tak jako tak* přiřadit – chceme tedy nejdřív tu nejproblematictější, ať případný spor přijde co nejdřív a ořízne velký podstrom. U hodnoty naopak chceme rychle najít *jedno* řešení, takže zkoušíme nejnadějnější hodnotu jako první.

2.8 Řešené příklady SZZ

Příklad – SZZ podzim 2023 č. 28: CSP, backtracking, konzistence, model Sudoku

(Reálná otázka SZZ: podzim 2023, č. 28 – Problém splňování podmínek (CSP, backtracking, arc consistency, forward checking, look-ahead, Sudoku))

Zadání.

Definujte problém splňování podmínek (CSP). Popište prohledávání s navracením (backtracking) jako způsob řešení CSP. Vysvětlíte pojmy „hranová konzistence“ (arc consistency), „kontrola dopředu“ (forward checking) a „pohled dopředu“ (look-ahead). Pomocí CSP namodelujte Sudoku (umístit čísla 1, ..., 9 do mřížky 9×9 tak, aby se v žádném řádku, sloupci ani 3×3 boxu žádná cifra neopakovala; některá čísla jsou předem zadána).

Řešení.

(1) Definice CSP. CSP je trojice (X, D, C) : konečná množina *proměnných* X , ke každé proměnné *doména* (konečná množina jejích možných hodnot; různé proměnné mohou mít různé domény) a konečná množina *podmínek* C . Podmínka je *relace* nad podmnožinou proměnných, tj. podmnožina

kartézského součinu jejich domén (zadaná výčtem n -tic nebo formulí). Řešením je úplné a konzistentní ohodnocení – každá proměnná má hodnotu a všechny podmínky jsou splněny.

(2) Backtracking. Hlubkové prohledávání nad částečnými ohodnoceními: vybereme nepřirazenou proměnnou, přiřadíme jí hodnotu a ověříme podmínky vůči už přiřazeným proměnným. Je-li ohodnocení konzistentní, rekurzivně pokračujeme na další proměnnou; nastane-li spor (žádná hodnota nejde přiřadit), *navrátíme se* k předchozí proměnné a zkusíme jinou hodnotu. Skončíme buď úplným konzistentním ohodnocením (řešení), nebo vyčerpáním všech možností (řešení neexistuje).

(3) Konzistenční techniky.

- *Hranová konzistence (arc consistency).* Hrana (X_i, X_j) je konzistentní, pokud pro každou hodnotu $x \in D_i$ existuje hodnota $y \in D_j$ splňující s x podmínku mezi X_i, X_j . Hodnoty bez takové opory z D_j vyřadíme. Jde o *předzpracování/inferenci*: prořeže domény nezávisle na konkrétním přiřazení.
- *Kontrola dopředu (forward checking).* Po přiřazení proměnné X projdeme *jen* podmínky obsahující X a z domén jejich nepřirazených sousedů vyřadíme hodnoty neslučitelné s přiřazením. Spor (prázdnou doménu) tak odhalíme dříve než holý backtracking.
- *Pohled dopředu (look-ahead).* Silnější než FC: po přiřazení obnovíme *plnou hranovou konzistenci* (spustíme AC-3), takže prořezání *šíříme* přes všechny zasažené podmínky, dokud se mění nějaká doména – ne jen přes podmínky s právě přiřazenou proměnnou. Vyřadí proto víc nekonzistentních hodnot.

(4) Model Sudoku.

- *Proměnné:* jedno políčko = jedna proměnná, $X_{r,c}$ pro řádek r a sloupec c (81 proměnných).
- *Domény:* $D_{r,c} = \{1, \dots, 9\}$.
- *Podmínky:* na každý *řádek*, každý *sloupec* a každý 3×3 *box* podmínka *all_different* (všech 9 proměnných různých) – ekvivalentně nerovnosti po dvojicích $X_{r,c} \neq X_{r',c'}$ pro políčka ve stejném řádku / sloupci / boxu.
- *Zadaná čísla:* buď zúžíme doménu políčka na jednoprvkovou ($D_{r,c} = \{k\}$), nebo přidáme podmínku $X_{r,c} = k$.

Sudoku se pak řeší backtrackingem s inferencí (FC, resp. look-ahead/MAC); často ho silné prořezávání vyřeší skoro bez navracení.

i Poznámka. Sudoku je hezká ukázka, že hranová konzistence *nestačí* ke globální konzistenci: existují rozpracovaná Sudoku, která jsou hranově konzistentní, a přesto vyžadují prohledávání (nebo silnější inferenci) k dořešení.

Příklad – SZZ jaro 2026 č. 27: binární CSP, hranová konzistence, běh AC-3

(Reálná otázka SZZ: jaro 2026, č. 27 – CSP (binární CSP, hranová konzistence, běh algoritmu AC-3))

Zadání.

(1) Definujte binární CSP (vstup a výstup) a vysvětlete pojem „hranová konzistence“. (2) Je-li problém hranově konzistentní, je zaručeně splnitelný? Může být splnitelný problém, který hranově konzistentní *není*? Zdůvodněte nebo dejte protipříklady. (3) Aplikujte AC-3 na CSP s celočíselnými proměnnými A, B, C, D , doménami $D_A = \{1\}$, $D_B = D_C = D_D = \{1, 2, 3\}$ a podmínkami $C_1 : C + D = 4$, $C_2 : A + B \leq 3$, $C_3 : B + C \geq 4$. Uvedte postup.

Řešení.

(1) Definice a hranová konzistence. Binární CSP je trojice (X, D, C) , kde každá podmínka váže nejvýše dvě proměnné. *Vstup:* proměnné, jejich domény a podmínky. *Výstup:* úplné konzistentní ohodnocení (řešení), nebo informace, že žádné neexistuje. Hrana (X_i, X_j) je *hranově konzistentní*, pokud pro každou hodnotu $x \in D_i$ existuje $y \in D_j$ tak, že (x, y) splňuje podmínku mezi X_i, X_j ; CSP je hranově konzistentní, je-li taková každá hrana v obou směrech.

(2) Co AC zaručuje. Hranově konzistentní problém *nemusí být splnitelný*. Protipříklad: X_1, X_2, X_3

s doménami $\{0, 1\}$ a podmínkami $X_i \neq X_j$ pro každou dvojici. Každá hrana je konzistentní (oporuou 0 je 1 a naopak), takže AC-3 nic nevyřadí, a přesto problém *nemá řešení* – tři proměnné nelze obarvit dvěma hodnotami po dvojicích různě. *Hranově nekonzistentní problém naopak může být splnitelný*: nekonzistence se týká jednotlivých hodnot, ne celého problému – ostatně i zadaný CSP z bodu (3) hranově konzistentní *není* (např. při $D_A = \{1\}$ podmínka $C_2 : A + B \leq 3$ vylučuje $B = 3$, takže hrana (B, A) konzistentní není), a přitom řešení má (viz dále). *Jediná jistota*: zbude-li po AC-3 prázdná doména, řešení *neexistuje*.

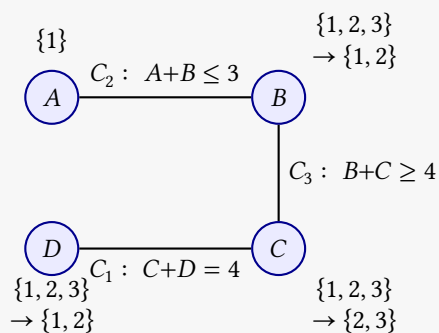
(3) Běh AC-3. Síť podmínek má hrany C_1 mezi C, D ; C_2 mezi A, B ; C_3 mezi B, C (viz obrázek). Fronta startuje se všemi hranami v obou směrech. Pro každou hranu REVIZE vyřadí z „první“ proměnné hodnoty bez opory; po zúžení vrátíme do fronty hrany vedoucí do zúžené proměnné. Pečlivý průchod (proměnná A s jednoprvkovou doménou je „pevná“ a tlačí na sousedy):

1. C_2 **zúží** B **podle** A . $A = 1$ a $A + B \leq 3$ dává $B \leq 2$, takže $B = 3$ nemá v D_A oporu: REVIZE(B, A) vyřadí 3, $D_B = \{1, 2\}$. (Hrana (A, B) konzistentní: $a = 1$ má oporu $b \leq 2 - D_A$ se nemění.)
2. C_3 **zúží** C **podle** B . $B \in \{1, 2\}$ a $B + C \geq 4$ dává $C \geq 4 - B \geq 4 - 2 = 2$, takže $C = 1$ nemá oporu: REVIZE(C, B) vyřadí 1, $D_C = \{2, 3\}$. (Hrana (B, C) : pro $b \in \{1, 2\}$ existuje $c = 3$ s $b + c \geq 4 - D_B$ beze změny.)
3. C **se změnila** $\Rightarrow$ **překontroluj hrany do** C *kromě* té, kterou jsme právě revidovali (pravidlo „kromě X_j “, tj. C_1 od D (nikoli C_3 od B). Hrana C_1 navíc zúží D podle C : $C + D = 4$ a $C \in \{2, 3\}$ dává $D = 4 - C \in \{1, 2\}$, takže $D = 3$ nemá oporu: REVIZE(D, C) vyřadí 3, $D_D = \{1, 2\}$.
4. D **se změnila** $\Rightarrow$ **překontroluj hrany do** D : C_1 od C . REVIZE(C, D): pro $c = 2$ je opora $d = 2$, pro $c = 3$ je opora $d = 1$ – obě v $D_D = \{1, 2\}$, takže D_C se nemění. Žádná další doména se už nemění, fronta se vyprázdní.

Výsledné hranově konzistentní domény:

$$D_A = \{1\}, \quad D_B = \{1, 2\}, \quad D_C = \{2, 3\}, \quad D_D = \{1, 2\}.$$

Problém je nyní splnitelný (řešení např. $A=1, B=2, C=2, D=2$ nebo $A=1, B=2, C=3, D=1$ – obě splní $C + D = 4, A + B \leq 3, B + C \geq 4$). To je v souladu s bodem (2): AC-3 jen prořezal domény; splnitelnost potvrdí teprve nalezení úplného ohodnocení.



Síť podmínek úlohy: proměnné A, B, C, D , hrany jsou binární podmínky $C_2(A, B), C_3(B, C), C_1(C, D)$. U každého uzlu je doména před AC-3 a po něm. Prořezání se šíří od „pevné“ proměnné $A = 1$: $C_2 \Rightarrow B$, pak $C_3 \Rightarrow C$, pak $C_1 \Rightarrow D$.

Příklad – SZZ léto 2022 č. 9: N-královen jako prohledávání a CSP

(Reálná otázka SZZ: léto 2022, č. 9 – N-královen)

Zadání.

Problém N-královen (rozmístit N královen na šachovnici $N \times N$ tak, aby se neohrožovaly – žádné dvě na stejném sloupci, řádku nebo diagonále). (a) Popište vhodnou abstrakci pro prohledávací algoritmus (stavy, přechodový model, cílovou podmínku). (b) Vyberte vhodný informovaný či neinformovaný

prohledávací algoritmus a zdůvodněte. (c) Ukažte, jak by se problém modeloval a řešil jako CSP.

Řešení.

(a) Abstrakce pro prohledávání. Stav = částečné rozmístění královen, např. jedna v každém z prvních k sloupců tak, že se vzájemně neohrožují. Počáteční stav = prázdná šachovnice. Přejchodový model = přidání královny do dalšího $(k+1.)$ sloupce na řádek, kde neohrožuje žádnou již položenou královnu. Cílová podmínka = rozmístěno N královen (všechny sloupce zaplněny). Tím, že na každý sloupec přijde právě jedna královna, se prohledávací strom drasticky zúží (místo N^2 políček volíme jen řádek v jednom sloupci).

(b) Volba algoritmu. Neinformované prohledávání do hloubky (DFS) s navracením (backtracking): řešení leží v hloubce N , strom je konečný a DFS je paměťově nenáročný. Informované metody (A^* , heuristiky) se obvykle nevyplatí, protože jde o splnění podmínek (constraint satisfaction), ne o hledání nejkratší cesty – všechna řešení jsou ve stejné hloubce a cena cesty zde nehraje roli.

(c) Model jako CSP. Proměnné $Q_1, \dots, Q_N$ (po jedné na sloupec), doména $D_i = \{1, \dots, N\}$ (řádek královny v i -tém sloupci). Sloupcová podmínka je už zakódovaná do volby proměnných. Pro všechna $i < j$:

$$Q_i \neq Q_j \quad (\text{různé řádky}), \quad |Q_i - Q_j| \neq |i - j| \quad (\text{ne na společné diagonále}).$$

Řeší se backtrackingem se šířením podmínek (forward checking / AC-3) a heuristikou výběru proměnné MRV. Jde o tentýž rozklad „právě jedna na sloupec“ jako v příkladu *N dam jako CSP* výše.

Originální zadání těchto otázek (a případné oficiální náčrty řešení) jsou ve složce příklady/.

3 Logické uvažování (DPLL, dopředné a zpětné řetězení, rezoluce)

Druhým velkým paradigmatem AI je *logické uvažování*. Místo prohledávání stavů popíšeme svět množinou logických tvrzení (*znalostní bázi*) a ptáme se, co z ní logicky plyne. *Agent založený na znalostech* drží znalostní bázi KB (množinu vět nějakého jazyka), kterou rozšiřuje operací TELL (přidej nové pozorování či poznatek) a dotazuje se operací Ask (plyne z KB daná věta?). Logická *inference* mu dovolí odvodit i to, co není přímo pozorovatelné. V tomto okruhu zůstáváme u *výrokové logiky* a díváme se na ni *aplikačně* – jako na nástroj pro algoritmy (rozhodnutí splnitelnosti, dokazování důsledku). Syntaxi a sémantiku logiky systematicky buduje společný okruh M6; zde stačí pracovní minimum.

3.1 Model, splnitelnost, důsledek

Definice – Model, splnitelnost, důsledek

Model M je ohodnocení (přiřazení pravdivostních hodnot) všem výrokovým proměnným. M je *modelem* věty α , pokud je α v M pravdivá; množinu všech modelů α značíme $M(\alpha)$.

- Věta je **splnitelná** (satisfiable), je-li pravdivá v *nějakém* modelu (např. $A \vee B$).
- Věta je **nesplnitelná** (unsatisfiable), není-li pravdivá v *žádném* modelu (např. $A \wedge \neg A$).
- α (**sémanticky**) **vyplývá** z KB (entails), píšeme $KB \models \alpha$, je-li α pravdivá v každém modelu KB , tj. $M(KB) \subseteq M(\alpha)$.

 **Intuice.** $KB \models \alpha$ znamená „kdykoli platí všechno v KB , platí i α “ – α je důsledek, ne nová informace. Ve světě Wumpa (klasický příklad z učebnice: agent prochází mřížku jeskyně, kde *závan* v poli signalizuje bezednou jámu v některém sousedním) pozorujeme „v poli $[1, 1]$ není závan“ a chceme zjistit, zda je sousední pole bezpečné. Vyjmenujeme modely shodné s pozorováním a ověříme, zda ve *všech* je dané pole bez jámy. Je-li tomu tak, bezpečnost *plyne* z KB ; existuje-li model KB , v němž tam jáma je, neplyne a jistotu nemáme.

Klíčový převod, na němž stojí algoritmy dokazování, vyjadřuje větu (důsledek) jazykem (ne)splnitelnosti:

★ Věta – Důsledek jako nespłnitelnost

Pro každou KB a větu α platí

$$KB \models \alpha \iff (KB \wedge \neg\alpha) \text{ je nespłnitelná.}$$

Intuice. Toto je výroková obdoba „důkazu sporem“. Pokud z KB plyne α , pak nemůže současně platit KB a $\neg\alpha$, takže konjunkce $KB \wedge \neg\alpha$ nemá model – je nespłnitelná. A naopak. Díky tomu se každá otázka „plyne α ?“ převede na otázku splnitelnosti, kterou řeší DPLL (hledá model) i rezoluce (odvozuje spor). Proto se v AI místo důsledku rovnou rozhoduje splnitelnost.

3.2 Konjunktivní a disjunktivní normální forma

Algoritmy chtějí formuli v jednotném tvaru. *Literál* je výroková proměnná nebo její negace (např. A , $\neg B$); *klauzule* je disjunkce literálů (např. $A \vee \neg B$); *elementární konjunkce* je konjunkce literálů.

☰ Definice – CNF a DNF

- **Konjunktivní normální forma (CNF)** je konjunkce klauzulí (AND disjunkcí), tj. tvar $\bigwedge_i \bigvee_j \ell_{ij}$.
Příklad: $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$.
- **Disjunktivní normální forma (DNF)** je disjunkce elementárních konjunkcí (OR konjunkcí), tj. tvar $\bigvee_i \bigwedge_j \ell_{ij}$. Příklad: $(A \wedge B) \vee (\neg A \wedge C)$.

Každá výroková formule je logicky ekvivalentní nějaké formuli v CNF i nějaké v DNF.

Intuice. CNF je tvar pro *dokazování*: formule v CNF je pravdivá, právě když je pravdivá každá klauzule, a klauzule je nepravdivá jen tehdy, jsou-li nepravdivé všechny její literály. Na tom stojí DPLL i rezoluce (rezoluce pracuje výhradně s klauzulemi). DNF je naopak tvar, v němž je splnitelnost triviální (stačí jedna splnitelná elementární konjunkce), zato rozhodnout, zda je formule v DNF tautologií (a tedy i důsledek), je obtížné – proto se pro dokazování normalizuje na CNF. Pozn.: převod do CNF může formuli exponenciálně nafouknout (roz Distribučování $\vee$ přes $\wedge$), takže ani „jen“ převod na CNF není zadarmo.

🔗 Příklad – Převod na CNF z definice ekvivalence

Převedeme $B \Leftrightarrow (P_1 \vee P_2)$ do CNF – typický krok při kódování pravidel světa do klauzulí.

1. *Odstranění ekvivalence* $a \Leftrightarrow b \equiv (a \Rightarrow b) \wedge (b \Rightarrow a)$:

$$(B \Rightarrow (P_1 \vee P_2)) \wedge ((P_1 \vee P_2) \Rightarrow B).$$

2. *Odstranění implikace* $a \Rightarrow b \equiv \neg a \vee b$:

$$(\neg B \vee P_1 \vee P_2) \wedge (\neg(P_1 \vee P_2) \vee B).$$

3. *De Morgan* $\neg(P_1 \vee P_2) \equiv (\neg P_1 \wedge \neg P_2)$ a roznásobení ($\vee$ přes $\wedge$):

$$(\neg B \vee P_1 \vee P_2) \wedge (\neg P_1 \vee B) \wedge (\neg P_2 \vee B).$$

Postup je obecný: (1) odstran $\Leftrightarrow$ a $\Rightarrow$, (2) zatlač negace dovnitř (De Morgan, $\neg\neg a \equiv a$), (3) roz Distribuuji $\vee$ přes $\wedge$. Výsledkem jsou tři klauzule – to je hledaná CNF.

3.3 Algoritmus DPLL

Splnitelnost CNF (problém SAT) lze rozhodnout vyčíslením pravdivostní tabulky (2^n modelů), to je ale neúnosné. **DPLL** (Davis–Putnam–Logemann–Loveland) je chytřejší: prohledávání modelů s backtrackin-

gem, urychlené *inferenci* (stejně spojení prohledávání a inference jako u CSP). Pracuje nad *částečnými* modely a používá tři zrychlení.

Definice – Tři klíčové prvky DPLL

- **Předčasné ukončení** (early termination): klauzule je pravdivá, je-li pravdivý byť jediný její literál; formule je nepravdivá, je-li nepravdivá být jediná klauzule (všechny literály nepravdivé). Lze tak rozhodnout už nad částečným modelem, bez doplnění zbylých proměnných.
- **Čistý literál** (pure symbol): proměnná, která se ve všech (dosud nesplněných) klauzulích vyskytuje se *stejným znaménkem*. Takovou proměnnou bezpečně nastavíme tak, aby tyto literály byly pravdivé – nemůže to uškodit, protože v žádné klauzuli nevystupuje opačně.
- **Jednotková klauzule** (unit clause): klauzule, v níž po dosazení částečného modelu zbývá *jediný* dosud neohodnocený literál. Ten *musí* být pravdivý, jinak je klauzule nepravdivá; proměnnou tedy vynutíme (*jednotková propagace*). Nastavení jednoho literálu může vyrobít další jednotkové klauzule – řetězí se.

Algoritmus – DPLL – rozhodnutí splnitelnosti CNF

```

DPLL-SATISFIABLE?(s) :
    clauses ← klauzule CNF formule s; symbols ← proměnné v s
    return DPLL(closures, symbols, {})                                ▷ prázdný model
DPLL(closures, symbols, model):
    if každá klauzule je v model pravdivá: return true                ▷ model nalezen
    if některá klauzule je v model nepravdivá: return false          ▷ backtrack
    P, hodnota ← Find-Pure-Symbol(symbols, closures, model)          ▷ čistý literál
    if P ≠ null: return DPLL(closures, symbols-P, model ∪ {P=hodnota})
    P, hodnota ← Find-Unit-Clause(closures, model)                  ▷ jednotková klauzule
    if P ≠ null: return DPLL(closures, symbols-P, model ∪ {P=hodnota})
    P ← First(symbols); rest ← Rest(symbols)                         ▷ větvení
    return DPLL(closures, rest, model ∪ {P=true})
    or DPLL(closures, rest, model ∪ {P=false})

```

Pořadí pravidel je podstatné: nejdřív se zkusí ukončit (pravda/spor), pak levná *inference* bez větvení (čisté literály, jednotkové klauzule), a teprve když nic z toho nezabere, se *větví* na zvolené proměnné s možností backtrackingu. DPLL je **korektní a úplný** algoritmus rozhodující splnitelnost CNF: vrátí TRUE právě tehdy, má-li formule model (důkaz neuvádíme). Moderní SAT solvery DPLL rozšiřují o analýzu komponent, heuristiky volby proměnné (stupňová, aktivitní), náhodné restarty, sledované literály a *učení klauzulí* z konfliktů.

3.4 Řešený příklad SZZ: DPLL a perfektní párování

Příklad – SZZ jaro 2025, č. 27: DPLL a perfektní párování

(Reálná otázka SZZ: jaro 2025, č. 27 – DPLL a párování)

Zadání.

Nechť $G = (V, E)$ je neorientovaný graf s n vrcholy; *perfektní párování* je $M \subseteq E$ tak, že každý vrchol je incidentní s právě jednou hranou z M . (1) Navrhněte formuli ve výrokové logice splnitelnou právě tehdy, má-li G perfektní párování, obecně v CNF. (2) Vysvětlete *čistý literál* a *jednotkovou klauzuli* a jejich roli v DPLL. (3) Napište formuli z (1) pro cestu na čtyřech vrcholech a, b, c, d a demonstруйте na ní obě heuristiky.

Řešení.

(1) Kódování do CNF. Pro každou hranu $e \in E$ zavedeme proměnnou m_e s významem „hrana e je v párování“. Pro každý vrchol u s incidentními hranami $e_1, \dots, e_k$ vyjádříme podmínku „právě jedna“:

- *aspoň jedna* (at-least-one): $m_{e_1} \vee m_{e_2} \vee \dots \vee m_{e_k}$ – jedna klauzule;

- *nejvýš jedna* (at-most-one): pro každou dvojici různých incidentních hran $e_i \neq e_j$ klauzule $\neg m_{e_i} \vee \neg m_{e_j}$ – tedy $\binom{k}{2}$ klauzulí.

Konjunkce všech těchto klauzulí přes všechny vrcholy je hledaná CNF: její model přesně odpovídá výběru hran, kde každý vrchol pokrývá právě jedna – tedy perfektnímu párování. (Rozklad „právě jedna = aspoň jedna $\wedge$ nejvýš jedna“ je standardní obrat při kódování úloh do SAT.)

(2) Čistý literál a jednotková klauzule.

- *Jednotková klauzule* obsahuje (po dosazení dosavadního modelu) jediný literál; ten musí být pravdivý, jinak je klauzule nesplněná. DPLL proto danou proměnnou ihned vynutí (jednotková propagace) bez větvení.
- *Čistý literál* je proměnná, jejíž všechny výskyty v dosud nesplněných klauzulích mají stejné znaménko. Nastavíme ji tak, aby tyto literály byly pravdivé; nemůže to nikde uškodit, takže opět bez větvení.

Obě pravidla nahrazují slepé větvení *vynucenou* dedukcí, čímž drasticky ořežou strom prohledávání.

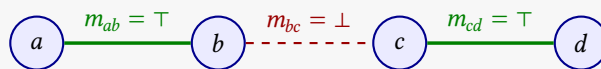
(3) *Cesta* $a-b-c-d$. Hrany jsou ab, bc, cd ; proměnné m_{ab}, m_{bc}, m_{cd} . Incidence: a jen s ab ; b s ab, bc ; c s bc, cd ; d jen s cd . Klauzule po vrcholech:

$$\underbrace{m_{ab}}_a \wedge \underbrace{(m_{ab} \vee m_{bc}) \wedge (\neg m_{ab} \vee \neg m_{bc})}_b \wedge \underbrace{(m_{bc} \vee m_{cd}) \wedge (\neg m_{bc} \vee \neg m_{cd})}_c \wedge \underbrace{m_{cd}}_d.$$

Vrcholy a a d mají jen jednu incidentní hranu, takže jejich „aspoň jedna“ jsou rovnou *jednotkové klauzule* m_{ab} a m_{cd} . Běh DPLL:

1. Jednotková klauzule $m_{ab} \Rightarrow \boxed{m_{ab} = \text{TRUE}}$. Klauzule „nejvýš jedna“ u b , tj. $\neg m_{ab} \vee \neg m_{bc}$, se redukuje na jednotkovou $\neg m_{bc}$.
2. Jednotková klauzule $m_{cd} \Rightarrow \boxed{m_{cd} = \text{TRUE}}$. Klauzule „nejvýš jedna“ u c , tj. $\neg m_{bc} \vee \neg m_{cd}$, se rovněž redukuje na $\neg m_{bc}$.
3. Jednotková klauzule $\neg m_{bc} \Rightarrow \boxed{m_{bc} = \text{FALSE}}$. Zbylé klauzule (např. $m_{bc} \vee m_{cd}$) jsou už splněné díky m_{cd} .

Samotná *jednotková propagace* tedy bez jediného větvení najde model $m_{ab} = m_{cd} = \text{TRUE}$, $m_{bc} = \text{FALSE}$, tj. párování $\boxed{M = \{ab, cd\}}$, které je opravdu perfektní (pokrývá všechny čtyři vrcholy). Lze i poznamenat, že m_{bc} je po fixaci m_{ab}, m_{cd} *čistý literál* (zbude jen jeho negativní výskyt), takže by se na FALSE nastavila i tímto pravidlem. (Numericky ověřeno: jediný model formule odpovídá párování $\{ab, cd\}$.)



Cesta $a-b-c-d$. Vrcholy a a d mají stupeň 1, takže m_{ab} a m_{cd} jsou jednotkové klauzule a vynutí se na $\top$ (zelené, tlusté hrany v párování); klauzule „nejvýš jedna“ u b a c pak vynutí $m_{bc} = \perp$ (červená přerušovaná, mimo párování). Perfektní párování $\{ab, cd\}$ vznikne samotnou jednotkovou propagací.

Originál zadání a oficiální (stručnější) náčrt řešení této otázky jsou ve složce příklady/.

Příklad – SZZ podzim 2022 č. 17: Barvení grafu jako SAT a DPLL

(Reálná otázka SZZ: podzim 2022, č. 17 – Barvení grafu, DPLL)

Zadání.

Nechť $G = (V, E)$ je neorientovaný graf s N vrcholy. (1) Definujte konjunktivní normální formu (CNF) formule výrokové logiky. (2) Navrhněte formuli, která je splnitelná, právě když je G obarvitelný k barvami; popište ji obecně v CNF pro libovolný graf a libovolné $k > 0$. (3) Napište formuli pro úplný graf K_3 a $k = 3$ a demonstруйте na ní algoritmus DPLL.

Řešení.

(1) CNF. Formule je v *konjunktivní normální formě*, je-li to konjunkce klauzulí, kde každá klauzule

je disjunkce literálů a literál je výroková proměnná nebo její negace.

(2) **Kódování obarvitelnosti.** Zavedeme proměnné $x_{v,c}$ pro $v \in V, c \in \{1, \dots, k\}$ s významem „vrchol v má barvu c “. Klausule:

- *Pokrývací* (každý vrchol aspoň jednu barvu): pro každý vrchol v klauzule $(x_{v,1} \vee \dots \vee x_{v,k})$.
- *Různosti* (sousedé různou barvu): pro každou hranu $\{u, v\} \in E$ a každou barvu c klauzule $(\neg x_{u,c} \vee \neg x_{v,c})$.

Konjunkce všech těchto klauzulí je v CNF a je splnitelná právě tehdy, je-li G obarvitelný k barvami. Volitelnou podmínku „nejvýše jedna barva na vrchol“ ($\neg x_{v,c} \vee \neg x_{v,c'}$ pro $c \neq c'$) přidávat netřeba: dostane-li vrchol víc barev, stačí jednu ponechat.

(3) **Formule pro K_3 , $k = 3$ a běh DPLL.** K_3 má vrcholy 1, 2, 3 a hrany $\{1, 2\}, \{1, 3\}, \{2, 3\}$; pro $k = 3$ máme 9 proměnných $x_{v,c}$. Pokrývací klauzule:

$$(x_{1,1} \vee x_{1,2} \vee x_{1,3}), (x_{2,1} \vee x_{2,2} \vee x_{2,3}), (x_{3,1} \vee x_{3,2} \vee x_{3,3}),$$

a pro každou hranu a barvu klauzule různosti, např. pro $\{1, 2\}$: $(\neg x_{1,1} \vee \neg x_{2,1}), (\neg x_{1,2} \vee \neg x_{2,2}), (\neg x_{1,3} \vee \neg x_{2,3})$, obdobně pro $\{1, 3\}, \{2, 3\}$.

Běh DPLL. Rozhodneme $x_{1,1} = \text{TRUE}$. Jednotková propagace přes klauzule různosti $\neg x_{1,1} \vee \neg x_{2,1}$ a $\neg x_{1,1} \vee \neg x_{3,1}$ vynutí $x_{2,1} = x_{3,1} = \text{FALSE}$. Rozhodneme $x_{2,2} = \text{TRUE}$; propagace vynutí $x_{3,2} = \text{FALSE}$ (přes hranu $\{2, 3\}$). Vrcholu 3 pak v jeho pokrývací klauzuli po eliminaci $x_{3,1}, x_{3,2}$ zbude jediný literál $x_{3,3}$ (jednotková klauzule), takže $x_{3,3} = \text{TRUE}$. Splňující ohodnocení $x_{1,1}, x_{2,2}, x_{3,3} = \text{TRUE}$ (ostatní FALSE) odpovídá obarvení vrcholů 1, 2, 3 barvami 1, 2, 3. Roli hraje hlavně jednotková klauzule (vynucené dosazení); backtracking při tomto pořadí rozhodnutí není potřeba.

Originál zadání této otázky je ve složce příklady/ (v PDF bez oficiálního náčrtu řešení; náčrt tam je doplněný).

3.5 Rezoluce

Druhým přístupem (vedle hledání modelu) je *symbolické dokazování*: z klauzulí $KB \wedge \neg \alpha$ odvozovat nové klauzule, dokud nevznikne spor. Stojí na jediném inferenčním pravidle.

Definice – Rezoluční pravidlo

Mají-li dvě klauzule *komplementární* dvojici literálů (P v jedné, $\neg P$ v druhé), lze je *rezolovat* na novou klauzuli (rezolventu), která obsahuje všechny ostatní literály obou klauzulí (bez P a $\neg P$):

$$\frac{(\ell_1 \vee \dots \vee \ell_k) \quad (m_1 \vee \dots \vee m_n)}{\ell_1 \vee \dots \vee \ell_{i-1} \vee \ell_{i+1} \vee \dots \vee \ell_k \vee m_1 \vee \dots \vee m_{j-1} \vee m_{j+1} \vee \dots \vee m_n},$$

kde ℓ_i a m_j jsou komplementární. Rezolventa se přidá ke KB pro další rezolgování. *Prázdná klauzule* $\perp$ (prázdná disjunkce, vždy nepravdivá) značí spor.

Algoritmus – Rezoluce – rozhodnutí $KB \models \alpha$

```

PL-RESOLUTION( $KB, \alpha$ ):
  clauses  $\leftarrow$  klauzule CNF formule  $KB \wedge \neg \alpha$  ▷ důsledek = nesplnitelnost
  new  $\leftarrow \{\}$ 
  while true:
    for každou dvojici klauzulí  $C_i, C_j$  z clauses:
      resolventy  $\leftarrow$  PL-RESOLVE( $C_i, C_j$ ) ▷ všechny rezolventy dvojice
      if resolventy obsahují  $\perp$ : return true ▷ spor  $\Rightarrow KB \models \alpha$ 
      new  $\leftarrow$  new  $\cup$  resolventy
    if new  $\subseteq$  clauses: return false ▷ po všech dvojicích nic nového: pevný bod,  $KB \not\models \alpha$ 
  clauses  $\leftarrow$  clauses  $\cup$  new

```

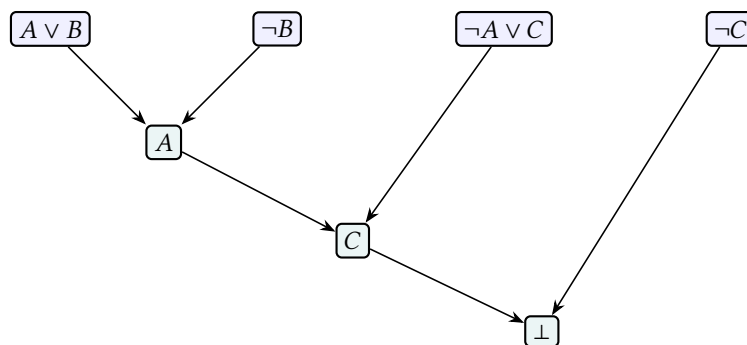
Algoritmus produkuje všechny rezolventy, dokud (a) nevyrobí prázdnou klauzuli – pak je $KB \wedge \neg\alpha$ nesplnitelná, tedy $KB \models \alpha$ – nebo (b) nedosáhne *pevného bodu* (žádná nová klauzule), pak je $KB \wedge \neg\alpha$ splnitelná a $KB \not\models \alpha$. Výroková rezoluce je **korektní a úplná** (vyvracecí úplnost: je-li $KB \wedge \neg\alpha$ nesplnitelná, rezoluce $\perp$ odvodí; bez důkazu).

Příklad – Rezoluční vyvrácení

Ukážeme, že $KB = \{A \vee B, \neg A \vee C, \neg B, \neg C\}$ je nesplnitelná. Rezolvujeme komplementární literály:

$$(A \vee B), (\neg B) \Rightarrow A; \quad A, (\neg A \vee C) \Rightarrow C; \quad C, (\neg C) \Rightarrow \perp.$$

Odvození prázdné klauzule dokazuje nesplnitelnost. Ekvivalentně: pro $KB' = \{A \vee B, \neg A \vee C, \neg B\}$ a $\alpha = C$ je $KB' \wedge \neg\alpha$ právě tato množina klauzulí, takže $KB' \models C$. Ověřeno i vyčíslením: KB nemá žádný model.



Rezoluční vyvrácení (refutace) pro $KB = \{A \vee B, \neg A \vee C, \neg B, \neg C\}$. Modré jsou vstupní klauzule, zelené rezolventy. Komplementární literály se v každém kroku „vyruší“: $B/\neg B$, pak $A/\neg A$, nakonec $C/\neg C$ dá prázdnou klauzuli $\perp$. Tím je KB nesplnitelná.

3.6 Hornovy klauzule, dopředné a zpětné řetězení

V praxi mají znalostní báze často speciální tvar, na němž jde uvažovat *lineárně rychle*.

Definice – Hornova a definitní klauzule

Hornova klauzule je disjunkce literálů, z nichž *nejvýš jeden* je pozitivní. **Definitní klauzule** má *právě jeden* pozitivní literál. Definitní klauzuli $\neg P_1 \vee \dots \vee \neg P_k \vee Q$ lze ekvivalentně psát jako implikaci (pravidlo)

$$P_1 \wedge \dots \wedge P_k \Rightarrow Q,$$

kde tělo (premise) je konjunkce proměnných a hlava (závěr) je jediná proměnná. *Fakt* je definitní klauzule s prázdným tělem (samotné Q). Příklad Hornovy báze: $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$.

 **Intuice.** Na Hornových klauzulích lze odvozování provést v *lineárním* čase vzhledem k velikosti KB – bez exponenciálního prohledávání. Odpovídají pravidlům „když premisy, pak závěr“ (logické jádro Prologu bez proměnných, expertních systémů, deduktivních databází). Dotaz „plyne výroková proměnná q ze KB ?“ se řeší dvěma duálními algoritmy, oba korektní a úplné pro Hornovy báze.

Algoritmus – Dopředné řetězení (forward chaining)

PL-FC-ENTAILS?(KB, q):

count[c] ← počet proměnných v premise klauzule c

▷ pro každé pravidlo

inferred[s] ← false pro všechny proměnné s

queue ← proměnné známe jako pravdivé v KB

▷ fakta

while queue $\neq \emptyset$:

```

p ← Pop(queue)
if p = q: return true
if inferred[p] = false:
    inferred[p] ← true
    for každé pravidlo c, kde p je v premise:
        count[c] ← count[c] - 1
        if count[c] = 0: přidej c.Závěr do queue
return false

```

▷ dotaz dokázán

▷ premisa splněna

▷ nový fakt

Dopředné řetězení jde od faktů k závěrům (řízeno daty). Pro každé pravidlo si drží počet dosud neo-
věřených premis; jakmile klesne na nulu, hlava pravidla se stává novým faktem a přidá se na agendu.
Odvodí postupně všechny důsledky, dokud nedojdou nebo dokud nedokáže dotaz. **Zpětné řetězení**
(backward chaining) jde naopak od dotazu k faktům (řízeno cílem): dotaz q rozloží přes pravidlo s hlavou
 q na pod-dotazy (jeho premisy) a ty rekurzivně dokazuje, dokud nenarazí na fakta. Hodí se, když nás
zajímá jediný dotaz – nemusí odvodit celou bázi, jen relevantní podstrom. Obě metody jsou speciálním
případem rezoluce (každý krok je modus ponens). Dopředné řetězení běží v lineárním čase vzhledem
k velikosti KB , zpětné bývá ještě levnější, protože se dotkne jen relevantních pravidel.

Příklad – Dopředné a zpětné řetězení na Hornově bázi

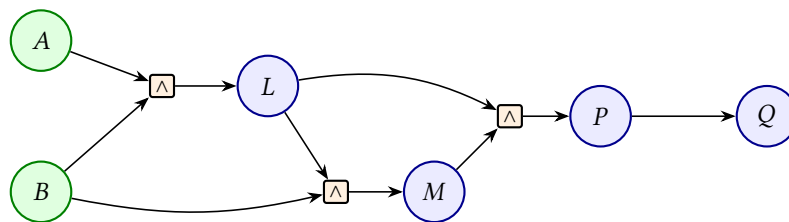
Mějme fakta A, B a pravidla

$$(A \wedge B \Rightarrow L), \quad (B \wedge L \Rightarrow M), \quad (L \wedge M \Rightarrow P), \quad (A \wedge P \Rightarrow L), \quad (P \Rightarrow Q).$$

Dotaz: plyne Q ?

Dopředné řetězení (od faktů): ze známých faktů A, B vyhoví pravidlo $A \wedge B \Rightarrow L$, takže L . Pak
 $B \wedge L \Rightarrow M$ dá M ; $L \wedge M \Rightarrow P$ dá P ; nakonec $P \Rightarrow Q$ dá Q . Dotaz tedy z KB plyne. (Ověřeno simulací:
odvozená báze je $\{A, B, L, M, P, Q\}$.)

Zpětné řetězení (od dotazu): cíl Q rozložíme pravidlem $P \Rightarrow Q$ na pod-cíl P ; ten pravidlem $L \wedge M \Rightarrow P$
na pod-cíle L a M . Cíl L dokáže pravidlo $A \wedge B \Rightarrow L$ (A, B jsou fakta, splněno); cíl M pravidlo $B \wedge L \Rightarrow M$
(B fakt, L už dokázáno). Tím je P , a tedy i Q , dokázáno. Všimněte si, že zpětné řetězení se dotklo jen
relevantních pravidel (pravidlo $A \wedge P \Rightarrow L$ vůbec nevyužilo), kdežto dopředné odvodilo celou bázi.



Hornova báze jako AND-OR graf. Zelené vrcholy jsou fakta (A, B), modré odvozené proměnné. Oranžový
uzel Δ je pravidlo: spojí všechny premisy a ukáže na hlavu. *Dopředné* řetězení postupuje zleva doprava
(od faktů A, B přes L, M, P k Q); *zpětné* řetězení jde zprava doleva (rozkládá dotaz Q přes $P \Rightarrow Q$ na
pod-dotaz P , ten přes $L \wedge M \Rightarrow P$ na L a M , atd., až ke faktům). Pro tuto bázi má každá hlava jediné
pravidlo, takže jsou potřeba jen AND-uzly; OR-uzel by vznikl, kdyby tutéž proměnnou odvozovalo víc
pravidel.

i Poznámka. [Kdy co použít] *DPLL* rozhoduje splnitelnost obecné CNF (najde model, nebo
prohlásí nesplnitelnost) – vhodné, když chceme *ohodnocení*. *Rezoluce* dokazuje důsledek $KB \models \alpha$
vyvrácením $KB \wedge \neg \alpha$ – úplná, ale obecně může vyrábět mnoho klauzulí. *Řetězení* je nejrychlejší
(lineární), ale jen pro *Hornovy* báze; dopředné, jdeme-li za *všemi* důsledky, zpětné, jdeme-li za *jedním*
dotazem. Všechny tři jsou korektní; *DPLL* a *rezoluce* úplné pro obecnou výrokovou logiku, řetězení
úplné pro *Hornovy* báze.

4 Pravděpodobnostní uvažování (Bayesovské sítě, exaktní a aproximační odvozování)

Logický agent věří každému tvrzení jako pravdivému, nepravdivému, nebo nemá názor. To je často příliš hrubé: ve světě s *částečnou pozorovatelností* (agent přesně nezná svůj stav) a *nedeterminismem* (nezná výsledek akce) by musel udržovat *stavy víry* (množiny možných světů) a *kontingenční plány* pro každou eventualitu, což exploduje. *Pravděpodobnostní* agent místo toho přiřadí každému tvrzení *stupeň víry* – číslo mezi 0 (jistě nepravda) a 1 (jistě pravda). Tato sekce shrnuje základy teorie pravděpodobnosti potřebné pro uvažování, ukáže, jak se odpovídá na dotazy vysčítáním sdružené distribuce, a zavede *Bayesovské sítě* jako kompaktní reprezentaci, nad níž se uvažuje exaktně (enumerace, eliminace proměnných) i aproximačně (Monte Carlo).

4.1 Základy teorie pravděpodobnosti

Pravděpodobnostní tvrzení mluví o *možných světech* (prvcích výběrového prostoru Ω), které jsou *vzájemně se vylučující* a *vyčerpávající*. Každému světu ω je přiřazena pravděpodobnost $P(\omega)$, přičemž $0 \leq P(\omega) \leq 1$ a $\sum_{\omega \in \Omega} P(\omega) = 1$. *Jev* je množina možných světů; jeho pravděpodobnost je součet pravděpodobností jeho světů. Svět popisujeme *faktorovaně* pomocí *náhodných veličin*: každá veličina (např. *Cavity*, *Toothache*) má doménu možných hodnot a úplné přiřazení hodnot všem veličinám určuje právě jeden svět. **Značení (důležité):** tučné $\mathbf{P}(X)$ značí celé *rozdělení* – vektor pravděpodobností pro všechny hodnoty X – zatímco obyčejné $P(x)$ je jedno *číslo*, pravděpodobnost konkrétní hodnoty. Tak třeba $\mathbf{P}(X | e) = \alpha \mathbf{P}(X, e)$ je rovnost dvou vektorů.

Definice – Úplná sdružená distribuce

Úplná sdružená pravděpodobnostní distribuce (full joint distribution) náhodných veličin $X_1, \dots, X_n$ je tabulka, která každému úplnému přiřazení (možnému světu) přiřadí jeho pravděpodobnost $P(X_1=x_1, \dots, X_n=x_n)$. Z ní lze *in principu* odpovědět na libovolný pravděpodobnostní dotaz. Má ale velikost $\prod_i |\text{dom}(X_i)|$, tedy pro n binárních veličin 2^n řádků – exponenciálně v počtu veličin.

Definice – Podmíněná pravděpodobnost, součinnové (product) pravidlo

Pro jev a a evidenci b s $P(b) > 0$ je *podmíněná* (aposteriorní) pravděpodobnost

$$P(a | b) = \frac{P(a \wedge b)}{P(b)}.$$

Přepsáním dostáváme *product rule* (součinnové pravidlo): $P(a \wedge b) = P(a | b) P(b) = P(b | a) P(a)$. Bez podmínky mluvíme o *apriorní* (prior) pravděpodobnosti.

Definice – Marginalizace, podmiňování, normalizace

Marginalizace (vysčítání, summing out) získá rozdělení podmnožiny veličin součtem přes hodnoty zbylých:

$$\mathbf{P}(Y) = \sum_{z \in Z} \mathbf{P}(Y, z), \quad (\text{varianta s podmiňováním}) \quad \mathbf{P}(Y) = \sum_{z \in Z} \mathbf{P}(Y | z) P(z).$$

Pro *podmíněný* dotaz se skrytými veličinami H (vše kromě dotazu Y a evidence E) platí

$$\mathbf{P}(Y | E=e) = \frac{\mathbf{P}(Y, E=e)}{P(E=e)} = \alpha \mathbf{P}(Y, E=e) = \alpha \sum_h \mathbf{P}(Y, E=e, H=h),$$

kde $\alpha = 1/P(E=e)$ je *normalizační konstanta*. Protože $\sum_y P(Y=y | e) = 1$, nemusíme $P(E=e)$ počítat zvlášť: spočteme nenormalizovaný vektor $\mathbf{P}(Y, e)$ pro všechny hodnoty Y a na závěr ho *znormalizujeme*, aby dal v součtu 1.

Příklad – Zubní diagnóza: dotaz vysčítáním a normalizací

Mějme binární veličiny *Cavity* (kaz), *Toothache* (bolest), *Catch* (sonda zachytí) s úplnou sdruženou distribucí. Pak třeba $P(\text{toothache}) = 0.108 + 0.012 + 0.016 + 0.064 = 0.2$ (sečteme všechny světy, kde bolí). Dotaz na rozdělení *Cavity* při bolesti vyřešíme přes normalizaci a vysčítání skryté veličiny *Catch*:

$$\begin{aligned} P(\text{Cavity} \mid \text{toothache}) &= \alpha P(\text{Cavity}, \text{toothache}) \\ &= \alpha [P(\text{Cavity}, \text{toothache}, \text{catch}) + P(\text{Cavity}, \text{toothache}, \neg \text{catch})] \\ &= \alpha [\langle 0.108, 0.016 \rangle + \langle 0.012, 0.064 \rangle] \\ &= \alpha \langle 0.12, 0.08 \rangle = \langle 0.6, 0.4 \rangle, \end{aligned}$$

kde první složka vektoru je *Cavity*=*true*, druhá *Cavity*=*false*, a $\alpha = 1/(0.12 + 0.08) = 5$.

Definice – Bayesovo pravidlo

Z rovnosti $P(a \mid b)P(b) = P(b \mid a)P(a)$ plyne *Bayesovo pravidlo*

$$P(a \mid b) = \frac{P(b \mid a)P(a)}{P(b)}, \quad \text{obecně} \quad P(Y \mid X) = \frac{P(X \mid Y)P(Y)}{P(X)} = \alpha P(X \mid Y)P(Y).$$

 **Intuice.** [Kauzální vs. diagnostický směr] Píšeme-li $a = \text{cause}$ (příčina) a $b = \text{effect}$ (důsledek), pak $P(\text{effect} \mid \text{cause})$ je *kauzální* (příčinná) znalost a $P(\text{cause} \mid \text{effect})$ *diagnostická*. Bayesovo pravidlo počítá diagnostiku z kauzality. Proč to tak děláme? *Kauzální* znalost je robustnější: popisuje, jak svět funguje. Příklad – meningitida způsobí ztuhlou šíjí v 70 % ($P(s \mid m) = 0.7$), apriorní $P(m) = 1/50\,000$, $P(s) = 0.01$. Pak

$$P(m \mid s) = \frac{P(s \mid m)P(m)}{P(s)} = \frac{0.7 \cdot \frac{1}{50000}}{0.01} = 0.0014.$$

Kdyby vypukla epidemie, $P(m)$ vzroste a s ním i $P(m \mid s)$, kdežto kauzální vztah $P(s \mid m)$ (jak meningitida působí) se nezmění. Proto ukládáme kauzální čísla.

4.2 Nezávislost a podmíněná nezávislost

Plnou sdruženou distribuci lze *zhustit*, využijeme-li nezávislosti.

Definice – (Absolutní) nezávislost a podmíněná nezávislost

Veličiny X, Y jsou **nezávislé**, právě když

$$P(X \mid Y) = P(X) \iff P(Y \mid X) = P(Y) \iff P(X, Y) = P(X)P(Y).$$

X, Y jsou **podmíněně nezávislé** při dané Z , právě když

$$P(X, Y \mid Z) = P(X \mid Z)P(Y \mid Z) \iff P(X \mid Y, Z) = P(X \mid Z).$$

 **Intuice.** Absolutní nezávislost je vzácná (dvě kostky), zato *podmíněná* nezávislost je všudypřítomná. Příklad: *Toothache* a *Catch* *nejsou* nezávislé (obojí je důsledkem kazu), ale *při dané Cavity* už ano – známe-li stav kazu, výsledek sondy a bolest spolu nesouvisejí: $P(\text{Catch}, \text{Toothache} \mid \text{Cavity}) = P(\text{Catch} \mid \text{Cavity})P(\text{Toothache} \mid \text{Cavity})$. Jednu velkou tabulku tak nahradíme několika menšími. Přesně tento princip systematizují Bayesovské sítě.

Definice – Naivní Bayes

Jsou-li všechny důsledky $Effect_1, \dots, Effect_n$ podmíněně nezávislé při dané příčině $Cause$, faktorizuje se sdružená distribuce na

$$P(Cause, Effect_1, \dots, Effect_n) = P(Cause) \prod_{i=1}^n P(Effect_i | Cause).$$

Tento model se nazývá *naivní Bayes* (či *idiot's Bayes*). Místo $O(2^n)$ čísel stačí $O(n)$ malých tabulek. Používá se i tam, kde podmíněná nezávislost ve skutečnosti neplatí přesně – často překvapivě dobře (např. klasifikace textu, viz okruh S3).

4.3 Bayesovské sítě: definice, konstrukce, vztah ke sdružené distribuci

Definice – Bayesovská síť

Bayesovská síť je orientovaný acyklický graf (DAG), kde

- **uzly** odpovídají náhodným veličinám,
- **orientovaná hrana** $X \rightarrow Y$ znamená, že X je rodič Y (přímý vliv),
- každý uzel X má *tabulku podmíněných pravděpodobností* (CPT) $P(X | Parents(X))$, která kvantifikuje vliv rodičů na X .

Síť kompaktně reprezentuje úplnou sdruženou distribuci.

★ Věta – Síť reprezentuje sdruženou distribuci – řetízkové pravidlo

Bayesovská síť určuje úplnou sdruženou distribuci součinem CPT:

$$P(x_1, \dots, x_n) = \prod_{i=1}^n P(x_i | parents(X_i)).$$

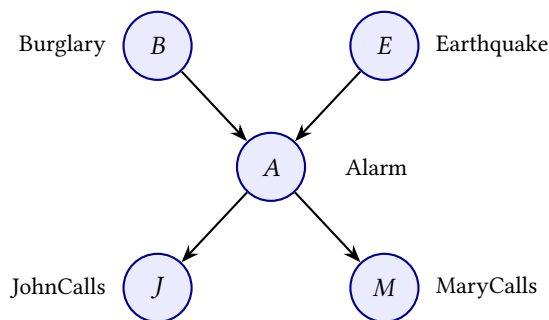
Intuice. [Proč to funguje] Obecné řetízkové (*chain*) pravidlo platí pro libovolné očíslování veličin:

$$P(x_1, \dots, x_n) = \prod_i P(x_i | x_{i-1}, \dots, x_1).$$

Bayesovská síť navíc *předpokládá*, že každý uzel je podmíněně nezávislý na svých *ne-potomcích* při daných rodičích, tedy $P(x_i | x_{i-1}, \dots, x_1) = P(x_i | parents(X_i))$, je-li $Parents(X_i) \subseteq \{x_1, \dots, x_{i-1}\}$. Dosazením do řetízkového pravidla vznikne součin CPT. Tím je vztah „síť $\Leftrightarrow$ sdružená distribuce“ oboustranný: ze sítě sdruženou distribuci *sestavíme* (součinem) a síť ji *reprezentuje* kompaktně, protože ukládá jen lokální CPT místo celé exponenciální tabulky.

Poznámka. [Konstrukce sítě] Síť se staví takto: (1) vyber a *seřaď* veličiny – libovolné pořadí funguje, ale *příčiny před důsledky* dávají menší a přirozenější síť; (2) pro $i = 1, \dots, n$ vyber mezi již zařazenými $\{x_1, \dots, x_{i-1}\}$ *minimální* množinu rodičů $Parents(X_i)$ tak, aby $P(X_i | Parents(X_i)) = P(X_i | x_{i-1}, \dots, x_1)$; (3) veď hranu od každého rodiče k X_i a zapiš CPT. Špatné pořadí (důsledky před příčinami) vede k hustším sítím s nepřirozenými hranami a většími CPT, ale stále korektně reprezentuje stejnou sdruženou distribuci.

Klasickým příkladem je síť *alarm*: vloupání (B) nebo zemětřesení (E) spustí alarm (A); alarm slyší sousedé John (J) a Mary (M) a zavolají. Sousedé nevnímají vloupání ani zemětřesení přímo, jen alarm, a neradí se spolu, takže J a M jsou podmíněně nezávislé při daném A .



$$P(B=T) = 0.001$$

$$P(E=T) = 0.002$$

CPT pro A , tj. $P(A=T \mid B, E)$:

B	E	$P(A=T \mid B, E)$
T	T	0.95
T	F	0.94
F	T	0.29
F	F	0.001

$$P(J=T \mid A): 0.90 / 0.05$$

$$P(M=T \mid A): 0.70 / 0.01$$

Síť *alarm*: DAG s pěti binárními veličinami a CPT u uzlů (poslední dva řádky znamenají $P(\cdot \mid A=T) / P(\cdot \mid A=F)$). Místo $2^5 - 1 = 31$ čísel úplné sdružené tabulky stačí $1+1+4+2+2 = 10$ čísel v CPT. Síť reprezentuje celou sdruženou distribuci součinem $P(B, E, A, J, M) = P(B) P(E) P(A \mid B, E) P(J \mid A) P(M \mid A)$, obecně $P(x_1, \dots, x_n) = \prod_i P(x_i \mid \text{parents}(X_i))$. Např. $P(j, m, a, \neg b, \neg e) = 0.90 \cdot 0.70 \cdot 0.001 \cdot 0.999 \cdot 0.998 \approx 0.000628$.

4.4 Exaktní odvozování: enumerace

Cílem *odvozování* (inference) je spočítat aposteriorní rozdělení dotazované veličiny X při dané evidenci e , přičemž zbylé veličiny Y jsou *skryté*:

$$\mathbf{P}(X \mid e) = \alpha \mathbf{P}(X, e) = \alpha \sum_y \mathbf{P}(X, e, y),$$

kde každý člen $P(x, e, y)$ se vyčíslí jako součin CPT podle řetízkového pravidla. Pro síť *alarm* a dotaz $\mathbf{P}(B \mid j, m)$ (vloupání, když volá John i Mary) je

$$\begin{aligned} \mathbf{P}(B \mid j, m) &= \alpha \sum_e \sum_a P(B) P(e) P(a \mid B, e) P(j \mid a) P(m \mid a) \\ &= \alpha P(B) \sum_e P(e) \sum_a P(a \mid B, e) P(j \mid a) P(m \mid a). \end{aligned}$$

Vytknutím konstantních faktorů ven ze sumace dostaneme strom výpočtu, který se vyhodnocuje zdola nahoru.

⚙️ Algoritmus – ENUMERATION-ASK – exaktní odvozování enumerací

```

ENUMERATION-ASK( $X, e, bn$ ):
   $Q(X) \leftarrow$  prázdné rozdělení nad  $X$ 
  for každou hodnotu  $x_i$  veličiny  $X$ :
     $Q(x_i) \leftarrow$  ENUMERATE-ALL( $bn.vars, e \cup \{X=x_i\}$ )
  return NORMALIZE( $Q(X)$ )

ENUMERATE-ALL( $vars, e$ ):
  if  $vars$  je prázdné: return 1.0
   $V \leftarrow$  FIRST( $vars$ ) ▷ veličiny v topologickém pořadí
  if  $V$  má v  $e$  hodnotu  $v$ :
    return  $P(v \mid \text{parents}(V)) \cdot$  ENUMERATE-ALL( $REST(vars), e$ )
  else ▷ skrytá veličina -- vysčítej přes její hodnoty
    return  $\sum_v P(v \mid \text{parents}(V)) \cdot$  ENUMERATE-ALL( $REST(vars), e \cup \{V=v\}$ )
  
```

💡 Intuice. Enumerace je v podstatě hloubkové procházení stromu, jehož větvení odpovídá hodnotám skrytých veličin – podobně jako řešení CSP nebo SAT. Je správná, ale *neefektivní*: některé podvýpočty se opakují. Ve výrazu výše se faktory $P(j \mid a) P(m \mid a)$ počítají *znovu* pro každou hodnotu e , ačkoli na e nezávisí. To napravuje eliminace proměnných.

4.5 Exaktní odvozování: eliminace proměnných

Eliminace proměnných (variable elimination) je dynamické programování nad výrazem enumerace: *mezi* výsledky se uloží a znovu použijí. Pracuje s *faktory* – maticemi (tabulkami) indexovanými hodnotami

veličin, odpovídajícími CPT. Výraz pro $\mathbf{P}(B \mid j, m)$ zapíšeme jako součin faktorů

$$\mathbf{P}(B \mid j, m) = \alpha \underbrace{f_1(B)}_{P(B)} \sum_e \underbrace{f_2(E)}_{P(e)} \sum_a \underbrace{f_3(A, B, E)}_{P(a|B,e)} \underbrace{f_4(A)}_{P(j|a)} \underbrace{f_5(A)}_{P(m|a)}$$

a vyhodnocujeme *zprava doleva*: opakovaně bodově vynásobíme faktory obsahující právě eliminovanou veličinu a tu pak *vysčítáme*.

Definice – Operace s faktory

Bodový součin (pointwise product) dvou faktorů dá faktor nad *sjednocením* jejich veličin; každý řádek je součin odpovídajících řádků (shoda na sdílených veličinách). **Vysčítání** (summing out) veličiny V z faktoru sečte podmatice vzniklé fixací V na jednotlivé hodnoty – V z faktoru zmizí.

Algoritmus – ELIMINATION-ASK – eliminace proměnných

ELIMINATION-ASK(X, e, bn):

 factors $\leftarrow []$

for každou veličinu V v ORDER($bn.vars$):

 factors $\leftarrow$ [MAKE-FACTOR(V, e)] + factors

▷ CPT V jako faktor

if V je skrytá veličina:

 factors $\leftarrow$ SUM-OUT($V, factors$)

▷ vynásob faktory s V a vysčítej V

return NORMALIZE(POINTWISE-PRODUCT(factors))

Podprogramy – co se skrývá za těmi jmény:

MAKE-FACTOR(V, e):

$f \leftarrow$ CPT $P(V \mid parents(V))$ jako tabulka

 zafixuj v f veličiny z e

▷ jejich rozměr zmizí

return f

POINTWISE-PRODUCT($f_1, \dots, f_k$):

$h \leftarrow$ tabulka nad $\bigcup_i vars(f_i)$

$h(x) \leftarrow \prod_i f_i(x)$ pro každý řádek x

▷ f_i dle své části

return h

SUM-OUT($V, factors$):

 touch $\leftarrow$ faktory obsahující V ; rest $\leftarrow$ ostatní

$g \leftarrow$ POINTWISE-PRODUCT(touch)

▷ spoj vše s V

$g' \leftarrow \sum_v g(V=v, \dots)$

▷ V zmizí

return rest + [g']

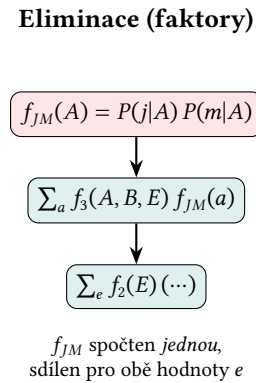
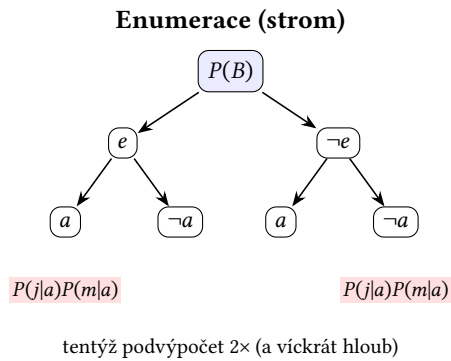
ORDER(vars): obrácené topologické pořadí

▷ libovolné korektní

NORMALIZE(f): poděl f součtem prvků

▷ ať součet = 1

i Poznámka. Algoritmus je správný pro *libovolné* pořadí eliminace, ale pořadí ovlivňuje rychlost: *složitost je dána velikostí největšího faktoru*, který během běhu vznikne. Heuristika: eliminuj vždy veličinu, která vede k nejmenšímu nově vytvořenému faktoru. (Pro polystromy – sítě bez neorientovaných cyklů – je eliminace lineární; pro obecné sítě je exaktní odvozování #P-těžké.) Klíčový rozdíl oproti enumeraci: faktor $\sum_a P(a \mid B, e)P(j \mid a)P(m \mid a)$ se spočítá jednou a sdílí místo opakování pro každé e .



Enumerace (vlevo) prochází strom hodnot skrytých veličin a tentýž podvýpočet $P(j|a)P(m|a)$ opakuje v každé větvi e . Eliminace proměnných (vpravo) si mezivýsledek uloží jako *faktor* $f_{JM}(A)$ a znovu ho použije, čímž sdílí výpočet (dynamické programování). Pořadí eliminace zde: nejdřív A , pak E .

4.6 Aproximační odvozování: metody Monte Carlo

Pro velké, *vícenásobně propojené* sítě je exaktní odvozování neúnosné. Pak sáhneme k *aproximaci* metodami Monte Carlo: vygenerujeme mnoho *vzorků* (úplných ohodnocení veličin) a hledanou pravděpodobnost odhadneme statisticky – více vzorků znamená větší přesnost. Vzorky se generují podle rozdělení daného sítí: veličiny se berou v *topologickém pořadí* a každá se navzorkuje z $P(X_i | \text{parents}(X_i))$ podmíněné hodnotami už navzorkovaných rodičů (algoritmus PRIOR-SAMPLE). Pak platí $P(x_1, \dots, x_n) = \lim_{N \rightarrow \infty} N(x_1, \dots, x_n)/N$, kde $N(\cdot)$ je počet výskytů události mezi N vzorky.

i Poznámka. [Zamítací vzorkování (rejection sampling)] Chceme ale $P(X | e)$. Nejjednodušší cesta: generuj vzorky ze sítě (jako PRIOR-SAMPLE) a *ponech jen ty konzistentní s evidencí e , ostatní zamítni*. Odhad je $P(X | e) \approx N(X, e)/N(e)$ (podíl, kolikrát mezi konzistentními vzorky vyšlo $X=x$). **Hlavní slabina:** při málo pravděpodobné evidenci se zamítne drtivá většina vzorků (počet konzistentních vzorků klesá exponenciálně s počtem evidenčních veličin), takže odhad je drahý a nepřesný.

Algoritmus – LIKELIHOOD-WEIGHTING – vážení věrohodností

```

LIKELIHOOD-WEIGHTING( $X, e, bn, N$ ):
     $W \leftarrow$  vektor vážených počtů pro hodnoty  $X$ , nuly
    for  $j = 1$  to  $N$ :
         $\mathbf{x}, w \leftarrow$  WEIGHTED-SAMPLE( $bn, e$ )
         $W[\mathbf{x}] \leftarrow W[\mathbf{x}] + w$   $\triangleright x = \text{hodnota } X \text{ ve vzorku } \mathbf{x}$ 
    return NORMALIZE( $W$ )

WEIGHTED-SAMPLE( $bn, e$ ):
     $w \leftarrow 1$ ;  $\mathbf{x} \leftarrow$  událost s hodnotami evidence z  $e$  napevno
    for  $i = 1$  to  $n$   $\triangleright$  veličiny v topologickém pořadí:
        if  $X_i$  je evidenční s hodnotou  $x_i$  v  $e$ :
             $w \leftarrow w \cdot P(X_i = x_i | \text{parents}(X_i))$   $\triangleright$  nevzorkuj, jen zvaž
        else  $\mathbf{x}[i] \leftarrow$  vzorek z  $P(X_i | \text{parents}(X_i))$ 
    return  $\mathbf{x}, w$ 

```

Intuice. Místo zamítání generuje vážení věrohodností *jen* vzorky konzistentní s e : evidenční veličiny se *nevzorkují*, ale fixují na pozorované hodnoty. Tím se ale zkresluje rozdělení – vzorek vznikne s pravděpodobností $\prod_i P(z_i | \text{parents}(z_i))$ jen přes *nevidenční* veličiny z , kdežto „chybí“ váha $w(z, e) = \prod_j P(e_j | \text{parents}(e_j))$ za evidenční veličiny. Každý vzorek proto dostane *váhu* $w =$ součin pravděpodobností (věrohodností) evidence vzhledem k jejím rodičům a odhad $P(x | e)$ je normalizovaný součet vah těch vzorků, v nichž $X=x$ (vážené součty místo prostých počtů). **Hlavní slabina:** jsou-li evidenční veličiny „dole“ v síti, většina vah je nepatrná a odhad ovládne pár vzorků (degenerace). Přesto je likelihood weighting typicky mnohem účinnější než zamítací vzorkování.

4.7 Řešený příklad SZZ

Příklad — SZZ léto 2024 č. 28: Bayesovské sítě (definice, konstrukce, odvozování)

(Reálná otázka SZZ: léto 2024, č. 28 – Bayesovské sítě)

Zadání.

Definujte Bayesovskou síť a napište, jak se takové sítě používají a jak se konstruují. Jaký je vztah mezi Bayesovskou sítí a úplnou sdruženou pravděpodobnostní distribucí? Pro síť na obrázku (čtyři binární veličiny $P_1, \dots, P_4$ s uvedenými CPT) a pozorování $P_3=F, P_4=T$ spočítejte $P(P_1=T | P_3=F, P_4=T)$.

Řešení (teoretická část). Bayesovská síť je orientovaný acyklický graf, jehož uzly jsou náhodné veličiny, hrana $X \rightarrow Y$ značí přímý vliv (rodičovství) a každý uzel nese CPT $\mathbf{P}(X | \text{Parents}(X))$. Používá se k odpovídání na dotazy $\mathbf{P}(X | E=e)$ (rozdělení dotazované veličiny při evidenci) – exaktně enumerací nebo eliminací proměnných, aproximace vzorkováním. Konstruuje se seřazením veličin (ideálně příčiny před důsledky) a výběrem minimální množiny rodičů pro každou veličinu, aby platilo $P(X_i | \text{Parents}(X_i)) = P(X_i | X_{i-1}, \dots, X_1)$. Vztah ke sdružené distribuci: síť ji *plně reprezentuje* přes řetízkové pravidlo $P(x_1, \dots, x_n) = \prod_i P(x_i | \text{parents}(X_i))$ – ze sítě sdruženou distribuci sestavíme součinem CPT a naopak síť ji ukládá kompaktně (jen lokální CPT místo exponenciální tabulky).

Řešení (výpočet). Struktura sítě je řetěz s rozvětvením $P_1 \rightarrow P_2 \rightarrow \{P_3, P_4\}$ (uzel P_2 je rodičem P_3 i P_4). CPT:

$$P(P_1=T) = 0.4; \quad P(P_2=T | P_1=T) = 0.8, \quad P(P_2=T | P_1=F) = 0.5;$$

$$P(P_3=T | P_2=T) = 0.2, \quad P(P_3=T | P_2=F) = 0.3; \quad P(P_4=T | P_2=T) = 0.8, \quad P(P_4=T | P_2=F) = 0.5.$$

(1) Sdružená distribuce z řetízkového pravidla. Pro libovolné hodnoty

$$P(P_1, P_2, P_3, P_4) = P(P_1) P(P_2 | P_1) P(P_3 | P_2) P(P_4 | P_2).$$

(2) Vysčítání skryté veličiny P_2 . Veličina P_2 je skrytá (není v dotazu ani evidenci), proto ji marginalizujeme. Pro $P_1=T$ a evidenci $P_3=F, P_4=T$:

$$\begin{aligned} P(P_1=T, P_3=F, P_4=T) &= P(P_1=T) \sum_{P_2} P(P_2 | P_1=T) P(P_3=F | P_2) P(P_4=T | P_2) \\ &= 0.4 \left[\underbrace{0.8}_{P_2=T} \cdot \underbrace{0.8}_{P_3=F|P_2=T} \cdot \underbrace{0.8}_{P_4=T|P_2=T} + \underbrace{0.2}_{P_2=F} \cdot \underbrace{0.7}_{P_3=F|P_2=F} \cdot \underbrace{0.5}_{P_4=T|P_2=F} \right] \\ &= 0.4 [0.512 + 0.070] = 0.4 \cdot 0.582 = 0.23280. \end{aligned}$$

(Použili jsme $P(P_3=F | P_2=T) = 1 - 0.2 = 0.8$ a $P(P_3=F | P_2=F) = 1 - 0.3 = 0.7$.)

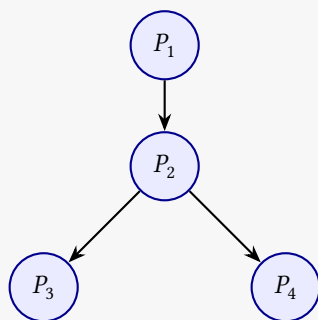
Analogicky pro $P_1=F$ (zde $P(P_1=F) = 0.6, P(P_2=T | P_1=F) = 0.5$):

$$\begin{aligned} P(P_1=F, P_3=F, P_4=T) &= 0.6 [0.5 \cdot 0.8 \cdot 0.8 + 0.5 \cdot 0.7 \cdot 0.5] \\ &= 0.6 [0.320 + 0.175] = 0.6 \cdot 0.495 = 0.29700. \end{aligned}$$

(3) Normalizace. Evidence $P(P_3=F, P_4=T) = 0.23280 + 0.29700 = 0.52980$, tedy

$$P(P_1=T | P_3=F, P_4=T) = \frac{0.23280}{0.52980} = \boxed{0.4394} \quad (\approx 43.9\%).$$

(Doplňkově $P(P_1=F \mid P_3=F, P_4=T) = 0.29700/0.52980 \approx 0.5606$; součet je 1.)



$$P(P_1=T) = 0.4$$

P_1	$P(P_2=T \mid P_1)$
T	0.8
F	0.5

P_2	$P(P_3=T \mid P_2)$	$P(P_4=T \mid P_2)$
T	0.2	0.8
F	0.3	0.5

Bayesovská síť k SZZ úloze: řetěz s rozvětvením $P_1 \rightarrow P_2 \rightarrow \{P_3, P_4\}$ a příslušné CPT. Dotaz $P(P_1=T \mid P_3=F, P_4=T)$ řešíme sestavením sdružené distribuce přes řetízkové pravidlo, vysčítáním skryté P_2 a normalizací; výsledek ≈ 0.4394 . Tato otázka nemá v PDF oficiální náčrt řešení; výpočet je ověřen numericky.

i Poznámka. Pozorování ($P_3=F, P_4=T$) změnilo víru o P_1 z apriorních $P(P_1=T) = 0.4$ na aposteriorních 0.439. Důsledky P_3, P_4 tedy nesou informaci o vzdálené příčině P_1 – to je celé kouzlo Bayesovských sítí: lokální CPT a globální (diagnostické i kauzální) uvažování přes ně.

Příklad – Obměna úlohy: kauzální směr na téže síti

SZZ dotaz výše byl *diagnostický* (evidence P_3, P_4 pod dotazem P_1). V opačném, *kauzálním* směru – evidence *nad* dotazem – se výpočet zjednoduší. **Zadání:** na téže síti $P_1 \rightarrow P_2 \rightarrow \{P_3, P_4\}$ spočítejte $P(P_4=T \mid P_1=T)$.

Řešení. Relevantní jsou jen dotaz, evidence a jejich *předkové*, tj. $\{P_1, P_2, P_4\}$. Veličina P_3 není předkem P_4 ani P_1 – je to *barren* uzel (nepřispívá) a vypadne, protože $\sum_{P_3} P(P_3 \mid P_2) = 1$. Zbude suma jen přes skrytou P_2 :

$$P(P_4=T \mid P_1=T) = \sum_{P_2} P(P_2 \mid P_1=T) P(P_4=T \mid P_2) = \underbrace{0.8 \cdot 0.8}_{P_2=T} + \underbrace{0.2 \cdot 0.5}_{P_2=F} = 0.74.$$

Normalizace netřeba: $P(P_4=F \mid P_1=T) = 0.8 \cdot 0.2 + 0.2 \cdot 0.5 = 0.26$ a součet $0.74 + 0.26 = 1$ vyjde sám – v kauzálním směru je nenormalizovaný výsledek rovnou rozdělením (kdežto diagnostické $P(P_1 \mid P_3, P_4)$ výše normalizovat α muselo).

Extrém – pozorování přímého rodiče: $P(P_4=T \mid P_2=T) = 0.8$ je rovnou řádek CPT, žádná suma ani normalizace.

5 Reprezentace znalostí (situační kalkulus, Markovovy modely, HMM)

Tato kapitola se zabývá *reprezentací znalostí o měnícím se světě*. Nejdřív logicky: *situační kalkulus* pojmenuje jednotlivé stavy světa termy prvořádové logiky a vyřeší *problém rámce*. Pak pravděpodobnostně: agent svět nepozoruje úplně ani spolehlivě, a proto skrytý stav X_t pouze *odhaduje* z posloupnosti zašuměných pozorování $E_{1:t}$. To vede k *uvažování v čase* s Markovovými předpoklady, ke čtyřem základním inferenčním úlohám (filtrace, predikce, vyhlazování, nejpravděpodobnější průchod) a ke *skrytým Markovovým modelům* (HMM) coby maticově řešitelnému speciálnímu případu *dynamických Bayesovských sítí* (DBN).

5.1 Situační kalkulus a problém rámce

Mění se svět by se dal popsat výrokovými proměnnými *indexovanými časem* (tzv. *fluenty*), např. $L_{x,y}^t$ – „agent je v poli (x, y) v čase t “ (takto s nimi pracuje např. výrokový plánovač SATPlan). Takový popis funguje, ale je upovídaný: tytéž axiomy musíme opisovat pro každý čas t zvlášť. *Situační kalkulus* totéž dělá elegantněji v *prvořádové logice*: stavy (situace) dostanou jména v podobě termů, takže o změnách lze uvažovat čistou logickou inferencí.

Definice – Situační kalkulus

Svět popisujeme termy a relacemi prvořádové logiky:

- **situace** (stavy) jsou pojmenovány termy: s_0 je počáteční situace a $\text{RESULT}(s, a)$ je situace vzniklá provedením akce a v situaci s ;
- **fluenty** jsou relace, jejichž platnost *závisí na situaci* – přidají argument situace, např. $at(robot, loc, s)$. Relace, které se v čase nemění (*rigidní* predikáty), argument situace nemají, např. $connected(l, l')$;
- **axiom použitelnosti** (possibility axiom) říká, kdy je akce proveditelná: $\Phi(s) \Rightarrow Poss(a, s)$, kde $\Phi(s)$ popisuje předpoklady akce. Např. $at(a, l, s) \wedge connected(l, l') \Rightarrow Poss(go(a, l, l'), s)$;
- **plánování** se převede na logický dotaz: existuje situace, v níž platí cíl? $\exists s : HaveGold(a, s) \wedge ClimbedOut(a, s)$.

Problém rámce. Jak v situačním kalkulu popsat *účinek* akce? Nejprímější způsob jsou *efektové axiomy* – co se po akci změní (např. akce *Grab* způsobí *Holding*). Tyto axiomy ale mlčí o tom, co se *nezměnilo*. Z „agent jde dopředu“ nelze odvodit, že stále drží šíp, protože žádný axiom to neříká. To je **problém rámce** (frame problem): bez explicitního popisu setrvačnosti se po každé akci „rozpadne“ celý zbytek poznání o světě.

Naivní řešení jsou *rámecové axiomy* (frame axioms) – pro každou akci a každý fluent vyjmenovat, že se nemění, např. $Forward^t \Rightarrow (HaveArrow^t \Leftrightarrow HaveArrow^{t+1})$. Těch je ale obrovské množství (akce $\times$ fluenty) a uvažování se zahltí.

Definice – Successor-state axiom (řešení problému rámce)

Místo axiomů o akcích píšeme *jeden axiom na každý fluent* – tzv. **successor-state axiom** (axiom následnického stavu), který určuje pravdivost fluentu v situaci $\text{RESULT}(s, a)$ vyčerpávajícím způsobem:

$$Poss(a, s) \Rightarrow \left(F(\text{RESULT}(s, a)) \Leftrightarrow \underbrace{a \text{ činí } F \text{ pravdivým}}_{\text{efekt akce}} \vee \underbrace{(F(s) \wedge \neg(a \text{ činí } F \text{ nepravdivým}))}_{\text{setrvačnost}} \right).$$

Fluent F platí po akci právě tehdy, když ho akce *nastavila*, *nebo* platil už předtím a akce ho *nezrušila*.
Příklad: $HaveArrow^{t+1} \Leftrightarrow (HaveArrow^t \wedge \neg Shoot^t)$.

 **Intuice.** Klíčový obrat: efektový axiom je orientovaný „na akci“ (co tahle akce udělá), zatímco successor-state axiom je orientovaný „na fluent“ (jak se tento fluent vyvíjí pod *všemi* akcemi). Tím že do pravé strany ekvivalence zahrneme i větev setrvačnosti, je popis *úplný* – co axiom neuvede jako příčinu změny, se prostě nemění, a problém rámce mizí. Cena: pro každý fluent musíme vyjmenovat všechny akce, které ho ovlivňují.

Situační kalkulus tak umožní plánovat čistou logickou inferencí, bez programového „obalu“. Klasickému plánování s operátory (faktorová reprezentace, dopředné a zpětné plánování) se věnuje samostatná sekce o automatickém plánování.

5.2 Uvažování v čase: Markovovy předpoklady

Přejdeme k nejistotě. Svět vnímáme jako posloupnost *časových řezů* (time slices). V každém čase t je

- **skrytý stav** X_t – náhodná veličina (či vektor veličin), kterou *nepozorujeme* (skutečné počasí, poloha robota);
- **pozorování** E_t s pozorovanou hodnotou e_t – to, co o stavu *vidíme* (zda kolega přinesl deštník, údaje senzorů).

Čas bereme diskrétní s rovnoměrným krokem. Zápis $X_{a:b}$ značí posloupnost $X_a, \dots, X_b$. Abychom mohli uvažovat, potřebujeme popsat (1) jak se stavy vyvíjejí a (2) jak na stavu závisí pozorování. Obojí zjednoduší *Markovovy předpoklady*.

Definice – Přejchodový a senzorový model, Markovovy předpoklady

Přejchodový model udává rozdělení dalšího stavu vzhledem k minulosti $\mathbf{P}(X_t | X_{0:t-1})$. *Markovův předpoklad (prvního řádu)*: stav závisí jen na bezprostředně předchozím stavu,

$$\mathbf{P}(X_t | X_{0:t-1}) = \mathbf{P}(X_t | X_{t-1}).$$

Senzorový model udává, jak pozorování závisí na stavech $\mathbf{P}(E_t | X_{0:t}, E_{1:t-1})$. *Senzorový Markovův předpoklad*: pozorování závisí jen na aktuálním stavu,

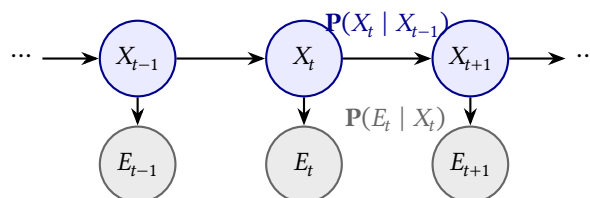
$$\mathbf{P}(E_t | X_{0:t}, E_{1:t-1}) = \mathbf{P}(E_t | X_t).$$

Stacionarita: tabulky $\mathbf{P}(X_t | X_{t-1})$ a $\mathbf{P}(E_t | X_t)$ jsou *stejné pro všechna* t (proces se v čase nemění).

Intuice. Markovův předpoklad říká, že *přítomnost stíní minulost*: jakmile znám stav X_{t-1} , starší stavy mi o X_t už nic dalšího neřeknou. Stacionarita znamená, že stačí *jedna* přechodová a *jedna* senzorová tabulka, ne nová pro každý čas. Spolu s priorem $\mathbf{P}(X_0)$ tím dostáváme úplné sdružené rozdělení

$$\mathbf{P}(X_{0:t}, E_{1:t}) = \mathbf{P}(X_0) \prod_{i=1}^t \mathbf{P}(X_i | X_{i-1}) \mathbf{P}(E_i | X_i).$$

Tyto závislosti lze nakreslit jako Bayesovskou síť rozprostřenou v čase: řetěz skrytých stavů s „visícími“ pozorováními.



Markovův model jako Bayesovská síť v čase (svět deštníků). Horní řada jsou *skryté* stavy (počasí) spojené *přejchodovým modelem* $\mathbf{P}(X_t | X_{t-1})$; svislé hrany dolů jsou *senzorový model* $\mathbf{P}(E_t | X_t)$, který každý stav spojuje s jeho pozorováním (deštník). Šedé uzly jsou pozorované, modré skryté. Stacionarita znamená, že obě tabulky jsou ve všech řezech stejné.

5.3 Čtyři základní inferenční úlohy

Nad takovým modelem se ptáme na čtyři typické otázky. Liší se tím, *který* stav nás zajímá vůči času, do něhož máme pozorování.

Definice – Filtrace, predikce, vyhlazování, nejpravděpodobnější průchod

Mějme pozorování $e_{1:t}$ do času t . Pak:

- **Filtrace** (filtering): rozdělení *nynějšího* stavu, $\mathbf{P}(X_t | e_{1:t})$ – „kde jsem teď?“
- **Predikce** (prediction): rozdělení *budoucího* stavu, $\mathbf{P}(X_{t+k} | e_{1:t})$ pro $k > 0$ – „kde budu?“
- **Vyhlazování** (smoothing): rozdělení *minulého* stavu, $\mathbf{P}(X_k | e_{1:t})$ pro $0 \leq k < t$ – „kde jsem“

byl?“

- **Nejpravděpodobnější průchod** (most likely explanation): *posloupnost* stavů, která nejpravděpodobněji vygenerovala pozorování, $\operatorname{argmax}_{x_{1:t}} P(x_{1:t} | e_{1:t})$ – „kudy jsem šel?“

💡 **Intuice.** Rozdíl mezi nimi je, jaký díl pozorování o daném stavu „ví“. *Filtrace* odhaduje stav z pozorování *do* tohoto okamžiku; je to běžný režim agenta v reálném čase. *Predikce* jde dál do budoucnosti bez nových dat (po jisté době konverguje k *stacionárnímu rozdělení* procesu). *Vyhlazování* se ohlíží zpět a minulý stav přehodnocuje i podle *pozdějších* pozorování, takže je přesnější než filtrace v témže okamžiku. *Nejpravděpodobnější průchod* hledá nejlepší *celou cestu* – a pozor, to **není** totéž co vzít v každém čase nejpravděpodobnější vyhlazený stav (po sobě jdoucí lokálně nejlepší stavy nemusí tvořit přípustnou ani nejpravděpodobnější trajektorii).

5.4 Filtrace a dopředná rovnice

Klíčová vlastnost filtrace: lze ji počítat *rekurzivně*, bez vracení se k celé historii. Udržujeme aktuální odhad a jen ho aktualizujeme novým pozorováním (*rekurzivní odhad*). Hledáme funkci f takovou, že

$$P(X_{t+1} | e_{1:t+1}) = f(e_{t+1}, P(X_t | e_{1:t})).$$

★ Věta – Dopředná rovnice filtrace

Označme *dopřednou zprávu* (forward message) $f_{1:t} = P(X_t | e_{1:t})$. Platí rekurze

$$P(X_{t+1} | e_{1:t+1}) = \alpha \underbrace{P(e_{t+1} | X_{t+1})}_{\text{aktualizace senzorem}} \sum_{x_t} \underbrace{P(X_{t+1} | x_t) P(x_t | e_{1:t})}_{\text{predikce z minula}}$$

s inicializací $f_{1:0} = P(X_0)$, kde α je normalizační konstanta (aby vyšel pravděpodobnostní vektor). Zkráceně $f_{1:t+1} = \alpha \text{ FORWARD}(f_{1:t}, e_{t+1})$.

💡 **Intuice.** Krok rekurze má dvě fáze. Nejprve **predikce**: starý odhad $P(x_t | e_{1:t})$ protáhneme přechodovým modelem do času $t + 1$ (suma přes x_t). Pak **aktualizace**: výsledek převážíme tím, jak dobře každý stav vysvětluje nové pozorování e_{t+1} (násobení $P(e_{t+1} | X_{t+1})$), a znormalizujeme. Odvození využije Bayesovo pravidlo, senzorový Markovův předpoklad a podmiňování přes x_t ; jde o výpočet, ne o důkaz k memorování.

Predikce. Predikci dostaneme jako filtraci *bez* přidávání nových pozorování – jen opakovaně aplikujeme přechodový model:

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

5.5 Vyhlazování a zpětná rovnice

Vyhlazování $P(X_k | e_{1:t})$ pro $k < t$ kombinuje pozorování *před* časem k (dopředná zpráva $f_{1:k}$) a *po* něm (zpětná zpráva, backward message). Rozštěpíme pozorování na $e_{1:k}$ a $e_{k+1:t}$:

$$P(X_k | e_{1:t}) = \alpha \underbrace{P(X_k | e_{1:k})}_{f_{1:k}} \underbrace{P(e_{k+1:t} | X_k)}_{b_{k+1:t}} = \alpha f_{1:k} \times b_{k+1:t},$$

kde $\times$ je po složkách. Backward zpráva se počítá rekurzí *zprava doleva*:

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

zkráceně $b_{k+1:t} = \text{BACKWARD}(b_{k+2:t}, e_{k+1})$ s inicializací $b_{t+1:t} = \mathbf{1}$ (prázdná evidence). Algoritmus **forward-backward** provede jeden průchod dopředu (uloží všechny $f_{1:k}$) a jeden zpět (násobí backward zprávou), čímž vyhladí celou posloupnost v čase $O(t)$.

⚙️ Algoritmus – Forward-backward (vyhlazování celé posloupnosti)

```

FORWARD-BACKWARD(ev, prior):
    fv[0] ← prior                                ▷ forward zpráva
    for i = 1 to t: fv[i] ← FORWARD(fv[i-1], ev[i])
    b ← vektor samých 1                          ▷ backward zpráva
    for i = t downto 1:
        sv[i] ← NORMALIZE(fv[i] × b)
        b ← BACKWARD(b, ev[i])
    return sv                                    ▷ vyhlazené odhady pro 1, ..., t

```

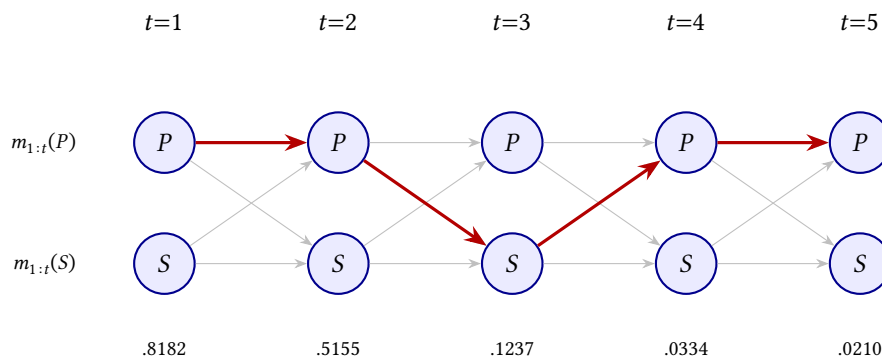
i Poznámka. Pro vyhlazování s *pevným zpožděním* d (online: vydávej vyhlazený odhad X_{t-d} , jakmile dorazí e_t) existuje algoritmus FIXED-LAG-SMOOTHING, který udržuje forward zprávu a maticovou transformaci pro posledních d kroků, takže pracuje v konstantním čase i paměti na krok (nezávisle na celkové délce sekvence).

5.6 Nejpravděpodobnější průchod: Viterbi

Najít nejpravděpodobnější celou cestu $\arg\max_{x_{1:t}} P(x_{1:t} | e_{1:t})$ řeší **Viterbiho algoritmus**. Je analogický filtraci, jen *sumu nahradí maximem*: pravděpodobnost nejlepší cesty končící ve stavu X_{t+1} je

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

kde $m_{1:t}(x) = \max_{x_1, \dots, x_{t-1}} P(x_1, \dots, x_{t-1}, X_t=x | e_{1:t})$ je pravděpodobnost nejlepší cesty do stavu x v čase t . Algoritmus jde mříží (trellis) zleva doprava, pro každý stav si pamatuje nejlepšího předchůdce (ukazatel zpět), a po dosažení času t *zpětně* cestu rekonstruuje od $\arg\max$ u.



pozorování: D, D, N, D, D

Viterbiho mříž (trellis) pro svět deštníku: dva stavy $\{P, S\}$ v každém čase. Tenké šedé hrany jsou všechny možné přechody, **tlustá červená cesta** je nejpravděpodobnější průchod (zde P, P, S, P, P) pro pozorování D, D, N, D, D . Pod každým sloupcem je hodnota nejlepší cesty do stavu na červené cestě (maxima $m_{1:t}$, odpovídají standardnímu příkladu z AIMA/slídů). Na rozdíl od vyhlazování (které pro každý čas vrací *marginální* rozdělení nezávisle) Viterbi vrací jednu *souvislou* trajektorii s ukazateli zpět.

5.7 Skryté Markovovy modely (HMM) a maticová formulace

Definice – Skrytý Markovův model (HMM)

Skrytý Markovův model (HMM) je Markovův model, v němž je stav procesu popsán *jedinou* diskretní náhodnou veličinou X_t (a je i *jediná* pozorovací veličina E_t). Nechť X_t nabývá hodnot z $\{1, \dots, S\}$. HMM je dán:

- priorem $\mathbf{P}(X_0)$,
- přechodovým modelem $\mathbf{P}(X_t | X_{t-1})$ – matice $\mathbf{T}$ velikosti $S \times S$, $\mathbf{T}(i, j) = P(X_t=j | X_{t-1}=i)$,
- sensorovým modelem $\mathbf{P}(E_t | X_t)$, který pro pozorovanou hodnotu e_t zapíšeme jako *diagonální* matici $\mathbf{O}_t$ s $\mathbf{O}_t(i, i) = P(E_t=e_t | X_t=i)$.

Omezení na jednu stavovou veličinu se vyplatí: všechny čtyři úlohy se zapíší jako *maticové operace*. Pokud dopřednou zprávu $f_{1:t}$ bereme jako sloupcový vektor, pak

$$f_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top f_{1:t} \quad (\text{filtrace}), \quad b_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} b_{k+2:t} \quad (\text{vyhlazování}).$$

Násobení $\mathbf{T}^\top$ provede predikční krok (suma přes předchozí stavy), násobení diagonální $\mathbf{O}_{t+1}$ provede aktualizaci senzorem. Vyhlažování v jedné formuli je $\mathbf{P}(X_k | e_{1:t}) = \alpha f_{1:k} \times b_{k+1:t}$.

5.8 HMM versus dynamické Bayesovské sítě (DBN)

Definice – Dynamická Bayesovská síť (DBN)

Dynamická Bayesovská síť (DBN) je Bayesovská síť reprezentující časový pravděpodobnostní model, v níž jsou proměnné a hrany *přesně replikovány z řezu na řez*. Díky tomu stačí popsat *jeden* řez: prior $\mathbf{P}(X_0)$, přechodový model $\mathbf{P}(X_1 | X_0)$ a sensorový model $\mathbf{P}(E_1 | X_1)$. Každá stavová proměnná má rodiče buď ve stejném, nebo v předchozím řezu (Markovův předpoklad).

 **Intuice.** Vztah HMM a DBN je tentýž jako mezi *plně tabelovaným sdruženým rozdělením* a *Bayesovskou sítí*: oba popisují totéž, ale DBN *strukturovaně* (více menších CPT pro jednotlivé proměnné řezu), zatímco HMM *monoliticky* (jedna veličina, jedna velká tabulka). Každý HMM je speciální případ DBN a naopak každou DBN lze zakódovat jako HMM tak, že vytvoříme *jednu* veličinu, jejíž hodnoty jsou n -tice hodnot stavových proměnných DBN. To je ale draho:

 **Poznámka.** [Proč na struktuře záleží] DBN s 20 booleovskými stavovými proměnnými, každá se třemi rodiči, má přechodový model o $20 \cdot 2^3 = 160$ pravděpodobnostech. Tatáž věc jako HMM má *jednu* veličinu s 2^{20} hodnotami a přechodovou matici o $2^{20} \times 2^{20} \approx 10^{12}$ číslech. HMM tedy potřebuje řádově víc paměti a inference je výrazně dražší. **Závěr:** HMM je elegantní pro *málo* stavů (maticové vzorce), DBN se vyplatí, jakmile stav rozpadá na *mnoho* dílčích proměnných. (Pro DBN existuje i přibližná inference *částicovým filtrem*, který populaci vzorků převáží podle evidence.)

5.9 Řešený příklad SZZ

Příklad – SZZ léto 2023 č. 5: Skrytý Markovův model (deštník)

(Reálná otázka SZZ: léto 2023, č. 5 – Skrytý Markovův model (HMM))

Zadání. Definujte HMM a Markovův předpoklad; vysvětlete rozdíl mezi filtrací, predikcí a vyhlazováním. Počasí ve dni i je $X_i \in \{P = \text{déšť}, S = \text{slunečno}\}$, pozorujeme jen, zda kolega přinesl deštník $E_i \in \{D, N\}$. Přechod $P(X_i=P | X_{i-1}=P) = 0,7$, $P(X_i=P | X_{i-1}=S) = 0,3$; senzor $P(D | P) = 0,9$,

$P(D | S) = 0,2$; prior $P(X_0=P) = P(X_0=S) = 0,5$. Pomocí

$$\mathbf{P}(X_t | e_{1:t}) = \alpha \mathbf{P}(e_t | X_t) \sum_{x_{t-1}} \mathbf{P}(X_t | x_{t-1}) \mathbf{P}(x_{t-1} | e_{1:t-1})$$

určete $P(X_2=P | E_1=D, E_2=D)$.

Řešení.

Pojmy. HMM je Markovův model s jedinou skrytou stavovou veličinou X_t a jedinou pozorovací veličinou E_t , daný priorem, přechodovou maticí $\mathbf{T}$ a senzorovým modelem. *Markovův předpoklad:* X_t závisí jen na X_{t-1} (a E_t jen na X_t). *Filtrace* počítá $\mathbf{P}(X_t | e_{1:t})$ (nynější stav z dosavadních pozorování) – to je náš případ; *predikce* $\mathbf{P}(X_{t+k} | e_{1:t})$ míří do budoucna bez nových dat; *vyhlazování* $\mathbf{P}(X_k | e_{1:t})$, $k < t$, přehodnocuje minulý stav i podle pozdějších pozorování.

Použijeme dvoukrokovou filtraci. Forward zprávu píšeme jako vektor $\langle P, S \rangle$. Zaokrouhlujeme na 4 desetinná místa.

Inicializace. $f_0 = \mathbf{P}(X_0) = \langle 0,5; 0,5 \rangle$.

Den 1 – predikce. Protáhneme přechodovým modelem:

$$P(X_1=P) = 0,7 \cdot 0,5 + 0,3 \cdot 0,5 = 0,5, \quad P(X_1=S) = 0,5.$$

Den 1 – aktualizace $E_1=D$. Vynásobíme $P(D | X_1) = \langle 0,9; 0,2 \rangle$:

$$\langle 0,9 \cdot 0,5; 0,2 \cdot 0,5 \rangle = \langle 0,45; 0,10 \rangle, \quad \text{součet } 0,55,$$

$$f_1 = \frac{1}{0,55} \langle 0,45; 0,10 \rangle = \langle 0,8182; 0,1818 \rangle.$$

Den 2 – predikce.

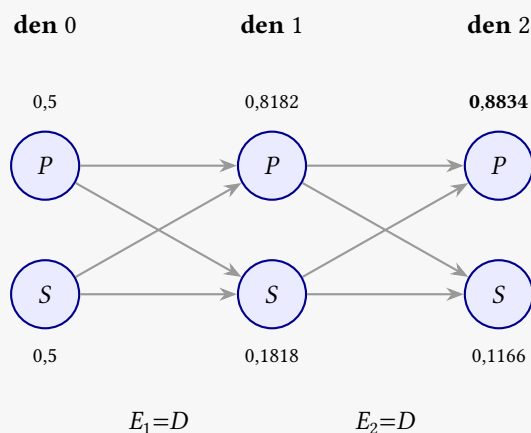
$$P(X_2=P) = 0,7 \cdot 0,8182 + 0,3 \cdot 0,1818 = 0,6273, \quad P(X_2=S) = 0,3 \cdot 0,8182 + 0,7 \cdot 0,1818 = 0,3727.$$

Den 2 – aktualizace $E_2=D$.

$$\langle 0,9 \cdot 0,6273; 0,2 \cdot 0,3727 \rangle = \langle 0,5645; 0,0745 \rangle, \quad \text{součet } 0,6391,$$

$$f_2 = \frac{1}{0,6391} \langle 0,5645; 0,0745 \rangle = \langle 0,8834; 0,1166 \rangle.$$

Odpověď. $P(X_2=P | E_1=D, E_2=D) \approx 0,883$ (tj. asi 88,3 %). Dva po sobě jdoucí deštníky výrazně posílily víru v déšť (z priorních 50 % na téměř 90 %).



Dvoukroková filtrace pro svět deštníku. Pod každým uzlem je vyfiltrovaná pravděpodobnost $P(X_t | e_{1:t})$ po aktualizaci pozorováním $E_t=D$. Hodnota pro P roste $0,5 \rightarrow 0,8182 \rightarrow \mathbf{0,8834}$: každý pozorovaný deštník posiluje víru v déšť.

V příklady/ je u této otázky původní zadání s tabulkami (sada léto 2023; originál byl v PDF bez náčrtu řešení, výpočet výše je ověřen numericky).

6 Automatické plánování (dopředné a zpětné plánování)

Plánování je hledání posloupnosti akcí, která agenta dovede k jeho cíli. V principu jde o tutéž úlohu jako prohledávání (sekce 1), ale s jedním zásadním rozdílem v *reprezentaci stavu*. Prohledávání pracovalo se stavem *atomickým* – nedělitelným černým boxem, takže heuristiku bylo nutno vymyslet zvlášť pro každou doménu. *Klasické (faktorové) plánování* naproti tomu reprezentuje stav *strukturovaně*, jako množinu pravdivých tvrzení (atomů) o světě, a akce popisuje obecnými *operátory* s předpoklady a efekty. Z této struktury lze automaticky odvodit *doménově nezávislé* heuristiky – tytéž pro libovolnou doménu. To je hlavní motivace celého formalismu.

💡 Intuice. Faktorová reprezentace je nutnost už z důvodu velikosti. Barták uvádí příklad logistiky: 5 lokací, 3 hromady na lokaci, 100 kontejnerů a 3 roboty dají řádově 10^{277} stavů – o 10^{190} víc, než je odhadovaný počet částic ve vesmíru. Takový prostor nelze vypsát ani uložit; lze ho jen *implicitně* generovat aplikací operátorů a ořezávat heuristikou, kterou si plánovač sám odvodí ze struktury operátorů.

Pozn.: *situační kalkulus* a *problém rámce* (plánování pomocí logické inference v predikátové logice) patří do reprezentace znalostí a jsou probrány v sekci 5. Zde se věnujeme *klasickému plánování* s faktorovou reprezentací, jak je formuluje učebnice AIMA i slidy přednášejícího.

6.1 Stavy, atomy a cíle

Stav světa popíšeme tím, co v něm *platí*. Základním stavebním kamenem je *konkretizovaný atom*, tj. predikát aplikovaný na konkrétní konstanty (objekty), bez proměnných – například $\text{na}(r_1, \text{loc}_2)$ („robot r_1 je na lokaci loc_2 “).

Definice – Stav, fluenty a rigidní atomy, předpoklad uzavřeného světa

Stav s je **množina konkretizovaných atomů** (bez proměnných). Atomů konečně mnoho $\Rightarrow$ stavů je *konečně* mnoho.

- **Fluent** (proměnný atom) – atom, jehož pravdivost se mezi stavy mění; jen fluenty smí měnit akce. Příklad: $\text{na}(r_1, \text{loc}_2)$. (Obdoba fluentů ze situačního kalkulu v sekci 5, zde bez argumentu situace.)
- **Rigidní atom** – atom, který platí ve všech stavech stejně (akce ho nemění). Příklad: $\text{soused}(\text{loc}_1, \text{loc}_2)$.

Platí **předpoklad uzavřeného světa** (closed-world assumption): *atom, který v s není uveden, ve stavu s neplatí*. Stav tedy vyjmenovává právě pravdivé atomy a vše ostatní je nepravda.

Definice – Cíl a jeho splnění

Cíl g je **množina konkretizovaných literálů** (atom nebo jeho negace). Rozložíme ho na kladnou a zápornou část:

$$g^+ = \{\text{atomy, které mají v cíli platit}\}, \quad g^- = \{\text{atomy, jejichž negace je v cíli (mají neplatit)}\}.$$

Stav s **splňuje** cíl g právě tehdy, když

$$g^+ \subseteq s \quad \wedge \quad g^- \cap s = \emptyset,$$

tj. všechny požadované atomy ve stavu jsou a žádný zakázaný tam není. Cíl typicky popisuje *množinu* cílových stavů (nemusí předepisovat hodnotu každého atomu).

6.2 Operátory a akce

Definice – Operátor

Operátor (schéma akce) o je trojice

$$o = (\text{name}(o), \text{precond}(o), \text{effects}(o)) :$$

- $\text{name}(o) = n(x_1, \dots, x_k)$ – jméno operátoru (n je jednoznačný symbol) a jeho *parametry* (proměnné). Musí obsahovat *všechny* proměnné vystupující v definici operátoru.
- $\text{precond}(o)$ – *předpoklady*: literály, které musí ve stavu platit, aby byl operátor použitelný.
- $\text{effects}(o)$ – *efekty*: literály, které po provedení začnou platit (kladný efekt) nebo přestanou platit (záporný efekt). V efektech smí být *jen fluenty* (rigidní atomy se nemění).

Operátor je *parametrizovaná* (lifted) reprezentace: popisuje akci obecně, nezávisle na konkrétních objektech.

Definice – Akce

Akce je **plně konkretizovaný operátor**: za parametry $x_1, \dots, x_k$ dosadíme konstanty. Z jednoho operátoru tak vznikne mnoho akcí. Pro akci a značíme $\text{precond}^+(a)$, $\text{precond}^-(a)$ kladné a záporné předpoklady (atomy, které mají, resp. nemají platit) a podobně $\text{effects}^+(a)$, $\text{effects}^-(a)$.

Příklad – Operátory světa kostek (blockworld)

Robotická ruka přemísťuje kostky a, b, c . Predikáty: $\text{ontable}(x)$ (kostka na stole), $\text{on}(x, y)$ (kostka x na y), $\text{clear}(x)$ (na x nic neleží, lze ji vzít), $\text{holding}(x)$ (ruka drží x), handempty (ruka je prázdná). Čtyři operátory:

operátor	precond	effects
$\text{unstack}(x, y)$	$\text{on}(x, y), \text{clear}(x), \text{handempty}$	$\neg \text{on}(x, y), \neg \text{clear}(x), \text{clear}(y), \neg \text{handempty}, \text{holding}(x)$
$\text{stack}(x, y)$	$\text{holding}(x), \text{clear}(y)$	$\neg \text{holding}(x), \neg \text{clear}(y), \text{on}(x, y), \text{clear}(x), \text{handempty}$
$\text{pickup}(x)$	$\text{ontable}(x), \text{clear}(x), \text{handempty}$	$\neg \text{ontable}(x), \neg \text{clear}(x), \neg \text{handempty}, \text{holding}(x)$
$\text{putdown}(x)$	$\text{holding}(x)$	$\neg \text{holding}(x), \text{ontable}(x), \text{clear}(x), \text{handempty}$

Dosazením konstant vznikne akce, např. $\text{unstack}(a, c)$: použitelná, je-li a na c , a je volná a ruka prázdná; efektem ruka uchopí a a c se uvolní.

6.3 Použitelnost akce, výsledek a relevance

Definice – Použitelnost a výsledek akce

Akce a je **použitelná** ve stavu s právě tehdy, když

$$\text{precond}^+(a) \subseteq s \quad \wedge \quad \text{precond}^-(a) \cap s = \emptyset,$$

tj. všechny kladné předpoklady ve stavu jsou a žádný záporný v něm není. Výsledek aplikace použitelné akce a na stav s je stav

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

nejprve odstraníme atomy, které mají přestat platit, pak přidáme ty, které mají začít. (Pořadí dává smysl: nejdřív „vymaž“, pak „připiš“.)

Použitelnost a výsledek řídí pohyb *vpřed*: ze stavu s generujeme nástupce $\gamma(s, a)$ pro každou použitelnou akci. Pro pohyb *vzad* (od cíle) potřebujeme duální pojmy – které akce mohly vést k danému cíli

a jaký stav jim musel předcházet.

Definice – Relevance akce a regresní množina

Akce a je **relevantní** pro cíl g právě tehdy, když:

- *přispívá k cíli*: $g \cap \text{effects}(a) \neq \emptyset$ (aspoň jeden efekt je v cíli žádaný), a zároveň
- *efekty nejsou v rozporu s cílem*: $g^- \cap \text{effects}^+(a) = \emptyset \wedge g^+ \cap \text{effects}^-(a) = \emptyset$ (akce neudělá pravdivým nic zakázaného ani nezruší nic žádaného).

Pro relevantní akci a je **regresní množina** cíle g množina (pod)cílů, které musely platit *před* provedením a , aby po něm platil g :

$$\gamma^{-1}(g, a) = (g \setminus \text{effects}(a)) \cup \text{precond}(a).$$

(Z cíle odebereme to, co a sama zařídí, a přidáme předpoklady, které a vyžaduje.)

6.4 Plánovací doména, problém a plán

Definice – Plánovací doména, problém, plán

Sada operátorů O tvoří *doménový model* (popisuje schopnosti agenta i použité predikáty). **Plánovací problém** je trojice

$$P = (O, s_0, g),$$

kde O je doménový model, s_0 je *počáteční stav* (dodává konkrétní konstanty – objekty) a g je *cíl* (množina konkretizovaných literálů). **Plán** je posloupnost akcí $\langle a_1, a_2, \dots, a_k \rangle$. Plán p je **řešením** problému P právě tehdy, když je celý proveditelný z s_0 (každá akce je použitelná v okamžiku své aplikace) a výsledný stav $\gamma^*(s_0, p)$ splňuje cíl g , kde $\gamma^*(s_0, p) = \gamma(\dots \gamma(\gamma(s_0, a_1), a_2) \dots, a_k)$.

6.5 Dopředné a zpětné plánování

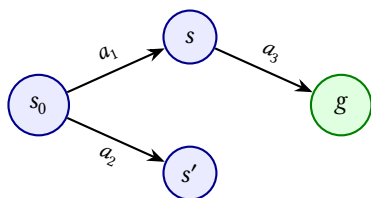
Plánovací problém řešíme *prohledáváním stavového prostoru* (sekce 1) – jen stavy a přechody teď nejsou atomické, nýbrž dané faktorovou reprezentací. Dva základní směry:

Definice – Dopředné (progrese) a zpětné (regrese) plánování

- **Dopředné plánování (progrese)**: prohledáváme *od počátečního stavu s_0 k cíli*. V každém uzlu (stavu s) generujeme nástupce $\gamma(s, a)$ pro všechny akce a *použitelné* v s ; končíme, jakmile stav splní g .
- **Zpětné plánování (regrese)**: prohledáváme *od cíle g zpět k s_0* . V každém uzlu (pod)cíli generujeme předchůdce $\gamma^{-1}(g, a)$ pro akce a *relevantní* pro g ; končíme, jakmile počáteční stav s_0 splňuje dosažený podcíl.

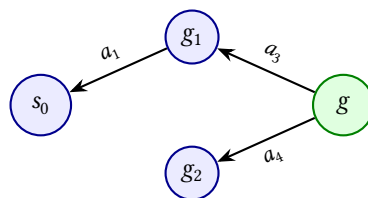
Intuice. **Dopředné** plánování pracuje s konkrétními (úplnými) stavy, ale často má velký *faktor větvení*: ve stavu může být použitelných mnoho akcí, z nichž většina k cíli vůbec nesměruje. Neinformované prohledávání proto zvládne jen malé úlohy; v praxi se nasazuje *informované* (A^*) s dobrou heuristikou.

Zpětné plánování zkoumá jen akce *relevantní* pro cíl, takže má obvykle *menší faktor větvení*. Daní je, že pracuje s *množinami stavů* (částečnými podcíli), ne s jednotlivými stavy – je proto těžší pro ně vymyslet dobré heuristiky. Oba směry jsou korektní; volba je věcí toho, kde je prostor užší.



dopředné (progrese)

$s \rightarrow \gamma(s, a)$, akce *použitelné*



zpětné (regrese)

$g \rightarrow \gamma^{-1}(g, a)$, akce *relevantní*

Dva směry plánování. Vlevo dopředné: z konkrétního počátečního stavu s_0 aplikujeme *použitelné* akce ($s \mapsto \gamma(s, a)$) a hledáme stav splňující cíl g . Vpravo zpětné: od cíle g aplikujeme *relevantní* akce regrese ($g \mapsto \gamma^{-1}(g, a)$), čímž vznikají postupně podcíle, dokud nějaký nesplní s_0 . Větvení vpravo bývá menší (jen relevantní akce), ale uzly jsou *množiny* stavů.

6.6 Plánovací heuristiky

Aby informované prohledávání (A^*) bylo efektivní, potřebuje heuristiku $h(s)$ odhadující „vzdálenost“ stavu s k cíli. Síla faktorové reprezentace je v tom, že heuristiku lze odvodit *automaticky a doménově nezávisle* – vyřešením *relaxovaného* problému (z původního odebereme některá omezení, čímž se zjednoduší). Délka řešení relaxace je dolním odhadem délky skutečného řešení, tedy *přípustnou* heuristikou pro A^* .

Definice – Relaxace: ignoruj předpoklady / ignoruj záporné efekty

- **Ignoruj předpoklady** (ignore preconditions): každá akce je vždy použitelná (vypustíme precondition). Úloha se redukuje na výběr *nejmenší množiny akcí pokrývajících cílové atomy* (problém množinového pokrytí, set cover). Pozor: tato relaxace zanedbává i to, že akce mohou rušit efekty jiných akcí.
- **Ignoruj záporné efekty** (ignore delete lists): z akcí vypustíme záporné efekty (effects^-). Žádná akce pak neníčí předpoklady jiné, takže množina pravdivých atomů jen *monotónně roste*. Lze tedy postupně přidávat všechny použitelné akce, dokud nejsou pokryty všechny cílové atomy; nejmenší taková množina dá heuristiku.

Intuice. Obě relaxace činí úlohu snáz řešitelnou, a protože odebrání omezení nemůže řešení *prodloužit*, dávají *přípustný* (nikdy nenadhodnocující) odhad. To je přesně podmínka, za níž A^* najde optimální plán (sekce 1). Klíčové je, že tyto recepty fungují pro *libovolnou* doménu popsanou operátory – proto „doménově nezávislé“ heuristiky. (Plánovací heuristiky požadavky okruhu výslovně nejmenují; uvádíme princip a obě základní relaxace.)

6.7 Řešený příklad SZZ

Příklad – SZZ jaro 2023 č. 5: Plánování (robot a krabice)

(Reálná otázka SZZ: jaro 2023, č. 5 – Plánování (robot, krabice))

Zadání. Robot v mřížce 4×4 umí akce **Jdi** (na sousední volné políčko; nese-li krabici, navíc nesmí na políčko s jinou krabicí), **Seber** (krabici na své pozici, jen nedrží-li už jinou) a **Polož** (nesenou krabici na svou pozici). Stav popisují predikáty $\text{soused}(X, Y)$ (z X lze přejít na Y , na Y není zeď), $\text{na}(X, K)$ (na políčku X je krabice K) a $\text{robot}(X)$ (robot je na X). Zdi (neprůchozí pole): **B1**, **B3**, **B4**, **C3**. Počátek: robot R na **C1**, krabice A na **A1**, krabice B na **C2**. Cíl: dostat krabici A na pole **D4** (hvězdička). Úkoly: (1) formalizovat jako plánovací problém (operátory, počáteční a cílový stav); (2) popsat dopředné a zpětné plánování; (3) napsat konkrétní řešící plán.

(1) **Formalizace.**

Konstanty. Políčka $A1, \dots, D4$ (16 jmen), krabice A, B . *Predikáty (fluenty):* $\text{robot}(X)$, $\text{na}(X, K)$, navíc pomocné $\text{nese}(K)$ („robot nese krabici K “), $\text{volno}(X)$ („na políčku X neleží žádná krabice“) a volnaruka („robot nic nenese“); pomáhají vyjádřit, kam smí robot s krabicí vstoupit a zda může sbírat. *Rigidní predikát* $\text{soused}(X, Y)$ kóduje mapu (sousední dvojice obou volných polí); v počátečním stavu ho podle zadání nevypisujeme.

Operátory. Podle nápovědy potřebujeme *dvě* verze pohybu – robot bez krabice smí na libovolné volné sousední pole, robot s krabicí navíc jen na pole bez jiné krabice:

operátor	precond	effects
$\text{Jdi}(X, Y)$ (bez krabice)	$\text{robot}(X), \text{soused}(X, Y),$ volnaruka	$\neg \text{robot}(X), \text{robot}(Y)$
$\text{JdiSK}(X, Y, K)$ (nese K)	$\text{robot}(X), \text{soused}(X, Y),$ $\text{nese}(K), \text{volno}(Y)$	$\neg \text{robot}(X), \text{robot}(Y)$
$\text{Seber}(X, K)$	$\text{robot}(X), \text{na}(X, K), \text{volnaruka}$	$\neg \text{na}(X, K), \text{nese}(K), \neg \text{volnaruka}, \text{volno}(X)$
$\text{Poloz}(X, K)$	$\text{robot}(X), \text{nese}(K), \text{volno}(X)$	$\neg \text{nese}(K), \text{na}(X, K), \text{volnaruka}, \neg \text{volno}(X)$

Pomocný fluent volnaruka nahrazuje záporný předpoklad „nedrží žádnou krabici“ (faktorová reprezentace tak vystačí s kladnými předpoklady, jako handempty ve světě kostek). $\text{Seber}(X, K)$ uvolní pole X (efekt $\text{volno}(X)$, krabice z něj odešla do ruky); JdiSK jen přesune robota – nesená krabice K zůstává „v ruce“ přes fluent $\text{nese}(K)$ a na podlahu ji položí teprve Poloz (atom na tedy mění jen Seber a Poloz , ne pohyb). Předpoklad $\text{volno}(Y)$ u JdiSK brání vstupu s krabicí na pole, kde už leží jiná krabice.

Počáteční stav s_0 (vypisujeme jen měnící se fluenty, soused ne):

$$s_0 = \{ \text{robot}(C1), \text{volnaruka}, \text{na}(A1, A), \text{na}(C2, B) \} \cup \{ \text{volno}(X) : X \text{ volné}, X \neq A1, C2 \}.$$

Cíl $g = \{ \text{na}(D4, A) \}$ (jediný požadovaný literál; $g^+ = \{ \text{na}(D4, A) \}$, $g^- = \emptyset$).

(2) Dopředné a zpětné plánování. *Dopředné (progrese):* začneme v s_0 , generujeme nástupce $\gamma(s, a)$ jen pro akce *použitelné* v s (splnění předpoklady), prohledáváme stavový prostor vpřed (A^* s heuristikou, např. manhattanská vzdálenost krabice A k $D4$ plus vzdálenost robota ke krabicí) a končíme ve stavu, kde platí $\text{na}(D4, A)$. *Zpětné (regrese):* začneme cílem $g = \{ \text{na}(D4, A) \}$; relevantní je akce, jejíž efekt $\text{na}(D4, A)$ přidává – to je $\text{Poloz}(D4, A)$. Regresní množina

$$\gamma^{-1}(g, \text{Poloz}(D4, A)) = (g \setminus \text{effects}) \cup \text{precond} = \{ \text{robot}(D4), \text{nese}(A), \text{volno}(D4) \}.$$

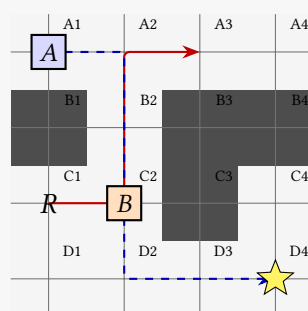
Z tohoto podcíle pokračujeme vzad (co muselo platit, aby tu robot s A stál na $D4$ – musel sem dojít akcí JdiSK apod.), dokud podcíl nesplní s_0 .

(3) Konkrétní plán. Topologie mřížky je zde klíčová. Volná pole jsou jen $A1, A2, A3, A4, B2, C1, C2, C4, D1, D2, D3, D4$ a kvůli zdem $B1, B3, B4, C3$ se mapa rozpadá na *dvě oblasti* – horní levou $\{A1, A2, A3, A4, B2\}$ a dolní $\{C1, C2, D1, D2, D3, D4, C4\}$ – spojené **jediným** přechodem $B2 \leftrightarrow C2$. Jenže na $C2$ leží krabice B , takže robot s *krabicí* A přes tento jediný most projít nemůže. Naivní plán „seber A a odnes ji do $D4$ “ proto **neexistuje**. Robot musí nejdřív *uvolnit most*: krabicí B z $C2$ odnést do horní oblasti, položit ji stranou, vrátit se pro A a teprve pak ji volným $C2$ pronést dolů k $D4$. Validní plán (ověřeno prohledáním stavového prostoru, je to nejkratší řešení, 16 akcí):

1. $\text{Jdi}(C1, C2)$
2. $\text{Seber}(C2, B)$
3. $\text{JdiSK}(C2, B2, B)$
4. $\text{JdiSK}(B2, A2, B)$
5. $\text{JdiSK}(A2, A3, B)$
6. $\text{Poloz}(A3, B)$
7. $\text{Jdi}(A3, A2)$

8. $Jdi(A2, A1)$
9. $Seber(A1, A)$
10. $JdiSK(A1, A2, A)$
11. $JdiSK(A2, B2, A)$
12. $JdiSK(B2, C2, A)$
13. $JdiSK(C2, D2, A)$
14. $JdiSK(D2, D3, A)$
15. $JdiSK(D3, D4, A)$
16. $Poloz(D4, A)$

Krabici B jsme odložili na $A3$ (stačilo by kterékoli volné pole horní oblasti mimo trasu). Výsledný stav obsahuje na $(D4, A)$, cíl je splněn.



Mapa úlohy (4×4 , řádky A nahoře po D dole). Tmavé buňky jsou zdi ($B1, B3, B4, C3$). Robot R startuje v $C1$, krabice A v $A1$, krabice B v $C2$, cílová hvězdička je v $D4$. Volná pole tvoří dvě oblasti spojené jediným mostem $B2 \leftrightarrow C2$, na němž stojí krabice B . Červeně: robot nejprve odkládí B z mostu do horní oblasti ($C1 \rightarrow C2 \rightarrow B2 \rightarrow A2 \rightarrow A3$). Modře čárkovaně: pak odnese A z $A1$ už volným mostem dolů do $D4$ ($A1 \rightarrow A2 \rightarrow B2 \rightarrow C2 \rightarrow D2 \rightarrow D3 \rightarrow D4$). (Schéma obou fází; proběhnou postupně, ne současně.)

i Poznámka. [Past v zadání] Tato otázka nemá v PDF oficiální náčrt řešení. Naivní „seber A , jdi do $D4$ “ selže – jediná spojnice obou oblastí ($B2 - C2$) je blokována krabicí B , kterou robot s nákladem nesmí minout. Validitu plánu (každá akce použitelná, žádný vstup s krabicí na obsazené pole, dosažení cíle) i jeho *minimalitu* (16 akcí) jsme ověřili prohledáním stavového prostoru. Bez tohoto kroku by „odnes A rovnou“ znělo věrohodně, ale bylo by chybné.

V příklady/ je u této otázky původní oříznutý obrázek mřížky z PDF (zadání bez oficiálního náčrtu řešení).

7 Markovovy rozhodovací procesy (Bellmanova rovnice, iterace hodnot a strategií, POMDP)

Doposud jsme předpokládali prostředí *deterministické*: každá akce měla jednoznačný výsledek a agent mohl celou posloupnost akcí naplánovat dopředu. Reálný svět je ale *stochastický* – robot, který chce jet vpřed, občas sklouzne stranou. Markovův rozhodovací proces (MDP) je formální model pro *sekvenční rozhodování v plně pozorovatelném, ale stochastickém prostředí*. Agent nemůže slepě provést pevnou posloupnost akcí (mohl by skončit jinde, než čekal); místo toho musí mít předpis, co dělat *v každém stavu*, kam se může dostat. Takovému předpisu říkáme *strategie* (policy) a úkolem je najít tu, která maximalizuje očekávaný (diskontovaný) užitek.

7.1 Rozhodování za nejistoty a princip MEU

Nejdřív si připomeňme, jak agent volí *jednu* akci, je-li její výsledek nejistý. Preference agenta zachycuje *užitková funkce* $U(s)$ – číslo vyjadřující, jak je stav s pro agenta žádoucí. Pokud akce a vede do stavů s'

s pravděpodobnostmi $P(s' | a, e)$ (za pozorování e), je její *očekávaný užitek*

$$EU(a | e) = \sum_{s'} P(s' | a, e) U(s').$$

Definice – Princip maximálního očekávaného užtku (MEU)

Racionální agent volí akci, která maximalizuje jeho očekávaný užitek:

$$a^* = \operatorname{argmax}_a EU(a | e) = \operatorname{argmax}_a \sum_{s'} P(s' | a, e) U(s').$$

Intuice. Princip MEU je jádro celé teorie rozhodování: *co agent věří* popisuje teorie pravděpodobnosti, *co agent chce* popisuje užtková funkce a *co má agent dělat* pak plyne z jejich spojení – vybrat akci s nejvyšším očekávaným užtkem. Užitek loterie $L = [p_1, S_1; \dots; p_n, S_n]$ (výsledky S_i s pravděpodobnostmi p_i) je $U(L) = \sum_i p_i U(S_i)$. MDP je rozšíření tohoto principu z jediného rozhodnutí na *posloupnost* rozhodnutí.

7.2 Markovův rozhodovací proces

Definice – Markovův rozhodovací proces (MDP)

MDP je sekvenční rozhodovací úloha pro plně pozorovatelné, stochastické prostředí s *Markovovým* přechodovým modelem a *aditivní* odměnou. Skládá se z:

- množiny **stavů** S (s počátečním stavem s_0) a množiny **akcí** $A(s)$ použitelných ve stavu s ;
- **přechodového modelu** $P(s' | s, a)$ – pravděpodobnost, že akce a ve stavu s vede do stavu s' ;
- **odměny** $R(s)$, kterou agent dostane ve stavu s (může být kladná i záporná, musí být omezená).

Markovova vlastnost: pravděpodobnost přechodu do s' závisí jen na aktuálním stavu s a akci a , *ne na historii* dřívějších stavů.

Poznámka. Odměna $R(s)$ je „krátkodobá“ (co získám tady a teď); užitek U bude „dlouhodobý“ součet odměn za celou budoucí trajektorii. Variantu s odměnou závislou i na akci a cíli, $R(s, a, s')$, lze na tuto formu vždy převést; my držíme jednodušší $R(s)$ ze slidů. V SZZ příkladu níže se odměna připisuje „při průchodu stavem“.

Diskontovaný užitek posloupnosti

Užitek závisí na celé posloupnosti navštívených stavů. Aby byl konečný i pro nekonečné trajektorie a aby se budoucí odměny počítaly méně než ty bezprostřední, používáme *diskontování* faktorem $\gamma \in [0, 1]$:

$$U([s_0, s_1, s_2, \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots = \sum_{t=0}^{\infty} \gamma^t R(s_t).$$

Intuice. *Diskontovaná* odměna říká, že odměna o jeden krok později má pro nás cenu jen γ -násobnou. Pro $\gamma < 1$ je součet *konečný* i na nekonečné trajektorii: je-li $R(s) \leq R_{\max}$, pak

$$U([s_0, s_1, \dots]) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = \frac{R_{\max}}{1 - \gamma}.$$

Diskontování má i ekonomickou interpretaci (peníze dnes jsou cennější než zítra) a modeluje nejistotu, zda se agent budoucnosti vůbec dožije. Pro $\gamma \rightarrow 1$ se blížíme k prostému součtu odměn.

Strategie a optimální strategie

V stochastickém prostředí *nelze* použít pevnou posloupnost akcí: agent může skončit v jiném stavu, než zamýšlel. Řešením MDP proto musí být předpis, co dělat *v každém* stavu.

Definice – Strategie a optimální strategie

Strategie (policy) π je funkce, která každému stavu s přiřadí akci $\pi(s) \in A(s)$. *Očekávaný užitek* strategie π při startu ve stavu s je

$$U^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(S_t) \right],$$

kde S_t je (náhodný) stav v čase t při řízení strategií π . **Optimální strategie** π^* maximalizuje očekávaný užitek:

$$\pi^* = \underset{\pi}{\operatorname{argmax}} U^\pi(s).$$

 **Intuice.** Optimální strategie nezávisí na počátečním stavu (pro diskontovaný užitek na nekonečném horizontu): kdyby se dvě optimální strategie startující z různých stavů setkaly ve společném stavu c , není důvod, aby se v c rozhodovaly různě. Proto má smysl mluvit o jediné optimální strategii π^* a o *skutečném užitku stavu*

$$U(s) = U^{\pi^*}(s).$$

Známe-li tyto pravé užitky, dostaneme optimální akci jednoduše principem MEU – vybereme akci, která maximalizuje očekávaný užitek následujícího stavu:

$$\pi^*(s) = \underset{a \in A(s)}{\operatorname{argmax}} \sum_{s'} P(s' | s, a) U(s').$$

7.3 Bellmanova rovnice

Mezi užitek stavu a užitek jeho sousedů platí přímý vztah: užitek stavu je jeho vlastní odměna plus diskontovaný očekávaný užitek toho, co bude následovat, zvolím-li nejlepší akci.

★ Věta – Bellmanova rovnice

Pro skutečné užitky $U(s) = U^{\pi^*}(s)$ platí pro každý stav s

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

Pro *koncový* (terminální) stav, z něž se nepokračuje, je $U(s) = R(s)$.

 **Intuice.** Bellmanova rovnice je rozklad dlouhodobého užitku na *okamžitou* odměnu $R(s)$ a *diskontovaný* užitek budoucnosti. Vnitřní suma $\sum_{s'} P(s' | s, a) U(s')$ je očekávaný užitek následníka po akci a (to je *EU* z principu MEU); maximem přes a vybíráme nejlepší akci. Pro n neterminálních stavů dostaneme soustavu n rovnic o n neznámých $U(s)$. Soustava je ale *nelineární* kvůli operátoru $\max$, takže ji obvykle neřešíme přímo, ale iterativně (iterace hodnot níže). Veličinu uvnitř maxima, $Q(s, a) = R(s) + \gamma \sum_{s'} P(s' | s, a) U(s')$, nazýváme *akční hodnotou*; pak $U(s) = \max_a Q(s, a)$ a $\pi^*(s) = \operatorname{argmax}_a Q(s, a)$.

3	0,812 →	0,868 →	0,918 →	+1
2	0,762 ↑	zeď	0,660 ↑	-1
1	0,705 ↑	0,655 ←	0,611 ←	0,388 ←
	1	2	3	4

Klasický *gridworld* 3×4 (AIMA): koncové stavy +1 (pole 4, 3) a -1 (pole 4, 2), zeď na poli 2, 2. Akce {nahoru, dolů, vlevo, vpravo} vyjde podle úmyslu s pravd. 0,8, jinak agenta vychýlí kolmo (0,1 + 0,1). Modré šipky ukazují optimální strategii, čísla jsou užitky $U(s)$ (pro $R(s) = -0,04$ v neterminálních polích, $\gamma = 1$). Každý užitek splňuje Bellmanovu rovnici $U(s) = R(s) + \gamma \max_a \sum_{s'} P(s' | s, a)U(s')$; např. ve startu (1, 1) je optimální jít *nahoru* (snaha o horní cestu k +1 při vyhýbání se -1).

7.4 Iterace hodnot (value iteration)

Bellmanova rovnice je základem algoritmu *iterace hodnot*. Protože je soustava nelineární (kvůli max), nepočítáme ji přímo, ale opakovaně aplikujeme Bellmanovu rovnici jako *aktualizační pravidlo* (Bellman update), dokud se užitky neustálí.

🔧 Algoritmus – Iterace hodnot (value iteration)

VALUE-ITERATION(mdp , ϵ):

▷ ϵ = maximální dovolená chyba užitku libovolného stavu

$U \leftarrow$ vektor nul (pro všechny stavy); $U' \leftarrow U$

repeat

$U \leftarrow U'$; $\delta \leftarrow 0$

▷ δ = největší změna užitku v tomto kole

for každý stav $s \in S$:

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

▷ Bellman update

if $|U'[s] - U[s]| > \delta$: $\delta \leftarrow |U'[s] - U[s]|$

until $\delta \leq \epsilon(1 - \gamma)/\gamma$

return U

💡 **Intuice.** Začneme s libovolnými (typicky nulovými) užitky a v každém kole každý stav *aktualizujeme* podle Bellmanovy rovnice z aktuálních užiteků sousedů. Operátor Bellmanova updatu je *kontrakce* (pro $\gamma < 1$ zmenšuje rozdíl mezi libovolnými dvěma odhady faktorem γ), takže iterace *konverguje* k jedinému pevnému bodu – pravým užitkům $U(s)$. Důležité je, že updaty pro všechny stavy v jednom kole čteme z *téhož* starého vektoru U (synchronní varianta). Ukončovací podmínka $\delta \leq \epsilon(1 - \gamma)/\gamma$ zaručí, že chyba užitku je nejvýš ϵ . Z konvergovaných užiteků pak jednorázově odečteme optimální strategii $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s' | s, a)U(s')$.

7.5 Iterace strategií (policy iteration)

Často nás zajímá jen *optimální strategie*, ne přesné užitky. Optimální akce se přitom ustálí dříve než užitky (na pořadí akcí stačí znát užitky přibližně). Toho využívá *iterace strategií*: střídá dva kroky – vyhodnocení strategie a zlepšení strategie – dokud se strategie přestane měnit.

🔧 Algoritmus – Iterace strategií (policy iteration)

POLICY-ITERATION(mdp):

$U \leftarrow$ vektor nul; $\pi \leftarrow$ libovolná počáteční strategie

repeat

▷ 1) vyhodnocení strategie (policy evaluation)

$U \leftarrow \text{POLICY-EVALUATION}(\pi, U, \text{mdp})$

▷ 2) zlepšení strategie (policy improvement)

```

beze_zmeny ← true
for každý stav  $s \in S$ :
     $a^* \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' | s, a) U[s']$ 
    if  $\sum_{s'} P(s' | s, a^*) U[s'] > \sum_{s'} P(s' | s, \pi[s]) U[s']$ :
         $\pi[s] \leftarrow a^*$ ; beze_zmeny ← false
until beze_zmeny
return  $\pi$ 

```

Intuice. **Vyhodnocení strategie** (policy evaluation): pro pevnou strategii π_i spočítáme její užitky U^{π_i} . Protože akce je daná, zmizí operátor max a Bellmanovy rovnice se stanou lineárními:

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

Pro malý počet stavů n je vyřešíme přímo (Gaussova eliminace) v čase $O(n^3)$; pro velké prostory uděláme pár kol zjednodušené iterace hodnot (bez max).

Zlepšení strategie (policy improvement): z užitků U^{π_i} spočítáme novou strategii principem MEU, $\pi_{i+1}(s) \leftarrow \operatorname{argmax}_a \sum_{s'} P(s' | s, a) U^{\pi_i}(s')$. Pokud se žádná akce nezměnila, je strategie optimální a končíme. Strategie se nikdy nezhoršuje a možných strategií je konečně mnoho, takže algoritmus skončí po konečně mnoha krocích.

Poznámka. **Iterace hodnot vs. iterace strategií.** Obě řeší tutéž úlohu. Iterace hodnot dělá v každém kole jeden levný Bellman update na stav (s max), ale potřebuje víc kol. Iterace strategií dělá méně kol, ale každé je dražší (v kroku vyhodnocení řeší lineární soustavu $O(n^3)$, případně sama iterací). Test optimality strategie – „nezmění-li krok zlepšení žádnou akci, je strategie optimální“ – je přímo *Bellmanovo kritérium optimality* a použijeme ho i v SZZ příkladu níže.

7.6 Částečně pozorovatelný MDP (POMDP)

V MDP agent vždy ví, ve kterém stavu je (plná pozorovatelnost), takže může provést akci $\pi(s)$ předepsanou pro daný stav. V reálném světě tomu tak často není: agent stav přímo nevidí, jen ho odhaduje z nepřesných pozorování.

Definice – Částečně pozorovatelný MDP (POMDP)

POMDP je jako MDP – má přechodový model $P(s' | s, a)$, akce $A(s)$ a odměnu $R(s)$ – navíc ale obsahuje **senzorový model** $P(e | s)$: pravděpodobnost pozorování (evidence) e ve stavu s . Agent neví, ve kterém stavu přesně je.

Intuice. Protože agent nezná svůj stav jistě, udržuje místo něj *stav víry* (belief state) b – pravděpodobnostní rozdělení přes všechny možné stavy. Po každé akci a pozorování b aktualizuje filtrací (jako u HMM). Klíčové pozorování: *optimální chování závisí jen na aktuálním stavu víry*, takže POMDP lze chápat jako (obyčejný) MDP nad *spojitým* prostorem stavů víry. Tím se ale úloha výrazně ztíží – stavů víry je nekonečně mnoho a techniky iterace hodnot je nutné upravit na spojitý případ (řeší se přes po částech lineární užitkové funkce, viz POMDP-VALUE-ITERATION). V praxi se rozhodnutí dělají rozvíjením budoucích posloupností akcí a pozorování dopředu (dynamická rozhodovací síť) a výběrem nejlepší větve, podobně jako algoritmus expektimax u stochastických her. Požadavky okruhu žádají jen *základní definici*: POMDP = MDP + senzorový model $P(e | s)$, řešený nad stavy víry.

7.7 Řešený příklad SZZ

Příklad – SZZ léto 2025 č. 29: Bellmanova rovnice užítu (MDP, iterace strategií)

(Reálná otázka SZZ: léto 2025, č. 29 – Bellmanova rovnice užítu (MDP, iterace strategií))

Zadání.

Robot se pohybuje v konečném stavovém prostoru. Pro každý stav s známe množinu akcí $A(s)$ a přechodový model $P(s' | s, a)$. Při průchodu stavem s dostane robot odměnu $R(s)$, užitek je diskontován faktorem $\gamma = 0,9$. Graf níže udává stavy, jejich odměny a dvě možné akce z každého neterminálního stavu. Zvolená akce vede do stavu daného svou hranou s pravděpodobností 0,8, jinak (0,2) do cíle druhé akce daného stavu. Úkoly: (1) napište rovnici pro maximální užitek $U(s)$; (2) napište soustavu rovnic pro vyhodnocení strategie $B \rightarrow C, C \rightarrow D, D \rightarrow F$; (3) jakým algoritmem ji vyřešíme; (4) jak ověříme, že tato volba dává maximální užitek pro všechny stavy; (5) jak najdeme akce maximalizující užitek pro všechny stavy.

Řešení.

(1) Bellmanova rovnice. Maximální užitek stavu je jeho odměna plus diskontovaný očekávaný užitek nejlepší akce:

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

s koncovými stavy $U(E) = R(E) = 5$ a $U(F) = R(F) = 6$.

(2) Soustava pro vyhodnocení strategie $B \rightarrow C, C \rightarrow D, D \rightarrow F$. Pevná strategie odstraní max, takže rovnice jsou *lineární*. U každého stavu vede zvolená akce do svého cíle s pravd. 0,8 a do cíle druhé akce s pravd. 0,2 (druhá akce: B má $i \rightarrow D, C$ má $i \rightarrow E, D$ má $i \rightarrow B$):

$$\begin{aligned} U(B) &= -3 + 0,9[0,8U(C) + 0,2U(D)], \\ U(C) &= 1 + 0,9[0,8U(D) + 0,2U(E)], \\ U(D) &= 1 + 0,9[0,8U(F) + 0,2U(B)], \\ U(E) &= 5, \quad U(F) = 6. \end{aligned}$$

(3) Algoritmus řešení. Jde o *vyhodnocení strategie* (policy evaluation). Soustava je lineární (tři rovnice o třech neznámých $U(B), U(C), U(D)$), takže ji vyřešíme *přímo* – Gaussovou eliminací v čase $O(n^3)$. Alternativně lze iterovat zjednodušený Bellman update bez max (iterace hodnot pro pevnou strategií), dokud se užítiky neustálí. Dosazením $U(E) = 5, U(F) = 6$ a vyřešením soustavy:

$$U(B) \approx 2,383, \quad U(C) \approx 6,039, \quad U(D) \approx 5,749.$$

(Kontrola dosazením, např. $U(D) = 1 + 0,9[0,8 \cdot 6 + 0,2 \cdot 2,383] = 1 + 0,9 \cdot 5,277 \approx 5,749$.) Pár kol iterace dává $U^{(1)} = (-3,0; 1,90; 5,32), U^{(2)} = (-0,67; 5,73; 4,78), U^{(3)} = (1,99; 5,34; 5,20), \dots$, postupně se ustaluje na hodnotách výše.

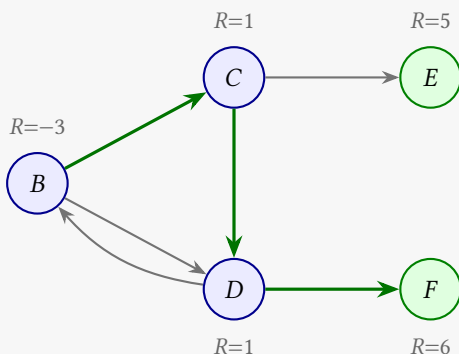
(4) Test optimality strategie. Pro každý stav ověříme *Bellmanovo kritérium optimality*: zda zvolená akce dosahuje $\max_a \sum_{s'} P(s' | s, a) U(s')$, tj. zda by krok *zlepšení strategie* (policy improvement) některou akci změnil. Spočítáme očekávaný užitek obou akcí v každém stavu (z užiteků výše):

- B : akce $\rightarrow C$ dá $0,8 \cdot 6,039 + 0,2 \cdot 5,749 = 5,981$; akce $\rightarrow D$ dá $0,8 \cdot 5,749 + 0,2 \cdot 6,039 = 5,807$. Lepší je $\rightarrow C$ (zvolená).
- C : akce $\rightarrow D$ dá $0,8 \cdot 5,749 + 0,2 \cdot 5 = 5,599$; akce $\rightarrow E$ dá $0,8 \cdot 5 + 0,2 \cdot 5,749 = 5,150$. Lepší je $\rightarrow D$ (zvolená).
- D : akce $\rightarrow F$ dá $0,8 \cdot 6 + 0,2 \cdot 2,383 = 5,277$; akce $\rightarrow B$ dá $0,8 \cdot 2,383 + 0,2 \cdot 6 = 3,106$. Lepší je $\rightarrow F$ (zvolená).

V žádném stavu by se akce nezměnila, takže zadaná strategie **je optimální** a spočtené užítiky jsou maximální možné. (Kdyby aspoň v jednom stavu vyšla jiná akce lépe, strategie by optimální nebyla.)

(5) Hledání optimálních akcí. Obecně použijeme *iteraci strategií* (policy iteration): střídáme *vyhodnocení strategie* (vyřeš lineární soustavu jako v bodě 3) a *zlepšení strategie* (v každém stavu

vezmi akci s nejvyšším očekávaným užtkem jako v bodě 4), dokud se strategie přestane měnit. Výsledná strategie je optimální pro všechny stavy. (Alternativně lze rovnou spustit *iteraci hodnot* s $\max$ a z konvergovaných užtků odečíst $\pi^*(s) = \operatorname{argmax}_a \sum_{s'} P(s' | s, a) U(s')$.) Zde už jsme v bodě 4 ověřili, že strategie $B \rightarrow C, C \rightarrow D, D \rightarrow F$ je optimální, takže iterace strategií by skončila hned.



tučné zelené hrany = strategie $B \rightarrow C, C \rightarrow D, D \rightarrow F$

Stavový graf MDP ze zadání: neterminální stavy $B(-3), C(1), D(1)$, terminální $E(5), F(6)$. Z každého neterminálního stavu vedou dvě akce (orientované hrany): $B \rightarrow C, B \rightarrow D; C \rightarrow D, C \rightarrow E; D \rightarrow F, D \rightarrow B$. Zvolená akce vede do svého cíle s pravd. 0,8, do cíle druhé akce s pravd. 0,2. Tučně zeleně je vyznačena vyhodnocovaná strategie $B \rightarrow C, C \rightarrow D, D \rightarrow F$, o níž výpočet ukázal, že je navíc optimální.

V příklady/ je u této otázky původní zadání a oříznutý stavový graf z PDF (originál byl bez náčrtu řešení; hodnoty výše jsme ověřili výpočtem).

8 Hry a teorie her (minimax, alfa-beta, Nashovo ekvilibrium, aukce)

Dosud jsme uvažovali jednoho agenta v prostředí, které se samo aktivně nesnaží jeho plánům bránit. *Teorie her* (obor ekonomie) naopak studuje prostředí s *více agenty*, jejichž rozhodnutí se navzájem podstatně ovlivňují. V umělé inteligenci nás nejčastěji zajímají *deterministické, tahové, dvouhráčové hry s nulovým součtem a úplnou informací* (šachy, dáma, go): pravidla jsou přesná, akcí je málo, hra se snadno reprezentuje, a přesto je těžké ji vyřešit. Vedle nich se dotkneme *jednotahových* (simultánních) her a jejich pojmů (dominantní strategie, Nashovo ekvilibrium, Pareto optimalita) a *mechanism designu* (návrh pravidel, zejména aukcí).

8.1 Herní strom a optimální strategie

Tahovou hru dvou hráčů, které tradičně značíme **MAX** a **MIN**, lze formálně chápat jako *prohledávání proti soupeři* (adversarial search) v *herním stromu*.

Definice – Hra jako prohledávací úloha

Hra je dána:

- **počátečním stavem** s_0 (výchozí pozice) a funkcí $\text{To-Move}(s)$, která určí hráče na tahu;
- množinou **akcí** $\text{ACTIONS}(s)$ a přechodovým modelem $\text{RESULT}(s, a)$ (výsledný stav);
- **testem koncovosti** $\text{IS-TERMINAL}(s)$ (je hra u konce?);
- **ohodnocovací (užitkovou) funkcí** $\text{UTILITY}(s, p)$, která koncovému stavu přiřadí číselnou výplatu hráče p (např. výhra +1, prohra -1, remíza 0).

Ve hře s *nulovým součtem* je výplata jednoho hráče opačná než druhého, takže stačí jediné číslo: užitek z pohledu MAX (který ho maximalizuje), zatímco MIN ho minimalizuje.

Intuice. Řešením hry *není jediná posloupnost akcí* (jako u prohledávání jednoho agenta), nýbrž **strategie**: tah ve výchozím stavu a pak tah na každou možnou odpověď soupeře. *Optimální strategie* vede k výsledku alespoň tak dobrému jako kterákoli jiná, hraje-li proti *neomylnému* soupeři. Herní strom střídá patra MAX a MIN; jeho listy jsou koncové stavy s konkrétní výplatou.

8.2 Minimax

Optimální strategii určíme z *minimaxové hodnoty* každého uzlu – užitku, který v daném stavu vznikne, hrají-li *oba hráči optimálně*.

Definice – Minimaxová hodnota

$$\text{MINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & n \text{ je koncový stav,} \\ \max_{s \in \text{succ}(n)} \text{MINIMAX}(s) & \text{na tahu je MAX,} \\ \min_{s \in \text{succ}(n)} \text{MINIMAX}(s) & \text{na tahu je MIN.} \end{cases}$$

Minimaxové rozhodnutí je akce v kořeni, která vede do následníka s nejvyšší minimaxovou hodnotou.

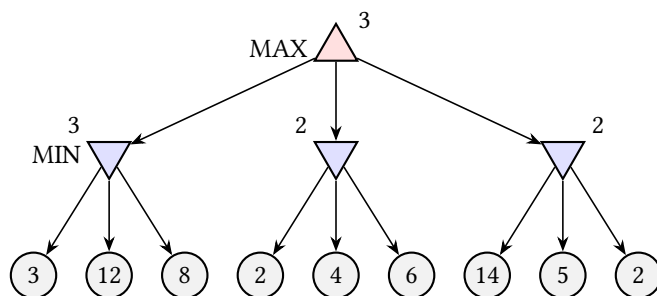
Hodnoty se počítají *zdola nahoru*: list dostane svou výplatu, vnitřní uzel MAX vezme maximum hodnot dětí, uzel MIN minimum. Algoritmus je v jádru prohledávání do hloubky celého stromu.

Algoritmus – Minimax (rekurzivně, návrat hodnoty i tahu)

```
MINIMAX-SEARCH(hra, stav):
    v, move ← MAX-VALUE(hra, stav)                                ▷ hodnota kořene a optimální tah
    return move

MAX-VALUE(hra, stav):
    if hra.IS-TERMINAL(stav): return hra.UTILITY(stav), null
    v ← -∞
    for každou akci a ze stavu:
        v2, _ ← MIN-VALUE(hra, RESULT(stav,a))
        if v2 > v: v, move ← v2, a                                ▷ lepší tah pro MAX
    return v, move

MIN-VALUE(hra, stav):
    if hra.IS-TERMINAL(stav): return hra.UTILITY(stav), null
    v ← +∞
    for každou akci a ze stavu:
        v2, _ ← MAX-VALUE(hra, RESULT(stav,a))
        if v2 < v: v, move ← v2, a                                ▷ lepší tah pro MIN
    return v, move
```



Propagace minimaxových hodnot. Uzly MIN (modré trojúhelníky dolů) berou minimum svých listů: $\min(3, 12, 8) = 3$, $\min(2, 4, 6) = 2$, $\min(14, 5, 2) = 2$. Kořen MAX (červený trojúhelník nahoru) bere maximum: $\max(3, 2, 2) = 3$. Minimaxové rozhodnutí je tah do levého následníka.

Poznámka. Minimax projde celý strom: časová složitost je $O(b^m)$, paměťová $O(bm)$ (b = počet tahů ve stavu, m = hloubka). Pro reálné hry je to nepoužitelné – šachový strom má kolem 35^{100} uzlů – a slouží jen jako základ praktičtějších algoritmů.

8.3 Alfa-beta prořezávání

K výpočtu *správného* minimaxového rozhodnutí není nutné prohlédnout každý uzel stromu. **Alfa-beta prořezávání** vrací stejnou hodnotu jako minimax, ale celé podstromy přeskočí, jakmile je jasné, že nemohou ovlivnit rozhodnutí v kořeni.

Definice – Meze α a β

Podél cesty od kořene udržujeme dvě meze:

- α = hodnota nejlepší (tj. nejvyšší) volby, kterou MAX dosud na cestě našel; je to *dolní mez* na hodnotu, kterou si MAX zajistí.
- β = hodnota nejlepší (tj. nejnižší) volby, kterou MIN dosud na cestě našel; je to *horní mez*.

Pravidlo řezu. V uzlu MIN s průběžnou hodnotou v : jakmile $v \leq \alpha$, zbylé následníky *nezkoumáme* (α -řez) – MAX nad námi už má lepší alternativu (α) a tudý nepůjde. Symetricky v uzlu MAX: jakmile $v \geq \beta$, nastává β -řez.

Algoritmus – Alfa-beta (úprava minimaxu o meze α, β)

ALPHA-BETA-SEARCH(hra, stav):

$v, \text{move} \leftarrow \text{MAX-VALUE}(\text{hra}, \text{stav}, -\infty, +\infty)$

return move

MAX-VALUE(hra, stav, α, β):

if hra.IS-TERMINAL(stav): **return** hra.UTILITY(stav), null

$v \leftarrow -\infty$

for každou akci a ze stavu:

$v_2, _ \leftarrow \text{MIN-VALUE}(\text{hra}, \text{RESULT}(\text{stav}, a), \alpha, \beta)$

if $v_2 > v$: $v, \text{move} \leftarrow v_2, a$; $\alpha \leftarrow \max(\alpha, v)$

if $v \geq \beta$: **return** v, move

▷ β -řez: MIN nad námi tudý nepůjde

return v, move

MIN-VALUE(hra, stav, α, β):

if hra.IS-TERMINAL(stav): **return** hra.UTILITY(stav), null

$v \leftarrow +\infty$

for každou akci a ze stavu:

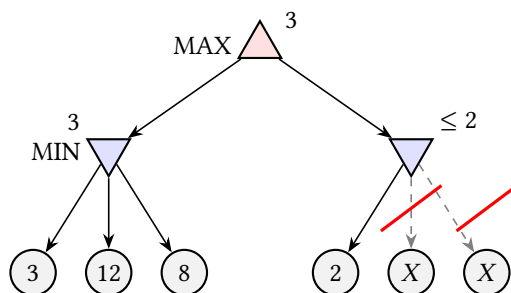
$v_2, _ \leftarrow \text{MAX-VALUE}(\text{hra}, \text{RESULT}(\text{stav}, a), \alpha, \beta)$

if $v_2 < v$: $v, \text{move} \leftarrow v_2, a$; $\beta \leftarrow \min(\beta, v)$

if $v \leq \alpha$: **return** v, move

▷ α -řez: MAX nad námi tudý nepůjde

return v, move



Levý uzel MIN dá $\min(3, 12, 8) = 3$, takže kořen MAX má $\alpha = 3$. Pravý uzel MIN nejprve vidí list 2; tím je jeho hodnota $\leq 2 \leq \alpha = 3$. MAX si už zajistil 3 a tato větev mu lepší výsledek nedá, proto se zbylé dva listy (X) *prořezou* (červené škrty) a vůbec se neprohlédnou. Výsledek kořene = 3 je stejný jako u čistého minimaxu.

i Poznámka. [Závislost na pořadí] Účinnost řezů silně závisí na *pořadí*, v jakém procházíme tahy: při *ideálním* uspořádání (nejlepší tah první) zkoumá alfa-beta jen $O(b^{m/2})$ uzlů místo $O(b^m)$, tedy zhruba dvojnásobnou hloubku za stejný čas. *Hodnota kořene a optimální tah jsou na pořadí nezávislé* – prořezává se jen práce, ne výsledek; špatné pořadí jen řezy promešká.

8.4 Nedokonalá rozhodnutí: ohodnocovací funkce a ořezání hloubky

I s alfa-beta nelze u reálných her vygenerovat celý strom. Proto prohledávání *utneme dříve* (v hloubce d) a v takto vzniklých „listech“ použijeme místo skutečné užitkové funkce **ohodnocovací (evaluation) funkci** $\text{EVAL}(s)$ – *odhad* očekávaného užitku pozice (přesně jako heuristika u A^*). Typicky se počítá z rysů stavu a kombinuje se lineárně, $\text{EVAL}(s) = w_1 f_1(s) + \dots + w_n f_n(s)$ (v šachu např. materiál: pěšec 1, jezdec/střelec 3, věž 5, dáma 9). Lineární tvar předpokládá, že příspěvky rysů jsou nezávislé; jinak je třeba nelineární kombinace.

i Poznámka. [Stochastické hry, expectiminimax] Hry s náhodným prvkem (kostky) modelujeme přidáním **uzlů náhody** (chance) do stromu. V nich se počítá *očekávaná* hodnota:

$$\text{EXPECTIMINIMAX}(n) = \sum_{s \in \text{succ}(n)} P(s) \cdot \text{EXPECTIMINIMAX}(s) \quad (\text{uzel náhody}),$$

zatímco v uzlech MAX/MIN se bere max/min jako dříve. U stochastických her musí být ohodnocovací funkce navržena *pečlivěji*: záleží na konkrétních hodnotách (kvůli vážení pravděpodobnostmi), ne jen na jejich pořadí.

8.5 Jednotahové (simultánní) hry

V *jednotahové hře* všichni hráči volí akci *naráz* (simultánně) a výsledek je dán touto kombinací akcí. Hru zadáme trojicí: *hráči*, jejich *akce* a *výplatní funkce*, která každé kombinaci akcí přiřadí užitek pro každého hráče. Výplaty zapisujeme do *výplatní matice*.

Definice – Strategie, dominance, Nash, Pareto

- **Čistá strategie** je deterministická volba jediné akce; **smíšená strategie** je náhodná volba akce podle pravděpodobnostního rozdělení.
- Strategie s hráče p (**striktně**) **dominuje** strategii s' , je-li výsledek s pro p *lepší* než výsledek s' *při každé* volbě ostatních hráčů. *Dominantní strategie* dominuje všechny ostatní strategie téhož hráče.
- Profil strategií je **Nashovo ekvilibrium**, jestliže *žádný* hráč si nepolepší jednostrannou změnou své strategie, drží-li ostatní svou. (Lokální optimum vůči jednostranným odchylkám.)
- Výsledek je **Pareto-dominován** jiným výsledkem, pokud by jej *všichni* hráči preferovali. Výsledek je **Pareto optimální**, neexistuje-li jiný výsledek, který by preferovali všichni (nelze polepšit jednomu, aniž si jiný pohorší).

💡 Intuice. Pojmy se snadno pletou. *Dominance* se týká **jedné strategie jednoho hráče** (je nejlepší bez ohledu na soupeře). *Nashovo ekvilibrium* se týká **profilu** (dvojice voleb) – je stabilní vůči jednostranné odchylce, ale *neříká nic o globální dobrotě*. *Pareto optimalita* se týká **výsledku** z hlediska *kolektivu* – nelze ho pro všechny zlepšit. Klíčové ponaučení věžňova dilematu: profil v dominantních strategiích (= Nash) může být Pareto-dominovaný, tedy pro všechny horší než jiný, nestabilní výsledek.

Příklad – Věžňovo dilema

Dva lupiči, Alice a Bob, jsou vyslýcháni odděleně. Každý se může rozhodnout *svědčit* (zradit druhého), nebo *mlčet*. Výplaty jsou roky vězení se znaménkem mínus (více let = horší); v políčku je nejdřív výplata Boba (řádky), pak Alice (sloupce):

	Alice: svědčí	Alice: mlčí
Bob: svědčí	−5, −5	0, −10
Bob: mlčí	−10, 0	−1, −1

Svědčit je **dominantní strategie** obou: ať soupeř dělá cokoli, svědčit je vždy lepší ($-5 > -10$, $0 > -1$). Profil (svědčí, svědčí) je proto Nashovo ekvilibrium (rovnováha v dominantních strategiích) s výplatou $(-5, -5)$. Jenže výsledek $(-1, -1)$ z oboustranného mlčení je *Pareto lepší* – oba by si polepšili. Ekvilibrium je tak **Pareto-dominováno** výsledkem, který racionální hráči nedosáhnou: jádro dilematu.

8.6 Smíšené strategie a maximová technika

Některé hry s nulovým součtem (např. hra na prsty „two-finger Morra“) nemají žádné ekvilibrium v čistých strategiích. Optimální *smíšenou* strategii lze najít **maximinovou technikou**: hru si představíme jako tahovou (druhý hráč zná tah prvního, je tedy zvýhodněn) a *minimaxem* spočítáme dolní a horní mez na užitek. Odhalí-li první hráč svou *smíšenou* strategii (rozdělení $[p : \text{akce}_1; (1 - p) : \text{akce}_2]$), druhý odpoví nejlepší *čistou* odpovědí; první proto volí p tak, aby maximalizoval svůj nejhorší výsledek. Pro Morru vychází optimum $[7/12; 5/12]$ s hodnotou hry $-1/12$ (hra zvýhodňuje jednoho z hráčů). Podstatný je pojem: *smíšená strategie randomizuje volbu akce* a právě ona zaručuje existenci ekvilibria i tam, kde čisté selhává.

8.7 Opakované hry

Opakovaná hra je nejjednodušší vícetahová hra: hráči čelí téže volbě opakovaně, pokaždé se znalostí historie, a výplaty se v čase sčítají. U opakovaného věžňova dilematu hraného *pevně* 100 kol zůstává racionální stále *svědčit* (poslední kolo už není opakované, takže se v něm zradí, čímž argument zpětně padá na všechna kola). Je-li ale konec *nejistý* (např. 99 % šance na další setkání), spolupráce se vyplácí. Robustní a jednoduchá je strategie **tit-for-tat** (*půjčka za oplátku*): začni *mlčením* a pak opakuj soupeřův minulý tah. Je velmi odolná proti široké škále protistrategií a udržuje spolupráci.

8.8 Mechanism design a aukce

Mechanism design (inverzní teorie her) navrhuje *pravidla prostředí* tak, aby při racionálním (sobeckém) chování agentů vznikl co největší *kolektivní* užitek. Typickým mechanismem je **aukce** – prodej zboží skupině dražitelů. Každý dražitel i má soukromou hodnotu v_i a dělá nabídku b_i ; nejvyšší nabídka vyhrává, ale *cena, kterou vítěz zaplatí, závisí na mechanismu*. Dobrý mechanismus maximalizuje *očekávaný výnos* prodejce nebo je *efektivní* (zboží získá ten, kdo si ho cení nejvíce). Žádoucí je, aby měli dražitelé *dominantní strategii* – pak nemusí ztrácet čas odhadováním strategií ostatních.

Poznámka. [Typy aukcí]

- **Anglická** (vzestupná): cena roste od vyvolávací, dokud někdo přihazuje; vyhrává poslední a platí svou nabídku. Jednoduchá dominantní strategie (přihazuj, dokud $\text{cena} \leq v_i$). Nevýhody: může odradit konkurenci a má vysoké komunikační náklady.
- **Holandská** (sestupná): cena klesá z vysoké, dokud někdo neakceptuje. Rychlá (prodej květin, ovoce).
- **Obálková se zapečetěnou nabídkou** (sealed-bid, první cena): každý podá *jedinou* tajnou nabídku, vyhrává nejvyšší a platí ji. *Nemá* jednoduchou dominantní strategii (optimum je

přihodit těsně nad druhou nejvyšší nabídku, kterou ale neznáme).

- **Vickreyova** (sealed-bid, *druhá cena*): jediná tajná nabídka, vyhrává nejvyšší, ale *platí druhou nejvyšší*. Dominantní strategie je teď prostě nabídnout pravou hodnotu $b_i = v_i$ – to dělá mechanismus efektivním a jednoduchým.

i Poznámka. [Tragédie obecní pastviny] Když za užívání společného zdroje nikdo neplatí, bývá zdroj nadužíván k nižšímu celkovému užitku všech – *tragédie obecní pastviny*. Příklad: každá země volí mezi „snížit znečištění“ (náklad –10) a „znečišťovat dál“ (užitek –5 pro sebe, navíc –1 každé jiné zemi, protože vzduch je sdílený). Dominantní strategie každé země je znečišťovat – stejná struktura jako věžňovo dilema. Mechanism design to řeší *zpoplatněním* externalit (např. uhlíková daň; mechanismus Vickrey-Clarke-Groves, kde každý vítěz platí daň rovnou nejvyšší hodnotě mezi poraženými, má opět dominantní strategii $b_i = v_i$).

8.9 Řešené příklady SZZ

Příklad – SZZ podzim 2024 č. 28: Minimax a alfa-beta prořezávání

(Reálná otázka SZZ: podzim 2024, č. 28 – Minimax (alfa-beta prořezávání))

Zadání. (1) Popište algoritmus minimax pro hru dvou hráčů, jeho nedostatky a jejich řešení. (2) Zaveďte techniku alfa-beta prořezávání. (3) Ve stromu hry s nulovým součtem (Δ = tah maximalizujícího hráče, ∇ = minimalizujícího, $\circ$ = koncový stav s ohodnocením) doplňte minimaxové hodnoty s alfa-beta prořezáváním a vyznačte prořezané části. Tahy se procházejí *zleva doprava*.

Řešení (1) a (2). Minimax a alfa-beta jsou popsány výše (pseudokódy i meze α, β). *Nedostatkem* minimaxu je, že prochází celý strom velikosti $O(b^m)$, což je u reálných her neproveditelné. **Řešení:** (a) **alfa-beta prořezávání** (vrací tutéž hodnotu, ale přeskakuje podstromy, které rozhodnutí neovlivní; při dobrém pořadí tahů zdvojnásobí dosažitelnou hloubku); (b) **ořezání hloubky** a náhrada koncové funkce **ohodnocovací funkcí** EVAL v neukončených listech.

Řešení (3). Hodnoty počítáme zdola nahoru. Uzly nejnižší úrovně MIN (∇ nad listy):

$$\min(5, 6) = 5, \quad \min(4, 7, 5) = 4, \quad \min(3) = 3, \quad \min(6) = 6,$$

$$\min(5, 9) = 5, \quad \min(7) = 7, \quad \min(5) = 5, \quad \min(9, 8) = 8, \quad \min(6) = 6.$$

Úroveň MAX (Δ): zleva $\max(5, 4) = 5$, $\max(3) = 3$; $\max(6, 5) = 6$, $\max(7) = 7$; $\max(5) = 5$, $\max(8, 6) = 8$. Úroveň MIN (∇ , tři uzly pod kořenem):

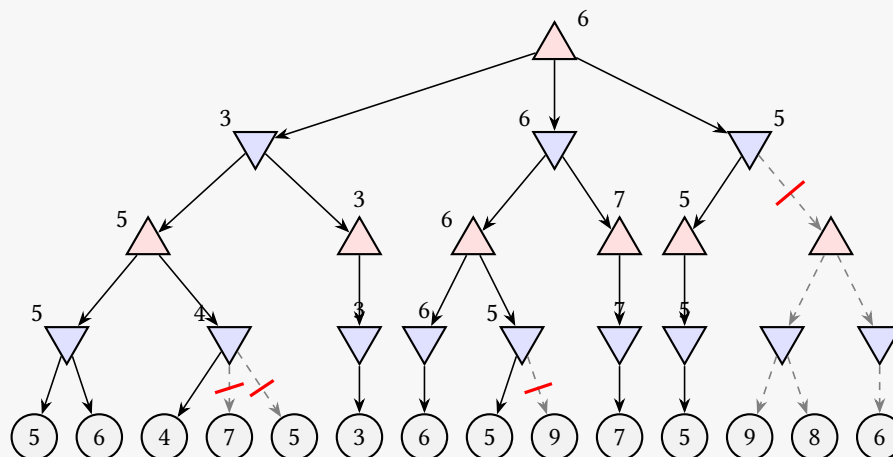
$$\min(5, 3) = 3, \quad \min(6, 7) = 6, \quad \min(5, 8) = 5.$$

Kořen MAX: $\max(3, 6, 5) = \boxed{6}$. Optimální první tah je do prostředního následníka (hodnota 6).

Prořezávání (procházení zleva doprava) prořeže 6 z 14 listů, prohlédne se jich 8 (v pořadí 5, 6, 4, 3, 6, 5, 7, 5). Tři řezy:

- *MIN nad listy 4, 7, 5:* sourozenec vlevo dal MAX hodnotu 5, takže $\alpha = 5$. Tento MIN po přechzení listu 4 má hodnotu $\leq 4 \leq \alpha$, proto se zbylé listy 7, 5 **prořezou**.
- *MIN nad listy 5, 9:* sourozenec vlevo dal MAX hodnotu 6, tedy $\alpha = 6$; po listu 5 je $\leq 5 \leq \alpha$, proto se list 9 **prořeže**.
- *Pravý uzel MIN pod kořenem:* kořen už má z levých větví hodnotu $\max(3, 6) = 6$, takže $\alpha = 6$. Jeho první potomek MAX dá 5, čímž je tento $\min \leq 5 \leq \alpha = 6$; celý druhý podstrom MAX (listy 9, 8, 6) se proto **prořeže**.

(Hodnoty i prořezání ověřeny vlastním Python skriptem implementujícím minimax a alfa-beta s počítáním navštívených listů; kořen = 6, navštíveno 8/14 listů.)



Rekonstruovaný strom s dopočtenými minimaxovými hodnotami (tučně u uzlů) a vyznačenými řezy alfa-beta (červené škrty, šedé čárkované hrany). Kořen MAX = 6. Prořezáno je 6 listů: {7, 5} pod druhým MIN zleva, {9} pod MIN nad listy 5, 9 a celý pravý podstrom MAX s listy {9, 8, 6}. Pořadí čtení listů: 5, 6, 4, 3, 6, 5, 7, 5 (8 z 14).

V příklady/ je u této otázky původní oříznutý herní strom z PDF (originál byl bez náčrtu řešení).

Příklad – SZZ podzim 2025 č. 27: Jednotahové hry (dominance, Nash, Pareto)

(Reálná otázka SZZ: podzim 2025, č. 27 – Jednotahové hry (Nash, dominantní strategie, Pareto))

Zadání. (A) Definujte *dominantní strategii*, *Nashovo ekvilibrium* a *Pareto optimální výsledek*. (B) Pro tři níže uvedené hry (hráč A = řádky Left/Right, hráč B = sloupce Left/Right, oba *maximalizují*; políčko = (zisk A, zisk B)) najděte všechny *dominantní strategie*, *Nashova ekvilibria* a *Pareto optimální výsledky*. (C) Vysvětlete na nich rozdíly mezi pojmy.

(A) Definice – viz výše (dominance = nejlepší strategie hráče bez ohledu na soupeře; Nash = profil stabilní vůči jednostranné odchylce; Pareto optimum = výsledek, který nelze zlepšit všem zároveň).

(B) Tři hry.

Hra 1.

	B: Left	B: Right
A: Left	10, 5	7, 8
A: Right	0, 6	15, 7

Hráč B: při A = Left je $5 < 8$, při A = Right je $6 < 7$ – B vždy preferuje **Right**, to je jeho *dominantní strategie*. Hráč A dominantní strategii nemá (při B = Left preferuje Left $10 > 0$, při B = Right preferuje Right $15 > 7$). B tedy zahraje Right a A na to odpoví Right. **Nash (čistý):** (R, R) = (15, 7) (ověř: nikdo si jednostranně nepolepší); **smíšené žádné** – B hraje ostře dominantní Right s pravd. 1, takže nerandomizuje. **Pareto optimální:** (15, 7) a (7, 8) (žádný jiný výsledek není lepší pro oba; tyto dva jsou navzájem neporovnatelné – vyšší zisk A proti vyššímu zisku B).

Hra 2 (koordinační).

	B: Left	B: Right
A: Left	100, 100	0, 0
A: Right	0, 0	50, 50

Dominantní strategie: žádná (oba hráči chtějí volit *totéž* co soupeř). **Nash (čistý):** dvě – (L, L) = (100, 100) i (R, R) = (50, 50) (z obou se jednostranně nepolepší: odchylka vede na 0). **Nash (smíšený):** jedno – oba hrají Left s pravd. $\frac{1}{3}$, Right s pravd. $\frac{2}{3}$ (z indifference B: $100p = 50(1 - p) \Rightarrow p = \frac{1}{3}$);

očekávaný zisk $33\frac{1}{3}$, horší než obě čisté rovnováhy. Hra 2 má tedy *tři* ekvilibria. **Pareto optimální:** jen (100, 100) – výsledek (50, 50) je Pareto-dominován výsledkem (100, 100).

Hra 3 (struktura věžňova dilematu).

	B: Left	B: Right
A: Left	20, 20	0, 40
A: Right	40, 0	10, 10

Hráč A: při $B = \text{Left}$ je $40 > 20$, při $B = \text{Right}$ je $10 > 0$ – **Right** dominuje. Symetricky B: **Right** dominuje. **Nash (čisté):** $(R, R) = (10, 10)$ (rovnováha v dominantních strategiích); **smíšené žádné** – oba hrají ostře dominantní Right s pravd. 1. **Pareto optimální:** (20, 20), (0, 40), (40, 0). Nashův výsledek (10, 10) je Pareto-dominován výsledkem (20, 20) – oba by si polepšili, ale racionální dominantní hra je tam nepustí. To je **struktura věžňova dilematu**.

(C) Rozdíly mezi pojmy.

- *Dominance vs. Nash.* Dominance je vlastnost *jedné strategie jednoho hráče*; Nash je vlastnost *profilu*. Profil v dominantních strategiích je Nash (hra 1, hra 3), ale Nash *nemusí* pocházet z dominance: hra 2 nemá žádnou dominantní strategii, a přesto má dvě Nashova ekvilibria. *Existence Nashe tedy nevyžaduje dominanci.*
- *Nash vs. Pareto.* Nash je o *stabilitě* (nikdo neodbočí), Pareto o *kolektivní dobrotě*. V hře 3 je Nash (10, 10) Pareto-dominovaný (stabilní, ale pro oba špatný). V hře 1 Nash (15, 7) Pareto optimální je. *Nash a Pareto optimum se mohou, ale nemusí krýt.*
- *Dominance vs. Pareto.* Hra v dominantních strategiích nezaručuje Pareto optimum: hra 3 končí v Pareto-dominovaném (10, 10). *Sobecky optimální tah pro každého může být kolektivně nejhorší* – přesně poučení věžňova dilematu (a tragédie obecní pastviny).

(Všechny dominantní strategie, čistá Nashova ekvilibria a Pareto-optimální výsledky byly ověřeny prohledáním všech čtyř profilů každé hry; smíšená ekvilibria výpočtem přes princip indiference – vlastní smíšené existuje jen u hry 2.)

V příklady/ je u této otázky původní zadání s výplatními maticemi (originál byl bez náčrtu řešení).

9 Strojové učení – přehled (druhy učení, rozhodovací stromy, regrese, SVM, zpětnovazební učení)

Učení znamená zlepšování chování agenta na budoucích úlohách na základě pozorování. Programovat řešení napřímo často nejde: návrhář nedokáže předvídat všechny situace ani jejich změny v čase a někdy ani neví, jak řešení naprogramovat. Tato sekce dává **přehled strojového učení na úrovni úvodního kurzu** (NAIL120): druhy učení, rozhodovací stromy, lineární modely a SVM, Bayesovské a EM učení a zpětnovazební učení.

i Poznámka. [Vztah k okruhu S3] Strojové učení má vlastní samostatný státnicový okruh **S3 – Strojové učení**, kde se témata probírají do hloubky. *Hlubší rozbor* (jádrový trik a duální formulace SVM, evaluační metriky a křížová validace, regularizace a přeučení, prokletí dimenzionality, shlukování, PCA) patří do S3. Tato sekce ho **nezdvojuje**: drží úroveň „základní principy“ podle požadavků S1, tedy hlavní myšlenky, definice a klíčové algoritmy tak, jak je podává úvodní kurz UI.

9.1 Druhy učení

Strojové učení se dělí podle toho, jakou *zpětnou vazbu* má agent k dispozici.

☰ Definice – Tři základní druhy učení

- **Učení s učitelem** (supervised): agent dostane množinu příkladů ve formě dvojic *vstup–výstup* $(x_1, y_1), \dots, (x_N, y_N)$, kde $y_i = f(x_i)$ pro neznámou funkci f , a hledá *hypotézu* h , která f co nejlépe aproximuje.
- **Učení bez učitele** (unsupervised): agent dostane jen vstupy bez označení a hledá v nich *vzory* (typicky shluky nebo nízkodimenzionální strukturu).
- **Zpětnovazební učení** (reinforcement): agent se učí z posloupnosti *odměn a trestů*; zpětná vazba je řidší a opožděná, neříká „správný výstup“, jen ohodnotí důsledky akcí.

💡 **Intuice.** U učení s učitelem se podle typu výstupu rozlišuje:

- **klasifikace** – výstup y je z *konečné* množiny (např. *slunečno / oblačno / deštivo*);
- **regrese** – výstup y je *číslo* (např. teplota, cena).

Hypotéza je *konzistentní* s příkladem (x_i, y_i) , pokud $h(x_i) = y_i$. Konzistentních hypotéz bývá více; kterou vybrat?

☰ Definice – Ockhamova břitva

Mezi hypotézami konzistentními se stejnými daty preferujeme tu **nejjednodušší**. Jde o zásadu *indukce*: jednodušší hypotéza je méně náchylná k přeučení (zapamatování šumu místo zachycení zákonitosti) a lépe *generalizuje* na nová data.

9.2 Rozhodovací stromy

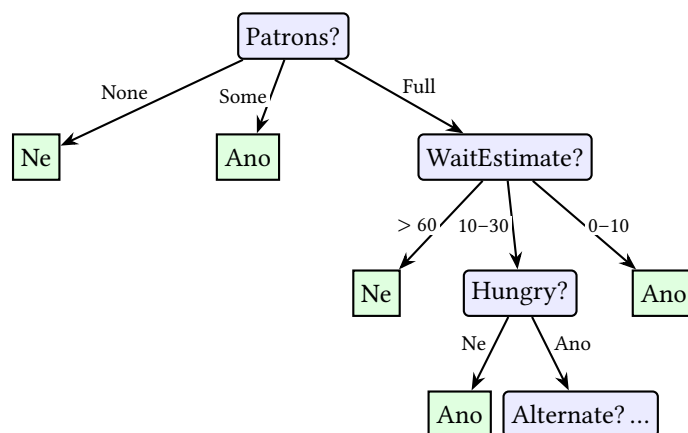
Rozhodovací strom je jedna z nejjednodušších a přitom velmi úspěšných forem naučené funkce. Na vstupu dostane vektor hodnot atributů a vrací *rozhodnutí* – jednu výstupní hodnotu.

☰ Definice – Rozhodovací strom

Rozhodovací strom dospívá k rozhodnutí posloupností *testů*:

- každý **vnitřní uzel** testuje hodnotu jednoho vstupního atributu,
- **větvě** z uzlu jsou označeny možnými hodnotami testovaného atributu,
- každý **list** udává výstupní hodnotu (rozhodnutí), kterou funkce vrátí.

Prostor hypotéz je množina všech rozhodovacích stromů; hledáme strom *konzistentní* s příklady a přitom co *nejmenší* (Ockhamova břitva).



Část rozhodovacího stromu pro úlohu „počká host na stůl v restauraci?“ (AIMA; větev 30–60 a další testy pod Hungry = Ano jsou vynechány). Vnitřní uzly testují atributy (kolik je hostů, odhad čekání, hlad), listy dávají rozhodnutí Ano/Ne. Strom se staví shora dolů a v každém uzlu se vybírá *nejinformativnější* atribut (viz informační zisk níže).

Konstrukce: hladová strategie rozděl a panuj

Malý konzistentní strom stavíme **hladovou** strategií „rozděl a panuj“: v kořeni vybereme *nejdůležitější* atribut, podle jeho hodnot rozdělíme příklady do skupin a v každé skupině rekurzivně postavíme podstrom. Rekurse končí, když jsou všechny zbývající příklady stejné třídy (vrátíme tu třídu), když dojdou atributy (vrátíme většinovou třídu), nebo když je skupina příkladů prázdná (vrátíme většinovou třídu rodiče).

Algoritmus – Učení rozhodovacího stromu (AIMA LEARN-DECISION-TREE)

```
LEARN-DECISION-TREE(examples, attributes, parent_examples):
  if examples je prázdná: return PLURALITY-VALUE(parent_examples) ▷ většinová třída rodiče
  else if všechny examples mají stejnou klasifikaci: return tu klasifikaci
  else if attributes je prázdná: return PLURALITY-VALUE(examples)
  else:
    A ← argmaxa ∈ attributes IMPORTANCE(a, examples) ▷ nejvyšší informační zisk
    tree ← nový strom s testem A v kořeni
    for každou hodnotu v atributu A:
      exs ← { e ∈ examples : e.A = v }
      subtree ← LEARN-DECISION-TREE(exs, attributes - A, examples)
      přidej do tree větev s popiskem (A = v) a podstromem subtree
  return tree
```

Který atribut je nejdůležitější: entropie a informační zisk

Nejdůležitější atribut je ten, který nejvíc *rozliší* příklady – po jeho otestování zbude co nejméně nejistoty o klasifikaci. Nejistotu měříme **entropií**.

Definice – Entropie a informační zisk

Entropie náhodné veličiny V s hodnotami v_k a pravděpodobnostmi $p(v_k)$ (v *bitech*, $\log = \log_2$):

$$H(V) = - \sum_k p(v_k) \log_2 p(v_k).$$

Pro *booleovskou* veličinu, která je pravdivá s pravděpodobností q , píšeme

$$B(q) = -q \log_2 q - (1 - q) \log_2 (1 - q).$$

Mějme p kladných a n záporných příkladů; entropie celé množiny je $B\left(\frac{p}{p+n}\right)$. Atribut A rozděljuje příklady podle hodnot do skupin; k -tá skupina má p_k kladných a n_k záporných příkladů. *Zbytková* entropie po otestování A je vážený průměr

$$\text{Remainder}(A) = \sum_k \frac{p_k + n_k}{p + n} B\left(\frac{p_k}{p_k + n_k}\right).$$

Informační zisk (information gain) atributu A je očekávané *snížení* entropie:

$$\text{Gain}(A) = B\left(\frac{p}{p+n}\right) - \text{Remainder}(A).$$

V kořeni vybíráme atribut s *největším* $\text{Gain}(A)$.

Příklad – Information gain: atribut Patrons vs. Type (AIMA restaurace)

Trénovací množina má 12 příkladů, 6 kladných a 6 záporných (host počká / nepočká). Entropie celé množiny je

$$B\left(\frac{6}{12}\right) = B\left(\frac{1}{2}\right) = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1 \text{ bit}.$$

Atribut Patrons (kolik je hostů) má tři hodnoty s rozdělením příkladů

None: $(0^+, 2^-)$, Some: $(4^+, 0^-)$, Full: $(2^+, 4^-)$.

None a Some jsou *čisté* skupiny: $B(0) = B(1) = 0$ (žádná nejistota). Full má $B(\frac{2}{6}) = B(\frac{1}{3}) \approx 0,9183$. Zbytek

$$\text{Remainder}(\text{Patrons}) = \frac{2}{12}B(0) + \frac{4}{12}B(1) + \frac{6}{12}B(\frac{1}{3}) = \frac{1}{2} \cdot 0,9183 = 0,4591,$$

$$\text{Gain}(\text{Patrons}) = 1 - 0,4591 = 0,5409 \approx 0,541 \text{ bitu.}$$

Atribut Type (kuchyně: French, Italian, Thai, Burger) rozdělí příklady tak, že každá hodnota má vyrovnaný poměr 1 : 1 resp. 2 : 2, tedy $B(\frac{1}{2}) = 1$ v každé skupině:

$$\text{Remainder}(\text{Type}) = \frac{2}{12} \cdot 1 + \frac{2}{12} \cdot 1 + \frac{4}{12} \cdot 1 + \frac{4}{12} \cdot 1 = 1,$$

$$\text{Gain}(\text{Type}) = 1 - 1 = 0.$$

Závěr. Patrons je *dobrý* atribut (0,541 bitu), Type *bezcestný* (0 bitů) – po otestování Type je nejistota stejná jako předtím. Algoritmus proto v kořeni zvolí Patrons.

i Poznámka. Information gain mírně *nadhodnocuje* atributy s mnoha hodnotami (extrémně: jedinečné ID by každý příklad oddělilo do vlastní skupiny, Gain maximální, ale strom by negeneralizoval). V praxi se proto používá normalizace (gain ratio) a *prořezávání* (pruning) přečených větví; detaily patří do S3.

9.3 Lineární modely: regrese a klasifikace

Lineární regrese

Hypotézou je *lineární funkce* vstupů. Pro jeden vstup (přímka)

$$h_{\mathbf{w}}(x) = w_1 x + w_0, \quad \mathbf{w} = [w_0, w_1];$$

pro více vstupů $h_{\mathbf{w}}(\mathbf{x}) = w_0 + \sum_i w_i x_i$. Kvalitu proložení měříme **kvadratickou ztrátou** (L2):

$$\text{Loss}(h_{\mathbf{w}}) = \sum_j (y_j - h_{\mathbf{w}}(x_j))^2, \quad \mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}} \text{Loss}(h_{\mathbf{w}}).$$

💡 Intuice. Pro jednorozměrnou lineární regresi je řešení *jednoznačné* a lze ho dostat analyticky (položíme parciální derivace ztráty podle w_0 a w_1 rovny nule a vyřešíme soustavu). Obecněji – a vždy, když je prostor hypotéz složitější – použijeme **gradientní sestup**: začneme v libovolném bodě prostoru vah a opakovaně se posuneme proti gradientu ztráty,

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(h_{\mathbf{w}}),$$

kde α je *rychlost učení* (krok); α může být konstantní nebo s časem klesat. Pro L2 ztrátu vychází pravidlo $w_i \leftarrow w_i + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) x_{j,i}$ (znaménko se obrátí, protože derivace $(y - h)^2$ dá faktor $-2(y - h)$).

Lineární klasifikace a logistická regrese

Pro klasifikaci do dvou tříd hledáme *rozhodovací hranici* – v rovině přímku (obecně nadrovinu), která třídy odděluje. Říkáme, že data jsou **lineárně separabilní**, pokud taková nadrovina existuje. Lineární klasifikátor vrací

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x}), \quad \text{Threshold}(z) = \begin{cases} 1 & z \geq 0, \\ 0 & z < 0. \end{cases}$$

Váhy lze učit *perceptronovým pravidlem* $w_i \leftarrow w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) x_i$.

Intuice. Tvrdý práh je nespojitý (nelze přes něj počítat gradient) a perceptronové pravidlo na neseparabilních datech *nekonverguje*. Proto se práh „změkčí“ **logistickou funkcí**

$$\text{Threshold}(z) = \frac{1}{1 + e^{-z}},$$

jejíž výstup je z intervalu $(0, 1)$ a dá se číst jako *pravděpodobnost* třídy 1. To je **logistická regrese**, jedna z nepoužívanějších klasifikačních technik. Váhy se učí gradientním sestupem; využije se hezká vlastnost derivace logistické funkce $\text{Threshold}'(z) = \text{Threshold}(z)(1 - \text{Threshold}(z))$.

Parametrické vs. neparametrické modely; k-NN

Definice – Parametrický a neparametrický model

Parametrický model shrne data do *pevného počtu* parametrů (vah) nezávislého na počtu příkladů; po naučení lze trénovací data zahodit (lineární a logistická regrese, neuronové sítě). **Neparametrický** model si k reprezentaci hypotézy ponechá (část) původních dat; počet „parametrů“ roste s daty.

Typický neparametrický model je **k nejbližších sousedů** (k-NN): pro nový bod $\mathbf{x}$ najdeme k nejbližších trénovacích příkladů a odpověď složíme z jejich y – u klasifikace *většinovým hlasováním* sousedů, u regrese *průměrem*. Vzdálenost se měří *Minkowského* metrikou $L_p(\mathbf{x}_j, \mathbf{x}_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{1/p}$ ($p=1$ Manhattan, $p=2$ Euklid); na rozdílné škály atributů je dobré data *normalizovat*.

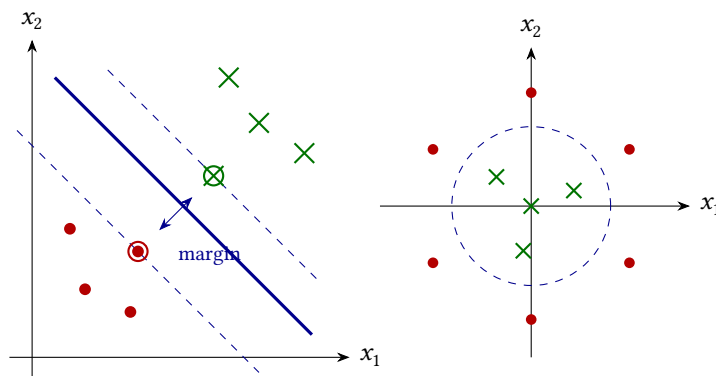
9.4 Support vector machines (SVM)

SVM je dnes jeden z nejpopulárnějších přístupů učení s učitelem „z krabice“. Stojí na dvou myšlenkách: *maximální margin* a *jádrová transformace*.

Definice – Maximum-margin separátor a podpůrné vektory

SVM hledá lineární oddělující nadrovinu s **největším možným odstupem** (margin) od nejbližších příkladů obou tříd. Příklady ležící *nejblíže* k separátoru jsou **podpůrné vektory** (support vectors); právě ony separátor určují (ostatní body lze odebrat, aniž se hranice změní). SVM je *neparametrická* metoda: hypotézu reprezentují podpůrné vektory.

Intuice. Mezi mnoha přímkami, které data oddělí, vybere SVM tu „nejodvážnější“ – s co největší rezervou na obě strany. Široký margin znamená lepší *generalizaci*: malé posuny bodů hranici nepřekročí. Hledání maximálního marginu je úloha kvadratického programování (konvexní, tedy s jednoznačným optimum); její duální tvar a detaily patří do S3.



(a) maximum-margin separátor (b) nelineárně separabilní data

(a) SVM volí oddělovací přímku s *maximálním marginem* (plná čára), ohraničeným čárkovaně; kroužkem zvýrazněné body na okraji marginu jsou *podpůrné vektory* a samy hranici určují. (b) Data, která nejdou oddělit přímkou (jedna třída uvnitř, druhá kolem): *jádrová funkce* je zobrazí do prostoru vyšší dimenze (např. atributy x_1^2 , x_2^2 , $\sqrt{2} x_1 x_2$), kde se stanou lineárně separabilní. Detail jádrového triku je v okruhu S3.

Intuice. [Jádrová transformace] Nejsou-li data lineárně separabilní, zobrazí je SVM *jádrovou funkcí* do prostoru vyšší dimenze, kde lineárně separabilní jsou; přímka v novém prostoru odpovídá zakřivené hranici v původním. Trik je v tom, že stačí umět počítat *skalární součiny* v novém prostoru (jádro), aniž bychom transformaci prováděli explicitně. Plný rozbor (Mercerova podmínka, polynomiální a RBF jádra, měkký margin) patří do S3.

9.5 Bayesovské učení a EM algoritmus

Učit lze i *pravděpodobnostní modely*, a to nástroji pravděpodobnostního usuzování (viz sekce 4 o Bayesovských sítích).

Definice – Maximálně věrohodná hypotéza a Bayesovská predikce

Mějme prostor hypotéz $\{h_i\}$ s apriorními pravděpodobnostmi $P(h_i)$ a pozorovaná data d .

- **Maximálně věrohodná** (ML) hypotéza maximalizuje *věrohodnost dat* $P(d | h)$:

$$h_{\text{ML}} = \underset{h}{\operatorname{argmax}} P(d | h).$$

- **Aposteriorní** pravděpodobnost hypotézy je $P(h_i | d) = \alpha P(d | h_i) P(h_i)$ (Bayes; α normalizuje).
- **Bayesovsky optimální predikce** neznámé veličiny X je *vážený průměr* predikcí všech hypotéz, vážený jejich aposteriorními pravděpodobnostmi:

$$P(X | d) = \sum_i P(X | h_i) P(h_i | d).$$

Intuice. Bayesovské učení nepředpovídá jednou „nejlepší“ hypotézou, ale *všemi naráz*, vážením podle jejich pravděpodobnosti. Při nezávislých stejně rozdělených pozorováních (i.i.d.) se věrohodnost faktorizuje, $P(d | h_i) = \prod_j P(d_j | h_i)$, takže každé pozorování její hodnotu jen násobí. Jak data přibývají, aposteriorní rozdělení se zostřeje kolem skutečné hypotézy a Bayesovská predikce konverguje k predikci ML hypotézy.

Příklad – SZZ jaro 2025 č. 30: Bayesovské učení (jablka od tří dodavatelů)

(Reálná otázka SZZ: jaro 2025, č. 30 – Bayesovské učení)

Zadání (zkráceně). Supermarket bere bedny jablek od dodavatelů A, B, C s apriorními pravděpodobnostmi $P(A) = \frac{1}{2}$, $P(B) = \frac{1}{3}$, $P(C) = \frac{1}{6}$. Naměřené četnosti velikostí (9/10/11 cm) jsou A: 5, 4, 1 (celkem 10); B: 9, 8, 3 (celkem 20); C: 1, 8, 1 (celkem 10). (1) Uveďte maximálně věrohodné odhady rozdělení velikostí pro každého dodavatele. (2) Definujte *maximálně věrohodnou hypotézu*. (3) Z jedné bedny jsme vzali dvě jablka, 10 cm a 11 cm; která hypotéza je maximálně věrohodná? (4) Jaká je *Bayesovsky optimální predikce* pravděpodobnosti 10 cm pro další jablko z téže bedny?

Řešení.

(1) ML odhady rozdělení. Pro diskrétní rozdělení je maximálně věrohodný odhad relativní četnost (podíl):

$P(\text{velikost} \mid h)$	9 cm	10 cm	11 cm
A	0,5	0,4	0,1
B	0,45	0,4	0,15
C	0,1	0,8	0,1

(2) Definice. Maximálně věrohodná hypotéza je hypotéza maximalizující věrohodnost dat $P(\text{data} \mid h)$ (tj. $h_{\text{ML}} = \operatorname{argmax}_h P(\text{data} \mid h)$).

(3) Která hypotéza je ML. Dvě nezávislá pozorování (10 cm a 11 cm), věrohodnost je součin:

$$P(d \mid A) = 0,4 \cdot 0,1 = 0,04, \quad P(d \mid B) = 0,4 \cdot 0,15 = 0,06, \quad P(d \mid C) = 0,8 \cdot 0,1 = 0,08.$$

Největší je 0,08, takže **maximálně věrohodná je hypotéza C**.

(4) Bayesovsky optimální predikce $P(\text{příští} = 10 \text{ cm})$. Spočteme aposteriorní pravděpodobnosti hypotéz $P(h \mid d) \propto P(h)P(d \mid h)$ a jimi vážíme predikce $P(10 \text{ cm} \mid h)$:

h	$P(h)$	$P(d \mid h)$	$P(h, d)$	$P(h \mid d)$	příspěvek $P(10 \mid h) \cdot P(h \mid d)$
A	$\frac{1}{2}$	0,04	0,02	$\frac{3}{16}$	$0,4 \cdot \frac{3}{16} = 0,15$
B	$\frac{1}{3}$	0,06	0,02	$\frac{1}{8}$	$0,4 \cdot \frac{1}{8} = 0,15$
C	$\frac{1}{6}$	0,08	0,0133	$\frac{1}{8}$	$0,8 \cdot \frac{1}{8} = 0,20$
			$P(d) = 0,16/3 \approx 0,0533$	$\Sigma = \mathbf{0,5}$	

Sdružené pravděpodobnosti $P(h, d) = P(h)P(d \mid h)$ se sečtou na $P(d) = 0,16/3$; normalizací vyjdou aposteriory $\frac{3}{8}, \frac{3}{8}, \frac{2}{8}$. **Bayesovsky optimální predikce** je

$$P(10 \text{ cm} \mid d) = 0,4 \cdot \frac{3}{8} + 0,4 \cdot \frac{3}{8} + 0,8 \cdot \frac{2}{8} = 0,15 + 0,15 + 0,20 = \boxed{0,5}.$$

Všimněte si, že ML hypotéza (C) sama by predikovala 0,8; Bayesovská predikce je nižší, protože bere v úvahu i hypotézy A a B, které jsou aposteriorně pravděpodobnější.

Originál zadání a oficiální náčrt řešení této otázky jsou ve složce příklady/.

EM algoritmus

Mnoho reálných modelů má **skryté proměnné**, jejichž hodnota není v datech pozorována (např. ze které ze dvou smíchaných beden bonbon pochází). Maximálně věrohodný odhad parametrů tehdy nejde spočítat přímo, protože neznáme, ke které „komponentě“ každý příklad patří.

Definice – EM (expectation–maximization)

EM hledá maximálně věrohodné parametry modelu se skrytými proměnnými iterací dvou kroků:

- **E-krok** (expectation): při *současném odhadu* parametrů dopočítej *očekávané hodnoty* (resp. očekávané čítače) skrytých proměnných – „doplň“ data libovolným inferenčním algoritmem.

- **M-krok** (maximization): ber doplněná data jako jistá a přepočítej parametry tak, aby *maximalizovaly věrohodnost*.

Kroky se střídají do *konvergence*. Každá iterace věrohodnost nezhorší.

Intuice. EM rozplete slepičí-vejce problém: kdybychom znali parametry, dopočítali bychom přiřazení ke komponentám; kdybychom znali přiřazení, dopočítali bychom parametry. EM oboje střídá – začne náhodnými parametry, v E-kroku spočítá *očekávané čítače* (např. $N(\text{Bedna}=1) = \sum_j P(\text{Bedna}=1 \mid \text{příklad}_j)$) místo tvrdého počtu) a v M-kroku z nich přepočítá parametry $\theta_{ijk} \leftarrow N(X_i=x_{ij}, U_i=u_{ik}) / N(U_i=u_{ik})$. Pozor: EM konverguje k *lokálnímu* maximu věrohodnosti, výsledek závisí na inicializaci. Algoritmus k-means (S3) je speciální případ EM s „tvrdým“ přiřazením.

9.6 Zpětnovazební učení (reinforcement learning)

Agent je umístěn do prostředí a má se v něm naučit úspěšně chovat. Pozorováním stavů (předpokládáme *plně pozorovatelné* prostředí) se může učit přechodový model. Aby věděl, co je dobré a co špatné, dostává zpětnou vazbu ve formě **odměny** (reinforcement). Úkolem je z pozorovaných odměn naučit se *optimální* (nebo *téměř optimální*) strategii. Pro prostředí má tvar Markovova rozhodovacího procesu (MDP, viz sekce 7): stavy s , akce a , přechodový model $P(s' \mid s, a)$, odměna $R(s)$, diskontní faktor γ ; cílem je strategie π . Na rozdíl od plánování v MDP agent přechodový model ani odměny *předem nezná*.

Definice – Pasivní a aktivní učení

- **Pasivní učení:** strategie π je *pevná* (ve stavu s agent vždy provede $\pi(s)$). Úkolem je naučit se, jak je tato strategie dobrá – tedy *užitkovou funkci*

$$U^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t) \right].$$

- **Aktivní učení:** agent strategii *nezná* a musí se rozhodovat, *co dělat* – naučit se nejen užítky, ale i optimální akce.

V obou případech agent provádí v prostředí *zkušební běhy* (trials) podle své strategie a z vnímání získává aktuální stav a obdrženou odměnu.

Pasivní učení: ADP a TD

Naivní *přímý odhad užitku* (direct utility estimation) bere užitek stavu jako průměrnou celkovou odměnu „dále z tohoto stavu“ (reward-to-go) přes mnoho běhů. Je to vlastně učení s učitelem (vstup = stav, výstup = reward-to-go), ale *neefektivní*: ignoruje, že užítky stavů spolu souvisí (sousední stavy se vážou Bellmanovými rovnicemi), a tak konverguje pomalu. Lepší jsou metody, které Bellmanovy rovnice využijí.

Definice – ADP a TD pro pasivní učení

- **Adaptivní dynamické programování** (ADP): agent se z pozorování naučí přechodový model $P(s' \mid s, \pi(s))$ (jako relativní četnost přechodů, např. $P((2, 3) \mid (1, 3), \text{Right}) = \frac{2}{3}$) a odměny $R(s)$, a pak vyřeší Bellmanovy rovnice pro pevnou strategii (např. iterací hodnot):

$$U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^\pi(s').$$

- **Učení s časovými rozdíly** (temporal-difference, TD): *nepotřebuje* model. Po každém pozorovaném přechodu $s \rightarrow s'$ posune odhad o malý kousek směrem ke konzistenci s Bellmanovou rovnicí.

🔧 Algoritmus – Aktualizace TD

Po pozorovaném přechodu ze stavu s (s odměnou $R(s)$) do stavu s' :

$$U^\pi(s) \leftarrow U^\pi(s) + \alpha \underbrace{\left(R(s) + \gamma U^\pi(s') - U^\pi(s) \right)}_{\text{časový rozdíl (TD error)},$$

kde α je rychlost učení. Závorka je rozdíl mezi „pozorovaným“ cílem $R(s) + \gamma U^\pi(s')$ a stávajícím odhadem $U^\pi(s)$.

💡 **Intuice.** ADP a TD jsou dva konce téhož kompromisu. **ADP** po každém pozorování upraví odhady tak, aby *všechny* stavy souhlasily se všemi nástupci (vyřeší soustavu) – rychle se učí (vzhledem k počtu pozorování), ale každý krok je drahý a vyžaduje model. **TD** udělá jen *jednu* úpravu na pozorovaný přechod (přiblíží stav jeho jedinému viděnému nástupci), model nepotřebuje a je levné; za to konverguje pomaleji a „zašuměněji“. Mají stejnou limitu: ustálené hodnoty U^π splní Bellmanovy rovnice.

Aktivní učení: **explorace vs. exploitace**, Q-učení, SARSA

Aktivní agent musí sám *volit akce*. Naivní „hladový“ agent vždy zahraje akci optimální podle dosud naučeného modelu – a snadno uvázne: jakmile najde nějakou cestu k odměně, drží se jí a nikdy neobjeví lepší. Naučený model totiž není skutečné prostředí; akce navíc neslouží jen k odměnám, ale i k *poznávání* modelu.

📖 Definice – Explorace vs. exploitace

Agent musí volit kompromis mezi **exploitací** (využít dosavadní znalost a maximalizovat okamžitou odměnu) a **explorací** (zkoušet i méně známé akce a zlepšit model kvůli dlouhodobému prospěchu). Čistá exploitace riskuje uvíznutí v suboptimu; čistá explorace nikdy znalost nevyužije.

💡 **Intuice.** Praktické *explorační strategie*: (a) zvolit s klesající pravděpodobností $O(1/t)$ náhodnou akci (jinak hladově) – konverguje k optimu, ale pomalu; (b) rozumnější je dávat *optimistický* odhad užítka málo vyzkoušeným dvojicím stav–akce. V iteraci hodnot se užije *explorační funkce* $f(u, n)$ rostoucí v u a klesající v n (počet vyzkoušení), např. $f(u, n) = R^+$ (optimistická konstanta), dokud $n < N_e$, jinak u . To agenta táhne i do neprozkoumaných oblastí, i kdyby byly daleko.

Aktivní TD učení obvykle pracuje s **Q-hodnotami**: $Q(s, a)$ je hodnota provedení akce a ve stavu s . Vztah k užítku: $U(s) = \max_a Q(s, a)$. Výhoda je, že se z Q dá rovnou číst nejlepší akci $\operatorname{argmax}_a Q(s, a)$ *bez modelu*.

📖 Definice – Q-učení a SARSA

Obě metody odhadují $Q(s, a)$ z pozorovaných přechodů s, a, r, s', a' .

- **Q-učení** (off-policy): aktualizace používá *nejlepší* akci v dalším stavu,

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma \max_{a'} Q(s', a') - Q(s, a) \right).$$

- **SARSA** (on-policy): aktualizace používá akci a' , kterou agent *skutečně provedl* v s' ,

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left(R(s) + \gamma Q(s', a') - Q(s, a) \right).$$

Pravidlo se aplikuje na konci každé pětice (s, a, r, s', a') – odtud název. Obě jsou *bez modelu* (model-free, nepotřebují $P(s' | s, a)$); stačí jim Q-hodnoty.

💡 Intuice. [Q-učení vs. SARSA] Liší se v tom, čím nahradí budoucí hodnotu. **Q-učení** dosadí $\max_{a'} Q(s', a')$ – hodnotu *optimální* budoucí akce bez ohledu na to, co agent vážně udělá; učí se proto rovnou optimální (hladovou) strategii i během explorační politiky, je *off-policy*. **SARSA** dosadí $Q(s', a')$ skutečně zahrané akce a' ; učí hodnotu *té strategie, kterou se agent řídí* (vč. explorační), je *on-policy*. Shodují-li se obě s hladovou volbou, splývají; při explorační politice je SARSA „opatrnější“ (počítá s občasnými exploračními chybami).

⚙️ Algoritmus – Aktivní Q-učící agent (AIMA Q-LEARNING-AGENT)

Q-LEARNING-AGENT(percept):

vstup: percept = aktuální stav s' a odměna r za poslední přechod (v konvenci odměn stavů $r = R(s)$)

stav: Q (tabulka hodnot dvojic stav--akce, zpočátku 0); N_{sa} (čítače); s, a (předchozí stav a akce)

if s není null:

$N_{sa}[s, a] + 1$

$Q[s, a] \leftarrow Q[s, a] + \alpha(N_{sa}[s, a]) (r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

$s, a \leftarrow s', \operatorname{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a'])$

▷ f = explorační funkce

return a

9.7 Shrnutí sekce

i Poznámka. [Co si odnést]

- Tři druhy učení podle zpětné vazby: *s učitelem* (dvojice vstup–výstup), *bez učitele* (vzory bez označení), *zpětnovazební* (odměny). U učení s učitelem: klasifikace (diskrétní výstup) vs. regrese (číslo); Ockhamova břitva preferuje nejjednodušší konzistentní hypotézu.
- *Rozhodovací strom* – testy v uzlech, rozhodnutí v listech; staví se hladově shora dolů, atribut se vybírá podle **informačního zisku** $= B(\frac{p}{p+n}) - \text{Remainder}(A)$ (Patrons 0,541 b., Type 0 b.).
- *Lineární regrese* – L2 ztráta, analyticky nebo gradientním sestupem; *logistická regrese* – změkčený práh, výstup jako pravděpodobnost. *k-NN* je neparametrický (drží data), lineární/logistická regrese parametrické (drží váhy).
- *SVM* – maximum-margin separátor určený *podpůrnými vektory*; *jádro* zobrazí neseparabilní data do vyšší dimenze. Detaily v S3.
- *Bayesovské učení* predikuje váženým průměrem všech hypotéz ($P(X | d) = \sum_i P(X | h_i)P(h_i | d)$); ML hypotéza maximalizuje $P(d | h)$. *EM* hledá ML parametry se skrytými proměnnými střídáním E-kroku (doplnění očekávaných čítačů) a M-kroku (přepočtení parametrů).
- *Zpětnovazební učení*: **pasivní** (pevná π , učíme U^π) přes *ADP* (naučí model, vyřeší Bellmana) a *TD* (bez modelu, $U(s) \leftarrow U(s) + \alpha(R(s) + \gamma U(s') - U(s))$); **aktivní** (volíme akce) řeší *exploraci* vs. *exploitaci* a učí Q-hodnoty: *Q-učení* (off-policy, $\max_{a'}$) a *SARSA* (on-policy, $Q(s', a')$).

Shrnutí

Okruh S1 prochází hlavní paradigmatu umělé inteligence. Společným motivem je *racionální agent*, který se za různých předpokladů (deterministicky/stochasticky, plně/částečně pozorovatelně, jeden/více agentů) rozhoduje tak, aby maximalizoval užitek. Důraz je na *aplikaci*: zformalizovat úlohu, popsat a odsimulovat algoritmus, spočítat pravděpodobnost či užitek.

i Poznámka. **Klíčové pojmy, algoritmy a vzorce (přehled k SZZ).** Devět bloků odpovídá tematickým skupinám z požadavků. Zkratky: NV = náhodná veličina, CWA = uzavřený svět, MEU = maximální očekávaný užitek, NE = Nashovo ekvilibrium.

Prohledávání – formulace, BFS/DFS/UCS, A^* , heuristiky (řešené otázky: jaro 2024, léto 2022).

- Úloha = $(s_0, A(s), \text{RESULT}(s, a), \text{test cílovosti}, c(s, a, s'))$; řešení = posloupnost akcí; optimální =

min cena.

- Strom (bez closed listu – opakuje stavy) vs. graf (closed list – každý stav rozvine nejvýš jednou).
- BFS (FIFO, nejmenší): úplné, optimální (cena neklesající s hloubkou), čas i paměť $O(b^d)$.
- DFS (LIFO, nejhlubší): neúplné (strom), neoptimální; paměť $O(bm)$, čas $O(b^m)$.
- Uniform-cost (g-min, Dijkstra): expanduje v pořadí ceny $g(n)$; úplné a optimální; test cíle při výběru uzlu (ne při generování!), jinak hrozí neoptimalita.
- A^* : $f(n) = g(n) + h(n)$. **Přípustná** $h(n) \leq h^*(n) \Rightarrow$ optimalita ve stromovém prohledávání. **Konzistentní** (monotónní) $h(n) \leq c(n, a, n') + h(n') \Rightarrow$ optimalita v grafovém; navíc f podél cesty neklesá. Konzistentní $\Rightarrow$ přípustná.
- Příklady heuristik pro 8-puzzle: počet špatně umístěných kostek h_1 a Manhattanská vzdálenost h_2 (obě přípustné i konzistentní). Dominance: $h_2 \geq h_1$ všude $\Rightarrow A^*$ s h_2 rozvine nejvýš tolik uzlů jako s h_1 .

CSP – backtracking, AC-3, MAC (řešené otázky: podzim 2023, jaro 2026, léto 2022).

- CSP = (X, D, C) : proměnné, domény (konečné), podmínky (relace – výčet, nebo formule). Řešení = úplné konzistentní ohodnocení. Splnitelnost CSP je NP-úplná.
- Binární CSP: podmínky arity ≤ 2 ; zobrazujeme jako síť podmínek (constraint network): vrcholy = proměnné, hrany = podmínky.
- Hranová konzistence: pro každou hodnotu $x \in D_i$ existuje opora $y \in D_j$ splňující podmínku (X_i, X_j) . Hranová konzistence splnitelnost nezaručuje; hranová nekonzistence splnitelnost nevylučuje.
- AC-3: fronta oblouků; po zúžení D_i přidej zpět oblouky (X_k, X_i) ; složitost $O(e d^3)$ (e = počet podmínek, d = max. doména).
- Backtracking: DFS nad částečnými ohodnoceními + ověření podmínek vůči přiřazeným. Heuristiky výběru proměnné: MRV (min remaining values), degree (max podmínek). Pořadí hodnot: LCV (least constraining value).
- Forward checking (FC): po přiřazení $X_i = v$ odstraň nekonzistentní hodnoty z domén sousedních proměnných.
- MAC (Maintaining Arc Consistency): po každém přiřazení spusť AC-3 – silnější než FC, odhalí spor dříve.

Logické uvažování – CNF/DNF, DPLL, řetězení, rezoluce (řešené otázky: jaro 2025, podzim 2022).

- Základní převod: $KB \models \alpha \Leftrightarrow (KB \wedge \neg \alpha)$ nesplnitelná. Na něm stojí DPLL i rezoluce.
- CNF: konjunkce klauzulí (AND disjunkcí); DNF: disjunkce elementárních konjunkcí (OR konjunkcí). Každá formule je ekvivalentní CNF i DNF (CNF může být exponenciálně větší).
- DPLL: backtracking po proměnných + dvě zkratky: (1) čistý literál – proměnná jen v jedné polaritě $\rightarrow$ nastav ji tak, aby literál byl pravdivý; (2) jednotková klauzule – klauzule s jediným neohodnoceným literálem $\rightarrow$ vynucená propagace. Tím se ořezávají větve bez explicitní enumerace.
- Hornovské klauzule: ≤ 1 pozitivní literál. Dopředné řetězení jde od faktů k závěrům (lineární čas); zpětné od dotazu k faktům (AND-OR rozklad, dotkne se jen relevantních pravidel).
- Rezoluce: pravidlo $(A \vee C) \wedge (\neg A \vee D) \vdash (C \vee D)$; pracuje na CNF; prázdná klauzule $\perp$ = spor; korektní a úplná pro vyvracení: $KB \models \alpha \Leftrightarrow KB \wedge \neg \alpha \vdash \perp$.

Pravděpodobnostní uvažování – Bayesovské sítě, exaktní i aproximační inference (řešená otázka: léto 2024).

- Podmíněná pravděpodobnost: $P(a | b) = P(a \wedge b) / P(b)$; součinnové pravidlo: $P(a \wedge b) = P(a | b)P(b)$.
- Bayesovo pravidlo: $P(a | b) = P(b | a)P(a) / P(b) = \alpha P(b | a)P(a)$ (kauzální $\rightarrow$ diagnostický směr).
- Marginalizace: $P(Y | e) = \alpha \sum_h P(Y, e, h)$ (vysčítej skrytá H , normalizuj $\alpha = 1/P(e)$).
- Podmíněná nezávislost $X \perp Y | Z$: $P(X | Y, Z) = P(X | Z)$; umožňuje faktorizovat sdruženou distribuci.
- Bayesovská síť: orientovaný acyklický graf; uzly = NV, hrany = přímý vliv; CPT pro každý uzel

X_i . Faktorizace: $P(x_1, \dots, x_n) = \prod_i P(x_i \mid \text{pa}(X_i))$; plně reprezentuje sdruženou distribuci.

- Exaktní inference: enumerace (sečti přes skrytá); eliminace proměnných (join faktorů + marginalizace; pořadí eliminace ovlivňuje složitost).
- Aproximace: *zamítací vzorkování* – generuj svět dle apriori, zamítň pokud $\neg e$, odhadni frekv. *Vážení věrohodností* – nastav e , váha = $\prod_i P(e_i \mid \text{pa}(E_i))$; neodmítá – efektivnější.

Reprezentace znalostí – situační kalkulus, Markovské modely, HMM (řešená otázka: léto 2023).

- Situační kalkulus: situace s_0 , $\text{RESULT}(s, a)$; fluenty $F(s)$ indexované situací; axiom použitelnosti $\Phi(s) \Rightarrow \text{Poss}(a, s)$.
- Problém rámce: efektové axiomy neříkají, co se *nezměnilo*. Řešení: *successor-state axiom* – 1 axiom na fluent: $F(\text{RESULT}(s, a)) \Leftrightarrow (\text{akce } F \text{ nastavila}) \vee (F(s) \wedge \text{akce } F \text{ nezrušila})$.
- Markovův předpoklad: $P(X_t \mid X_{0:t-1}) = P(X_t \mid X_{t-1})$; senzorový: $P(E_t \mid X_{0:t}, E_{1:t-1}) = P(E_t \mid X_t)$.
- **Filtrace** $\mathbf{P}(X_t \mid e_{1:t})$: rekurze predikce + update senzorem + normalizace:

$$\mathbf{P}(X_t \mid e_{1:t}) = \alpha P(E_t=e_t \mid X_t) \sum_{x_{t-1}} P(X_t \mid X_{t-1}=x_{t-1}) \mathbf{P}(X_{t-1} \mid e_{1:t-1}).$$

- Predikce: $\mathbf{P}(X_{t+k} \mid e_{1:t})$ ($k > 0$); vyhlazování: $\mathbf{P}(X_k \mid e_{1:t})$ ($k < t$) – přesnější, využívá pozdější pozorování.
- Viterbi: nejpravděpodobnější průchod (sekvence stavů); analogie s filtrací, ale max místo $\sum$ při rekurzi.
- HMM: diskretní X_t , diskretní E_t ; maticová formulace – přechodová matice $\mathbf{T}$, senzorová matice $\mathbf{O}_t$ (diagonální). DBN = generalizace HMM na více navzájem závislých veličin.

Automatické plánování – operátor, progrese, regrese (řešená otázka: jaro 2023).

- Stav = množina konkretizovaných atomů; CWA: atom neuvedený v s ve stavu s neplatí.
- Operátor $o = (\text{name}, \text{precond}, \text{effects})$; použitelný v s : $\text{precond}^+ \subseteq s$ a $\text{precond}^- \cap s = \emptyset$; efekt: $\gamma(s, o) = (s \setminus \text{eff}^-) \cup \text{eff}^+$.
- Dopředné plánování (progrese): od s_0 aplikuj použitelné operátory; prohledávání v konkrétních stavech.
- Zpětné plánování (regrese): od cíle g ; relevantní akce přispívá k cíli ($g \cap \text{effects} \neq \emptyset$) a nic z něj neruší ($g^- \cap \text{eff}^+ = \emptyset$, $g^+ \cap \text{eff}^- = \emptyset$); regresní množina $\gamma^{-1}(g, a) = (g \setminus \text{effects}) \cup \text{precond}$; menší větvení, ale pracuje s podmínkami, ne konkrétními stavy.
- Relaxační heuristiky: ignoruj záporné efekty (h^+) – přípustná, pomáhá informovat prohledávání.

MDP a POMDP – Bellmanova rovnice, iterace hodnot a strategií (řešená otázka: léto 2025).

- MDP = $(S, A(s), P(s' \mid s, a), R(s), \gamma)$; diskontovaný užitek $U([s_0, s_1, \dots]) = \sum_t \gamma^t R(s_t) \leq R_{\max}/(1 - \gamma)$.
- **Bellmanova rovnice**: $U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$ (charakterizuje optimální U).
- Iterace hodnot: inicializuj U_0 , opakuj $U_{k+1}(s) \leftarrow R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) U_k(s')$; konverguje (Bellman = kontrakce v $\|\cdot\|_\infty$).
- Iterace strategií: střídaj (1) *vyhodnocení* π (lineární soustava – max odpadá: $U^\pi(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^\pi(s')$) a (2) *zlepšení* π (greedy: $\pi'(s) = \arg \max_a \sum_{s'} P U^\pi(s')$); konverguje v konečném počtu kroků.
- POMDP: MDP + senzorový model $P(E_t \mid S_t)$; agent udržuje *stav víry* $b = \mathbf{P}(S_t \mid e_{1:t})$; rozhoduje v prostoru stavů víry (belief-state MDP).

Hry a teorie her – minimax, alfa-beta, Nash, aukce (řešené otázky: podzim 2024, podzim 2025).

- Minimax: $\text{MINIMAX}(n) = \text{Utility}(\text{list})$, max dětí (MAX-uzel), min dětí (MIN-uzel); čas $O(b^m)$, paměť $O(bm)$.
- Alfa-beta: drží α (nejlepší dosud pro MAX) a β (nejlepší dosud pro MIN); α -řez v MIN-uzlu, jakmile hodnota $\leq \alpha$; β -řez v MAX-uzlu, jakmile hodnota $\geq \beta$; vrací tutéž hodnotu jako minimax; při perfektním pořadí tahů $O(b^{m/2})$.

- Ořezání hloubky + ohodnocovací funkce: nelistový uzel → pseudolist; ohodnocení má odhadovat minimaxovou hodnotu pozice.
- **Dominantní strategie:** lepší než každá jiná bez ohledu na volbu soupeře.
- **Nashovo ekvilibrium:** profil, kde nikdo si jednostrannou změnou nepolepší; při neexistenci NE v čistých strategiích existuje smíšené (randomizovaná volba akcí).
- **Pareto optimalita:** nelze zlepšit všem hráčům zároveň. NE ≠ Pareto optimum – struktura věžňova dilematu (hra 3, podzim 2025): dominantní NE (Right, Right) = (10, 10) je Pareto-dominováno (Left, Left) = (20, 20).
- Mechanism design: návrh pravidel → požadované chování racionálních hráčů. Vickreyova aukce: výherce = nejvyšší nabídka, platí 2. nejvyšší nabídku; dominantní strategie = pravdivá valuace $b_i = v_i$.

Strojové učení – přehled (druhy, stromy, regrese, SVM, Bayesovské učení, EM, zpětnovazební; řešená otázka: jaro 2025).

- Druhy: *s učitelem* (dvojice (x_i, y_i) : klasifikace nebo regrese), *bez učitele* (jen x_i : shluky, vzory), *zpětnovazební* (agent v MDP bez znalosti R, P : odměny a tresty).
- Rozhodovací strom: vnitřní uzel = test atributu, list = rozhodnutí. Stavba: informační zisk $\text{Gain}(A) = B(p/(p+n)) - \text{Remainder}(A)$, kde $B(q) = -q \log_2 q - (1-q) \log_2 (1-q)$ a $\text{Remainder}(A) = \sum_k \frac{|E_k|}{|E|} B(p_k/(p_k+n_k))$; v každém kroku vyber A s max. ziskem.
- Lineární a logistická regrese: $h_w(x) = w^T x + b$; logistická: $P(y=1 | x) = \sigma(w^T x)$; trénink gradientním sestupem (u lineární regrese L2 ztráta).
- SVM: maximalizuje margin; podpůrné vektory = body nejbližší separátoru (samy ho určují). Jádrový trik: zobrazí data do vyšší dimenze – stačí počítat skalární součiny (jádro), bez explicitní transformace. Detaily (duální formulace, C , RBF/polynom) v S3.
- Bayesovské učení: posterior $P(h | d) = \alpha P(d | h) P(h)$; MAP = $\arg \max_h P(h | d)$; MLE = $\arg \max_h P(d | h)$ (bez prioru); Bayesovsky optimální predikce = $\sum_h P(h | d) P(x | h)$ (průměr přes hypotézy).
- EM algoritmus: (E-krok) sečti přes skrytá data – posteriorní $P(z | x, \theta)$; (M-krok) maximalizuj očekávanou log-věrohodnost v θ ; opakuj do konvergence (zaručena, ale jen lokální maximum).
- Zpětnovazební učení: *pasivní TD*: $U^\pi(s) \leftarrow U^\pi(s) + \alpha [R(s) + \gamma U^\pi(s') - U^\pi(s)]$ (temporal difference). *Q-učení* (off-policy): $Q(s, a) \leftarrow Q(s, a) + \alpha [R + \gamma \max_{a'} Q(s', a') - Q(s, a)]$; $\pi(s) = \arg \max_a Q(s, a)$. *SARSA* (on-policy): totéž, ale $Q(s', a')$ s aktuální π (ne max). Explorace vs. exploitace: explorační funkce $f(u, n)$ (optimistický odhad pro málo zkoušené dvojice).

Zdroje

- R. Barták: slidy přednášek předmětu NAIL120 *Úvod do umělé inteligence* (Introduction to Artificial Intelligence), MFF UK, anglická verze (primární zdroj výkladu).
https://ktiml.mff.cuni.cz/~bartak/ui_intro/
- S. Russell, P. Norvig: *Artificial Intelligence: A Modern Approach*, 3. vyd. 2010 a 4. vyd. 2020, Pearson (učebnice, na níž kurz stojí; značení a standardní příklady).
<https://aima.cs.berkeley.edu/>
- S. Russell, P. Norvig: *Pseudocode for the Algorithms in AIMA* (kanonický pseudokód algoritmů; kontrola správnosti).
<https://aima.cs.berkeley.edu/algorithms.pdf>
- MFF UK: požadavky ke státní závěrečné zkoušce (Informatika, Bc.) a zadání otázek z minulých termínů.
<https://www.mff.cuni.cz/cs/studenti/bakalarske-studium/statni-zaverecne-zkousky/bakalarske-statni-zkousky-studijniho-programu-informatika>