

# BENEŠOVA ČASŤ

10. Co je to A\*? Nepopisujte detailně, pouze zařaďte do správné kategorie. Co v tomto algoritmu znamená  $f(n) = g(n) + h(n)$ ? Vysvětlete význam symbolů  $f, g, h, n$ .

$A^*$  je informačná metoda prehľadania skorého priestoru

$$f(m) = g(m) + h(m) \quad \rightarrow m \text{ je aktuálny uzel}$$

$\hookrightarrow$  aktuálny uzel zo stavu  $m$  do cieľového stavu

$\rightarrow$  najmenšia cena cesty z počiatočného užla do užla  $m$

$\rightarrow$  celková cena z počiatočného užla do cieľového užla až užla  $m$

9. Pro A\* máme k dispozici 3 prípustné heuristiky  $h_1, h_2$  a  $h_3$ . S použitím týchto heuristik vytvořte prípustnou heuristiku  $h$ , ktorá bude mať najlepšiu možnú vlastnosť. Popište, jakým zpôsobom tuheuristiku vytvárite a proč. Ilustrujte nad stavy  $A, B$  a  $C$  s hodnotami heuristik  $h_1(A) = 2, h_1(B) = 3, h_1(C) = 3; h_2(A) = 1, h_2(B) = 4, h_2(C) = 1; h_3(A) = 3, h_3(B) = 2, h_3(C) = 2; h(A), h(B), h(C)$ .

$\rightarrow$  prípustná heuristika - odhad  $h(m)$  rehoľne väčší než reálna cena cesty do cieľu

$\rightarrow$   $h_A: h_1(a) > h_2(a) \Rightarrow h$ , je lepšia heuristika ale dosť všeobecnej, súvisiace smerom výplňou cest a sú jasnejšie

Heuristiky  $h_1, h_2, h_3$ :

|                        |                        |                        |
|------------------------|------------------------|------------------------|
| $A \quad h_1(A)=2$     | $B \quad h_1(B)=3$     | $C \quad h_1(C)=3$     |
| $\downarrow h_2(A)=1$  | $\downarrow h_2(B)=4$  | $\downarrow h_2(C)=1$  |
| $\underline{h_3(A)=3}$ | $\underline{h_3(B)=2}$ | $\underline{h_3(C)=2}$ |

Nova heuristika  $h$ :

$$h(m) = \max(h_1(m), h_2(m), h_3(m)) \quad h_1(A)=3 \quad h_1(B)=3 \quad h_1(C)=3$$

4. Máte dobré odladěnou implementaci algoritmu A\* a pro Vás prohledávací problém se Vám podařilo zhroustrovat dve heuristiky: Prípustnou  $h_1$ , ktorá pro každý stav s systematicky podesíva skutečnou cenu  $t(s)$  cestu do cieľu ( $\mathbb{E}[h_1(s)] = \frac{1}{3}t(s), 0 < h_1(s) < t(s)$ ), a neprípustnou  $h_2$ , ktorá ovšem dáva v průměru správné odhady ceny ( $\mathbb{E}[h_2(s)] = t(s), 0 < h_2(s)$ ). Porovnejte  $h_1$  a  $h_2$  z hlediska ceny nalezených riešení a paměti potřebné pro prohledávání. Nemá třeba se snažit o kvantifikaci případných rozdílů, stačí rozhodnutí větší/menší/stejná a zdůvodnění.

$\hookrightarrow$  CENA RIEŠENIA

- $\rightarrow h_1$  je jasnejšia a jednoduchšia  $\rightarrow A^*$  najde optimálnu cestu
- $\rightarrow h_2$  daže promene aktuálnu, ale može nadcezdiť  $\rightarrow A^*$  najde neoptimálnu cestu
- $\rightarrow h_1$  je lepšia ako  $h_2$

$\hookrightarrow$  PAMÄŤ

- $\rightarrow h_1$  jednoduchšia, keď  $A^*$  hore musí prechádzať viac užlo  $\rightarrow$  väčšia pamäť
- $\rightarrow h_2$  cesta je vždy správna, stavmy hore množ. užlo  $\rightarrow$  menšia pamäť
- $\rightarrow h_2$  je lepšia ako  $h_1$

1. Předpokládejte prohledávací úlohu s konstantní cenou hrany. Algoritmu A\* lze, podobně jako u prohledávání do hloubky s omezením hľadby (DLS), stanovit maximální zanoření, například z důvodu celkového omezení na paměť. Srovnejte takto omezený A\* (s prípustnou heuristikou) s DLS se stejně omezenou hľadkou: (1) Najde jeden řešení právě tehdyn, když druhý, nebo existuje úlohy, které jeden vyřeší a druhý ne? (2) Najdou stejná řešení? (3) Který algoritmus spotřebuje více paměti? U úloh 1 a 2 v případě shody krátce zdůvodněte, v případě rozdílu načtrněte příklad a vyznačte v něm, jak se liší. Rozdíl (nebo shodu) ve spotřebě paměti zdůvodněte, můžete i ilustrovat.

- 1) ak existuje riešenie do hľadky d,  $\hookrightarrow$  splňuje heuristiku  $A^*$ , keď ho najde ale;
  - ak tam existuje len suboptimalné riešenie, DLS ho najde, ale  $A^*$  ne;
  - ak do hľadky d neexistuje riešenie, nerajde ho ani jeden.
- 2.) normálna riešenie normálne riešenie
    - $\hookrightarrow$  DLS má iba jedno riešenie,  $\hookrightarrow$  najde (aj výše na druhom)
    - $\hookrightarrow$   $A^*$  hľadá iba riešenie riešenie,  $\hookrightarrow$  splňuje heuristiku
    - 3.)  $A^*$  vyzývá iba riešenie, keď si pamäť iba vezme (open, closed);
    - DLS sláčí cestu, keď a mesto na riešenie

4. Máte k dispozici odladěnou implementaci A\* a\_star(problem, g, h). Na vstupu očekává problem, který umí rozbalovat uzly a ověřovat, zda jsou cílem, funkci g, která popisuje cenu uražené cesty, kde cesta je seznam uzlů, a funkci h, která popisuje odhad zbývající ceny do cíle; původní funkce jsou k dispozici jako g\_star a h\_star. Popište pseudokódem, jaké g a h funkci a\_star předáte, aby simulovala hladové prohledávání (greedy search), uniform cost search (UCS) a prohledávání do šířky (BFS).

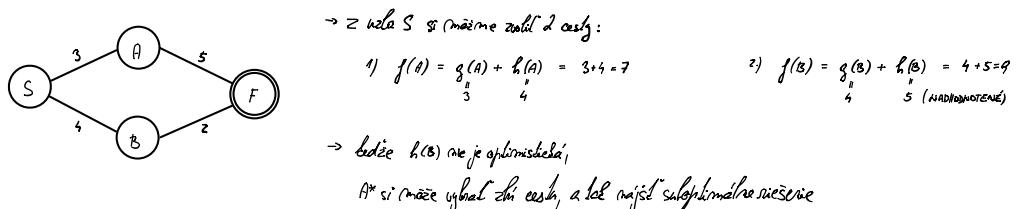
*GREEDY SEARCH : a\_star (problem, 0, h\_star)*

*UCS : a\_star (problem, g\_star, 0)*

*BFS : a\_star (problem, len(path), 0)*

*DFS : a\_star (problem, -len(path), 0)*

3. U heuristiky v A\* požadujeme, aby dávala optimistický odhad zbývající ceny do cíle. Demonstrejte na vhodném příkladu s jediným cílovým uzlem, že i při porušení této podmínky může dojít k nalezení optimálního řešení. Do prohledávacího grafu vepište jak ceny hran, tak hodnoty heuristiky.



4. Mějme prohledávací problém, kde má každý stav právě dva potomky, prostor je acylický, nejdélší cesta od kořene k uzlu měří 20 jednotek a jediný cíl se nachází v hloubce 10 jednotek. Napište, kolik uzelů ( $\pm 2$ ) bude mít při řešení maximálně rozbaleno v paměti prohledávání do šířky (BFS), prohledávání do hloubky (DFS), iterativní prohledávání do omezené hloubky (IDS) a A\* s heuristikou, která je perfektní. Předpokládejte, že acyklitost problému je známá a algoritmy si neudržují žádný seznam CLOSED / množinu EXPLORED. Pokud má metoda na výběr mezi prohledániem více uzelů, uvažujte, že si napřed vždy vybere ten, který nevede k cíli. Test na cíl předpokládejte u všech metod až při rozbalování uzlu, nikoliv při jeho objevení.

*BFS :  $2^{11} - 1$*

$\hookrightarrow$  prohledává na každou hloubku a všechny jeho potomky

*DFS : 20*

$\hookrightarrow$  pamatovali si len aktuálnu castu - stav naseká na max hloubku ale hledajúce uzel



*IDS : 10*

$\hookrightarrow$  pamatovali si len aktuálnu castu - posledný získaný uzel doje do max hloubky 10, kde naseká na riešenie

*A\* : 10*

$\hookrightarrow$  optimálne nájde uzel s najmenej nájdenou

6. Předpokládejte prohledávací úlohu, která odpovídá prohledávání v úplném binárním stromě, tedy každý uzel mimo listových má dva potomky, všechny listy jsou ve stejně úrovni. Hloubka problému je 5 (tzn. nejdélší cesta od kořene sestává z šesti uzelů, má cenu pět), existuje po jednom řešení v hloubkách 3, 4 a 5 a nejsou navzájem (pra)rodiči a (pra)potomky. Vzhledem k náhodnému uspořádání dětí v každém uzelu srovnajte prohledávání do šířky (BFS) a prohledávání do hloubky (DFS) z pohledu: (1) minimální ceny nalezeného řešení, (2) maximální ceny nalezeného řešení, (3) minimálního počtu navštívených uzelů, (4) maximálního počtu navštívených uzelů, (5) minimálního špičkového (peak) počtu uzelů v seznamu OPEN (množině FRONTIER) a (6) maximálního špičkového počtu uzelů tamtéž. Předpokládejte, že acyklitost problému je známá a ani jedna implementace si neudržuje žádný seznam CLOSED (množinu EXPLORED).

**\*\* kto toto ma pocitat dopici \*\***

- 1) BFS - nájde riešenie v hĺbke 3 najskor, DFS - random, moze najskor vybrat cestu k tomu v hlbke 5
- 2) BFS - nájde riešenie v hlbke 3 (asi), DFS - zase random, moze sa nahodne rozhodnut
- 3) BFS -  $1 + 2 + 4 + 1$  (1. node v 4. urovni), DFS -  $1 + 1 + 1 + 1$  (vyberie sa spravnou cestou)
- 4) BFS -  $1 + 2 + 4 + 8$  (last node v 4. urovni), DFS - vela??? spocitajte to sami
- 5/6) no idea, vela pocitania

1.) BFS nájde riešenie v hlbke 3 - min. cena 3

2.) - max. cena 3

3.) BFS - min. pocet navistiennych uzelov:  $1+2+4+1=8$

4.) - max - 11 - :  $1+2+4+8=15$

5.) BFS - min: 8

- max: 15

DFS vrati castu je juverna nájdenym riešenie,

zaležiac od smere prehledania a miestenia riešenia nájde v hlbke 3/4/5

$\rightarrow$  min cena 3

DFS - min. pocet uzelov:  $1+1+1+1=4$

- max - 11 - :  $1+2+4+7+13+26=53$



DFS - min: 4

- max: 6

2. Jaká je hlavní výhoda IDS (iterative deepening depth-first search) oproti BFS (breadth-first search)?

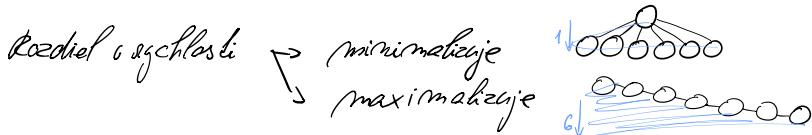
IDS funguje jako DFS s ohledem na délku, kterou od 0 iterativně inkrementuje o 1.

Nerozdíl od BFS má menší paměťové nároky - IDS si paměti hledá cestu  , BFS si paměti všechny podobné cesty 

1. Iterativní prohledávání do hloubky (IDS) lze považovat za paměťově šetrnou implementaci prohledávání do sírky (BFS), která je ale pomalejší. Uvažujte prohledávací stromy o 7 uzlech (cifrový uzel předpokládejte v největší hloubce, v dané hloubce je poslední prohledávaný) a nakreslete strom, který rozdíl v rychlosti minimalizuje, a strom, který rozdíl v rychlosti maximalizuje. Označte, který je který.

~~NOTE: SUPER~~

- asi prostě využít property ze IDS reexploruje, tzn najpomalsie bude když je délka rovná strom? (napr 7 chain)
- nejrychlejšie bude asi když bude 1 parent a 6 dětí?



1. Jakou neinformovanou metodu je nejlepší použít (nejméně paměti, nejkratší čas), pokud víme, že nejlevnější řešení prohledávací úlohy je ve 12. úrovni prohledávacího stromu, jehož struktura je jinak neznámá? Metodu pojmenujte nebo stručně popište. Odůvodněte, proč je metoda optimální z hlediska nalezeného řešení i výpočetních nároků.

DLS - depth limited search

- ↳ paměťová závislost je daná délka cesty (druh), kde 12
- ↳ DLS je výpočetně mnohem rychlejší než BFS až do IDS, když víme, že řešení je na řádu 12.

11. Uveďte, podle čeho se rozhodnete, jestli použít metody lokálního prohledávání oproti „tradičnímu“ (BFS, DFS, IDS, ...) prohledávání. Co je řešením tradičního a co lokálního prohledávání?

Riešením tradičného prohledávania je cesta k ceste,  
a lokálneho prohledávania to je je cieľový stav.

Dá sa rozhodovať podľa ↳ výšky úzky - tradičné majú závislosť problemu s pamäťou, keďže si musíš všetko pamätať  
zadĺenosť ťaženia - v lokálneho je všechno viedieť maximom a či je jedinečné,  
inak by sa mohlo jednať len o lokálne maximum, čo by viedlo k nesplneniu riešenia napäť s tradičným postupom.

5. Môže se pri simulovaném ťahaní stáť, že dojde v daném kroku k výberu horšieho stavu než predchozího?  
Popište mechanizmus, ktorý to zajišťuje / znemožňuje.

- tože môže byť horší stav je 1 z kluczových vlastností tohto algoritmu
- ak je nový stav lepší, určite ho vyberie
- ak je horší tak ho vyberie na základe energie súčasného stavu, energie ďalšieho stavu, súčasnej teploty (toto je ten point) a nejakých konštant K ktorá to reguluje
- tento mechanizmus umožňuje escapes z lokálnych minim

(funkcia malého leskha odporu)

} spôsoba  $\Delta E$ ,  $\rightarrow \propto \left(\frac{\Delta E}{T}\right)$   
v každej iterácii sa znižuje teplosa  $T$   
(čím nižšia teplosa, tým rýchšia pravoh. je horu horúci)

5. Evoluční algoritmy se od paralelního lokálního prohledávání s náhodou (stochastic beam search) liší především krokem křížení řešení. Co je hlavním předpokladem pro možnost křížit řešení? Uveďte příklad.

- řešenie sa musí dať "rozdelit" na podcasti, ktore súme o sebe davať zmysel keď sa skombinujú s podcastami z iných rodicov
- napr chceme najst max decimal cislo a odhadujeme ho v binarke

Parent 1)

11001

Parent 2)

01110

child)

- vezmeme 1. 3 bity od p1 a posledne 2 od p2, vznikne: 110 10

5. Pokud dokážete problém lokálního prohledávání formulovat jako úlohu s omezeními (CSP), můžete místo metod lokálního prohledávání použít backtracking, tedy postupné přiřazování hodnot do proměnných. Stručně popište alespoň dve místa, kde je v backtrackingu prostor pro optimalizaci, která je nezávislá na konkrétní řešení úlohy.

- 1) poradie priradovania hodnot premennym (lepsi vyber hodnot moze znamenat ze "vyhrame" skor)
- 2) poradie priradovania premennych (lepsi vyber ktoru premennu "testujeme" dalsiu)

5. Úlohy s omezeními (CSP) lze řešit pomocí metod lokálního prohledávání, jako např. hill-climbingu nebo simulovaného žhání. Navrhnete alespoň jedno vylepšení těchto metod, které přímo využívá skutečnost, že u CSP úlohy je k dispozici faktORIZOVANÁ reprezentace stavu, tj. jednotlivé proměnné a omezení.

- pri csp vieme ktore constraints su momentalne porusene, tij mozeme prioritizovat tieto premenne cim efektivne redukujeme search space
- napr prehľadavame iba tam kde sa menia premenne ktore sa vyskytuju v porusenych constraints
- mozme prioritizovat tie premenne ktore porusuju mnoho constraints

2. Máme úlohu CSP s proměnnými  $X_1$ ,  $X_2$  a  $X_3$  a odpovídajícími doménami  $D_1 = D_2 = D_3 = \{1, 2, \dots, 10\}$ . Pro omezení  $x_1^2 < x_3$ ,  $x_2 = 2x_3$  proveděte inferenci uzel a hran a zapište výsledek.

$$D_1 = \{1, 2\}$$

$$D_3 = \{2, 3, 4, 5\}$$

$$D_2 = \{4, 6, 8, 10\}$$

2. Máme úlohu CSP s proměnnými  $X_1$ ,  $X_2$  a  $X_3$  a odpovídajícími doménami  $D_1 = D_2 = D_3 = \{1, 2, \dots, 10\}$ . Pro omezení  $x_1^2 < x_3$ ,  $3x_2 < x_1 + x_2$  a  $x_3 < 2x_1$  proveděte inferenci uzel a hran a zapište výsledek.

$$x_1^2 < x_3 : \quad D_1 = \{1, 2, 3\}$$

$$D_3 = \{2, 3, 4, \dots, 10\}$$

$$\text{ale } x_1^2 < x_3 \quad \begin{matrix} \downarrow \\ x_1 < x_3 \end{matrix} \quad 3x_2 < x_1 + x_2 \quad \begin{matrix} \downarrow \\ 2x_2 < x_1 \end{matrix} \quad \text{zajusteju, kde domena premennych oslaiva jizdra: } D_1 = \emptyset \\ D_2 = \emptyset \\ D_3 = \emptyset$$

4. Máme CSP úlohu s proměnnými  $X_1$ ,  $X_2$ ,  $X_3$  a odpovídajícími doménami  $D_1 = \{1, 2, 3, 4, 5\}$ ,  $D_2 = D_3 = \{1, 2, \dots, 9, 10\}$ . Proveděte inferenci tak, abyste zajistili konzistence hran pro omezení  $3X_2 < X_3$ ,  $X_3 < X_1$ .

$$D_2 = \{1, 2, 3\} \Rightarrow D_2 = \{1\}$$

↑

$$D_3 = \{4, \dots, 10\} \Rightarrow D_3 = \{4\}$$

$$D_1 = \{5\}$$

↑

1. Pro nalezení vyhovujícího stavu v úlohách s omezeními (CSP, např. rozmístění přednášek do rozvrhu) může být použito lokální prohledávání nebo backtracking. Jak se liší? Srovnajte je z hlediska reprezentace stavu a použitelnosti pro mírně proměnlivé problémy (např. po naplánování semestru přibyde jedno omezení pro jednu přednášku).

#### **Backtracking:**

**Reprezentace stavu:** Prohledává celý vyhledávací prostor, každý krok přiřazuje hodnoty proměnným, a vrací se zpět při konfliktu.

**Použitelnost:** Méně flexibilní pro změny, pokud přidáme nové omezení, může vyžadovat prohledání celého prostoru znova. Výhodou je, že vždy najde globálně optimální řešení, pokud existuje.

#### **Lokální prohledávání:**

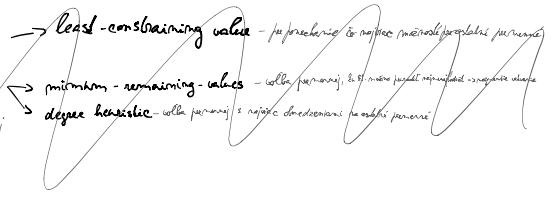
**Reprezentace stavu:** Pracuje s částečně vyřešenými stavami, mění hodnoty proměnných, aby minimalizovalo konflikty.

**Použitelnost:** Velmi flexibilní při změnách (např. přidání nového omezení), protože upravuje pouze části stavu, které jsou změnou ovlivněny. Rychlé pro problémy, kde se omezení často mění, ale nezaručuje globálně optimální řešení.

11. Pokud se rozhodneme řešit CSP úlohy bez použití lokálního prohledávání, používáme backtracking. Jakým způsobem upravujeme backtracking, aby došlo k urychlení prohledávání? Stručně odůvodňte, proč tyto úpravy mohou pomoci.

1) poradie priradovania hodnot premennym (lepsi výber hodnot moze znamenat ze "vyhram" skor)  
 2) poradie priradovania premennych (lepsi výber ktoru premennu "testujeme" dalsiu)

3) pracielse inferencie → Forward Checking - pre zavolenie konzistencie bodov alebo priradovej prekresky  
 4.) model pre konflicty → Conflict-Directed Backjumping - pre výber sa galasom ronaly chyby



↳ výber premennych → ale prvé výber premenné s najmenšími doménami (najmenšie tabuzy) → minimum-remaining-values

→ - - - - - , t. e. výber výberu najviac podmienok (najviac omezení) → degree heuristic

↳ výber hodnot do premennej → ale prvé pravidelne hodnoty, ktoré sú najmenšie obmedzia zošiel - least-constraining value

1. Implementujete solver CSP úloh a rozhodli jste se své řešení založené na čistém backtrackingu zrychlit pomocí inference unárních podmínek (node-consistency). Uvažujte problém o 6 proměnných s doménami o velikosti 10 prvků a jediném řešení v poslední prozkouvané větví. Unární omezení nechť vyřídí polovinu prvků z každé domény, jinak jsou v úloze jen omezení globální (tj. uplatnitelná až v listech). Uvedte a vysvětlete, (1) o kolik se zmenší počet prohledávaných stavů a (2) kolikrát v průběhu backtrackingu inferenci spustíte.

1) počet prohledávaných stavov se zmenší o  $10^6 \cdot 5^6$ .  
 2) pri volbe unárnej inferencie, je toto výsledné riešenie

1. Pracujete na řešení Sudoku na bázi CSP. Vaším výchozím bodem je naivní backtracking, který výřeší 1 úlohu za sekundu. Vypracovali jste zrychlení na základě chytrého backjumpingu; dosáhli jste rychlosti 12 úloh za sekundu. Kolega pracoval na dopředné inferenci a dosáhl rychlosti 50 úloh za sekundu. Merge by nebyl jednoduchý a Vás šéf chce vědět, jestli stojí za to. Odhadněte (udejte bodový odhad a rozumně jistý interval) rychlosť (v problémech za sekundu) solveru po zkombinování Vašich přístupů a odhad zdůvodněte.

Horní mez  $12 \cdot 50$ , každý řešil něco jiného a merge půjde udělat úplně dokonale. Taky se může stát, že ty optimalizace dělají úplně to samé, potom jsme pořád jen na 50 (uznával i něco jako 48 "protože režie").

Obecně ale inference a backjumping interferují, takže bodový odhad lze očekávat někde blíž k 50.

4. MiniMax je speciální případem obecnějšího algoritmu MaxN. Uveďte, jaké dvě podmínky navíc musí sekvenční hra splňovat, aby ji šlo řešit pomocí algoritmu MiniMax.

1) hra musí být pre 2 hrácov (maxm pacita s hocikolko)  
 2) hra musí být tzv "zero-sum" - t. j. gain pl je rovn loss p2

3. Minimax je populární algoritmus pro řešení dvouhráčových her, kde předpokládáme, že jsou oba hráči racionální, tedy že maximalizují svůj užitek. Přesto polovina algoritmu spočívá v hledání minimálních hodnot. Vysvětlete tento rozpor a pojmenujte odpovídající vlastnost her, pro něž minimax lze použít.

Minimax je hra 2 hráčov na báze "zero-sum", t. j. zisk jednoho hráče je pravou nejlepším dánkem. Teda na 1 hráčovi maximizujeme svůj výhodu a na druhém hráčovi minimizujeme. protože předpokládáme, že spoluhráč, který má maximizovat svou výhodu.

2. Jaká je motivace pro použití alpha-beta oproti minimaxu? Uveďte, kdy je minimax výhodnější.

- vzdly? alpha beta je len optimalizacia minimaxu

U Alpha Beta je těžké určit, kterou ze **stejně ohodnocených větví** vybrat a které zahodit. Může se teda lehce stát, že zahodíme větev, která má dále mnohem lepší výsledky.

Minimax je lepší než AlphaBeta, když **protivník dělá náhodné**, nebo **horší-než-nejlepší-možné tahy**. AlphaBeta totiž prohledává stavový prostor tak, že očekává protivníkův nejlepší tah. Pokud protivník zahráje něco mizerného, nejenže AlphaBeta musí prohledávat jinou část prostoru, než očekával, jeho heuristika nefunguje tak jak má. A když máte hru více hráčů a jeden protivník hraje výborně a další náhodně tak je težší natrénovat dobrou funkci pro ohodnocení tahů.

2. Hrajeme jednoduchou (jsme schopni prozkoumat všechny tahy) hru 2 hráčů. Tahy vybíráme pomocí metody minimax a ukládáme si skóre z každé hry. Kamarád nám řekl, že v průměru dosahuje statisticky významně lepšího skóre než my (předpokládejte, že náhodnost hry nemůže být příčinou rozdílu ve skóre), vysvětlete, proč je to možné/nemožné.

Je to možné, ak kamarát používa lepšiu heuristiku, hľať prehľadávanie alebo efektívnejšie prezérvávanie (napr. alfa-beta pruning), vďaka čomu hodnotí pozície presnejšie ako váš minimax.

4. Jak byste upravili minimax, pokud hrajete velký počet her proti soupeři, který hraje náhodně? Potom předpokládejte, že jste navíc schopni během těchto her trénovat model, který odhaduje, jak bude soupeř hrát (má na výstupu kategorické rozložení pravděpodobnosti, že si soupeř v daném stavu vybere danou akci). Jak dále upravíte minimax, aby výstupy z tohoto modelu zužitkovat?

↳ miesto vyberania minimaxu používame pravdepodobnosť, výberieme pravdepodobnosť  
 ↳ potom možno hat do výsledky pravd. rozloženie - miesto pravdepodobnosti,  
 že soupeře tak sa možno následne podla pravd. rozloženia, výberie tak

Ako by ste riešili situáciu pri hre 3 hrácov ak sa 2 hraci spoja a hraju proti tretiemu. Nakreslite rozhodovací strom 3 hrácov kde kazdy hrac ma na vyber 2 možnosti.

Rešenie:

Bude to klasický **MINIMAX**, len hráč MIN tam bude mať dva tropy po sebe. dobré/zlé?

4. Uvažujte sekvenční hru dvou hráčů s nulovým součtem, stavový prostor hry je úplný binární strom o hloubce  $D$ . Jakou časovou složitost má algoritmus minimax při hledání nejlepšího prvního tahu (začíná hráč, který maximalizuje)? Potom navrhnete, jak minimax zrychlit pomocí paralelizace, a uvedete, jak se ona časová složitost změní v ideálních podmínkách (nekonečno procesorů, mimožádná komunikační latence atp.).

↳ časová složitost minimaxu je  $O(2^d)$ , kde  $d$  je délka prvního hráčeho silnice  
 ↳ časová složitost minimaxu zrychleného paralelizací je  $O(\frac{2^d}{P})$ , kde  $P$  je počet paralelních procesorů, co může být iž i  $O(1)$

2. Váš agent hraje deterministickou sekvenční hru pro dva hráče pomocí algoritmu Minimax. Při hraní proti sade referenčních soupeřů na evaluaci sadě her dosáhl průměrného zisku na hru 42 a na referenčním stroji mu trvala volba tahů v průměru 13.57 sekund. Uveďte, jak očekáváte, že se tyto metriky změní (zvětší/zmenší), když Minimax nahradíte algoritmem Alfa-Beta. Odhadněte, jak moc se změní, příp. stručně popište, jaké informace Vám k takovému odhadu chybí.

Při použití Alfa-Beta místo Minimaxu se doba výpočtu zkrátí, protože Alfa-Beta provádí pruning a neprozkoumává neproduktivní větve. Průměrný zisk by měl zůstat podobný, protože algoritmus stále hledá optimální tahy. Zkrácení času závisí na faktoru větvení (b) a hloubce stromu (d), ale ideálně může být rychlosť až několikrát lepší. Chybí nám informace o velikosti herního stromu a efektivitě pruningu.

2. Uvažujte sekvenční deterministickou hru na obrázku. Jaké maximální úspory (v uzlech, které nemusíte prohledávat) lze dosáhnout, když pro hledání nejlepšího tahu namísto MiniMaxu použijete Alfa-Beta prořezávání? Vepište do obrázku ohodnocení listových uzlů, které k tomu povede, a vyznačte, které uzly se nebudou prohledávat.

