# Mesterséges intelligencia - 3. előadás

## Triciális heurisztikák, dominancia

### Triviális heurisztikák

- **zéró heurisztika** → nincs semmi info
- **egzakt heurisztika** → heurisztika maga a valós költség

### Dominancia

- azt fejezi ki, hogy az egyik heurisztika jobb a másiknál
- $h_a(n) \geq h_c(n)$, minden n-re , azaz az ***a* heurisztika jobb *b* heurisztikánál**
- heurisztikák → félhálóz alkotnak
- **$h(n) = max(h_a(n), h_c(n))$**
    - elfogadható heurisztikák maximuma szintén elfogadható

### Heurisztikák konzisztenciája

- elfogadható heurisztika → h(A) ≤ valós költség A-tól
- konzisztencia → h(A) - h(C) ≤ valós költség A-tól C-ig
- **konzisztencia következménye** → h(A) ≤ valós költség A-tól C-ig + h(C)
    - azaz az A pont heurisztikája legyen kisebb mint az A utáni C pont balós költsége plusz a C pont heurisztikája

### Optimalitás A*-nál

- **Fa** alapú keresés → A* optimális, ha a heurisztika elfogadható
- **Gráf** alapú keresés → A* optimális, ha a heurisztika konzisztens
- konzisztenciából következik az elfogadhatóság

## Lokálisan kereső algoritmusok

- specialitás: állapotok jóságát egy **hiperfelület** adja meg N-dimenziós paramétertérben
- felület magassága = adott állapot optimalitása
- cél: legmagasabb (vagy legalacsonyabb) csúcs keresése, mert az az optimális
- azon a helyen levő paraméterkonfiguráció = probléma optimális megoldása
- csak az aktuális állapotot vizsgáljuk, a közvetlen környezetében néz körbe
- fontos: nincs visszalépés (vagy nem igazán), **nincs nyilvántartott állapotlista** (hogy hol jártunnk)
    - memóriaigény kicsi :)
    - időigény → lineáris :)
    - **beragadhatunk** lokális szélsőhelyre :(
- különböző módszerek:

## Hegymászó algoritmus

- **mindig arra megy, amerre javít az aktuális állapoton (felfele)**
- több potenciális csomópont esetében véletlen szerűen választ
- 3 fő probléma:
  - *lokális maximum* → ide fenn tud ragadni
  - *fennsík* → minden irány egyformán jó, bolyongás lehet
  - *hegygerinc* → gerincen lassú feljutni a csúcsra

## Véletlen újraindítású hegymászás

- véletlen generált kiinduló állapot + hegymászás
- leáll: ha lejár az idő vagy nincs észrevehető előrelépés
- jó hír: néhány lokális minimum hellyel már meg tud birkózni, de egy "süni"-vel nem

## Szimulált lehűtés

- nem véletlenszerűen indítjuk útra a keresést, hanem néha direkt "rossz" fele is lépünk
- Boltzmann-eloszlással (egyre csökken a keresés előre haladásával a valószínűsége, hogy hibás utat választ) → mintha a hőmérséklet csökkenne
- cél: **ne ragadjunk be lokális maximumba**

## Lokális nyaláb keresés

- mindig *k* csomópontot tart számon
- *k* véletlen állapotból indul ki, az összes állapot mindegyikének összes követőit kifejti!
  - az újabb peremből *k*-t választ és azzal folytatja az előző pontot
  - ha célt talál, leáll

# Kényszerkielégítési problémák (CSP)

- állapot: az állapot a **leíró változók** és a **hozzájuk rendelt értékek** által definiált
  - $x_k$ változó értéke $D_k$ érték-tartományból van véve
- Célállapotteszt
  - minden $x_k$-hoz rendeltünk a hozzá tartozó $D_k$ tartományból értéket
  - az adott korlátok teljes halmazát kielégítettük
- példa: *Raktár-tervezési probléma*
  - $L_1$, ..., $L_n$ helyszín, k raktárra van szükségünk (k < n), minden helyszínen van max $CP_i$ kapacitás (hány boltot tud kiszolgálni az adott raktár)
  - minden bolthoz rendelünk raktárt
  - $S_j$ bolt ellátása $L_i$ helyről $P_{i,j}$ költségbe kerül
  - ezt a kiszolgálási összköltséget akarjuk minimalizálni
- **korlátgráf**: csomópontjai a változók, élei a korlátok

## CSP problémák típusai

### Diszkrét-folytonos

- Diszkrét változók
  - véges értéktartomány (pl Boole típusú), végtelen értéktartomány (pl egész számok)
- Folytonos változók
  - pl időpontok, fizikai állapotváltozók

### Kényszerek, korlátok alapján

- **Unáris** korlát: egy változóra vonatkozó megkötés, pl SA ≠ green
- **Bináris** korlát: két változó viszonyára vonatkozik, pl SA ≠ WA
- Magasabb-rendű korlát: 3 vagy több változó viszonyára vonatkozik
- **Preferencia-kényszer**: bizonyos értékeket jobban preferálunk másoknál (inkább legyen piros mint zöld)

## Probléma megfogalmazása

- **kezdeti állapot**: összes változó-hozzárendelés üres
- **operátor**: érték-hozzárendelés úgy, hogy ne ütközzön az eddigiekkel
  - **kudarc** = nincs megengedett hozzárendelés
- **célállapotteszt**: aktuális hozzárendelés teljes és minden kényszer teljesül

## Keresés

- változó hozzárendelése **kommutatív** a megoldás szempontjából
- jó lenne, ha vissza tudnánk lépni → legyen **mélységi keresés**
  - minden szinten egyetlen egy változó-hozzárendelés
  - ha sérül egy kényszer, visszalép
- fontos a probléma megfogalmazása → pl n királynő problémánál eleve minden oszlopba 1 királynőt rakjunk és csak az oszlopszámot tároljuk

### Előretekintő ellenőrzés

- minden egyes alkalommal ha X változó értéket kap, az X-hez kényszerrel kötött érték nélküli Y-t megvizsgál, és Y tartományából törli az X számára választott értékekkel inkonzisztens értékeket
- sok inkonzisztenciát észrevesz, de nem mindent
- hatékonyság növelése → heurisztikákkal
  - melyik változóval foglalkozzunk legközelebb?
  - milyen sorrendben vizsgáljuk az értékeit?
  - érzékelhető e jóval előre a kudarc? (korai nyesés)
  - egyéb tárgyfüggetlen heurisztikák

#### Legkevesebb fennmaradó érték ötlete (MRV)

- A **legkisebb számú megengedett értékkel rendelkező változóval kezdjük/folytassuk**
- az első kiválasztásánál nem segít :(

**Fokszám heurisztika**

- az első kezdő változó kiválasztását segíti
- **azzal kezdjük, akinek a legnagyobb a fokszáma** → ő van a legtöbb másik változóra hatással

**Legkevésbé korlátozó érték**

- azon **értéket részesítjük előnyben**, amely a **legkevesebb választást zárja ki** a kényszergráfban a szomszédos változóknál

**Élkonzisztencia**

- X-ből Y-ba húzott **él konzisztens** akkor és csak akkor, ha X minden x értékére létezik Y-nak valamilyen megengedett y értéke
- nehézség: ha X értéket veszít, a szomszédjait újra kell ellenőrizni
- alkalmazhatósága:
  - **előfeldolgozó lépésként** → keresés megkezdése előtt
  - **terjesztési lépésként** → minden egyes hozzárendelést követően

## CSP lokális kereséssel

- **kiindulás**: minden változónak van valamilyen (akár rossz) értéke
- operátorok: megváltoztatják a változók hozzárendelését, céljuk a sérült kényszerek számának csökkentése
- **változó szelekció**: véletlen módon bármely konfliktusban levő változót
- **minimális konfliktus heurisztika**
  - legkevesebb számú korlátot sértő állapot beállítása
  - h(n) = sérült korlátok száma, cél h(n) csökkentése → lokális minimum keresés (pl hegymászó algoritmussal)

# CSP struktúrája

- CSP gráfja független komponenseket tartalmaz, vagy többszörösen összekötött (hurkos) → nagyon nehéz lesz :(
- CSP fa gráf → könnyen megoldható lesz