

Mesterséges intelligencia

Gregorics Tibor
people.inf.elte.hu/gt/mi

Szakirodalom

❑ Könyvek

- Fekete István - Gregorics Tibor - Nagy Sára: Bevezetés a mesterséges intelligenciába, LSI Kiadó, Budapest, 1990, 1999. ELTE-Eötvös Kiadó, Budapest, 2006.
- Russel, J. S., Norvig, P.: MI - modern megközelítésben, Panem Kft, 2005.
- Futó Iván (szerk): Mesterséges intelligencia, Aula Kiadó, Budapest, 1999.

❑ Internet

- people.inf.elte.hu/gt/mi
- Neptun / MeetStreet / Virtuális terek / tantárgy / dokumentumok
- <http://www.inf.elte.hu/karunkrol/digitkonyv/Jegyzetek/mi.pdf>



I. Bevezetés

1. AZ MI FOGALMA

mesterséges intelligencia /MI (artificial intelligence - AI)

Erős MI

Az emberi gondolkodás reprodukálható számítógéppel.

MI szkeptikusok

A számítógép soha nem lesz okosabb az embernél.

Gyenge MI

Az MI kutatja, fejleszti, rendszerezi azokat az elméleteket és megoldási módszereket, amelyek hozzájárulnak az intelligens gondolkodás számítógéppel való reprodukálásához.

MI nem egy speciális részterülete az informatikának, hanem egy törekvés, hogy a számítógéppel olyan érdekes és nehéz problémákat oldjunk meg, amelyek megoldásában ma még az ember jobb.

Miről ismerhető fel egy szoftverben az MI?

❑ Megoldandó feladat: nehéz

- A feladat **problémátere** hatalmas,
- intuícióra, kreativitásra (azaz **heurisztikára**) van szükségünk ahhoz, hogy elkerüljük a **kombinatorikus robbanást**.

❑ Szoftver viselkedése: intelligens

- Turing teszt vs. kínai szoba elmélet
- „mesterjelölt szintű” mesterséges intelligencia

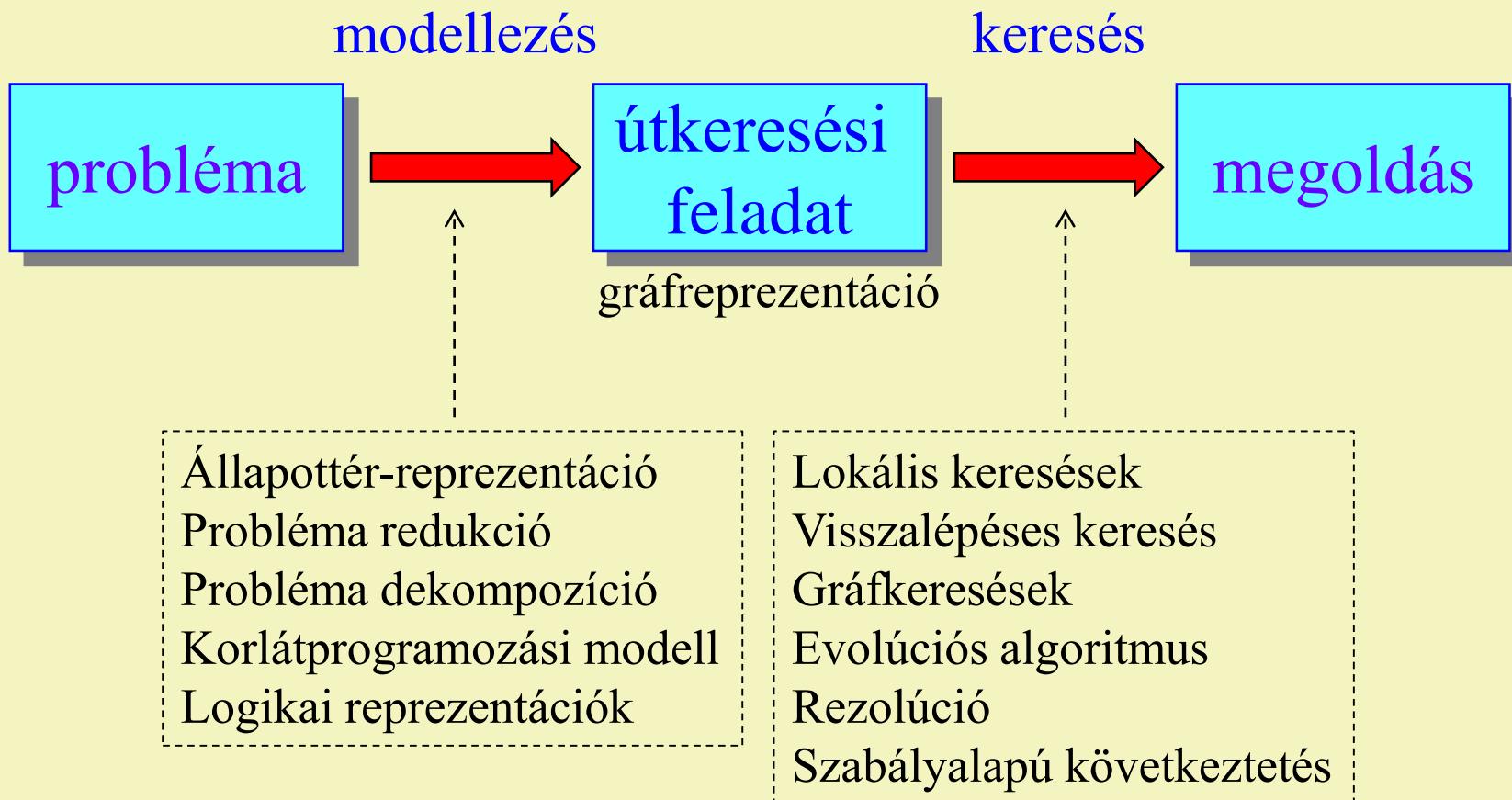
❑ Felhasznált eszközök: sajátosak

- átgondolt reprezentáció a feladat **modellezéséhez**
- heurisztikával megerősített hatékony **algoritmusok**

Intelligens szoftver jellemzői

- megszerzett ismeret tárolása
- automatikus következtetés
- tanulás
- term. nyelvű kommunikáció
- gépi látás, gépi cselekvés

2. MODELLEZÉS & KERESÉS



Mire kell a modellezésnek fókuszálni

- **Problémater elemei:** probléma lehetséges válaszai.
- **Cél:** egy helyes válasz (megoldás) megtalálása
- **Keresést segítő ötletek:**
 - Problémater hasznos elemeinek elválasztása a haszontalanoktól.
 - Az elemek szomszédsági kapcsolatainak kijelölése, hogy a probléma tér elemeinek szisztematikus bejárását segítsük.
 - Adott pillanatban elérhető elemek rangsorolása.
 - **Kiinduló elem** kijelölése.

Útkeresési probléma

- Egy probléma modelljét valamilyen modellezési módszer segítségével írjuk le.
- Ezek a módszerek sokszor **útkeresési problémává** fogalmazzák át a kitűzött problémát: a **problématerének** elemeit **egy speciális élsúlyozott irányított gráf csúcsai** vagy **útjai szimbolizálják**.
- A megoldást ennek megfelelően vagy **egy célcímszám**, vagy **egy startcímiből célcímszámba vezető** (esetleg a legolcsóbb ilyen) út megtalálása szolgáltatja.

Gráf fogalmak 1.

- csúcsok, irányított élek $N, A \subseteq N \times N$ (végtelen számosság)
- él n -ből m -be $(n, m) \in A \quad (n, m \in N)$
- n utódai $\Gamma(n) = \{m \in N \mid (n, m) \in A\}$
- n szülei $\pi(n) \in \Pi(n) = \{m \in N \mid (m, n) \in A\}$
- irányított gráf $R = (N, A)$
- véges sok kivezető él $|\Gamma(n)| < \infty \quad (\forall n \in N)$
- élköltség $c: A \rightarrow \mathbb{R}$
- δ -tulajdonság ($\delta \in \mathbb{R}^+$) $c(n, m) \geq \delta > 0 \quad (\forall (n, m) \in A)$
- δ -gráf δ -tulajdonságú, véges sok kivezető élű, élsúlyozott irányított gráf

Gráffogalmak 2.

- irányított út

δ-gráfokban ez végtelen sok út esetén is értelmes.

Értéke ∞ , ha nincs egy út se.

- út hossza
- út költsége
- opt. költség
- opt. költségű út

$$\alpha = (n, n_1), (n_1, n_2), \dots, (n_{k-1}, m)$$

$$= \langle n, n_1, n_2, \dots, n_{k-1}, m \rangle$$

$$n \rightarrow^\alpha m, n \rightarrow m,$$

$$n \rightarrow M, \{n \rightarrow m\}, \{n \rightarrow M\} \quad (M \subseteq N)$$

az út éleinek száma: $|\alpha|$

$$c(\alpha) = c^\alpha(n, m) := \sum_i c(n_{i-1}, n_i)$$

$$\text{ha } \alpha = \langle n = n_0, n_1, n_2, \dots, n_{k-1}, m = n_k \rangle$$

$$c^*(n, m) := \min_{\alpha \in \{n \rightarrow m\}} c^\alpha(n, m)$$

$$c^*(n, M) := \min_{\alpha \in \{n \rightarrow M\}} c^\alpha(n, m)$$

$$n \rightarrow^* m := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow m\} \}$$

$$n \rightarrow^* M := \min_c \{ \alpha \mid \alpha \in \{n \rightarrow M\} \}$$

Gráfprezentáció fogalma

- minden útkeresési probléma rendelkezik egy (a probléma modellezéséből származó) gráfprezentációval, ami egy (R, s, T) hármas, amelyben
 - $R = (N, A, c)$ δ -gráf az ún. **reprezentációs gráf**,
 - az $s \in N$ **startcsúcs**,
 - a $T \subseteq N$ halmazbeli **célcsúcsok**.
- és a probléma megoldása:
 - egy $t \in T$ cél megtalálása, vagy
 - egy $s \rightarrow T$, esetleg $s \rightarrow^* T$ optimális út megtalálása

Keresés

- Az útkeresési problémák megoldásához a reprezentációs gráfjaik nagy mérete miatt speciális (nem determinisztikus, heurisztikus) útkereső algoritmusokra van szükség, amelyek
 - a startcsúcsból **indulnak**, amely az első aktuális csúcs;
 - minden lépésben **nem-determinisztikus** módon új aktuális csúcsot **választanak** a korábbi aktuális csúcs(ok) gyerekei közül;
 - **tárolják** a már feltárt reprezentációs gráf egy részét;
 - **megállnak**, ha célcímsort találnak vagy nyilvánvalóvá válik, hogy erre semmi esélyük.

Kereső rendszer (KR)

Procedure KR

1. **ADAT** := *kezdeti érték*
 2. **while** \neg *terminálási feltétel(ADAT)* **loop**
 3. **SELECT SZ FROM alkalmazható szabályok**
 4. **ADAT** := **SZ(ADAT)**
 5. **endloop**
- end**

globális munkaterület

tárolja a keresés során megszerzett és megőrzött ismeretet (egy részgráfot)
(kezdeti érték ~ *start csúcs*,

vezérlési stratégia

végrehajtható szabályok közül
kiválaszt egy „megfelelőt”
(általános elv + heurisztika)

keresési szabályok

megváltoztatják a globális
munkaterület tartalmát
(előfeltétel, hatás)

Kereső rendszerek vizsgálata

- helyes-e (azaz korrekt választ ad-e)
- teljes-e (minden esetben választ ad-e)
- optimális-e (optimális megoldást ad-e)
- idő bonyolultság
- tár bonyolultság



II. Modellezés

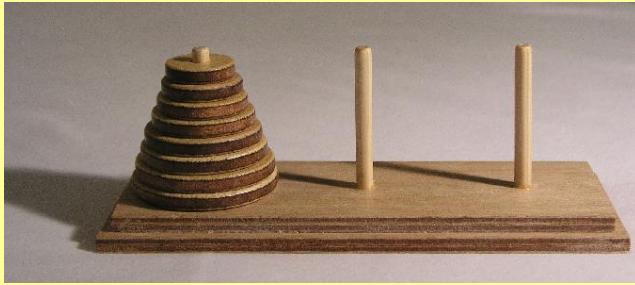
1. Állapottér-reprezentáció

- **Állapottér**: a probléma leírásához szükséges adatok által felvett érték-együttesek (azaz **állapotok**) halmaza
 - az állapot többnyire egy **összetett szerkezetű** érték
 - gyakran egy bővebb alaphalmazzal és egy azon értelmezett **invariáns állítással** definiáljuk
- **Műveletek**: állapotból állapotba vezetnek
 - megadásukhoz: **előfeltétel** és **hatás** leírása
 - invariáns tulajdonságot tartó leképezés
- **Kezdőállapot(ok)** vagy azokat leíró kezdeti feltétel
- **Végállapot(ok)** vagy célfeltétel

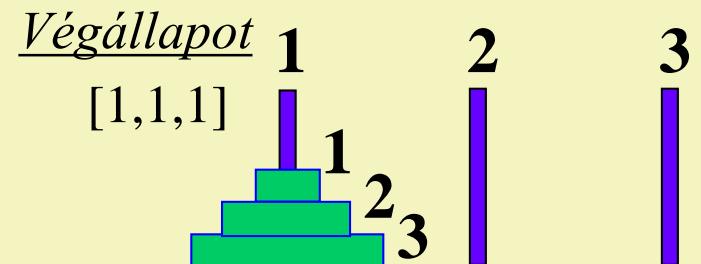
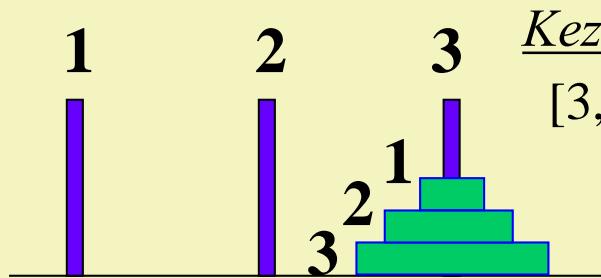
Állapottér-reprezentáció gráf-reprezentációja

- Állapot-gráf
 - állapot csúcs
 - művelet hatása irányított él
 - művelet költsége élköltség

- Reprezentációs gráf
 - δ -gráf állapot-gráf
 - startcsúcs kezdőállapot
 - célcsúcsok végállapotok
 - irányított út egy műveletsorozat hatása
 - irányított út a startból a célba egy megoldás



Hanoi tornyai probléma



Állapottér: $AT = \{1, 2, 3\}^n$

1..n intervallummal indexelt egydimenziós tömb, amelynek elemei 1,2,vagy 3 halmazbeliek.

megjegyzés : a tömb i -dik eleme mutatja az i -dik korong rúdjának számát; a korongok a rudakon méretük szerint fentről lefelé növekvő sorban vannak.

Művelet: **Rak**(*honnán*, *hova*): $AT \rightarrow AT$ *honnán*, *hova* $\in \{1, 2, 3\}$

HA a *honnán* és *hova* létezik és nem azonos, és van korong a *honnán* rúdon, és a *hova* rúd üres vagy a mozgatandó korong (*honnán* rúd felső korongja) kisebb, mint a *hova* rúd felső korongja,

AKKOR *this*[*honnán legfelső korongja*] := *hova*

this:AT az aktuális állapot

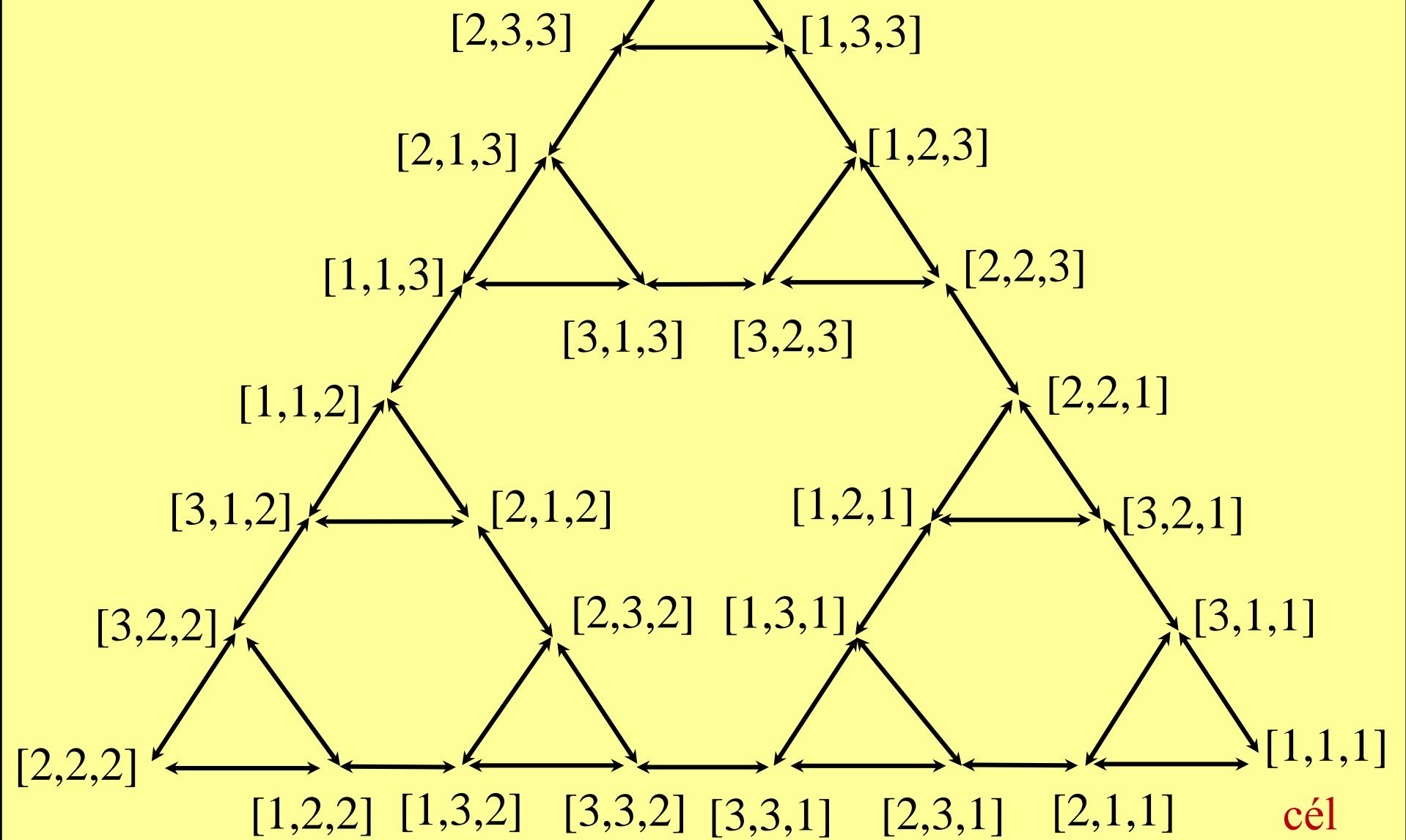
Implementáció

```
template <int n = 3>
class Hanoi {
    int _a[n];           // its elements are between 1 and 3
public:
    bool move (int from, int to) {
        if ((from<1 || from>3 || to<1 || to>3) || (from==to)) return false;
        bool l1; int i; // l1 ~ 'from' is not empty, i ~ upper disc on 'from'
        for(l1=false, i=0; !l1 && i<n; ++i) if (l1 = (_a[i]==from)) break;
        if (! l1) return false;
        bool l2; int j; // l2 ~ 'to' is not empty, j ~ upper disc on 'to'
        for(l2=false, j=0; !l2 && j<n; ++j) if (l2 = (_a[j]==to)) break;
        if (¬l2 || i<j ){ _a[i] = to; return true; } else return false;
    }
    bool final() const { for(int i=0;i<n;++i) if(_a[i]!=1) return false; return true; }
    void init() { for(int i=0;i<n;++i) _a[i] = 3; }
};
```

Hanoi tornyai

[3,3,3] **start**

állapot-gráfja



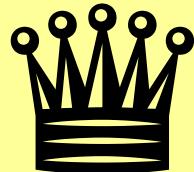
Állapottér vs. problématér

- A problématér elemeit többnyire a start csúcsból kiinduló utak szimbolizálják.
 - A Hanoi tornyai problémánál a problématér elemei nem az állapotok (csúcsok), hanem a kezdőállapotból kivezető műveletsorozatok (startcsúcsból induló irányított utak), hiszen a megoldás sem egyetlen állapot, hanem egy kezdőállapotot végállapotba vivő műveletsorozat (startcsúcsból célcsúcsba vezető irányított út).
 - Van amikor a megoldás egyetlen állapot (csúcs), de ebben az esetben is kell találni egy odavezető operátor-sorozatot (utat).
- Az állapottér-reprezentáció és a problématér között **szoros kapcsolat áll fenn**, de az állapottér többnyire **nem azonos** a problématérrel.

Állapot-gráf bonyolultsága

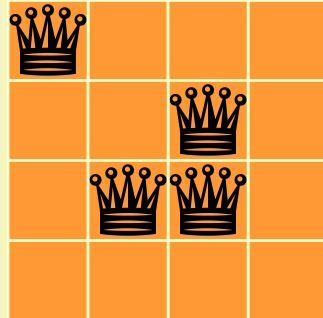


- A start csúcsból kivezető utak száma az oda-vissza lépések nélkül
Hanoi: a legfeljebb k hosszú utak: $1+2+\dots+2^k$, azaz $2^{k+1}-1$
 - csúcsok és élek száma
Hanoi: 3^n csúcs, $3 \cdot \frac{3^n - 1}{3 - 1}$ él
 - a csúcsok átlagos ki-foka
Hanoi: 3
 - körök gyakorisága, és hosszuk sokfélesége
Hanoi: 2, 3, 6, 7, 9, ...

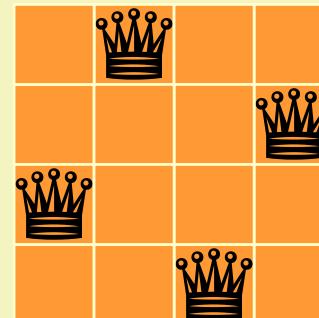


n-királynő probléma 1.

általános állapot



utófeltételnek megfelelő állapot



Állapottér: $AT = \{\text{퀸}, _\}^{n \times n}$

kétdimenziós tömb ($n \times n$ -es mátrix),
mely elemei {퀸, _} halmazbeliek

invariáns: egy állapot (tábla) pontosan n darab királynőt tartalmaz

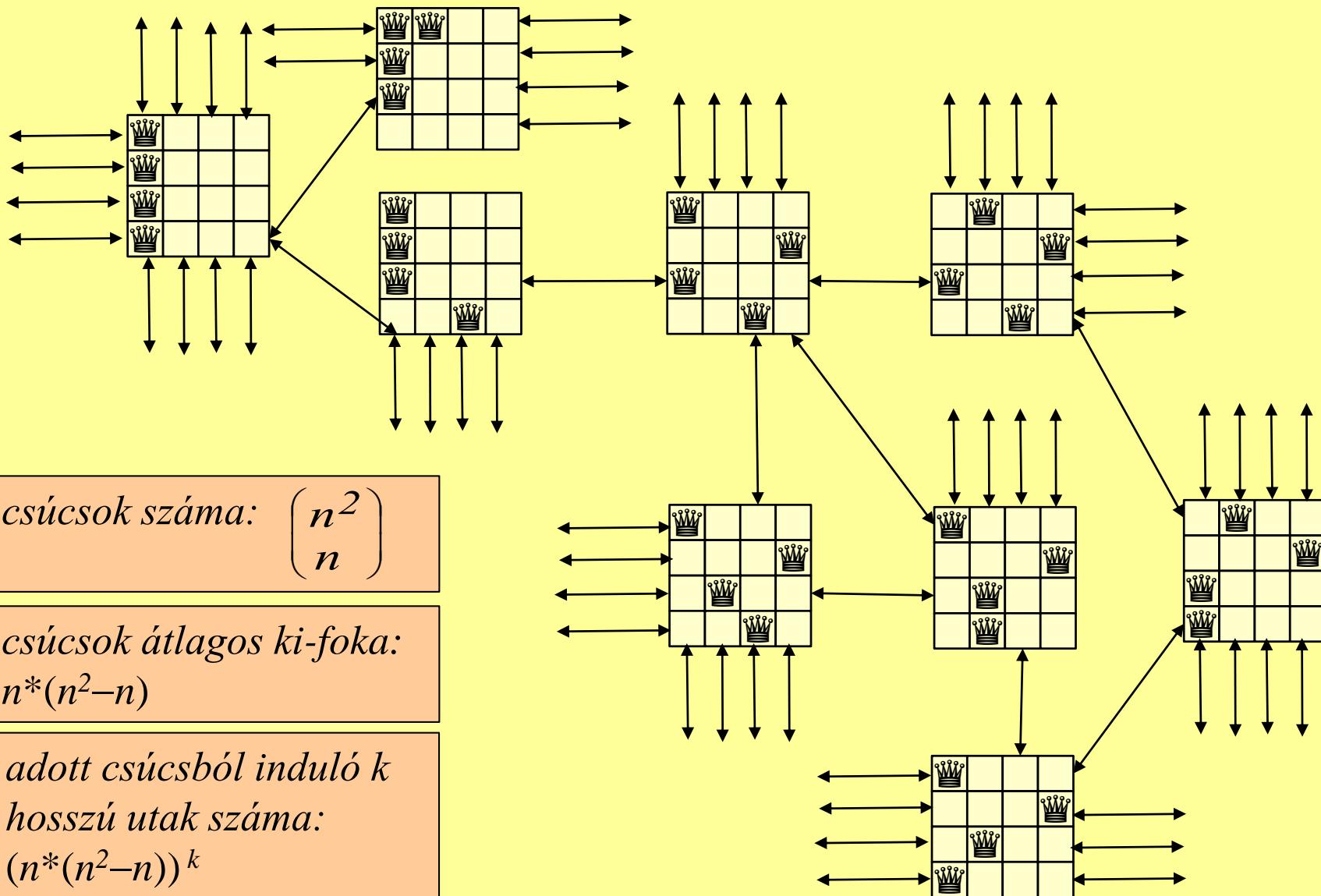
Művelet: $\text{Áthelyez}(x,y,u,v):AT \rightarrow AT$ $x,y,u,v \in [1..n]$ (this:AT)

HA $1 \leqq x,y,u,v \leqq n$ és $this[x,y] = \text{퀸}$ és $this[u,v] = _$

AKKOR $this[x,y] \leftrightarrow this[u,v]$

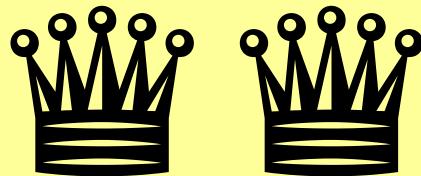
csere

Állapot-gráf részlet



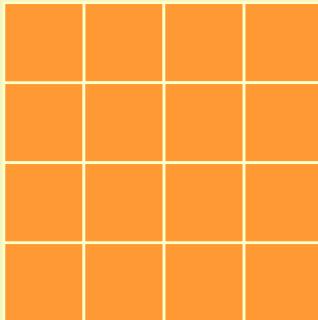
Csökkentsük a problématér méretét

- Ugyanannak a feladatnak több modellje lehet : érdemes olyat keresni, amely kisebb problémateret jelöl ki.
 - Az előző reprezentációnál a problématér mérete, azaz a lehetséges utak száma, óriási.
 - **Bővítsük az állapotteret** az n -nél kevesebb királynőt tartalmazó állásokkal, és **használjunk új műveletet** : királynő-felhelyezést (kezdő állás az üres tábla). Ekkor a pontosan n hosszú utak (ilyen a jó megoldás is) száma: $\binom{n^2}{n} \cdot n!$
 - **Műveletek előfeltételének szigorításával** csökken az állapotgráf átlagos ki-foka:
 - **Sorról sorra haladva csak egy-egy királynőt helyezzünk fel a táblára!** Ekkor az n hosszú utak száma: n^n .
 - **Ütést tartalmazó állásra ne tegyük királynőt!**

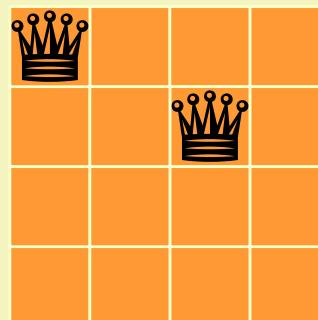


n-királynő probléma 2.

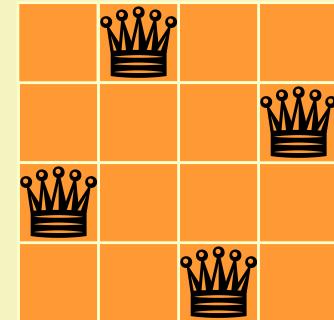
kezdőállapot



közülüső állapot



végállapot



Állapottér: $AT = \{ \text{퀸}, _ \}^{n \times n}$

nincs már üres sor és nincs ütés

invariáns: az első néhány sor egy-egy királynőt tartalmaz

Művelet: **Helyez(oszlop):** $AT \rightarrow AT$ $oszlop \in [1..n]$ (*this:AT*)

HA $1 \leq oszlop \leq n$ és a *this*-beli soron következő üres sor $\leq n$
és nincs ütés a *this*-ben

AKKOR *this*[a *this*-beli soron következő üres sor, *oszlop*] :=

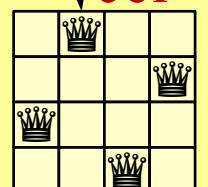
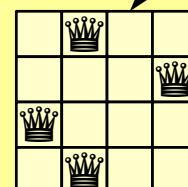
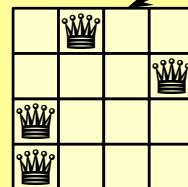
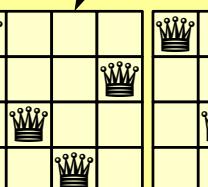
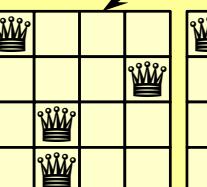
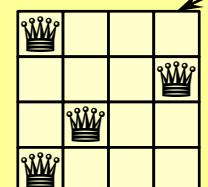
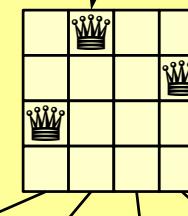
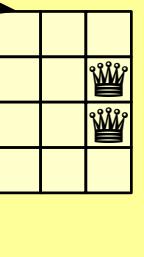
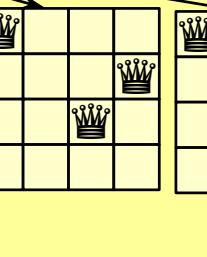
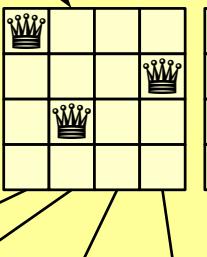
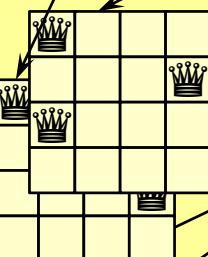
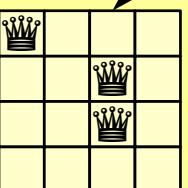
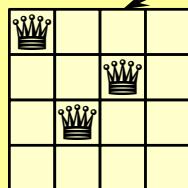
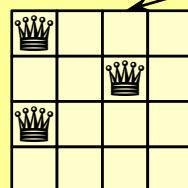
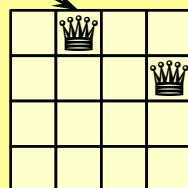
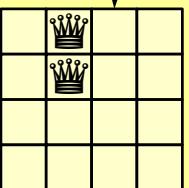
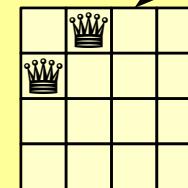
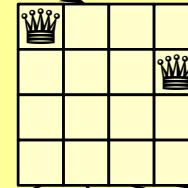
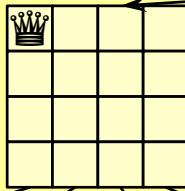
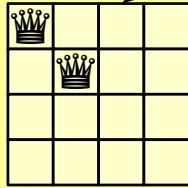
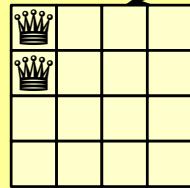
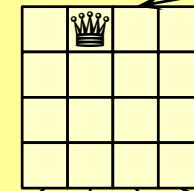
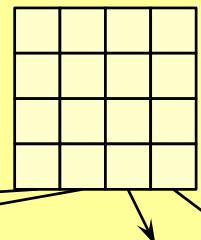
$csúcsok < (n^{n+1} - 1)/(n-1)$

csúcs ki-foka: n

lehetséges megoldások $< n^n$

Állapot-gráf

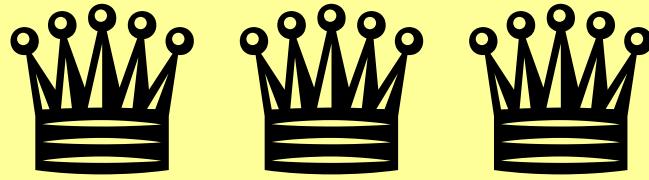
start



cél

Művelet végrehajtásának hatékonysága

- ❑ A művelet kiszámítási bonyolultsága csökkenthető, ha az állapotokat extra információval egészítjük ki, vagy az invariáns szigorításával szűkítjük az állapotteret.
- ❑ Például
 - A tábla soron következő üres sorának sorszámát eltárolhatjuk a tábla mellett az állapotokban, így új királynő elhelyezésekor ezt nem kell kiszámolni, ugyanakkor könnyen aktualizálhatjuk azt a növelésével egy művelet végrehajtásakor.
 - Ne engedjünk meg ütést létrehozni a táblán, hogy ne kelljen ezt a tulajdonságot külön ellenőrizni. Ennek céljából megjelöljük az ütés alatt álló üres (tehát már nem szabad) mezőket, amelyekre nem helyezhetünk fel királynőt. Egy mező státusza három féle lesz: szabad, ütés alatt álló vagy foglalt, amelyeket a művelet végrehajtásakor kell karbantartani.



n-királynő probléma 3.

kezdőállapot:

kövsor = 1

közbulső állapot:

kövsor = 3

👑	✗	✗	✗
✗	✗	👑	✗
✗	✗	✗	✗
✗		✗	✗

végállapot:

kövsor = 5

✗	👑	✗	✗
✗	✗	✗	👑
👑	✗	✗	✗
✗	✗	👑	✗

Állapottér: $AT = rec(t : \{ \text{👑}, \text{✗}, _ \}^{n \times n}, kövsor : \mathbb{N})$

invariáns:

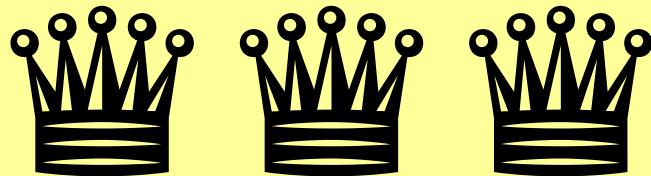
királynők nem ütik egymást,

kövsor $\leq n+1$,

az első *kövsor*-1 darab sor egy-egy királynőt tartalmaz,

✗ egy királynő által ütött üres mezőt jelöli,

_ az ütésben nem álló (szabad) üres mezőt jelöli.



n-királynő probléma 3. folytatás

Művelet: új királynő elhelyezése a soron következő üres sor szabad mezőjére

Helyez(oszlop): $AT \rightarrow AT$ $oszlop \in [1.. n]$ (*this:AT*)

HA $1 \leq oszlop \leq n$ és *this.kövsor* $\leq n$
és *this.t[this.kövsor,oszlop] := _*

AKKOR *this.t[this.kövsor,oszlop] :=*

előfeltétel számítás-
igénye: konstans

$\forall i \in [this.kövsor + 1.. n] : this.t[i, oszlop] := \mathbf{x}$

hatás számítás-
igénye: lineáris

this.t[i, i - this.kövsor + oszlop] := x

this.t[i, this.kövsor + oszlop - i] := x

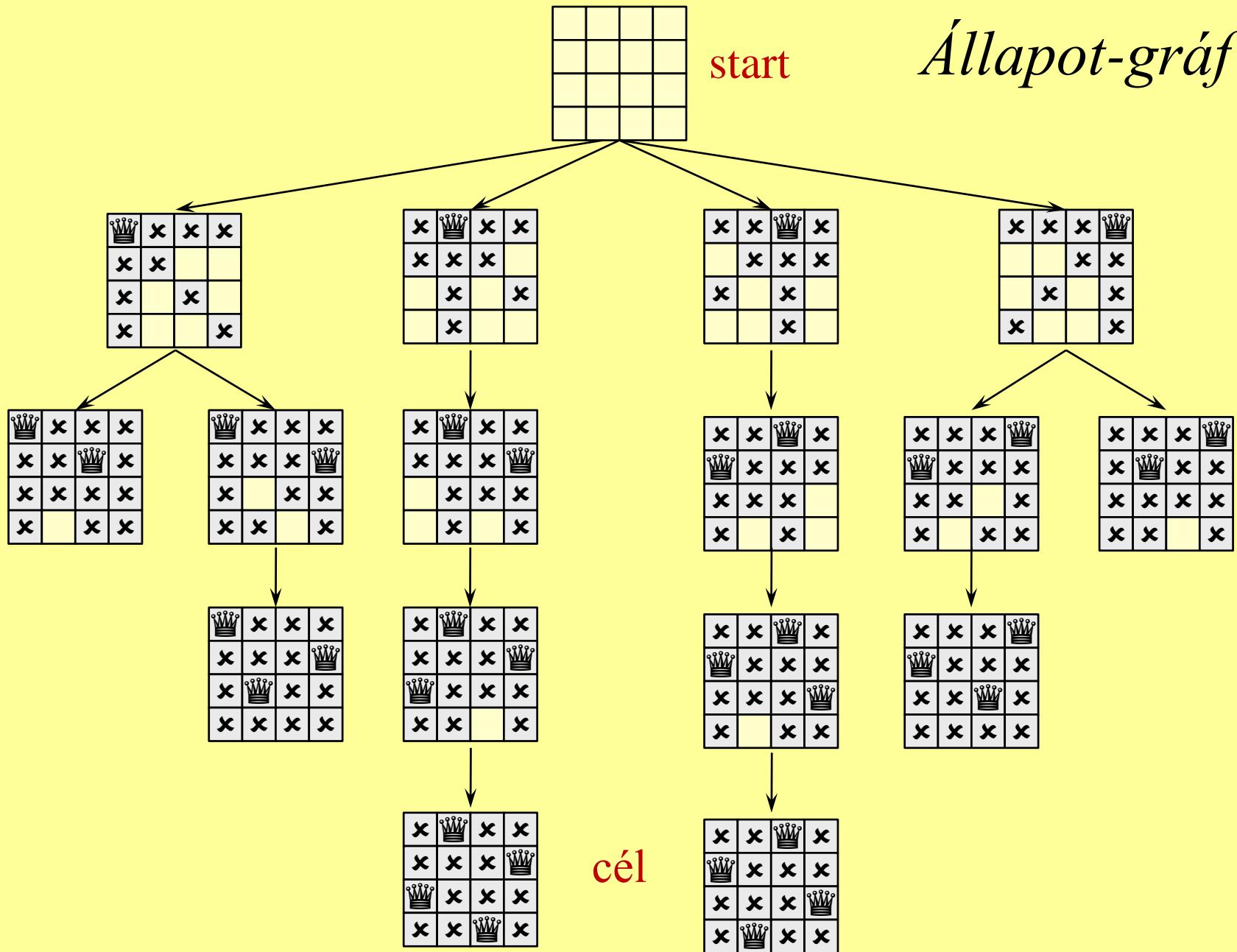
this.kövsor := this.kövsor + 1

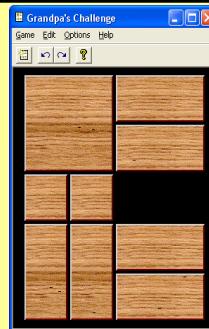
Kezdőállapot: *this.t* egy üres mátrix, *this.kövsor := 1*

Végállapot: *this.kövsor > n*

célfeltétel nagyon egyszerű lett

Állapot-gráf





Tologató játék (8-as, 15-ös)

Kezdőállapot:
tetszőleges

2	8	3
1	6	4
7		5



1	2	3
8		4
7	6	5

Végállapot:
szokásos

Állapottér: $AT = rec(\text{mátrix} : \{0..8\}^{3 \times 3}, \text{üres} : \{1..3\} \times \{1..3\})$

invariáns: egy állapot mátrixának sorfolytonos kiterítése a 0 .. 8 számok egy permutációja, az üres hely a 0 elem mátrixbeli sor és oszlopindexe.

Művelet: $Tol(irány) : AT \rightarrow AT$

koordinátánként értendő

HA

$irány \in \{(1,0), (0,1), (-1,0), (0,-1)\}$ és

$(1,1) \leq this.\text{üres} + irány \leq (3,3)$

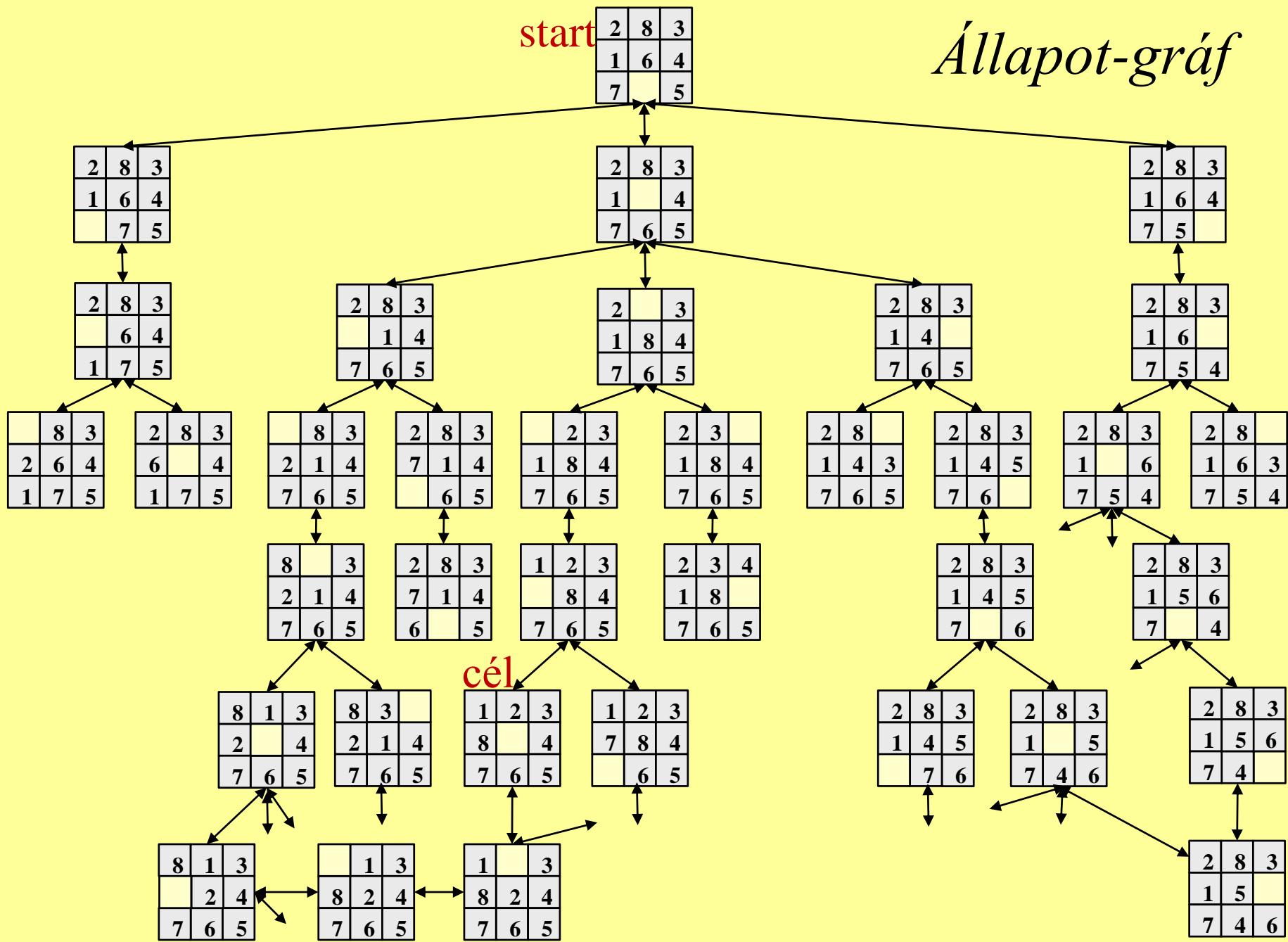
(this: AT)

AKKOR

$this.\text{mátrix}[this.\text{üres}] \leftrightarrow this.\text{mátrix}[this.\text{üres} + irány]$

$this.\text{üres} := this.\text{üres} + irány$

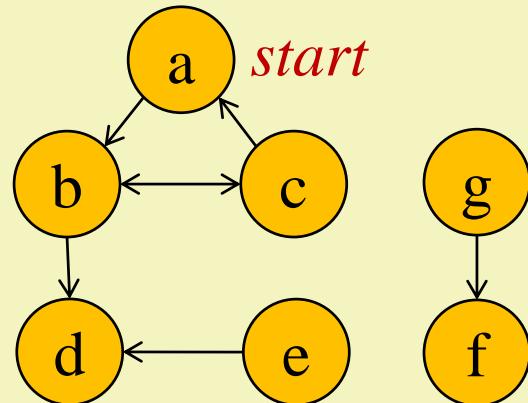
Állapot-gráf



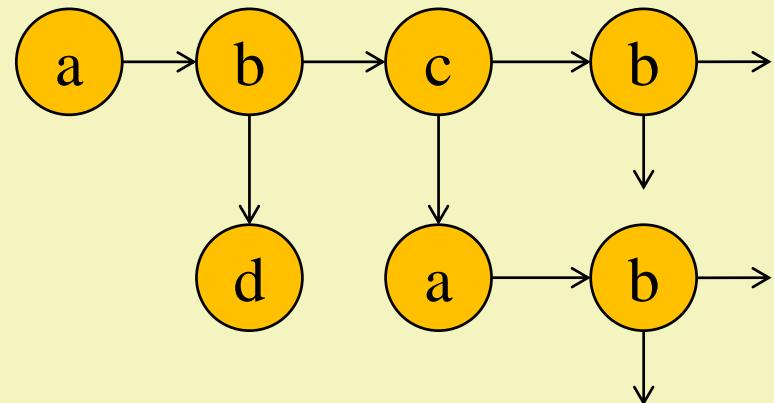
Hogyan „látja” egy keresés a reprezentációs gráfot?

- ❑ Egy keresés fokozatosan fedezi fel a reprezentációs gráfot: bizonyos részeihez el sem jut és a felfedezett részt sem feltétlenül tárolja el teljesen. Sőt kifejezetten **torzultan** „látja” a gráfot, ha egy csúcshoz érve nem vizsgálja meg, hogy ezt a csúcsot korábban már felfedezte-e, hanem ehelyett új csúcsként regisztrálja.

eredeti gráf:



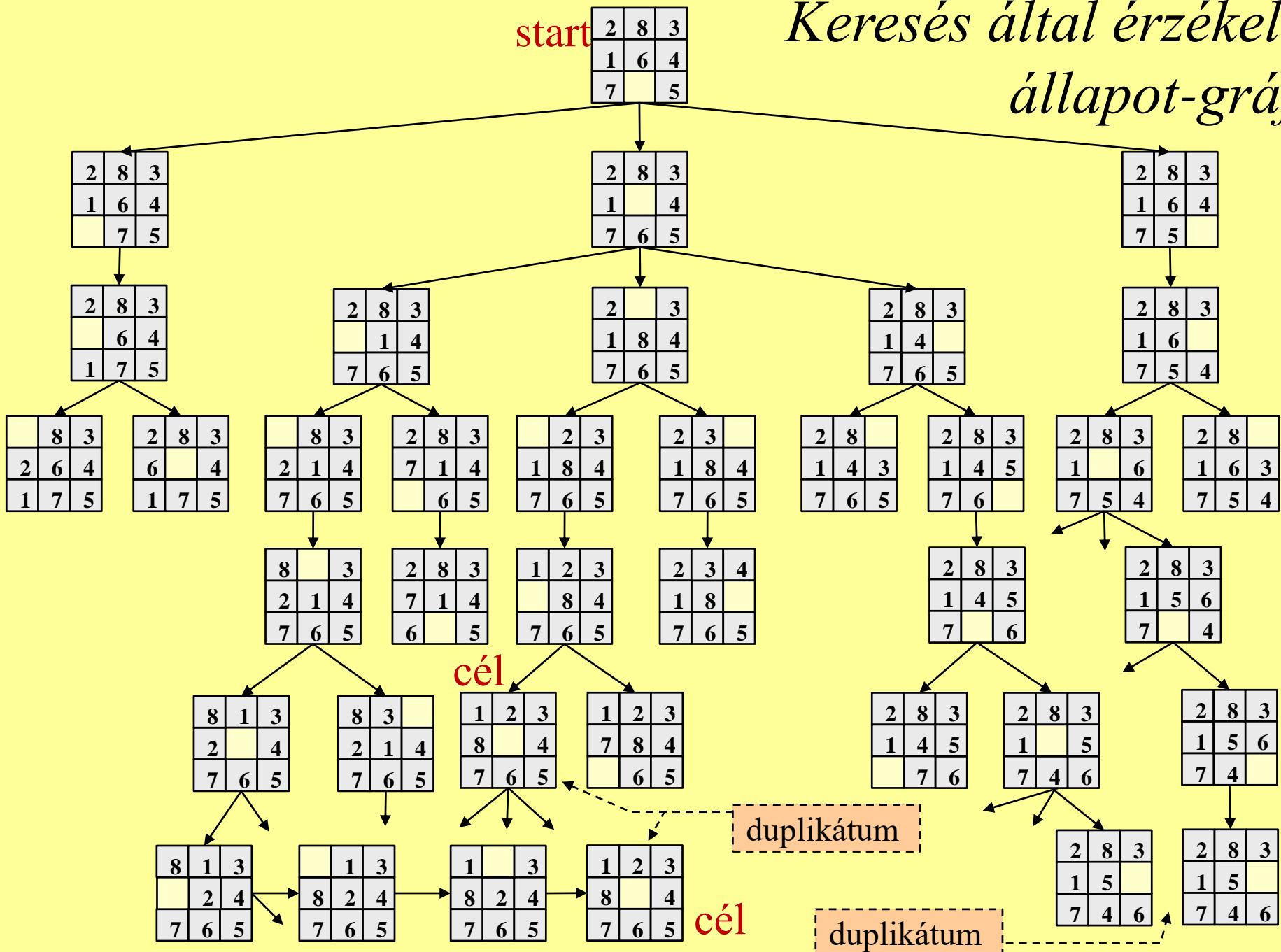
keresés által látott gráf:



Reprezentációs gráf „fává egyenesítése”

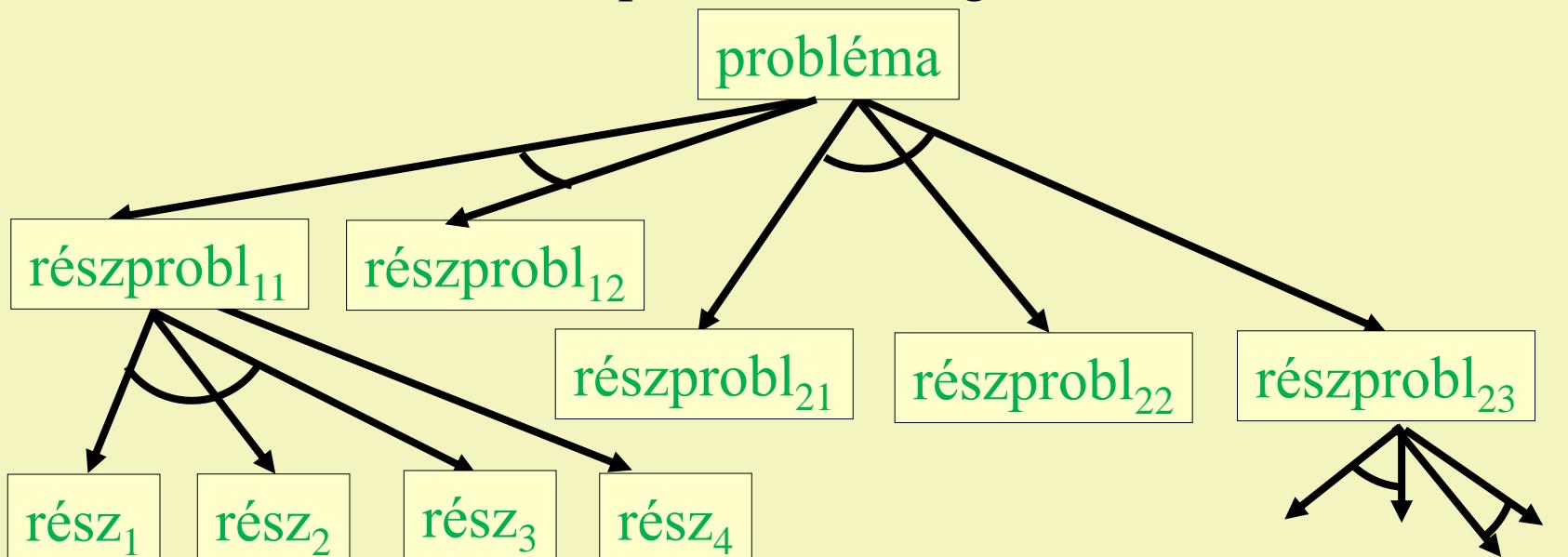
- Ha a keresés nem vizsgálja meg egy csúcsról, hogy korábban már felfedezte-e, akkor az eredeti reprezentációs gráf helyett valójában annak **fává kiegyenesített** változatában keres.
Előny: eltűnnek a körök, de a megoldási utak megmaradnak
Hátrány: duplikátumok jelennek meg, sőt a körök kiegyenesítése végtelen hosszú utakat eredményez
- A kétirányú (oda-vissza) élek különösen megnövelik a kiegyenesített fa méretét. Ezek kiszűrése azonban könnyen beépíthető minden keresésbe a megelőző aktuális csúcs (szülőcsúcs) eltárolásával, így az aktuális csúcsból oda **visszavezető él** felismerhető és figyelmen kívül hagyható.

Keresés által érzékelt állapot-gráf

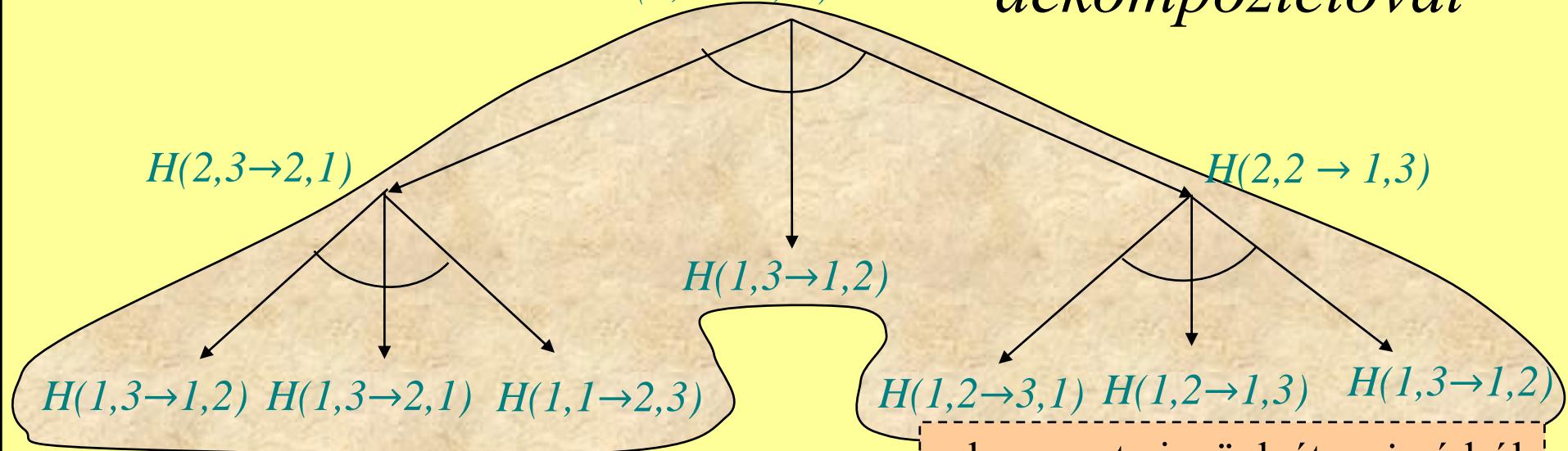


2. Probléma dekompozíció

- ❑ Egy probléma dekomponálása során a problémát részproblémákra bontjuk, majd azokat tovább részletezzük, amíg nyilvánvalóan megoldható problémákat nem kapunk.
- ❑ Sokszor egy probléma megoldását akár többféleképpen is fel lehet bontani részproblémák megoldásaira.



Hanoi tornyai probléma megoldása dekompozícióval



Probléma általános leírása: $H(n, i \rightarrow j, k)$

Kiinduló probléma: $H(3, 3 \rightarrow 1, 2)$

Egyszerű probléma: $H(1, i \rightarrow j, k)$

Dekomponálás: $H(n, i \rightarrow j, k) \rightsquigarrow < H(n-1, i \rightarrow k, j), H(1, i \rightarrow j, k), H(n-1, k \rightarrow j, i) >$

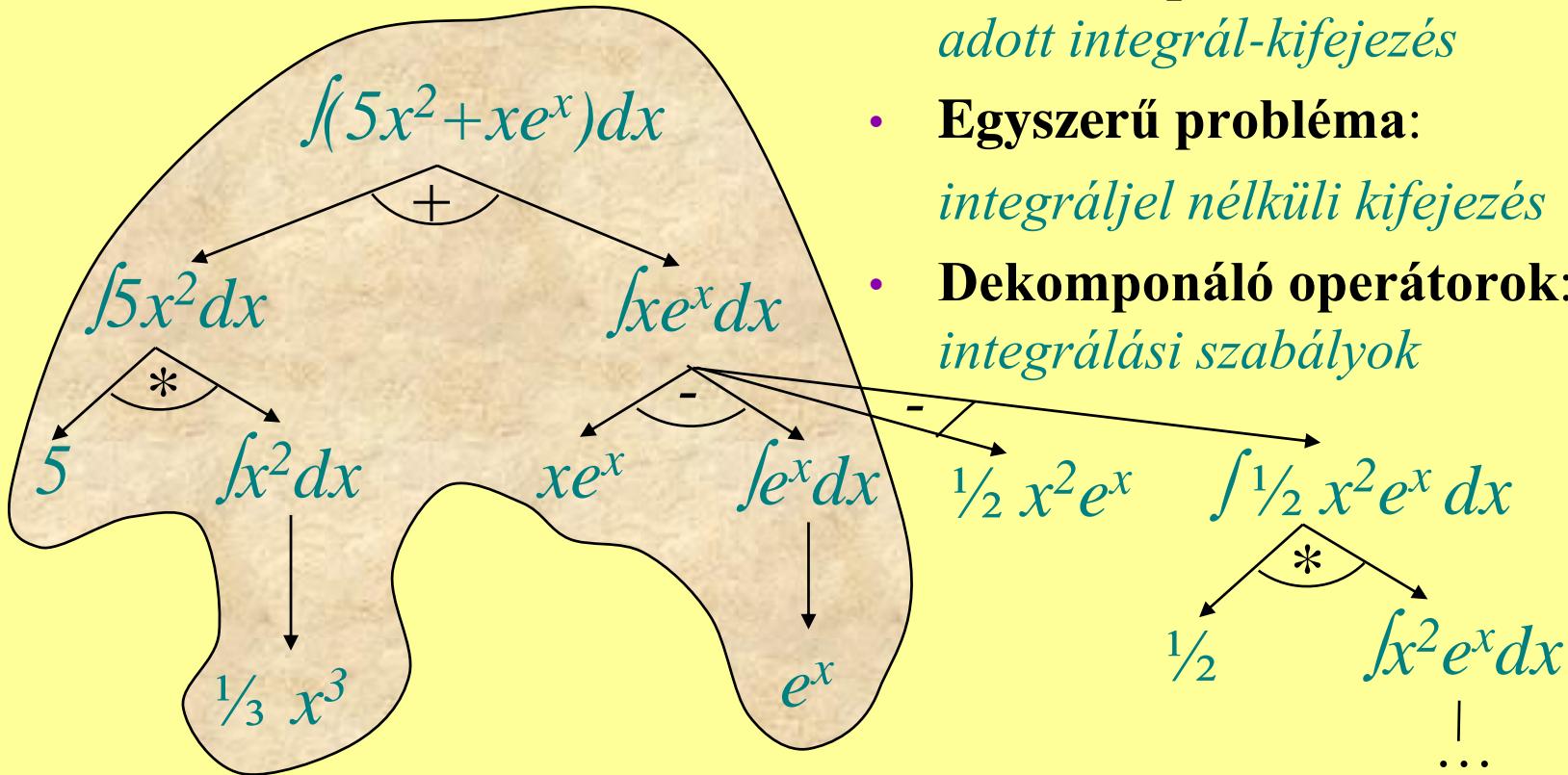
Megoldás gráf: kiinduló problémát egyszerű problémákra visszavezető fa

Megoldás: a részfa leveleit kell balról jobbra haladva összeolvasni

n korongot vigyük át az i . rúdról
a j . rúdra a k . rúd segítségével

eldönthető, hogy megoldható-e

Integrálszámítás



- **Probléma általános leírása:**
szintaktikusan helyes kifejezés
- **Kiinduló probléma:**
adott integrál-kifejezés
- **Egyszerű probléma:**
integráljel nélküli kifejezés
- **Dekomponáló operátorok:**
integrálási szabályok

Megoldás gráf: alternatívákat nem tartalmazó levezetés egy olyan részfa, amelynek belső csúcsaiból egyetlen élköteg indul ki, levelei egyszerű megoldható problémák

Megoldás: a megoldó részfa leveleit balról jobbra haladva kötjük össze a dekomponálások algebrai műveleteivel

Dekompozíciós reprezentáció fogalma

- A reprezentációhoz meg kell adnunk:
 - a feladat részproblémáinak általános leírását,
 - a kiinduló problémát,
 - az egyszerű problémákat, amelyekről könnyen eldönthető, hogy megoldhatók-e vagy sem, és
 - a dekomponáló műveleteket:
 - $D: \text{probléma} \rightarrow \text{probléma}^+$ és
$$D(p) = \langle p_1, \dots, p_n \rangle$$

A dekompozíció modellezése ÉS/VAGY gráffal

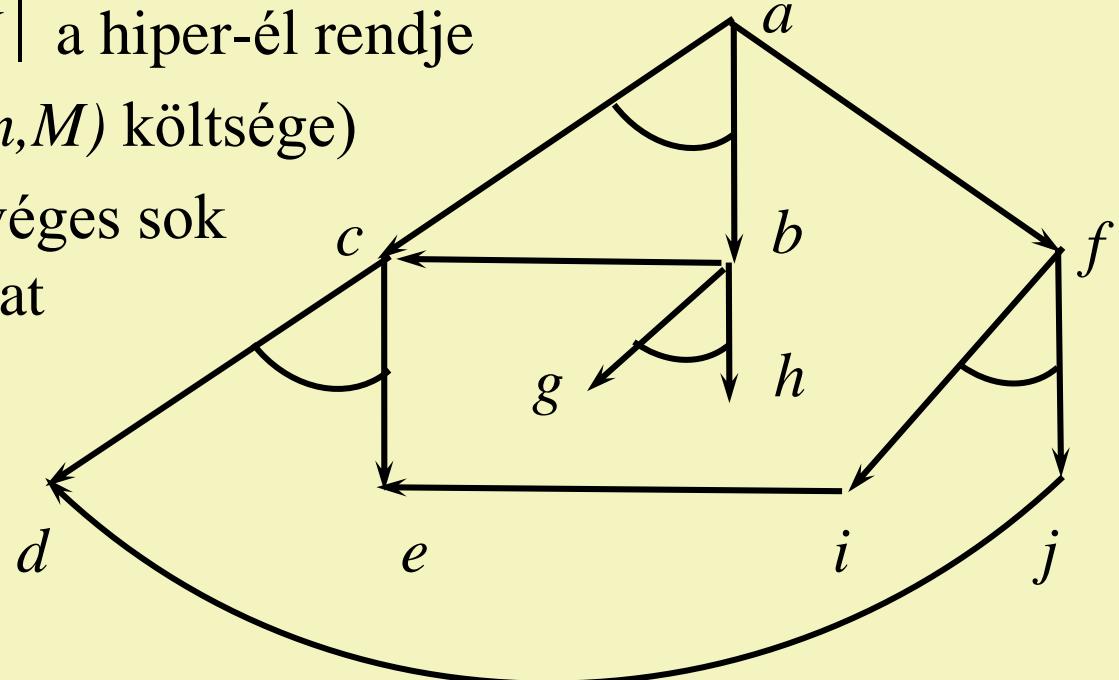
- ❑ Egy dekompozíciót egy ÉS/VAGY gráffal szemléltethetünk:
 - a csúcsok részproblémákat jelölnek, a kiinduló problémát a startcsúcs, a megoldható egyszerű problémákat a célcímszavak.
 - egy dekomponáló művelet hatását egy élkötégek mutatja, amely a dekomponált probléma csúcsából a dekomponálással előállított részproblémák csúcsaiba vezet.
 - egy élkötégek között ún. „ÉS” kapcsolat van: a dekomponált probléma megoldásához annak minden részproblémáját meg kell oldani;
 - egy csúcsból több élkötégek is indulhat, hiszen egy probléma többféleképpen dekomponálható. Ezen elkötegek között ún. „VAGY” kapcsolat áll fenn: választhatunk, hogy melyik élkötéget mentén oldunk meg egy problémát.

Megoldás-gráf

- ❑ Az eredeti problémát egyszerű problémákra visszavezető dekomponálási folyamatot az ÉS/VAGY gráf speciális részgráfja, az ún. **megoldás-gráf** jeleníti meg, amelyben
 - út vezet a startcsúcsból minden csúcsba, és minden csúcsból egy célcsúcsba
 - egy éllel együtt az összes azzal „ÉS” kapcsolatban álló éleket is tartalmazza (azaz teljes élkötegeket tartalmaz)
 - nem tartalmaz „VAGY” kapcsolatban álló él párokat
- ❑ A megoldás a megoldás-gráfból olvasható ki.
- ❑ Általában nincs összefüggés a megoldás-gráf költsége (bár hogyan is definiáljuk ezt) és a megoldás költsége között.

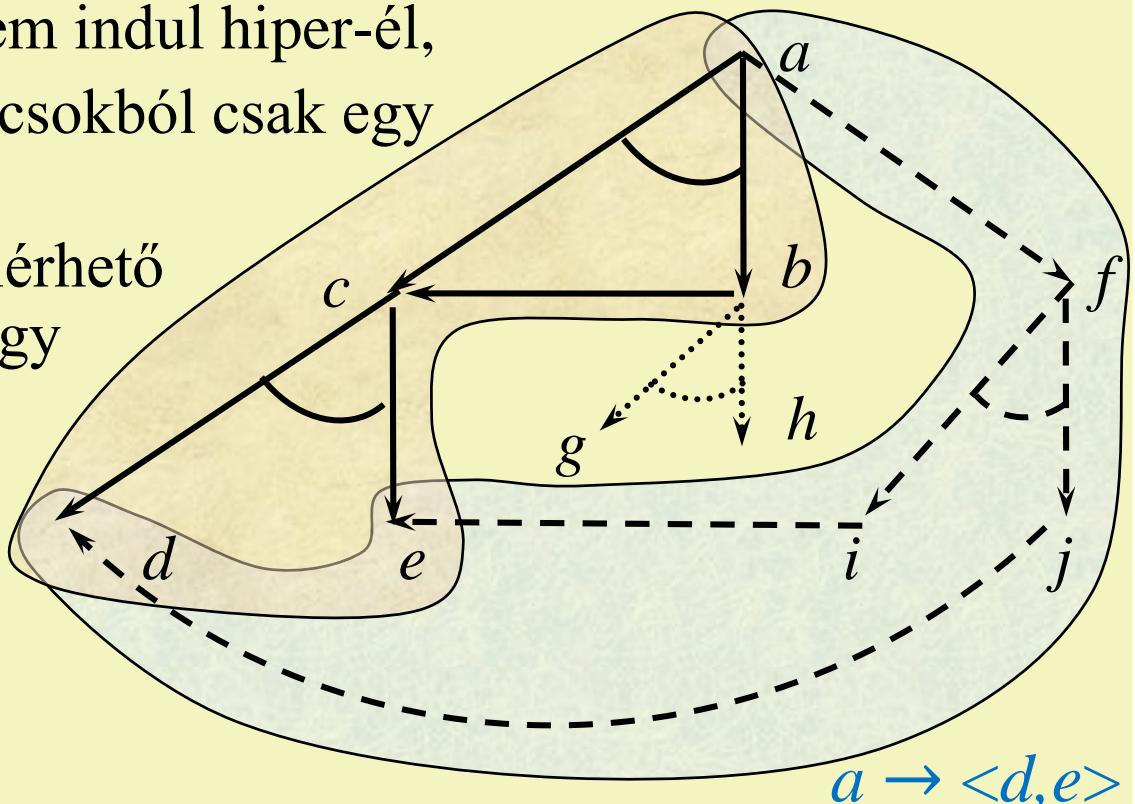
ÉS/VAGY gráfok

1. Az $R=(N,A)$ élsúlyozott irányított hiper-gráf, ahol az
 - N a csúcsok halmaza,
 - $A \subseteq \{ (n,M) \in N \times N^+ \mid 0 \neq |M| < \infty \}$ a hiper-élek halmaza, $|M|$ a hiper-él rendje
 - $(c(n,M))$ az (n,M) költsége)
2. Egy csúcsból véges sok hiper-él indulhat
3. $(0 < \delta \leq c(n,M))$



Az n csúcsból az M csúcs-sorozatba vezető irányított hiper-út fogalma

- ❑ Az $n^{\alpha} \rightarrow M$ hiper-út ($n \in N$, $M \in N^+$) egy olyan véges részgráf, amelyben
 - M csúcsaiból nem indul hiper-él,
 - M -en kívüli csúcsokból csak egy hiper-él indul,
 - minden csúcs elérhető az n csúcsból egy közönséges irányított úton.
- ❑ Hiper-út hossza az éleinek száma.
- ❑ Hiper-út költsége is definiálható.



A hiper-út és a megoldás kapcsolata

- ❑ Az $n^{\alpha} \rightarrow M$ hiper-út egy egyértelmű haladási irányt jelöl ki az n csúcsból az M csúcsaiba.
- ❑ A probléma dekompozíciójánál említett megoldás-gráf nem más, mint egy olyan hiper-út, amely a startcsúcsból egy célcsúcs-sorozatba vezet. Ez a célcsúcs-sorozat adja az eredeti probléma megoldását.
- ❑ A megoldás-gráf megtalálásához tehát a startcsúcsból kivezető hiper-utakat kell megvizsgálni.
- ❑ A startcsúcsból kivezető hiper-utak szerepe ugyanaz, mint a közönséges irányított gráfokban a startcsúcsból induló közönséges irányított utaké.

Különbség a közönséges út és a hiper-út bejárása között

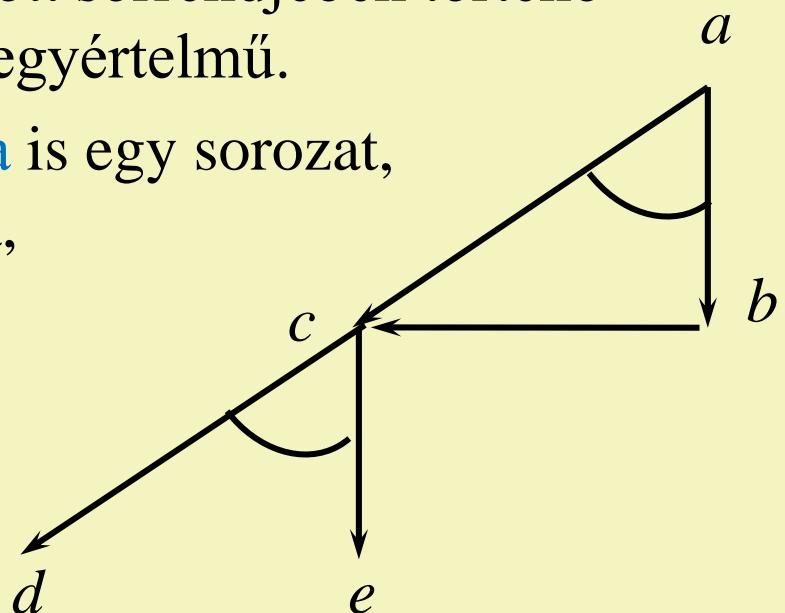
- Egy közönséges irányított út bejárásán az úton fekvő csúcsoknak az élek által mutatott sorrendjében történő felsorolását értjük. Ez minden egyértelmű.
- Egy irányított hiper-út bejárása is egy sorozat, de ez csúcs-sorozatok sorozata, ami nem egyértelmű.

Minden lépésben egy csúcs összes előfordulását kicseréljük a csúcsból induló hiper-él végcsúcs-sorozatára

bejárások:

$$\langle a \rangle \rightarrow \langle c, b \rangle \rightarrow \langle c, c \rangle \rightarrow \langle d, e, d, e \rangle$$

$$\langle a \rangle \rightarrow \langle c, b \rangle \rightarrow \langle d, e, b \rangle \rightarrow \langle d, e, c \rangle \rightarrow \langle d, e, d, e \rangle$$



Hiper-út bejárása

- Az $n \rightarrow M$ hiper-út egy bejárásán a hiper-út csúcsaiból képzett sorozatoknak a felsorolását értjük:
 - első sorozat: $\langle n \rangle$
 - a C sorozatot a $C^{k \leftarrow K}$ sorozat követi (ahol C -ben k minden előfordulásának helyére a K sorozatot írjuk), ha a hiper-útnak van olyan (k, K) hiper-éle, ahol $k \in C$ de $k \notin M$.
- Megjegyzés:
 - Egy hiperútnak véges sok véges hosszú bejárása van.
 - Egy $n \rightarrow T$ hiper-út (azaz egy megoldás-gráf) bejárásainak utolsó eleme egy csupa célcsúcsot tartalmazó sorozat.

Útkeresés ÉS/VAGY gráfban

- ❑ Egy ÉS/VAGY gráf startból induló hiper-útjainak (a potenciális megoldás gráfoknak) a bejárásai közönséges irányított utakként ábrázolhatók, és ezáltal egy közönséges irányított gráfot írnak le, amelynek csúcsai az eredeti ÉS/VAGY gráf csúcsainak véges sorozatai.
- ❑ Ha ebben a közönséges gráfban megoldási (azaz csupa célcímszámot tartalmazó sorozatba vezető) utat találunk, akkor az egyben az eredeti ÉS/VAGY gráf megoldás-gráfja is lesz.
- ❑ Az ÉS/VAGY gráfbeli megoldás-gráf keresést tehát közönséges irányított gráfbeli keresést végző útkereső algoritmusokkal végezhetjük el.



III. Keresések

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) loop

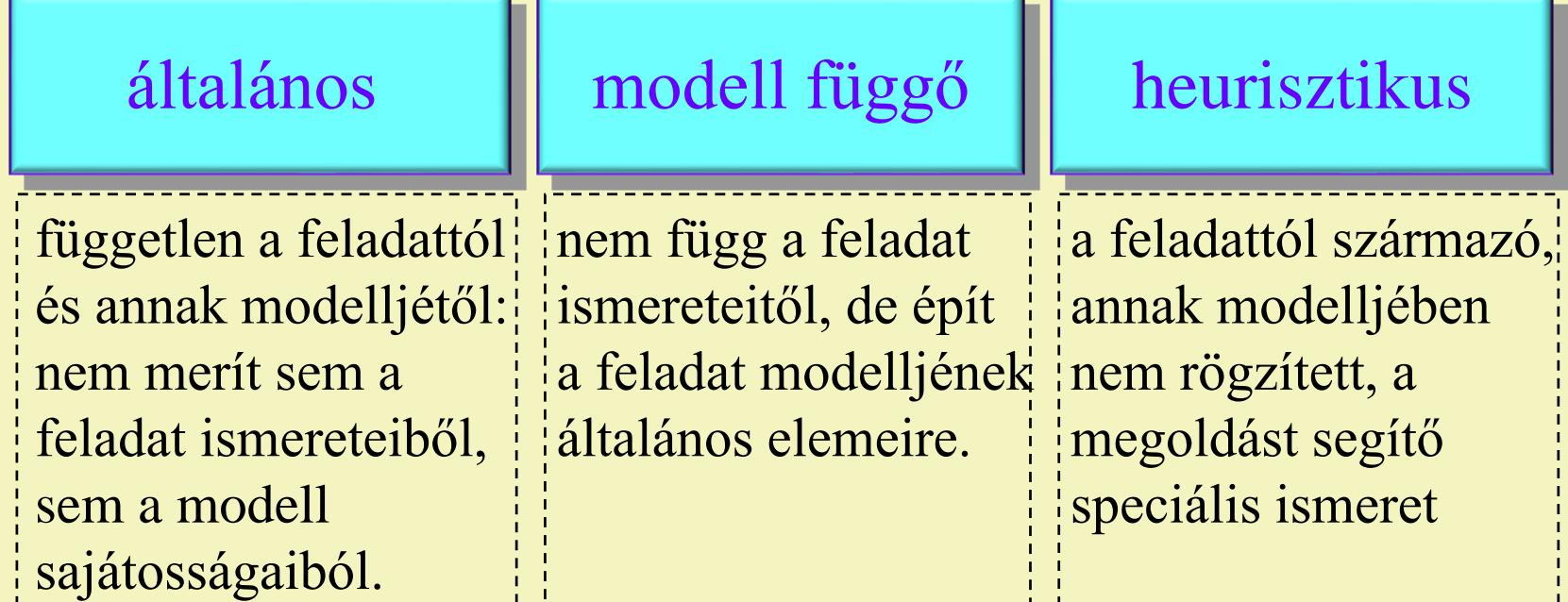
 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

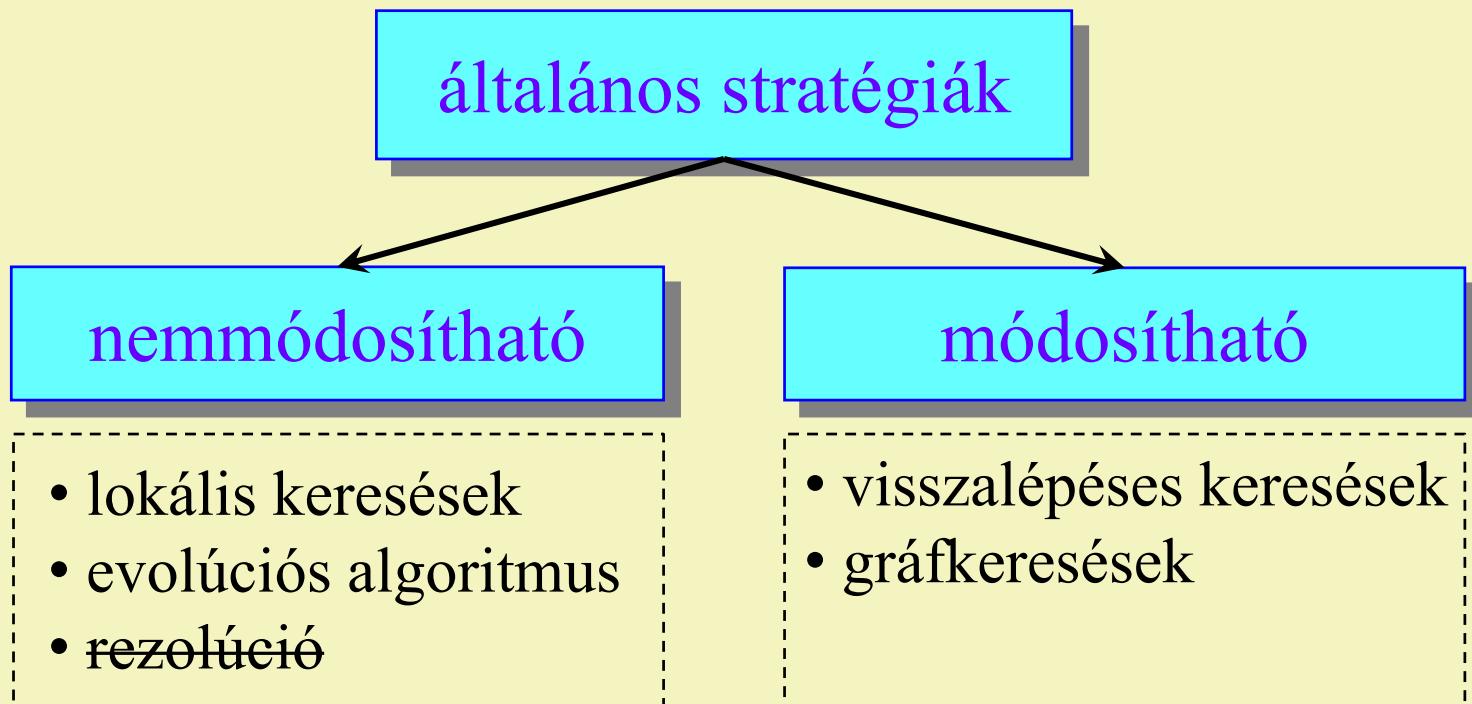
endloop

KR vezérlési szintjei

vezérlési stratégia



Általános vezérlési stratégiák



1. Lokális keresések

- ❑ A lokális keresés olyan KR, amely a megoldandó útkeresési probléma (reprezentációs) gráfjának egyetlen (az aktuális) csúcsát és annak szűk környezetét tárolja (a **globális munkaterületén**).
 - Kezdetben az aktuális csúcs a startcsúcs, és a keresés akkor áll le, ha az aktuális csúcs a célcsúcs lesz.
- ❑ Az aktuális csúcsot minden lépésben annak környezetéből vett „jobb” csúccsal cseréli le (**keresési szabály**).
- ❑ A „jobbság” eldöntéséhez (**vezérlési stratégia**) egy kiértékelő (cél-, rátermettségi-, heurisztikus) függvényt használ, amely reményeink szerint annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhöz.

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) loop

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Hegymászó algoritmus

- Mindig az aktuális (*akt*) csúcs legjobb gyermekére lép, amelyik lehetőleg nem a szülője.

1. *akt* := *start*

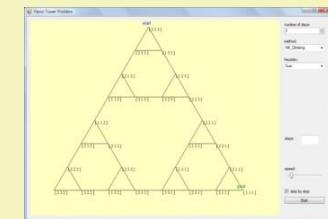
A bejárt út megadásához az
akt egymás után felvett
értékeit össze kell gyűjteni.

2. while *akt* $\notin T$ loop

3. *akt* := $\text{opt}_f(\Gamma(\text{akt}) - \pi(\text{akt}))$

4. endloop

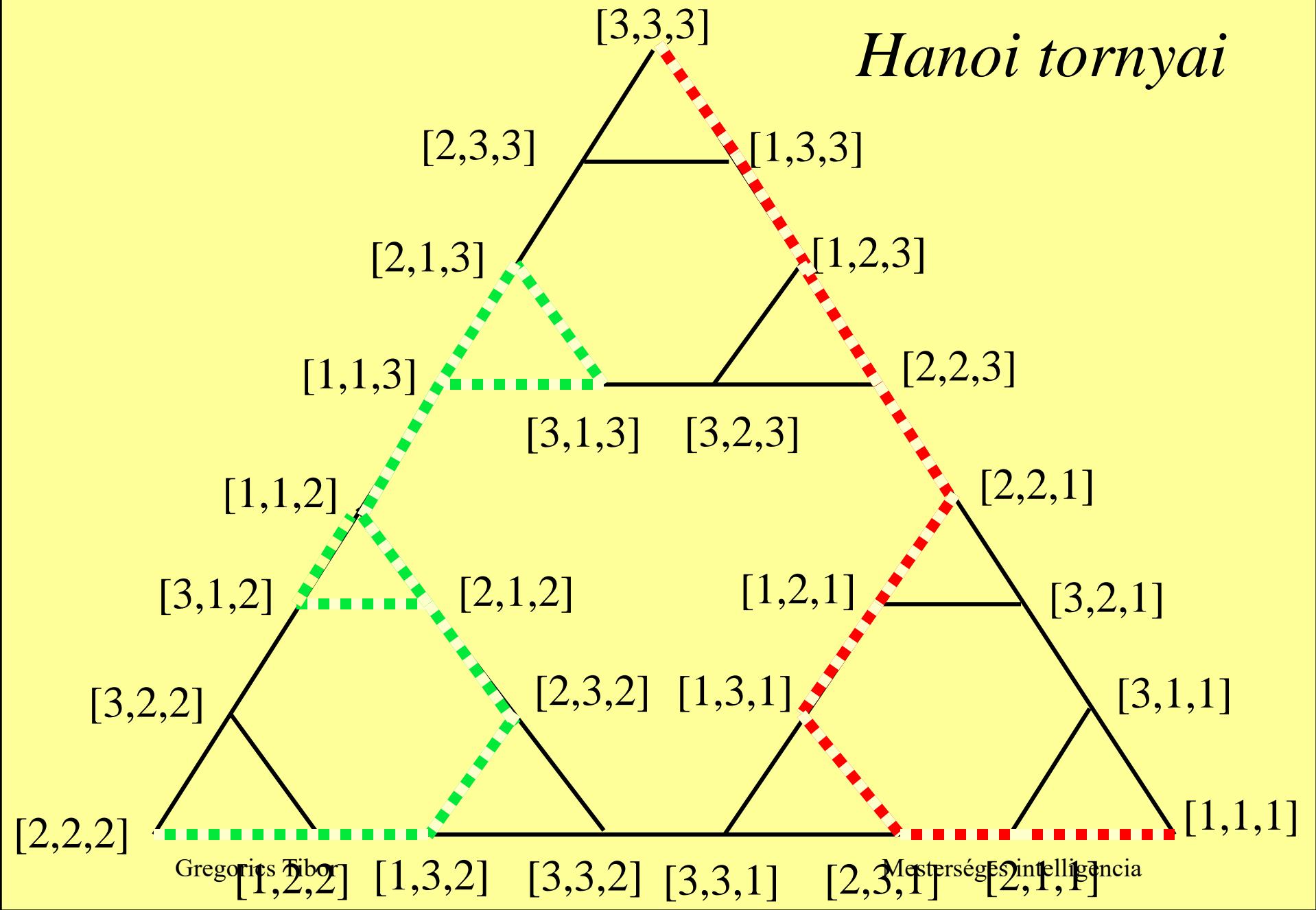
5. return *akt*



if $\Gamma(\text{akt}) = \emptyset$ then return nem talált megoldást
if $\Gamma(\text{akt}) - \pi(\text{akt}) = \emptyset$ then *akt* := $\pi(\text{akt})$
else *akt* := $\text{opt}_f(\Gamma(\text{akt}) - \pi(\text{akt}))$

- Az eredeti hegymászó algoritmus nem zárja ki a szülőre való lépest, viszont nem engedi meg, hogy az aktuális csúcsot egy rosszabb értékű csúcsra cseréljük (ilyenkor a keresés inkább leáll).

Hanoi tornyai



Hátrányok

- Csak erős heurisztika esetén lesz sikeres: különben „eltéved” (nem talál megoldást), sőt zsákutcába kerülve „beragad”. Segíthet, ha:
 - véletlenül választott startcsúcsból újra- és újraindítjuk
 - k aktuális csúcsnak a legjobb k gyerekére lépünk
 - gyengítjük a mohó stratégiát → szimulált hűtés
- Lokális optimum hely körül vagy ekvidisztans felületen (azonos értékű szomszédos csúcsok között) található körön, végtelen működésbe eshet. Segíthet, ha:
 - növeljük a memóriát → tabu keresés

Tabu keresés

- ❑ A **globális munkaterületén** az aktuális csúcson (*akt*) kívül nyilvántartja még
 - az utolsó néhány érintett csúcsot: *Tabu* halmaz
 - az eddigi legjobb csúcsot: optimális csúcs (*opt*)
- ❑ Egy **keresési szabály** minden lépésben
 - az aktuális csúcsnak a legjobb, de nem a *Tabu* halmazban levő, gyerekére lép
 - ha *akt* jobb, mint az *opt*, akkor *opt* az *akt* lesz
 - frissíti *akt*-tal a sorszerkezetű *Tabu* halmazt
- ❑ Terminálási feltételek:
 - ha az *opt* a célcímsúcs
 - ha az *opt* sokáig nem változik.

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) loop

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Tabu keresés algoritmusa

1. $akt, opt, Tabu := start, start, \emptyset$

2. while not ($opt \in T$ or opt régóta nem változik) loop

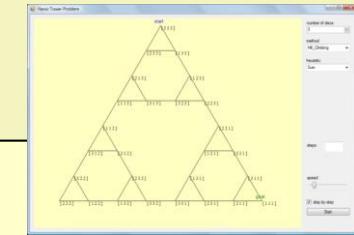
3. $akt := \text{opt}_f(\Gamma(akt) - Tabu)$

4. $Tabu := \text{Módosít}(akt, Tabu)$

5. if $f(akt)$ jobb, mint $f(opt)$ then $opt := akt$

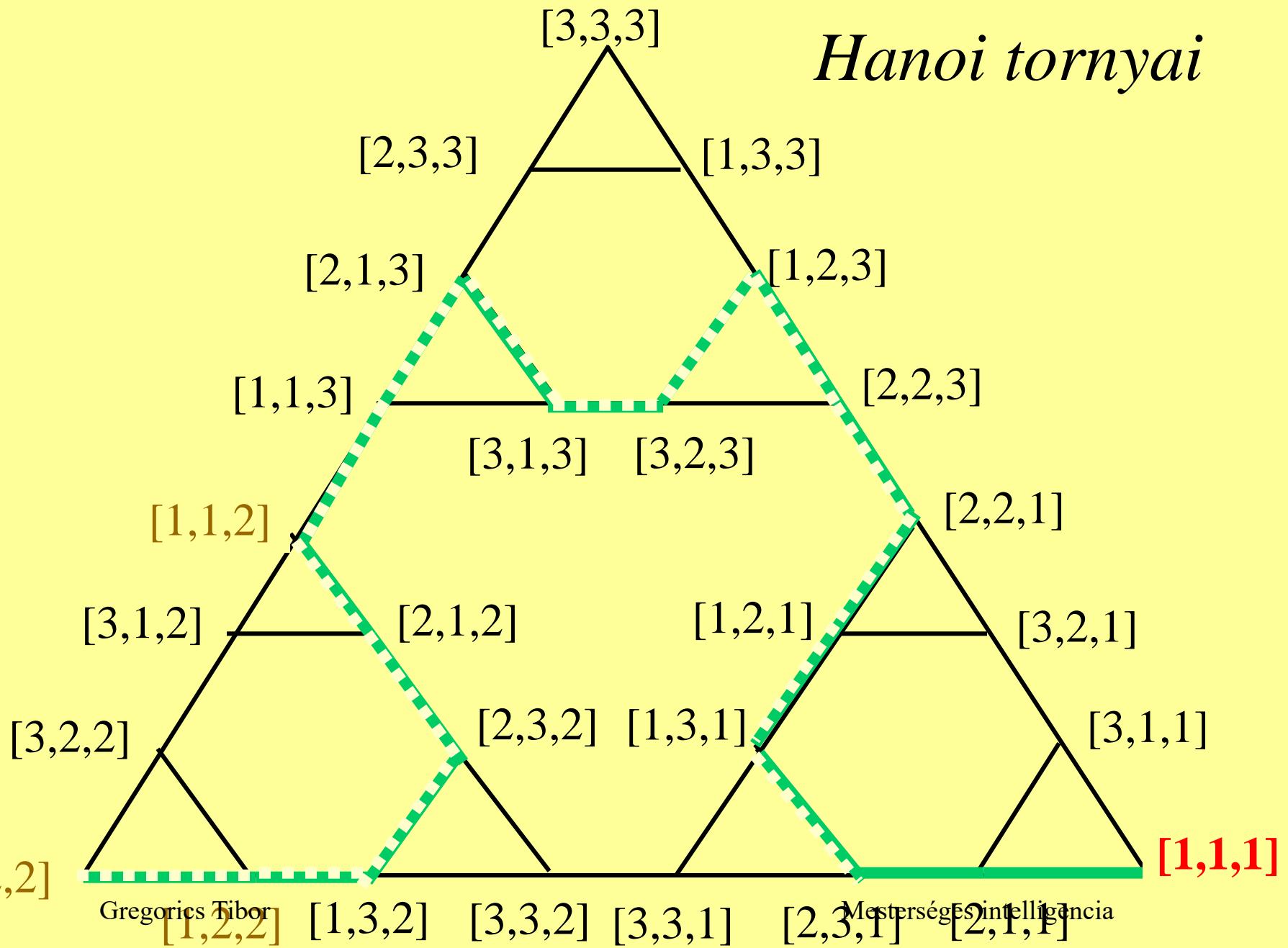
6. endloop

7. return akt



if $\Gamma(akt) = \emptyset$ then return nem talált megoldást
else if $\Gamma(akt) - Tabu = \emptyset$ then $akt := \text{opt}_f(\Gamma(akt))$
else $akt := \text{opt}_f(\Gamma(akt) - Tabu)$

Hanoi tornyai



Megjegyzés

❑ Előnyök:

- tabu méreténél rövidebb köröket észleli, és ez segíthet a lokális optimum hely illetve az ekvidisztans felület körüli körök leküzdésében.

❑ Hátrányok:

- a *Tabu* halmaz méretét kísérletezéssel kell belőni
- zsákutcába futva a nem-módosítható stratégia miatt beragad

Szimulált hűtés

- A **keresési szabály** a következő csúcsot **véletlenszerűen** választja ki az aktuális (*akt*) csúcs gyermekei közül.
- Ha az így kiválasztott *új* csúcs kiértékelő függvény-értéke nem rosszabb, mint az *akt* csúcsé (itt $f(\text{új}) \leq f(\text{akt})$), akkor elfogadjuk aktuális csúcsnak
- Ha az *új* csúcs függvényértéke rosszabb (itt $f(\text{új}) > f(\text{akt})$), akkor egy olyan véletlenített módszert alkalmazunk, ahol az *új* csúcs **elfogadásának valószínűsége** fordítottan arányos az $|f(\text{akt}) - f(\text{új})|$ különbséggel:

$$e^{\frac{f(\text{akt}) - f(\text{új})}{T}} > \text{random } [0,1]$$

Hűtési ütemterv

- Egy csúcs elfogadásának valószínűségét az elfogadási képlet kitevőjének T együtthatójával szabályozhatjuk. Ezt egy (T_k, L_k) $k=1,2,\dots$ ütemterv módosítja, amely L_1 lépésen keresztül T_1 , majd L_2 lépésen keresztül T_2 , stb. lesz.

$$e^{\frac{f(\text{akt}) - f(\text{új})}{T_k}} > \text{rand}[0,1]$$

$$f(\text{új}) = 120, f(\text{akt}) = 107$$

- Ha T_1, T_2, \dots szigorúan monoton csökken, akkor egy ugyanannyival rosszabb függvényértékű új csúcsot a keresés kezdetben nagyobb valószínűsséggel fogad majd el, mint később.

T	$\exp(-13/T)$
10^{10}	0.9999...
50	0.77
20	0.52
10	0.2725
5	0.0743
1	0.000002

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) loop

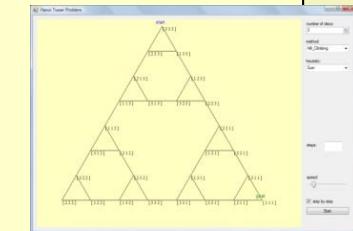
 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Szimulált hűtések algoritmusa

1. $akt := start ; k := 1 ; i := 1$
2. while not($akt \in T$ or $f(akt)$ régóta nem változik) loop
3. if $i > L_k$ then $k := k+1; i := 1$
4. $\acute{u}j := \text{select}(\Gamma(akt) - \pi(akt))$
5. if $f(\acute{u}j) \leq f(akt)$ or $e^{\frac{f(akt)-f(\acute{u}j)}{T_k}} > rand[0,1]$
~~or $f(\acute{u}j) > f(akt)$ and~~
6. then $akt := \acute{u}j$
7. *i* := *i*+1
8. endloop
9. return *akt*



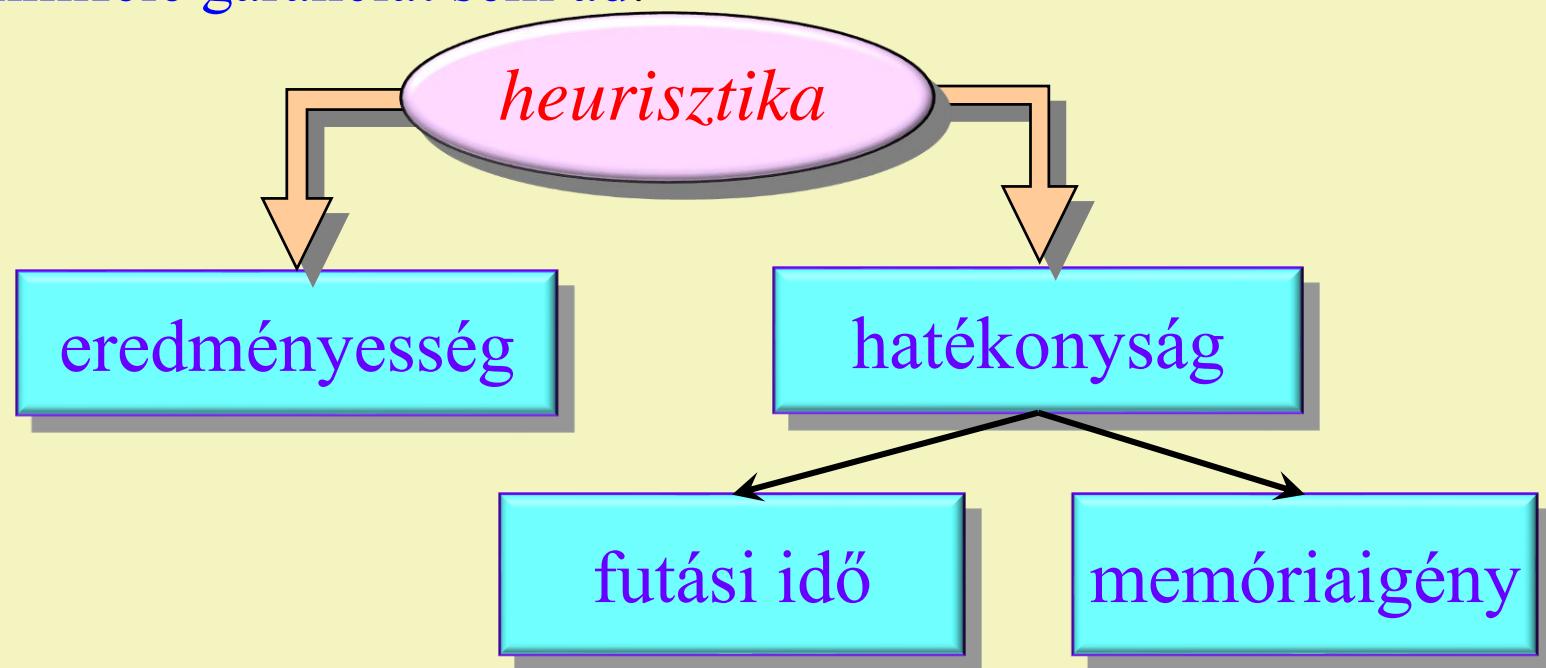
if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
if $\Gamma(akt) - \pi(akt) = \emptyset$ **then** $\acute{u}j := \pi(akt)$
else $\acute{u}j := \text{select}(\Gamma(akt) - \pi(akt))$

Lokális kereséssel megoldható feladatok

- Erős heurisztika nélkül nincs sok esély a cél megtalálására.
 - Jó heurisztikára épített kiértékelő függvényel elkerülhetőek a zsákutcák.
- A sikerhez az kell, hogy egy lokálisan hozott rossz döntés ne zárja ki a cél megtalálását!
 - Ez például erősen összefüggő reprezentációs-gráfban teljesül
 - Kifejezetten előnytelen, ha a reprezentációs-gráf egy irányított fa. (Például az n -királynő problémát csak tökéletes kiértékelő függvény esetén lehetne lokális kereséssel megoldani.)

A heurisztika hatása a KR működésére

A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon, habár erre általában semmiféle garanciát sem ad.



ADAT := kezdeti érték

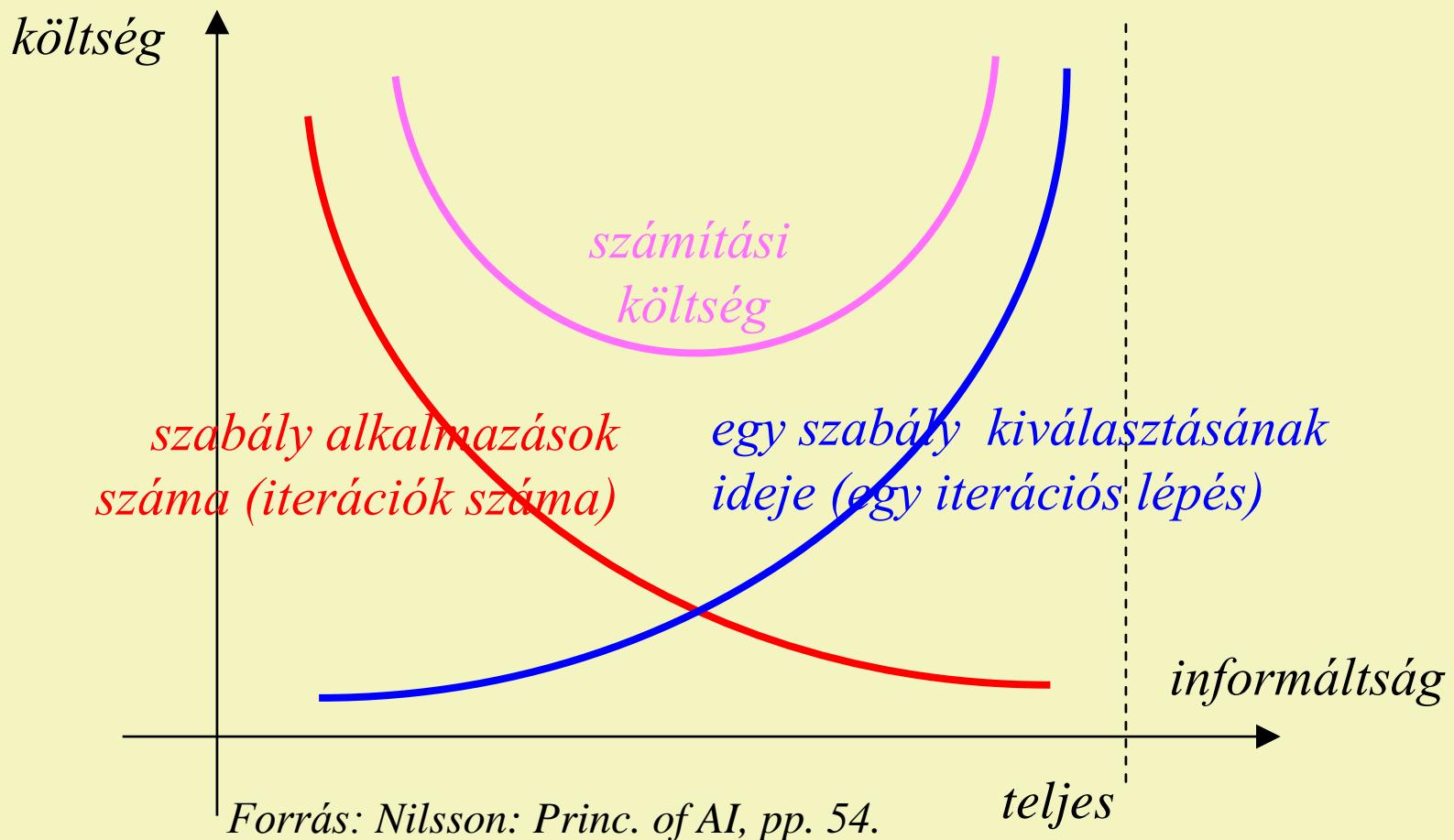
while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

KR hatékonysága

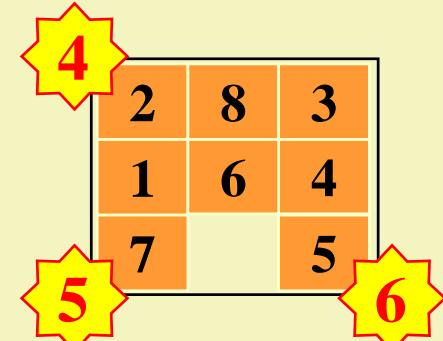


1	2	3
8		4
7	6	5

Heurisztikák a 8-as (15-ös) tologató játékra

- ❑ Rossz helyen levő lapkák száma:

$$W(this) = \sum_{i,j} \begin{cases} 1 & this[i,j] \neq 0 \wedge this[i,j] \neq cél[i,j] \\ 0 & \text{otherwise} \end{cases}$$



- ❑ Lapkák célbeli helyüktől vett minimális távolságainak összege (Manhattan):

$$P(this) = \sum_{i,j} (|i - célbelisor(this[i,j])| + |j - célbelioszlop(this[i,j])|)$$

célbelisor(this[i,j]) ~ a *this[i,j]* célállapotbeli helyének sora

célbelioszlop(this[i,j]) ~ a *this[i,j]* célállapotbeli helyének oszlopa

- ❑ „Széleken levő lapkák legyenek jók” (frame):

- Hány olyan lapka van a szélen, amelyiket nem a célbeli szomszédja követ az óra járásával megegyező irányban?
- Hány sarkokban ($\times 2$) nincs még a cél szerinti lapka?

W

Hegymászó keresés

	4	
2	8	3
1	6	4
7		5

	3	
2	8	3
1		4
7	6	5

	3	
2	8	3
1		4
7	6	5

l:5, u:3, r:5, d:-

l:3, u:3, r:4, d:

l: -, u:3, r: -, d:4

	3	
8	1	3
2		4
7	6	5

	3	
8		3
2	1	4
7	6	5

	3	
	8	3
2	1	4
7	6	5

l:3, u: -, r:4, d:4

l: -, u: -, r:4, d:3

l: -, u: -, r:3, d:

	3	
8	1	3
	2	4
7	6	5

	2	
8		3
8	2	4
7	6	5

	1	
1		3
8	2	4
7	6	5

	0	
1	2	3
8		4
7	6	5

l: -, u:2, r: -, d:4

l: -, u: -, r:1, d:

l: -, u: -, r:2, d:0

Csúcs utódainak sorrendje:
 < left, up, right, down >

P

Hegymászó keresés

Csúcs utódainak sorrendje:
 < left, up, right, down >

5

2	8	3
1	6	4
7		5

4

2	8	3
1		4
7	6	5

l:6, u:4, r:6, d:-

l:5, u:3, r:5, d:

W: l:3, u:3, r:4, d:

2

	2	3
1	8	4
7	6	5

3

2		3
1	8	4
7	6	5

l:-, u:-, r:, d:1

l:2, u:-, r:4, d:

1

1	2	3
	8	4
7	6	5

0

1	2	3
8		4
7	6	5

l:-, u:, r:0, d:2

F

Hegymászó keresés

Csúcs utódainak sorrendje:
 < left, up, right, down >

6

2	8	3
1	6	4
7		5

5

2	8	3
1		4
7	6	5

l:8, u:4, r:8, d:-

l:5, u:3, r:6, d:

W: l:3, u:3, r:4, d:

3

	2	3
1	8	4
7	6	5

3

2		3
1	8	4
7	6	5

l:-, u:-, r:, d:1

l:3, u:-, r:5, d:

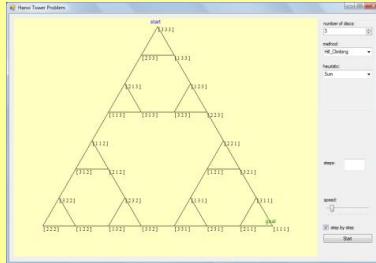
1

1	2	3
	8	4
7	6	5

0

1	2	3
8		4
7	6	5

l:-, u:, r:0, d:3



Heurisztikák a Hanoi tornyai problémára

❑ Darab:

$$C(this) = \sum_{\substack{i=1..n \\ this[i] \neq 1}} 1$$

❑ Súlyozott darab:

$$WC(this) = \sum_{\substack{i=1..n \\ this[i] \neq 1}} i$$

❑ Összeg:

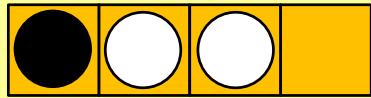
$$S(this) = \sum_{i=1..n} this[i]$$

❑ Súlyozott összeg:

$$WS(this) = \sum_{i=1..n} i \cdot this[i]$$

❑ Módosított összeg:

$$EWS(this) = WS(this) - \sum_{\substack{i=2..n \\ this[i-1] > this[i]}} 1 + \sum_{\substack{i=2..n-1 \\ this[i-1] = this[i+1] \wedge this[i] \neq this[i-1]}} 2$$



Heurisztikák a Fekete-fehér kirakóra

- Inverziószám:

$I(this)$ = minimálisan hány csere kell ahhoz, hogy minden fehér minden feketét megelőzzön

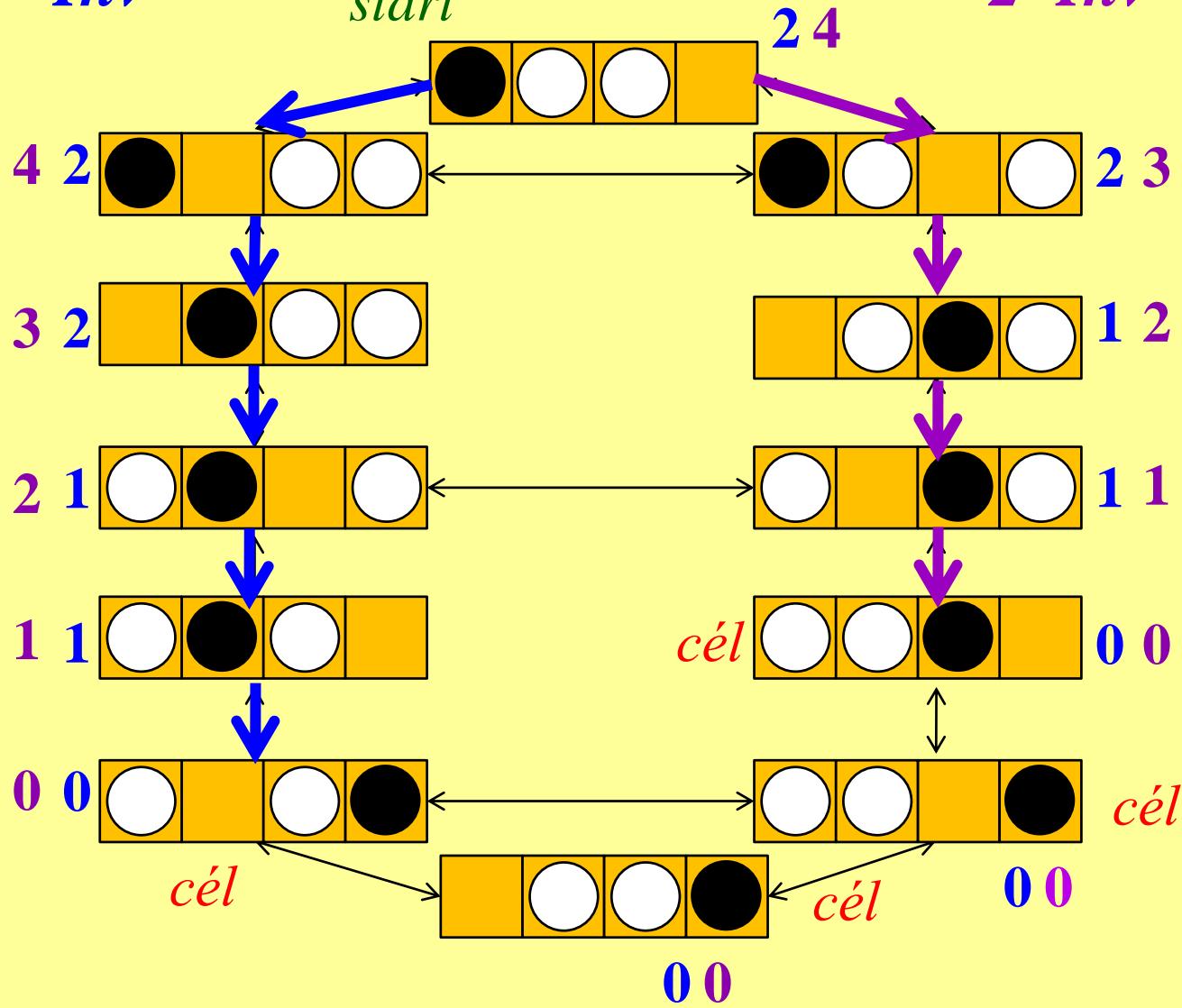
- Módosított inverziószám:

$M(this)= 2 \cdot I(this) -$
 $- (1, \text{ ha } this\text{-nek része } \square \square \square \text{ vagy } \square \square \square \square)$

Inv

start

$2*Inv - ?1$



2. Visszalépéses keresés



Visszalépéses keresés

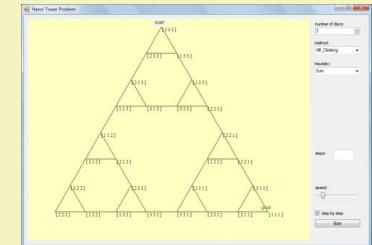
- A visszalépéses keresés egy olyan KR, amely
 - globális munkaterülete:
 - egy **út** a startcsúcsból az aktuális csúcsba (ezen kívül az útról leágazó még ki nem próbált élek)
 - kezdetben a startcsúcsot tartalmazó nulla hosszúságú út
 - terminálás célcíccsal vagy startcsúcsból való visszalépéssel
 - keresés szabályai:
 - a nyilvántartott út végéhez egy új (ki nem próbált) **él hozzáfűzése**, vagy a **legutolsó él törlése** (visszalépés szabálya)
 - vezérlés stratégiája a visszalépés szabályát csak a **legvégső esetben** alkalmazza

Visszalépés feltételei

- ❑ **zsákutca**: az aktuális csúcsból (azaz az aktuális út végpontjából) nem vezet tovább él
- ❑ **zsákutca torkolat**: az aktuális csúcsból kivezető utak nem vezettek célba
- ❑ **kör**: az aktuális csúcs szerepel már korábban is az aktuális úton
- ❑ **méliségi korlát**: az aktuális út hossza elér egy előre megadott értéket

Alacsonyabb rendű vezérlési stratégiák

- ❑ A vezérlési stratégia kiegészíthető:
 - **sorrendi szabály**: sorrendet ad az aktuális út végpontjából kivezető élek (utak) vizsgálatára
 - **vágó szabály**: megjelöli azokat az aktuális út végpontjából kivezető éleket (utakat), amelyeket nem érdemes megvizsgálni
- ❑ Ezek a szabályok lehetnek
 - másodlagos vezérlési stratégiák (a probléma modelljének sajátosságaiból származó ötlet)
 - heurisztikák (a probléma ismereteire támaszkodó ötlet)



Első változat: VL1

- A visszalépéses algoritmus első változata az, amikor a visszalépés feltételei közül **az első** kettőt építjük be a kereső rendszerbe.
- Bebizonyítható: *Véges körmentes irányított gráfokon a VL1 minden terminál, és ha létezik megoldás, akkor talál egyet.*
UI: véges sok adott startból induló út van.
- Rekurzív algoritmussal (VL1) szokták megadni
 - Indítás: *megoldás := VL1(startcsúcs)*

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

VLI

$A \sim \text{élek}$
 $seq(A) \sim \text{véges él sorozat}$

$N \sim \text{csúcsok}$

Recursive procedure $VLI(akt : N)$ **return** ($seq(A)$; $hiba$)

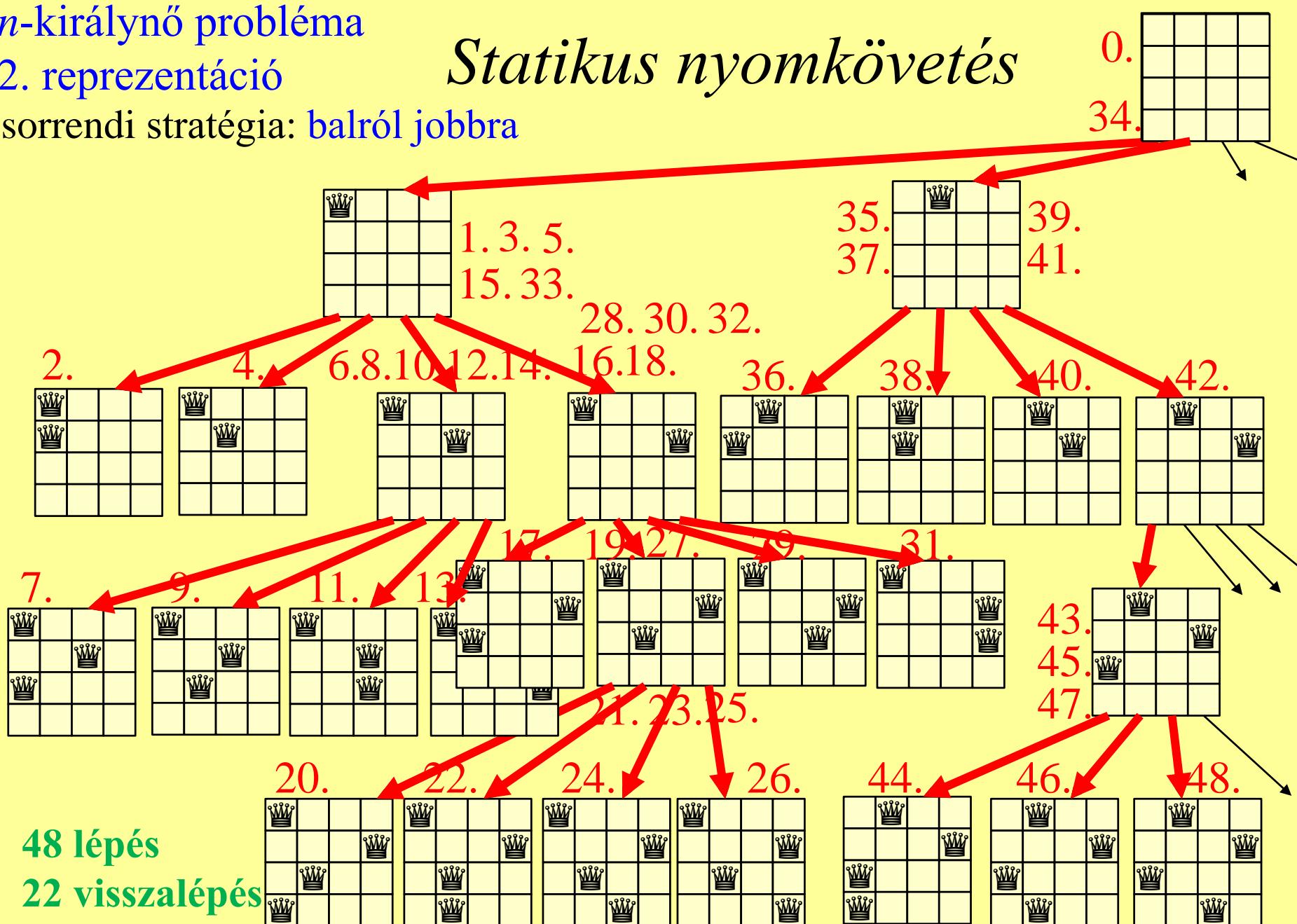
1. **if** $cél(akt)$ **then** **return**(nil) **endif**
2. **for** $\forall új \in \Gamma(akt)$ **loop**
 3. $megoldás := VLI(új)$
 4. **if** $megoldás \neq hiba$ **then**
 5. **return**($fűz((akt, új), megoldás)$) **endif**
 6. **endloop**
 7. **return**($hiba$)
- end**

n-királynő probléma

2. reprezentáció

sorrendi stratégia: balról jobbra

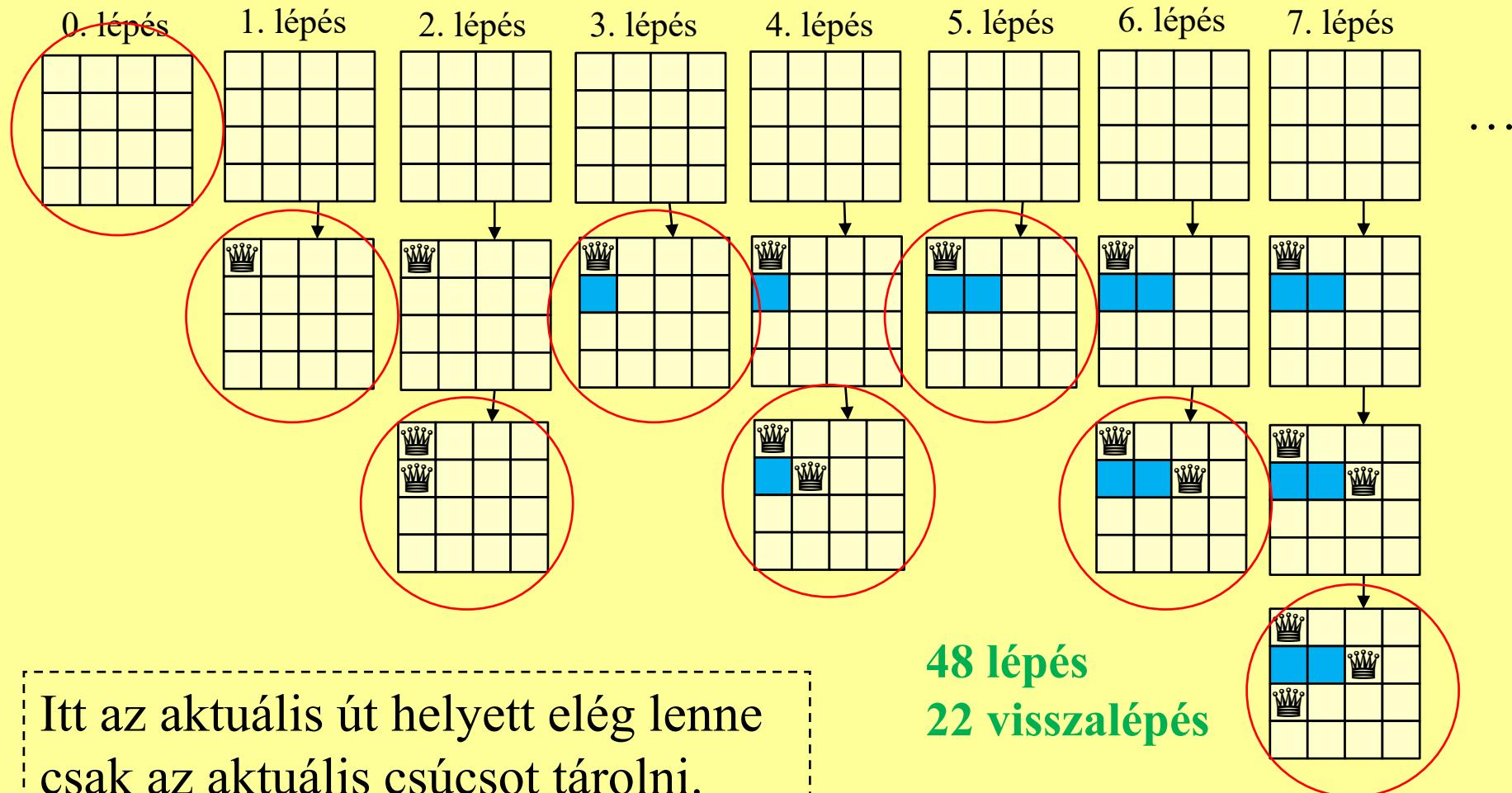
Statikus nyomkövetés



n-királynő probléma

2. reprezentáció

Dinamikus nyomkövetés



Sorrendi heurisztikák az n -királynő problémára

Egy királynő helyét annak sorrendjében próbáljuk majd megtalálni, hogy az adott sor mezői között milyen sorrendet jelölünk ki.

- **Diagonális:** a mezőn áthaladó *hosszabb átló hossza*.
- **Páratlan-páros:** a páratlan sorokban *balról jobbra*, a páros sorokban *jobbról balra* legyen a sorrend.
- **Ütés alá kerülő szabad mezők száma:** új királynő elhelyezésével hány szabad mező kerül ütésbe

4	3	3	4
3	4	4	3
3	4	4	3
4	3	3	4

1	2	3	4
4	3	2	1
1	2	3	4
4	3	2	1

👑	✗	✗	✗
✗	✗	3	2
✗		✗	
✗			✗

Heurisztikák az n -királynő problémára

diagonális + bal-jobb:

4	👑	3	4
	4	4	👑
👑	4	4	3
4		👑	4

8 lépés
2 visszalépés

diagonális + ptl-ps:

→	4/1	👑	3/3	4/4
	3/4	4/3	4/2	👑
→	👑	4/2	4/3	3/4
	4/4	3/3	👑	4/1

4 lépés
0 visszalépés

2. repr.	Nincs	Diag
$n = 4$	22/48	2/8
$n = 5$	10/25	10/25
$n = 6$	165/336	63/132
$n = 7$	35/77	80/167
$n = 8$	868/1744	196/400

$n = 4$	Nincs	Diag	Diag + ptl-ps
2. repr.	22/48	2/8	0/4
3. repr.	4/12	0/4	0/4

n-királynő probléma 3. reprezentáció

VLI
heurisztika nélkül

A k -adik királynő elhelyezése után a hátralevő üres sorokból töröljük az ütésbe került szabad mezőket.

for i=k+1 .. n loop

Töröl(i,k)

Töröl(i,k) : törli az i -dik sor azon szabad mezőit, amelyeket a k -dik királynő üt

$D_i = \{i\text{-dik sor szabad mezői}\}$

crown						
x	x	crown				
x	x	x	x	crown		
x	crown	x	x	x	x	
x	x	x	crown	x	x	
x	x	x	x	x	x	

$k=6$

VLI: if $D_k = \emptyset$ then visszalép.

Forward Checking

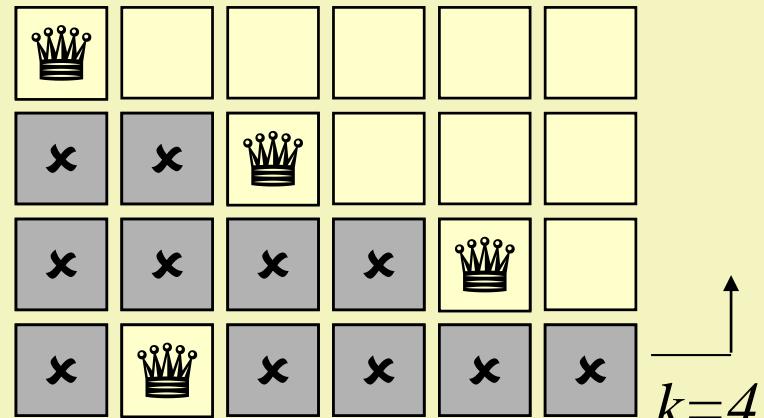
FC algoritmus:

VLI

+

if $\exists i \in [k+1.. n]: D_i = \emptyset$
then visszalép

$D_i = \{i\text{-dik sor szabad mezői}\}$



$$D_6 = \emptyset$$

Partial Look Forward

PLF algoritmus:

VLI

+

for $i=k+1..n$ **loop**

for $j=i+1..n$ **loop** ($i < j$)

Szűr(i,j)

if $\exists i \in [k+1..n]: D_i = \emptyset$

then visszalép

Szűr(i,j) : törli az i -edik sor azon szabad mezőit, amelyekhez nem található a j -edik sorban vele ütésben nem álló szabad mező

crown						
x	x		crown			
x	x	x	x		crown	
x		6	x	x	x	x
x			x		x	x
x	x	x			x	x

$k=3$

$$i = 4, j = 6 \quad D_4 = \emptyset$$

Look Forward

LF algoritmus:

VLI

+

for $i=k+1 \dots n$ **loop**

for $j=k+1 \dots n$ **and** $i \neq j$ **loop**

Szűr(i,j)

if $\exists i \in [k+1..n]: D_i = \emptyset$

then *visszalép*

crown						
x	x	crown				
x	x	x	x			
x			x	x	x	3
x	4	x		x	x	
x	4	x	4	5	x	
x	4	x	4	5	x	

$$i = 4, j = 3 \quad D_6 = \emptyset$$

$$i = 5, j = 4$$

$$i = 6, j = 4$$

$$i = 6, j = 5$$

Look Forward még egyszer

LF algoritmus:

VLI

+

for $i=k+1 \dots n$ **loop**

for $j=k+1 \dots n$ **and** $i \neq j$ **loop**

$Szűr(i,j)$

if $\exists i \in [k+1..n]: D_i = \emptyset$

then visszalép

x	x	x	x	crown		
crown	x	x	x	x	x	
x	x		x	x	x	
x	x	x	x		x	
4	x	4	x	4	5	

$k=3$

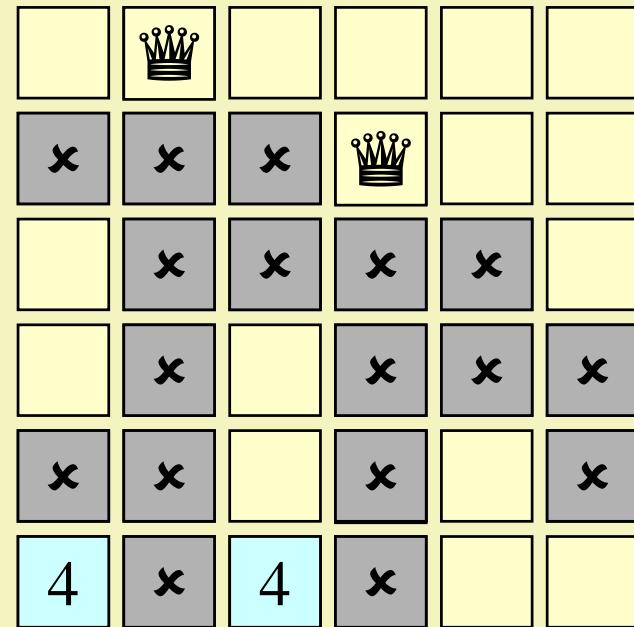
$$i = 6, j = 4 \quad D_6 = \emptyset$$

$$i = 6, j = 5$$

AC1 algoritmus:

VL1

+

repeat **for** $i = k+1 \dots n$ **loop** **for** $j = k+1 \dots n$ **and** $i \neq j$ **loop** $Szűr(i,j)$ **until** volt szűrés**if** $\exists i \in [k+1..n]: D_i = \emptyset$ **then** visszalép**1. menet** $i = 6, j = 4$

AC1 algoritmus:*VLI*

+

repeat **for** $i=k+1..n$ **loop** **for** $j=k+1..n$ **and** $i \neq j$ **loop** $Szűr(i,j)$ **until** volt szűrés**if** $\exists i \in [k+1..n]: D_i = \emptyset$ **then** visszalép

x	x	x	x			
	x	x	x	x	x	
	x		x	x	x	
x	x		x	6	x	
	x		x			

2. menet $i = 5, j = 6$

AC1 algoritmus:*VLI*

+

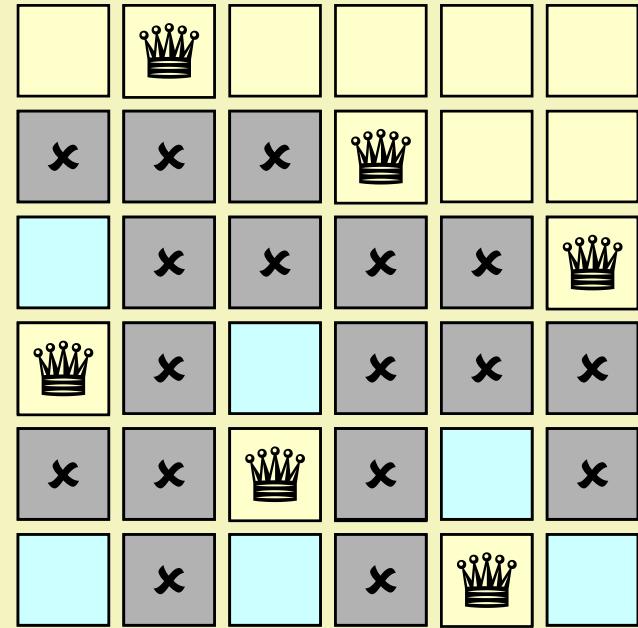
repeat **for** $i=k+1..n$ **loop** **for** $j=k+1..n$ **and** $i \neq j$ **loop** $Szűr(i,j)$ **until** volt szűrés**if** $\exists i \in [k+1..n]: D_i = \emptyset$ **then** visszalép

x	x	x	x			
5	x	x	x	x	x	
	x	5	x	x	x	x
x	x		x		x	
	x		x			3

3. menet $i = 3, j = 5$ $i = 4, j = 5$ $i = 6, j = 3$

AC1 algoritmus:*VL1*

+

repeat **for** $i=k+1..n$ **loop** **for** $j=k+1..n$ **and** $i \neq j$ **loop** $Szűr(i,j)$ **until** volt szűrés**if** $\exists i \in [k+1..n]: D_i = \emptyset$ **then** visszalép**4. menet**

Az n -királynő probléma új reprezentációs modellje

- Az előző módszerek átalakították az n -királynő probléma reprezentációját:
 - Tekintsük a D_1, \dots, D_n halmazokat, ahol $D_i = \{1 \dots n\}$ (ezek az i -dik sor szabad mezői).
 - Keressük azt az $(x_1, \dots, x_n) \in D_1 \times \dots \times D_n$ elhelyezést (x_i az i -dik sorban elhelyezett királynő oszloppozíciója),
 - amely nem tartalmaz ütést: minden i, j királynő párra:
 $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j \wedge |x_i - x_j| \neq |i - j|)$.
- A visszalépéses keresés e modell változóinak értékeit határozza meg, miközben a bemutatott vágó módszerek egyike redukálják ezen változók D_i halmazait.

Bináris korlát-kielégítési modell

- Keressük azt az $(x_1, \dots, x_n) \in D_1 \times \dots \times D_n$ n -est (D_i véges) amely kielégít néhány $C_{ij} \subseteq D_i \times D_j$ bináris korlátot.
- Példák:
 1. Házasságközvetítő probléma (n férfi, m nő; keressünk minden férfinak neki szimpatikus feleségjelöltet):
 - Az i -dik férfi ($i=1..n$) felesége (x_i) a $D_i = \{1, \dots, m\}$ azon elemei, amelyekre fenn áll, hogy $szimpatikus(i, x_i)$.
 - Az összes (i,j) -re: $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j)$ (azaz nincs bigámia)
 2. Gráfszínezési probléma (egy véges egyszerű irányítatlan gráf n darab csúcsát kell kiszínezni m színnel úgy, hogy a szomszédos csúcsok eltérő színűek legyenek):
 - Az i -dik csúcs ($i=1..n$) színe (x_i) a $D_i = \{1, \dots, m\}$ elemei.
 - minden i, j szomszédos csúcs párra: $C_{ij}(x_i, x_j) \equiv (x_i \neq x_j)$.

Modellfüggő vezérlési stratégia

- A korábban mutatott vágó módszereket az új modellben a bináris korlátok definiálják, de a korlátok jelentésétől függetlenül. Ezek a módszerek tehát nem heurisztikák, hanem **modellfüggő vágó stratégiák**:

Töröl(i,k): $D_i := D_i - \{e \in D_i \mid \neg C_{ik}(e, x_k)\}$

Szűr(i,j) : $D_i := D_i - \{e \in D_i \mid \forall f \in D_j : \neg C_{ij}(e, f)\}$

- Modellfüggő sorrendi stratégiák is konstruálhatók:
 - Mindig a legkisebb tartományú még kitöltetlen komponensnek válasszunk előbb értéket.
 - Ugyanazon korláthoz tartozó komponenseket lehetőleg közvetlenül egymás után töltsük ki.

Második változat: VL2

- A visszalépéses algoritmus második változata az, amikor a visszalépés feltételei közül mindenet beépítjük a kereső rendszerbe.
- Bebizonyítható: *A VL2 δ-gráfban mindig terminál. Ha létezik a mélységi korlátnál nem hosszabb megoldás, akkor megtalál egy megoldást.*
UI: véges sok adott korlátnál rövidebb startból induló út van.
- Rekurzív algoritmussal (VL2) adjuk meg
 - Indítás: *megoldás := VL2(<startcsúcs>)*

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

VL2

Recursive procedure VL2($\text{út} : \text{seq}(N)$) **return** ($\text{seq}(A)$; hiba)

1. $\text{akt} := \text{utolsó_csúcs}(\text{út})$
 2. **if** $\text{cél}(\text{akt})$ **then** **return**(nil) **endif**
 3. **if** $\text{hossza}(\text{út}) \geq \text{korlát}$ **then** **return**(hiba) **endif**
 4. **if** $\text{akt} \in \text{maradék}(\text{út})$ **then** **return**(hiba) **endif**
 5. **for** $\forall \text{új} \in \Gamma(\text{akt}) - \pi(\text{akt})$ **loop**
 6. $\text{megoldás} := \text{VL2}(\text{fűz}(\text{út}, \text{új}))$
 7. **if** $\text{megoldás} \neq \text{hiba}$ **then**
 8. **return**($\text{fűz}((\text{akt}, \text{új}), \text{megoldás}))$ **endif**
 9. **endloop**
 10. **return**(hiba)
- end**

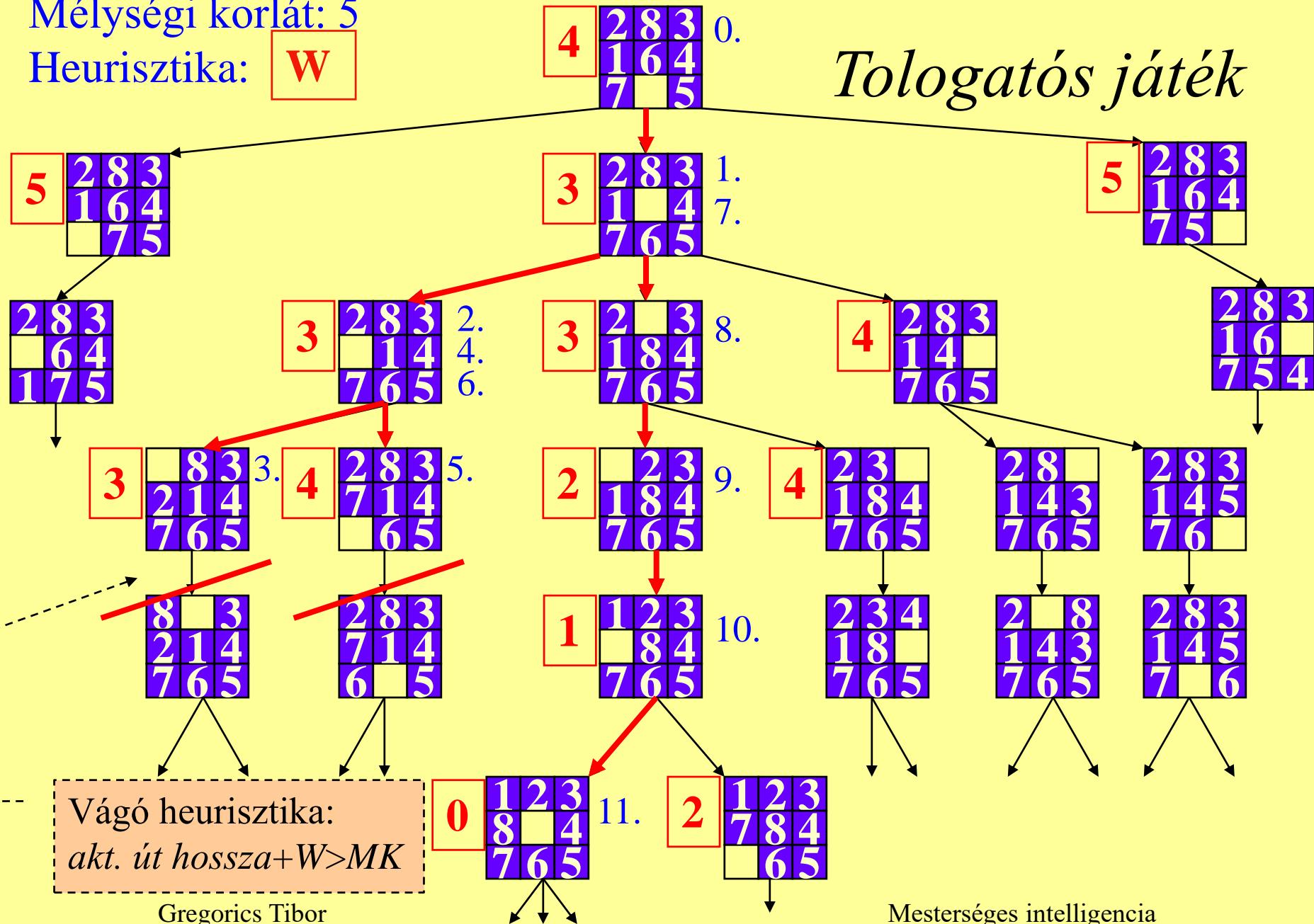
Mélységi korlát szerepe

- A VL2 nem talál megoldást (csak terminál), ha a megadott **mélységi korlátnál csak hosszabb megoldási utak** vannak.
- A **mélységi korlát önmagában** is biztosítja a terminálást körök esetén is.
 - Ez akkor előnyös, ha nincsenek rövid körök (a kettő hosszú köröket kiszűri a szülőcsúcs vizsgálat).
 - Ilyenkor nem kell a rekurzív hívásnál a teljes aktuális utat átadni : elég az út hosszát, az aktuális csúcsot és annak szülőjét.

Mélységi korlát: 5

Heurisztika: W

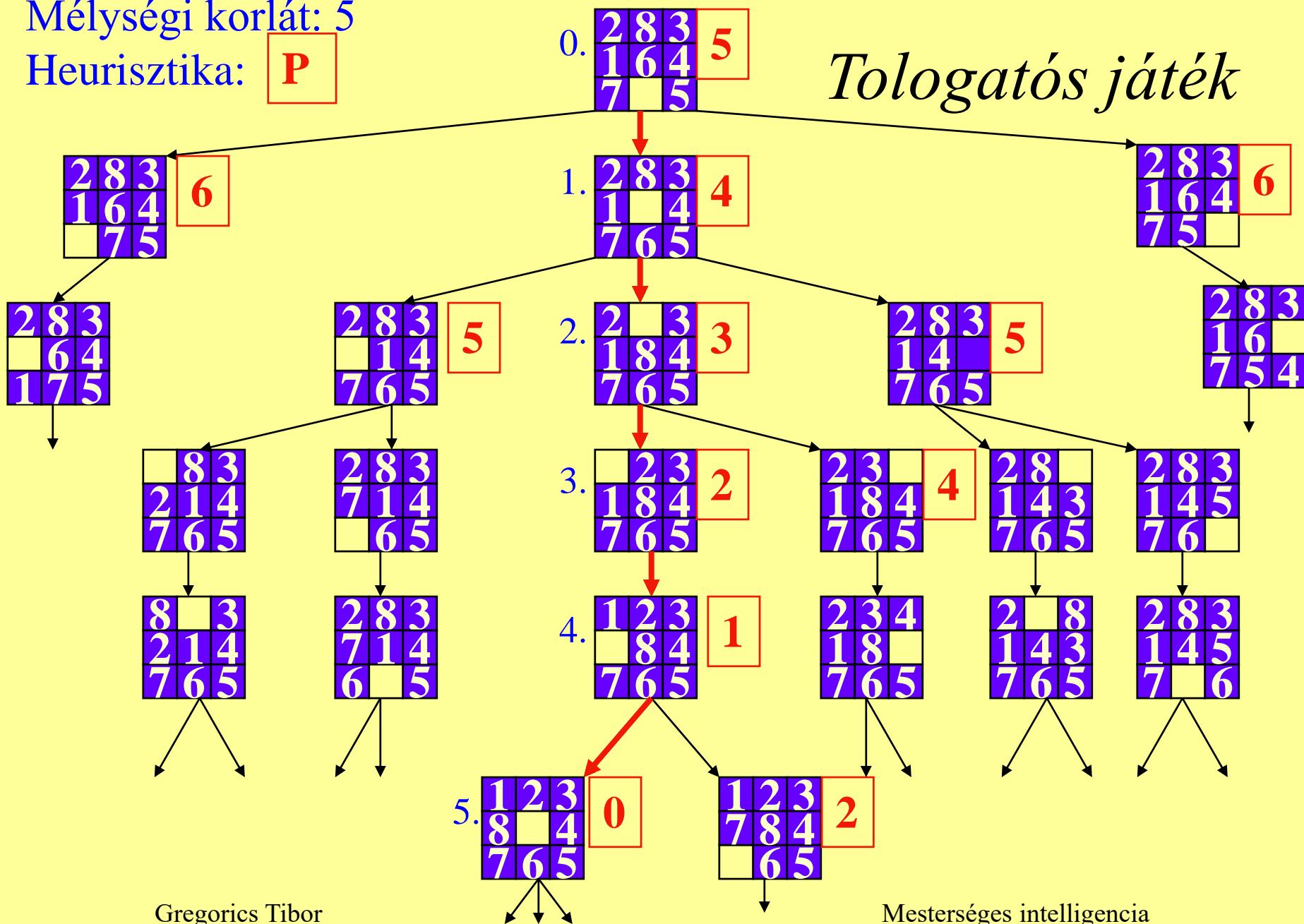
Tologatós játék



Mélységi korlát: 5

Heurisztika: P

Tologatós játék



Értékelés

□ ELŐNYÖK

- minden terminál, talál megoldást (a mélységi korláton belül)
- könnyen implementálható
- kicsi memória igény

□ HÁTRÁNYOK

- nem ad optimális megoldást. (iterációba szervezhető)
- kezdetben hozott rossz döntést csak sok visszalépés korrigál (visszaugrásos keresés)
- egy zsákutca részt többször is bejárhat a keresés

3. Gráfkeresés

- ❑ A gráfkeresés olyan KR, amelynek
 - globális munkaterülete: a reprezentációs gráf startcsúcsból kiinduló már feltárt útjait tárolja (tehát egy részgráfot), és külön az egyes utak végeit, a nyílt csúcsokat
 - kiinduló értéke: a startcsúcs,
 - terminálási feltétel: megjelenik egy célcsúcs vagy megakad az algoritmus.
 - keresés szabálya: egyik útvégi csúcs kiterjesztése
 - vezérlés stratégiája: a legkedvezőbb csúcs kiterjesztésére törekszik

3.1. Általános gráfkereső algoritmus

Jelölések:

- keresőgráf (G) : a reprezentációs gráf eddig
bejárt és eltárolt része
- nyílt csúcsok halmaza ($NYÍLT$) : kiterjesztésre várakozó
csúcsok, amelyeknek gyerekeit még nem vagy nem elégé
jól ismerjük
- kiterjesztett csúcsok halmaza ($ZÁRT$) : azok a csúcsok,
amelyeknek a gyerekeit már előállítottuk
- kiértékelő függvény ($f: NYÍLT \rightarrow \mathbb{R}$) : kiválasztja a
megfelelő nyílt csúcsot kiterjesztésre

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

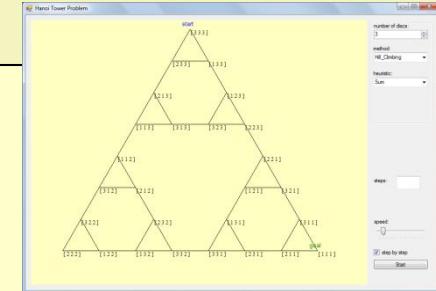
 ADAT := SZ(ADAT)

endloop

Kezdetleges verzió

~~Procedure GKO~~

1. $G := (\{start\}, \emptyset); NYÍLT := \{start\}$
2. **loop**
3. **if** empty($NYÍLT$) **then return** nincs megoldás
4. $n := \min_f(NYÍLT)$
5. **if** cél(n) **then return** van megoldás
6. $NYÍLT := NYÍLT - \{n\} \cup \Gamma(n) - \pi(n)$
7. $G := G \cup \{(n, m) \in A \mid m \in \Gamma(n) - \pi(n)\}$
8. **endloop**
- end**



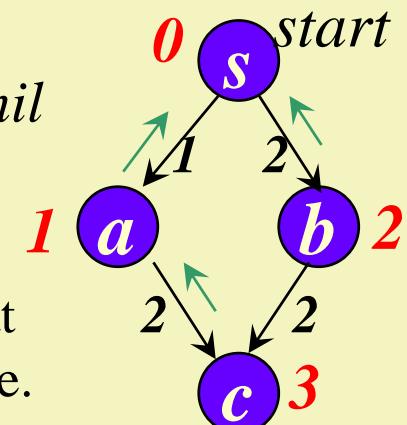
Kritika

- Nem olvasható ki a megoldási út a kereső gráfból
 - Meg kell jegyezni a felfedezett utak nyomát.
- Nem garantál optimális megoldást (sőt még a megoldást sem)
 - Tároljuk el egy csúcsnál az odavezető eddig talált legjobb út költségét.
- Körökre érzékeny
 - Ha ehhez a csúcshoz egy kört tartalmazó utat találunk, akkor annak költsége drágább lesz a tárolt értéknél, hiszen δ -gráfban vagyunk.

Gráfkeresés függvényei

□ $\pi: N \rightarrow N$ szülőre visszamutató pointer

- $\pi(n) = n$ csúcs már ismert szülője, $\pi(start) = nil$
 - π egy *start* gyökerű irányított feszítőfát jelöl ki *G*-ben: π -feszítőfa, π -út
 - Jó lenne ha a π -út optimális $start \rightarrow n$ *G*-beli utat jelölne ki: a π feszítőfa optimális lenne.



□ $g: N \rightarrow \mathbb{R}$ költség függvény

- $g(n) = c^\alpha(start, n)$ – egy már megtalált $\alpha \in \{start \rightarrow n\}$ út költsége
 - Jó lenne ha minden n -re a $g(n)$ a π -út költségét mutatná, azaz π és g konzisztens lenne.

Az n csúcs akkor **korrekt**, ha konzisztens és optimális:

$$g(n) = c^\pi(start, n) \text{ és } c^\pi(start, n) = \min_{\alpha \in \{start \rightarrow n\} \cap G} c^\alpha(start, n)$$

A *G* korrekt, ha minden csúcsa korrekt.

A korrektség fenntartása

- ❑ Kezdetben: $\pi(start) := nil$, $g(start) := 0$
- ❑ Az n csúcs kiterjesztése után minden $m \in \Gamma(n)$ csúcsra
 - 1. Ha m új csúcs

azaz $m \notin G$ akkor

$$\pi(m) := n, \quad g(m) := g(n) + c(n, m)$$

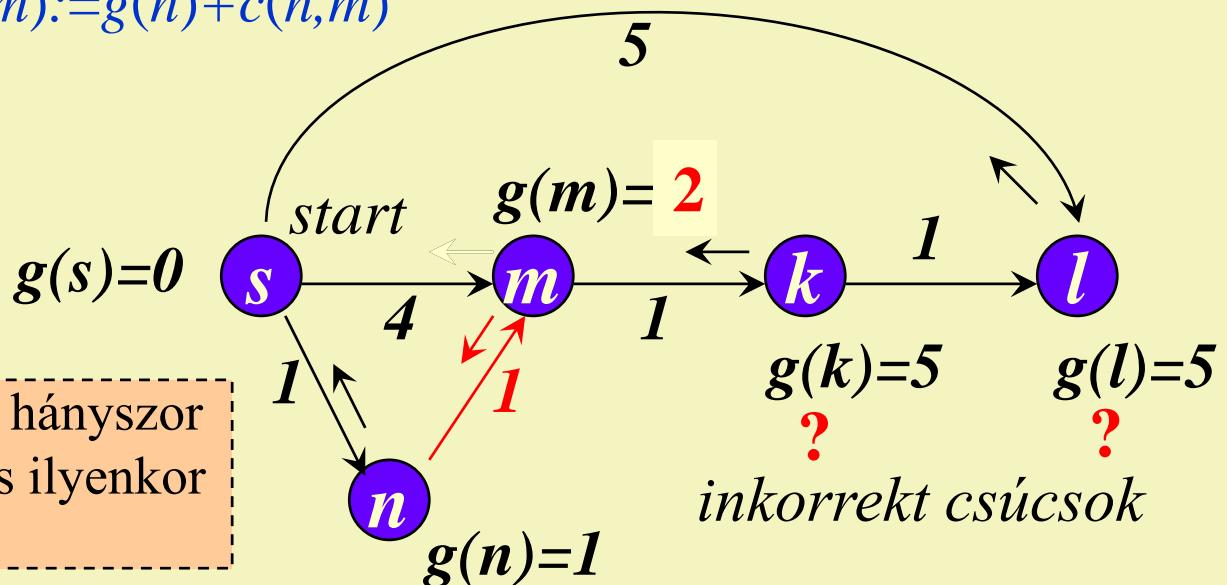
$$NYÍLT := NYÍLT \cup \{m\}$$

- 2. Ha m régi csúcs, amelyhez olcsóbb utat találtunk
azaz $m \in G$ és $g(n) + c(n, m) < g(m)$ akkor
- 3. Ha m régi csúcs, amelyhez nem találtunk olcsóbb utat
azaz $m \in G$ és $g(n) + c(n, m) \geq g(m)$ akkor *SKIP*

$g(n)$ értéke
ekkor csökken

Mégsem marad korrekt a kereső gráf

Ha $m \in G$ és $g(n) + c(n, m) < g(m)$, akkor
 $\pi(m) := n$, $g(m) := g(n) + c(n, m)$



- ❑ Mi legyen az olcsóbb úton újra megtalált m csúcs leszármazottaival?
 1. Járjuk be és javítsuk ki a pointereiket és költségeiket!
 2. Kerüljük el egy jó kiértékelő függvénytel, hogy ilyen történjen!
 3. Semmi más ne tegyünk, csak legyen az m csúcs újra nyílt!

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Általános gráfkereső algoritmus

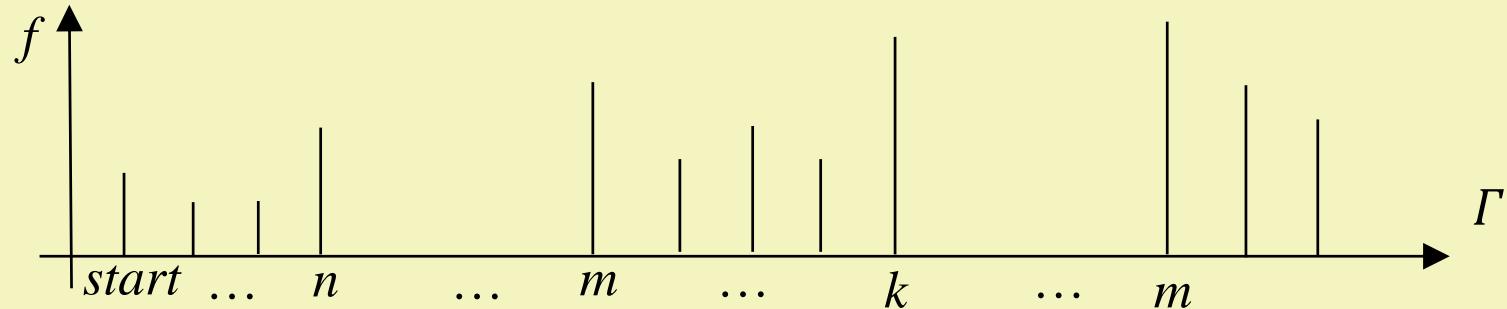
1. $G := (\{start\}, \emptyset); NYÍLT := \{start\}; g(start) := 0; \pi(start) := nil$
2. **loop**
3. **if** $empty(NYÍLT)$ **then return** nincs megoldás
4. $n := min_f(NYÍLT)$
5. **if** $cél(n)$ **then return** megoldás
6. $NYÍLT := NYÍLT - \{n\}$
7. **for** $\forall m \in \Gamma(n) - \pi(n)$ **loop**
8. **if** $m \notin G$ or $g(n) + c(n,m) < g(m)$ **then**
9. $\pi(m) := n; g(m) := g(n) + c(n,m); NYÍLT := NYÍLT \cup \{m\}$
10. **endloop**
11. $G := G \cup \{(n,m) \in A \mid m \in \Gamma(n) - \pi(n)\}$
12. **endloop**

Működés és eredmény

Bebizonyítható:

- A GK δ -gráfban a működése során egy csúcsot legfeljebb véges sokszor terjeszt ki.
(ebből következik például, hogy körökre nem érzékeny)
- A GK véges δ -gráfban mindenkor terminál.
- Ha egy véges δ -gráfban létezik megoldás, akkor a GK megoldás megtalálásával terminál.

Gráfkeresés működési grafikonja

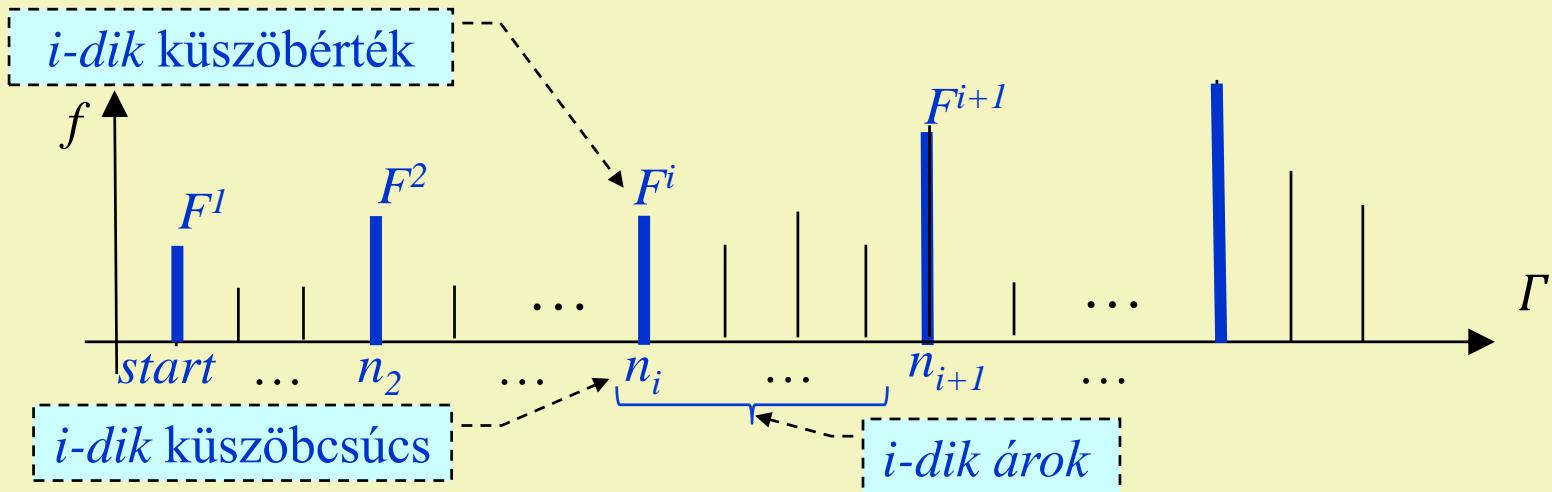


- Soroljuk fel a kiterjesztett csúcsokat kiterjesztésük sorrendjében (ugyanaz a csúcs többször is szerepelhet, hiszen többször is kiterjesztődhet) a kiterjesztésükkel mért f kiértékelő függvényértékükkel.

Csökkenő kiértékelő függvény

- Egy GK kiértékelő függvénye **csökkenő**, amennyiben a egy csúcs kiértékelő függvény értéke az algoritmus működése során nem növekszik, viszont mindenkor mindenkor csökken, valahányszor a korábbinál olcsóbb utat találunk hozzá.
- Csökkenő kiértékelő függvény mellett a GK időről időre automatikusan helyreállítja a kereső gráf **korrektségét**, azaz a π feszítő fájának optimálisságát és konzisztenciáját.

Mikor lesz a kereső gráf korrekt csökkenő kiértékelő függvény mellett?



- Válasszuk ki az értékekből azt az F^i ($i=1,2,\dots$) monoton növekedő részsorozatot, amely a legelső értékkel kezdődik, majd minden a legközelebbi nem kisebb értékkel folytatódik.
- Csökkenő kiértékelő függvény használata mellett a GK
 - kereső gráfja korrekt lesz valahányszor küszöbcsúcsot terjeszt ki
 - soha nem terjeszt ki inkorrekt csúcsot

3.2. Nevezetes gráfkereső algoritmusok

- Most az f kiértékelő függvény megválasztása következik.

Nem-informált

- mélységi (MGK)
- szélességi (SZGK)
- egyenletes (EGK)

- Az úgynevezett tie-breaking rule-ok (egyenlőséget feloldó szabályok) a nem-informált gráfkeresésekben is tartalmazhatnak heurisztikát.

Heurisztikus

- előre tekintő (mohó, best-first)
- A, A*, A^c
- A**, B

Nevezetes nem-informált algoritmusok

Kapcsolat a visszalépéses kereséssel

Algoritmus	Definíció	Eredmények
Mélységi gráfkeresés (MGK)	$f = -g$, $c(n,m) = 1$	végtelen gráfokban csak mélységi korláttal garantál megoldást
Szélességi gráfkeresés (SZGK)	$f = g$, $c(n,m) = 1$	optimális (legrövidebb) megoldást adja, ha van (még végtelen δ -gráfokban is) egy csúcsot legfeljebb egyszer terjeszti ki
Egyenletes gráfkeresés (EGK)	$f = g$	optimális (legolcsóbb) megoldást adja, ha van (még végtelen δ -gráfokban is) egy csúcsot legfeljebb egyszer terjeszti ki

Heurisztika a gráfkeresésekben

- Heurisztikus függvénynek nevezzük azt a $h:N \rightarrow \mathbb{R}$ függvényt, amelyik egy csúcsnál megbecsüli a csúcsból a célba vezető („hátralévő”) optimális út költségét.
- $h(n) \approx \min_{t \in T} c^*(n, t) = c^*(n, T) \Rightarrow h^*(n)$
 $(h^*:N \rightarrow \mathbb{R})$
- Ez egy az eddiginél szigorúbb definíciója a heurisztikának.
- Példák:
 - 8-kirakó : W, P
 - 0 (zéró függvény)?

hátralevő optimális költség

Heurisztikus függvények tulajdonságai

□ Nevezetes tulajdonságok:

- **Nem-negatív:** $h(n) \geq 0 \quad \forall n \in N$
- **Megengedhető** (admissible): $h(n) \leq h^*(n) \quad \forall n \in N$
- **Monoton megszorítás:** $h(n) - h(m) \leq c(n,m) \quad \forall (n,m) \in A$
(következetes)

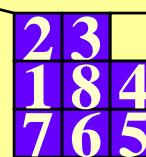
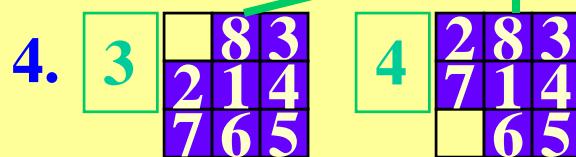
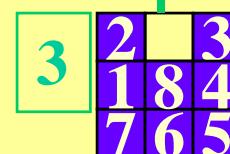
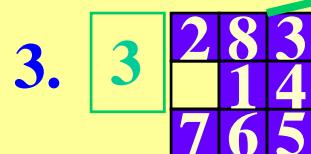
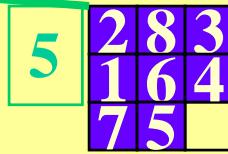
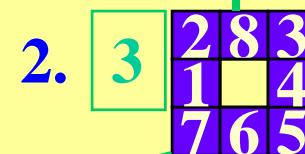
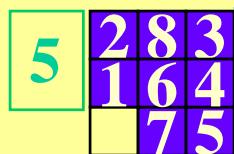
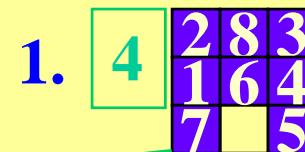
□ Megjegyzés:

- 8-kirakó : W és P minden tulajdonsággal bír.
- h monoton + h célban nulla $\Rightarrow h$ megengedhető
- Zéró függvény minden tulajdonsággal bír.

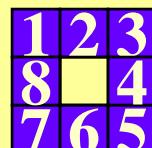
Nevezetes heurisztikus algoritmusok

Algoritmus	Definíció	Eredmények
<i>Előre tekintő gráfkeresés</i>	$f = h$	nincs említető extra tulajdonsága
<i>A algoritmus</i>	$f = g + h$ és $h \geq 0$	<ul style="list-style-type: none"> megoldást ad, ha van megoldás (még végtelen δ-gráfban is)
<i>A* algoritmus</i>	$f = g + h$ és $h \geq 0$ és $h \leq h^*$	<ul style="list-style-type: none"> optimális megoldást ad, ha van (még végtelen δ-gráfban is)
<i>A^c algoritmus</i>	$f = g + h$ és $h \geq 0$ és $h \leq h^*$ és $h(n) - h(m) \leq c(n, m)$	<ul style="list-style-type: none"> optimális megoldást ad, ha van (még végtelen δ-gráfban is) egy csúcsot legfeljebb egyszer terjeszt ki
		egyenletes gráfkeresés: $f = g + 0$

W



még 6 lépés



g+W

6	2	8	3
1	6	4	
7	5		

1. 4 | 2 8 3
 | 1 6 4
 | 7 5

2. 4 | 2 8 3
 | 1 4
 | 7 6 5

6 | 2 8 3
 | 1 6 4
 | 7 5

3. 5 | 2 8 3
 | 1 4
 | 7 6 5

4. 5 | 2 3
 | 1 8 4
 | 7 6 5

6 | 2 8 3
 | 1 4
 | 7 6 5

6 | 2 8 3
 | 2 1 4
 | 7 6 5

7 | 2 8 3
 | 7 1 4
 | 6 5

5. 5 | 2 3
 | 1 8 4
 | 7 6 5

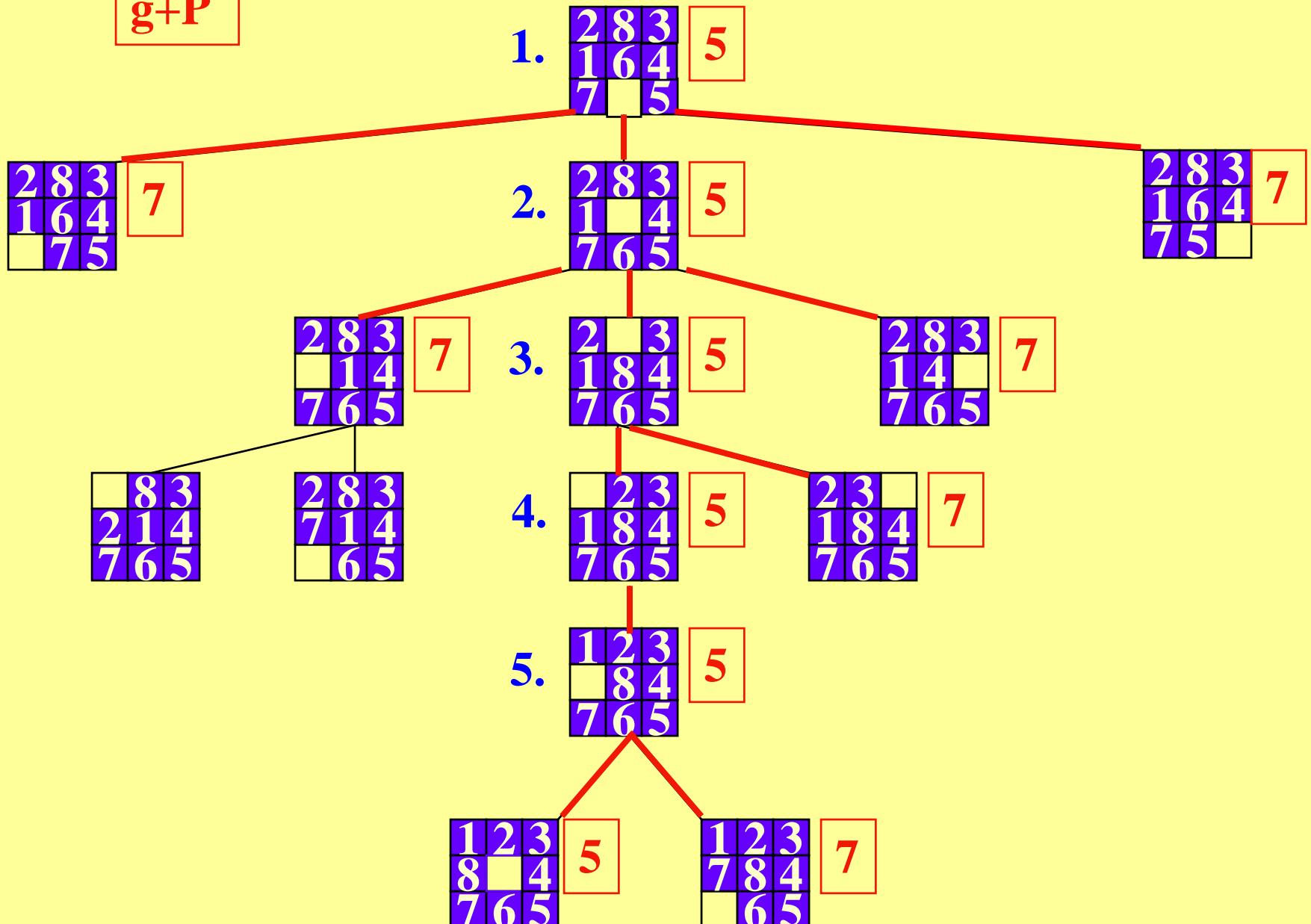
7 | 2 3
 | 1 8 4
 | 7 6 5

6. 5 | 1 2 3
 | 8 4
 | 7 6 5

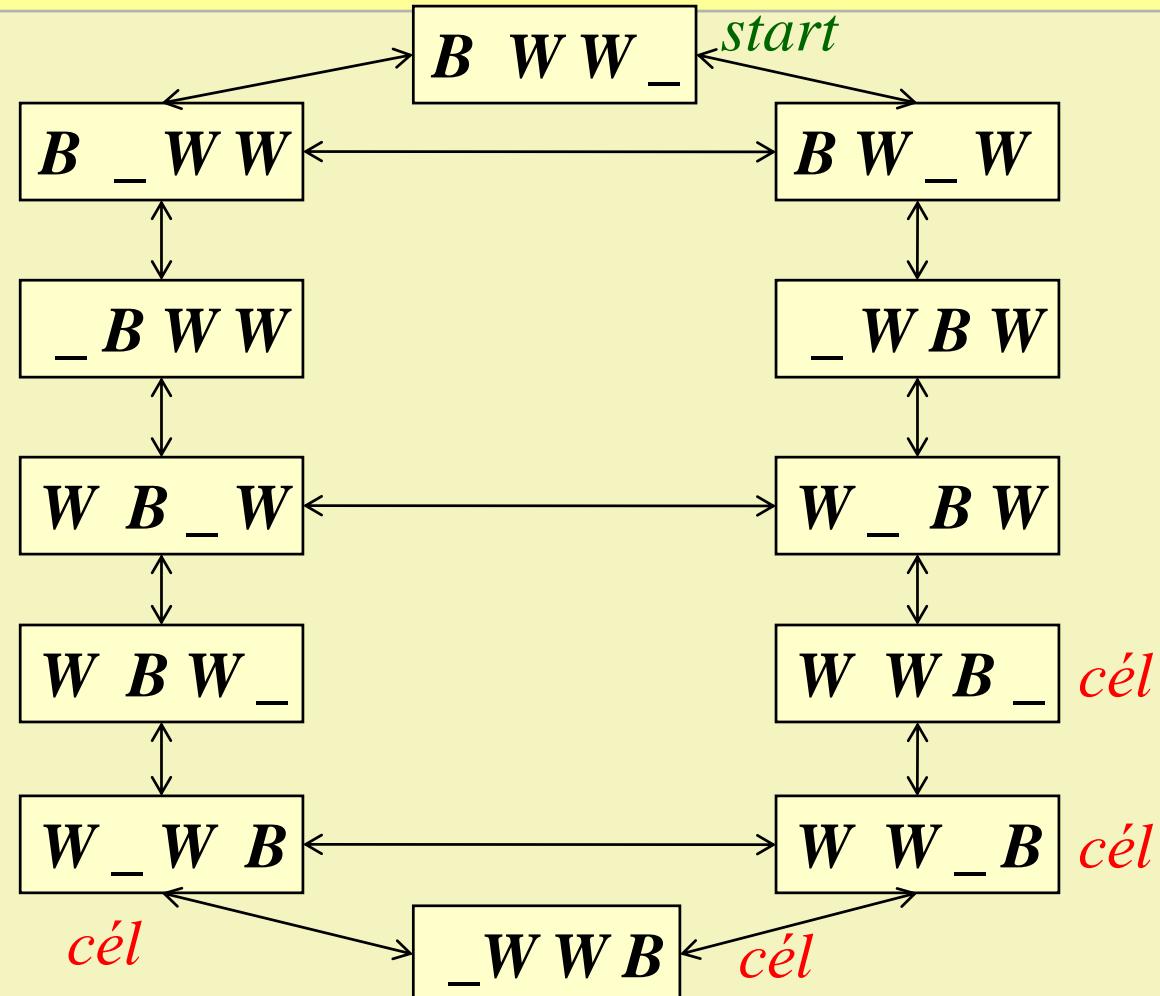
5 | 1 2 3
 | 8 4
 | 7 6 5

7 | 1 2 3
 | 7 8 4
 | 6 5

g+P

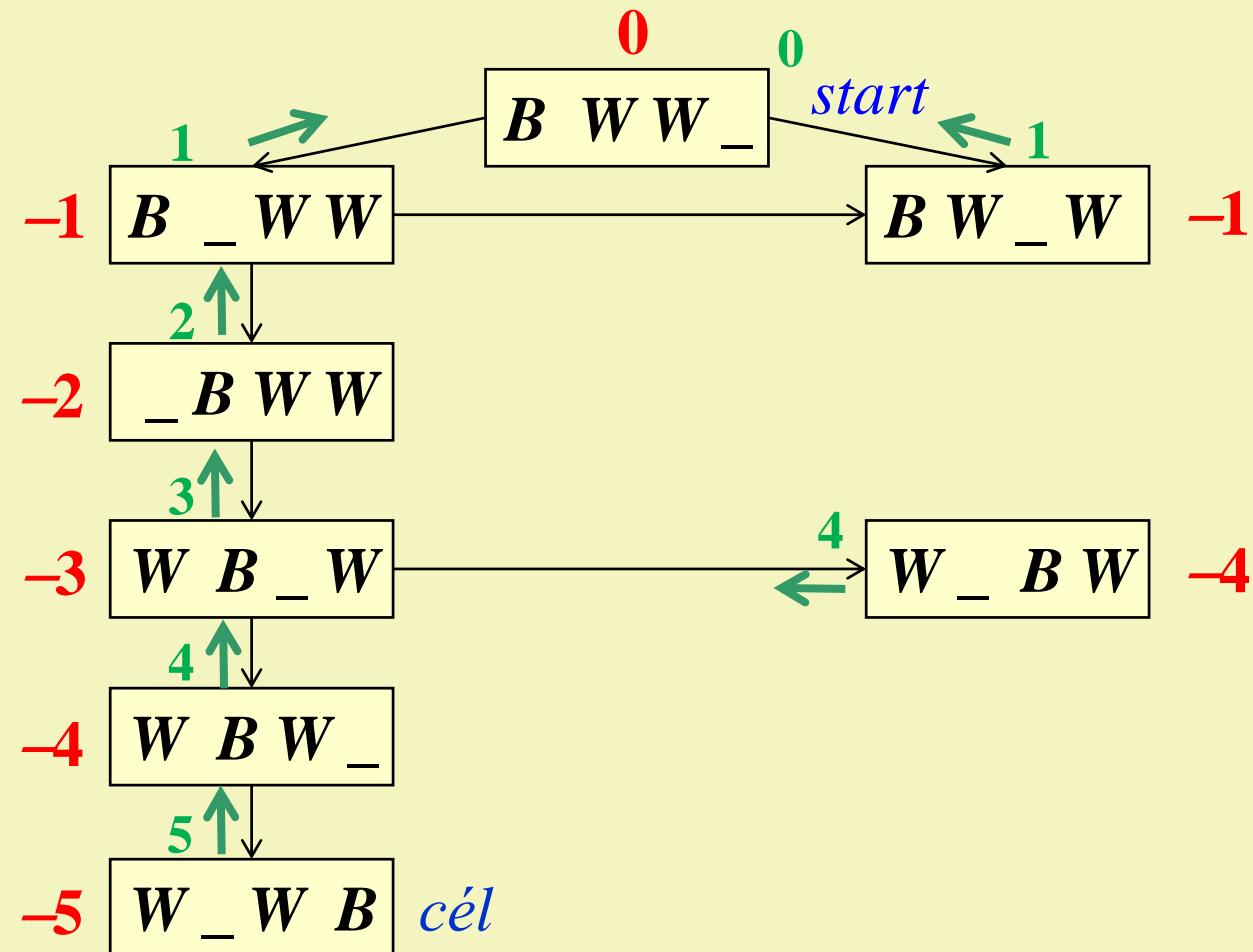


Fekete-fehér kirakó állapot gráfja



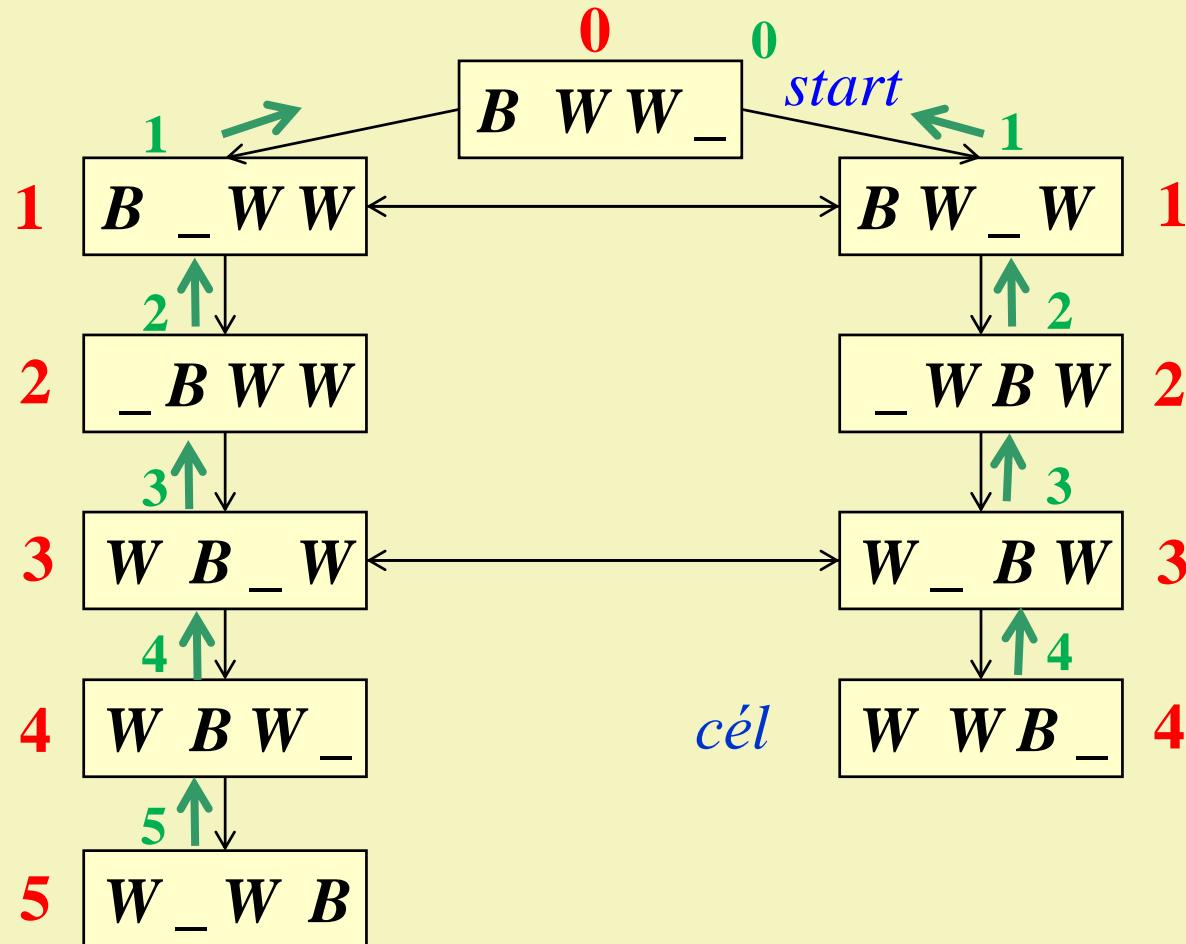
Mélységi gráfkeresés

$$f = -g$$



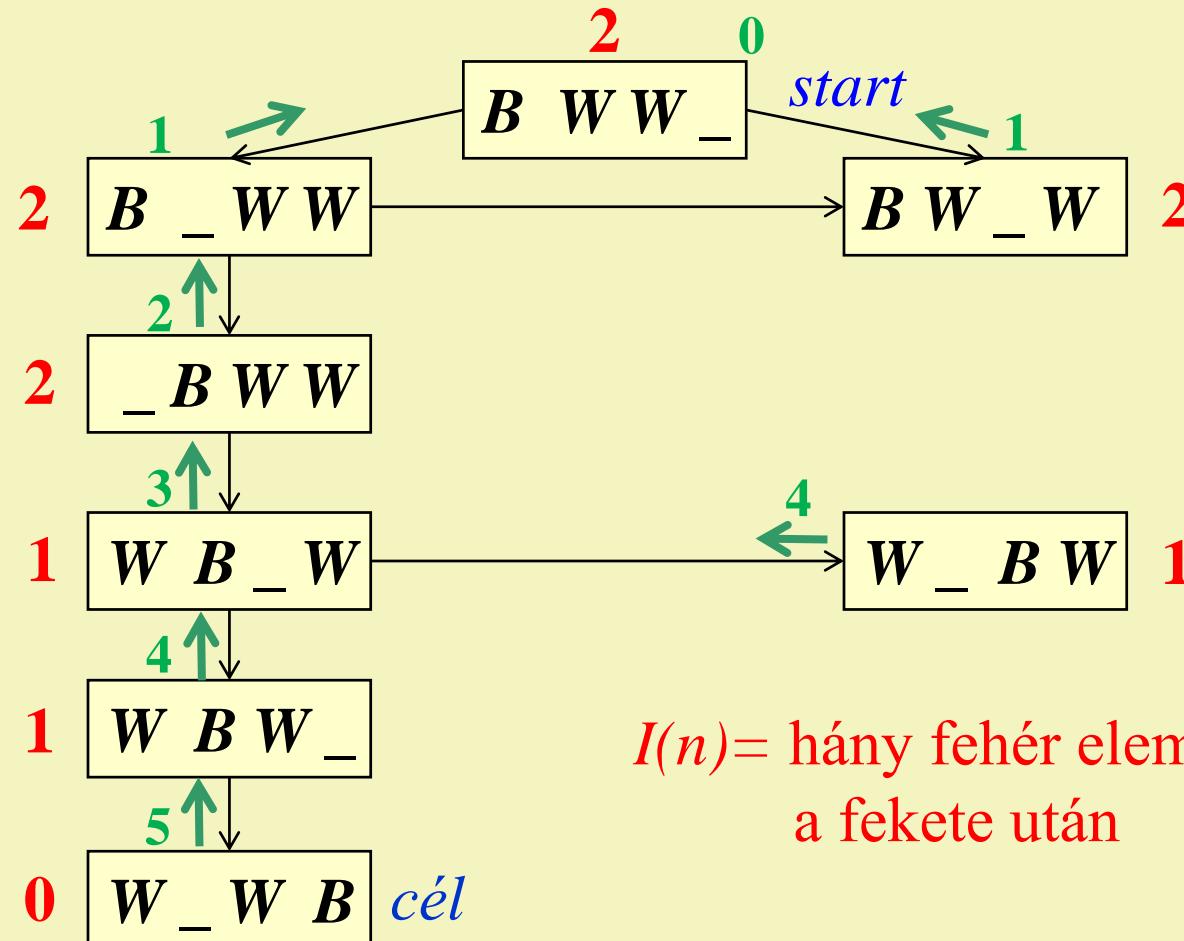
Szélességi gráfkeresés

$$f = g$$



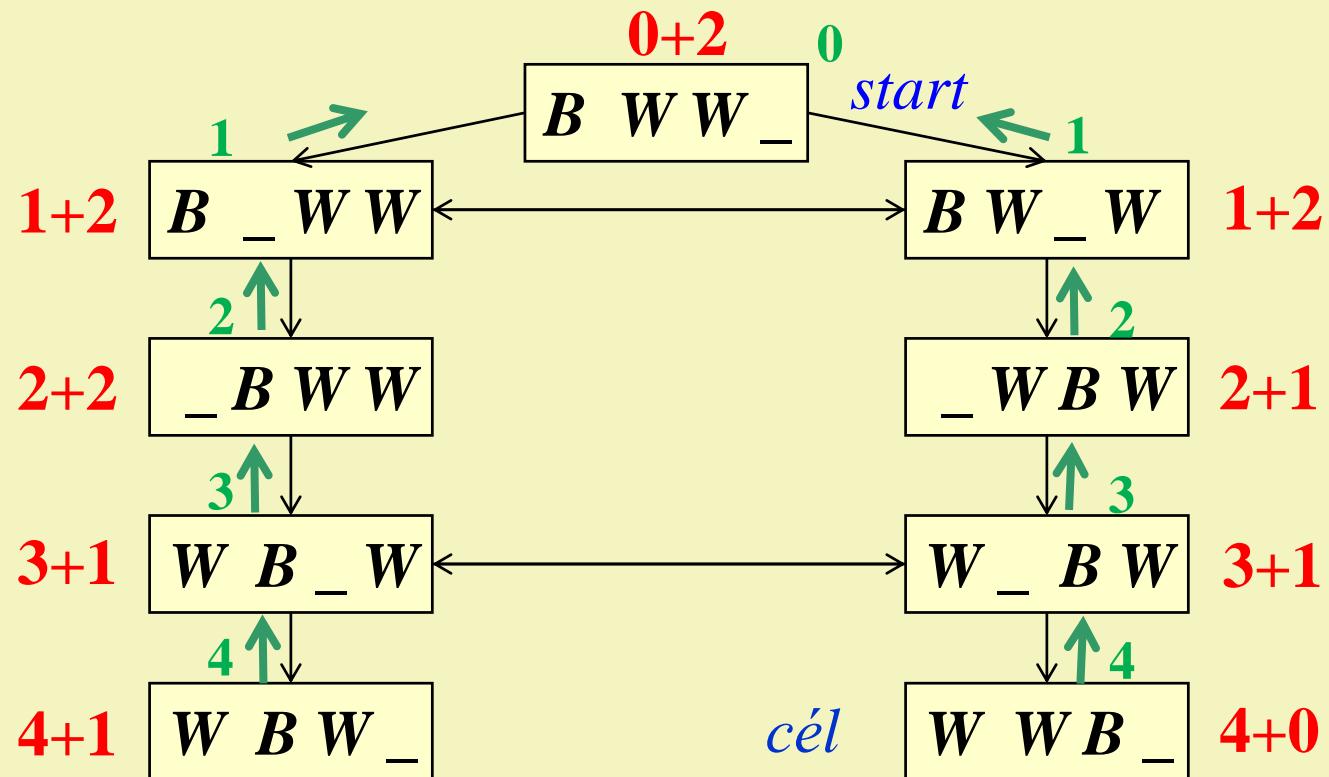
Előre tekintő gráfkeresés

$f = I$



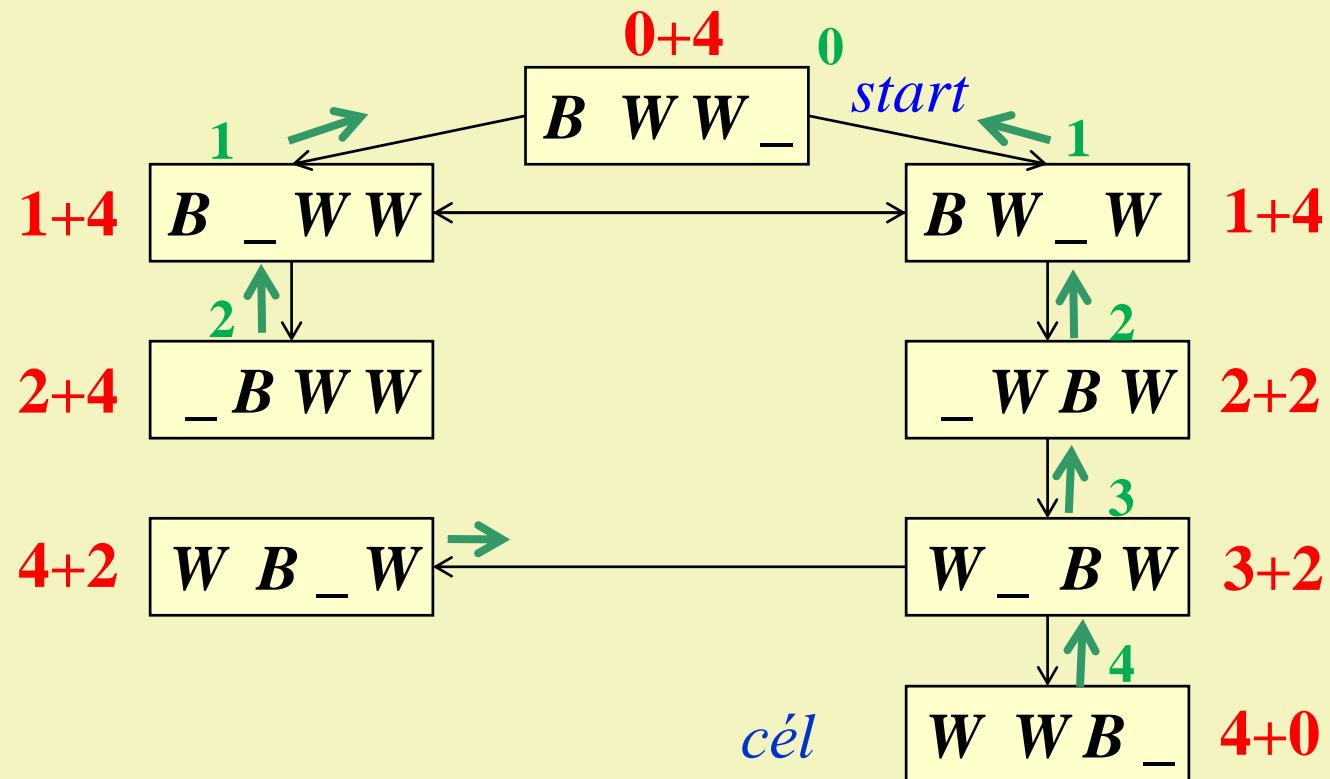
A algoritmus

$$f = g + I$$



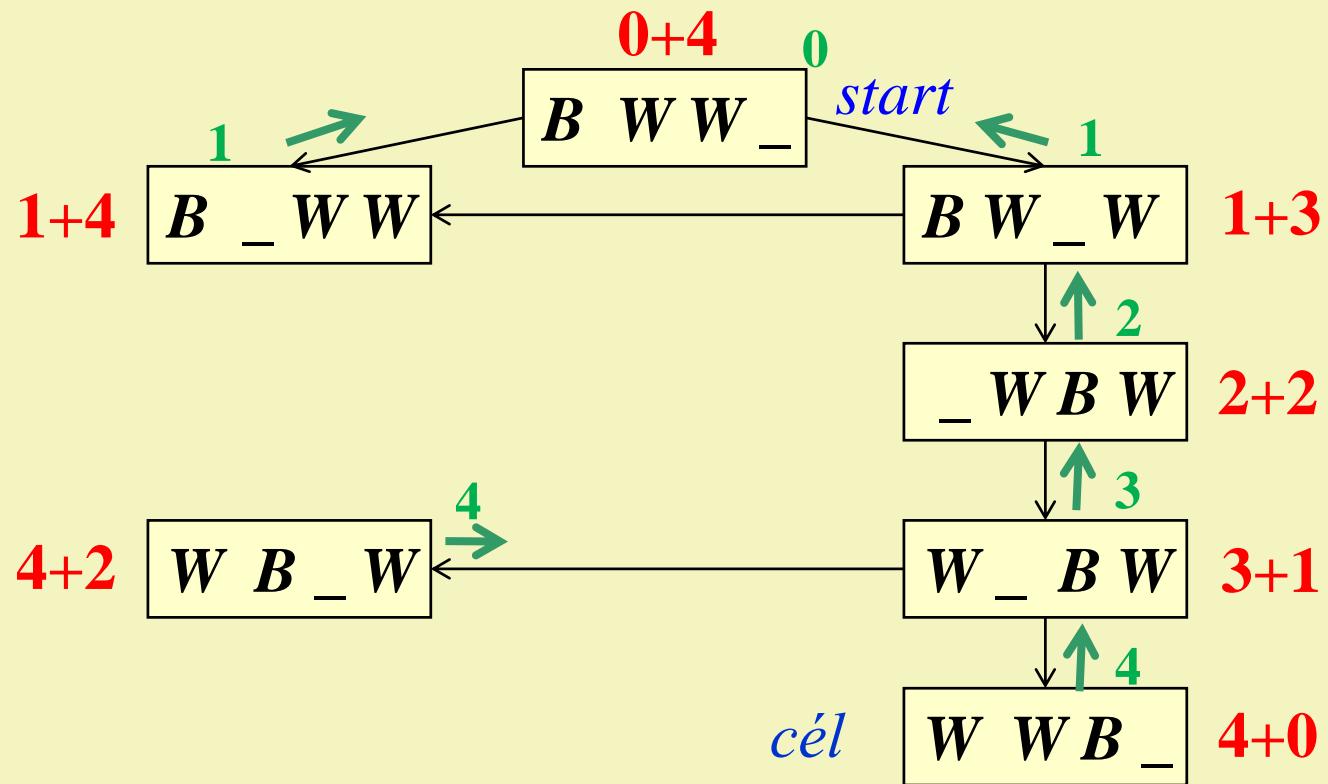
A algoritmus

$$f = g + 2*I$$



A algoritmus

$$f = g + 2*I - (1 \text{ ha van } BW_- \text{ vagy } _BW)$$



Elemzés

f	Alg	mo	G	Γ
$-g$	MGK	5	8	5
g	$SZGK$	4	10	8
l	<i>Előre tekintő</i>	5	8	5
$g+l$	$A \ alg$	4	9	7
$g+2*l$	$A \ alg$	4	8	6
$g+2*l-1(ha...)$	$A \ alg$	4	7	5

$A^c \ alg$

$A^c \ alg$

3.3. *A^{*} algoritmus* hatékonysága

Hatókonyság

Memória igény

Zárt csúcsok száma termináláskor jól jellemzi a kereső gráf méretét

Futási idő

Kiterjesztések száma a zárt csúcsok számához viszonyítva

A hatékonyságot a **megengedhető feladatokon** vizsgáljuk, amelyeknek van megoldása és ismert egy megengedhető heurisztikája, tehát az *A^{*} algoritmus* optimális megoldást talál hozzájuk.

3.3.1. A memória igény vizsgálata

- $ZART_S$ ~ az S gráfkereső algoritmus által lezárt (kiterjesztett) csúcsok halmaza
- Rögzítsünk egy feladatot és két, X és Y gráfkereső algoritmust
Az adott feladatra nézve
 - a. az X nem rosszabb az Y -nál, ha $ZART_X \subseteq ZART_Y$
 - b. az X jobb az Y -nál, ha $ZART_X \subset ZART_Y$
- Ezek alapján összevethető
 1. két eltérő heurisztikájú A^* algoritmus ugyanazon a feladaton, azaz a két heurisztika.
 2. két útkereső algoritmus, például az A^* algoritmus és egy másik – szintén optimális megoldást garantáló – gráfkereső algoritmus a megengedhető problémák egy részhalmazán.

Különböző heurisztikájú A^* algoritmusok memória igényének összehasonlítása

- Az A_1 (h_1 heurisztikával) és A_2 (h_2 heurisztikával) A^* algoritmusok közül az A_2 jobban informált, mint az A_1 , ha minden $n \in N \setminus T$ csúcsra teljesül, hogy $h_1(n) < h_2(n)$.

$$h_1(n) < h_2(n) \leq h^*(n)$$


- Bebizonyítható, hogy a jobban informált A_2 nem rosszabb a kevésbé informált A_1 -nél, azaz $ZART_{A_2} \subseteq ZART_{A_1}$

Megjegyzés

- A gyakorlatban a bizonyított állításnál enyhébb feltételek mellett látványosabb különbségekkel is találkozhatunk:
 - Sokszor akkor is jóval több csúcsot terjeszt ki az A_1 , mint A_2 ($ZART_{A_2} \subset ZART_{A_1}$), ha csak a $h_1 \leq h_2$ teljesül, esetleg nem is minden csúcsra.
 - Példák:
 - 8-as tologató: $0 \leq W \leq P$ ($\leq F$)
 - Fekete-fehér: $I \leq M$ ($\leq 2 \cdot I$)
- Minél jobban (közelebbről) becsli (ha lehet, alulról) a heurisztika a h^* -ot, várhatóan annál kisebb lesz a memória igénye.

15-kirakó

$f =$	$g+0$	$g+W$	$g+P$
6 lépések megoldás	117	7	6
13 lépések megoldás	32389	119	13
21 lépések megoldás	n.a.	3343	145
30 lépések megoldás	n.a.	n.a.	1137
34 lépések megoldás	n.a.	n.a.	3971

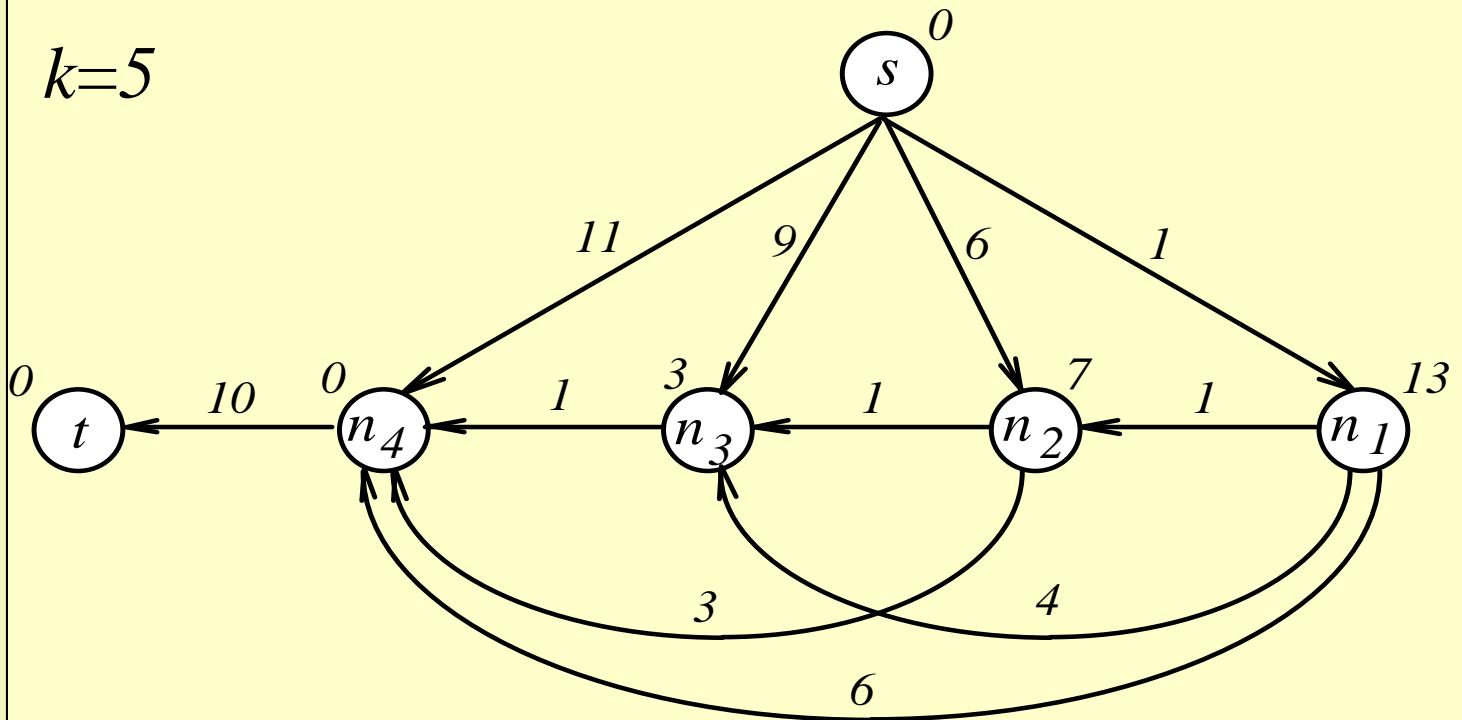
3.3.2. A futási idő elemzése

- Zárt csúcsok száma: $k = |\text{ZÁRT}|$
- Alsókorlát: k
 - Egy monoton megszorításos heurisztika mellett egy csúcs legfeljebb csak egyszer terjesztődik ki,
 - habár ettől még a kiterjesztett csúcsok száma igen sok is lehet (lásd egyenletes keresés)
- Felsőkorlát: 2^{k-1}
 - lásd. Martelli példáját

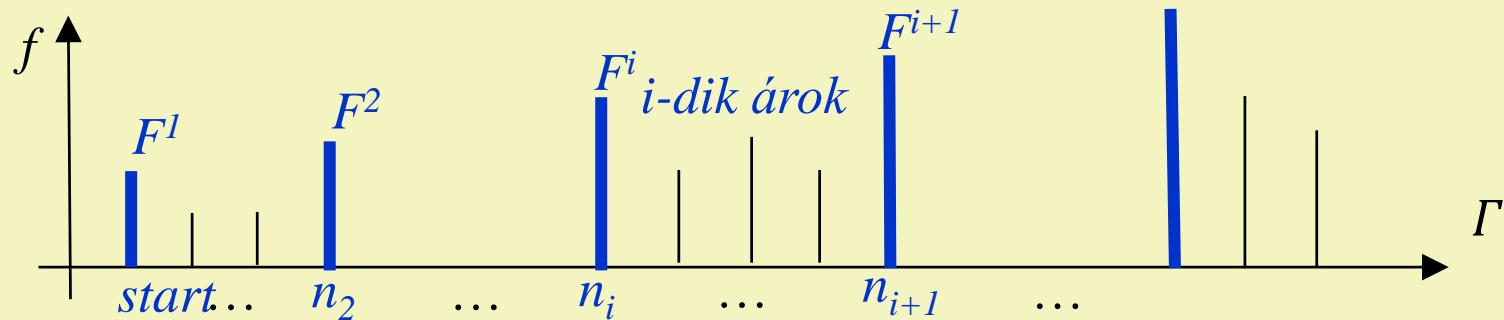
Megjegyzés

- ❑ Másik heurisztikával ugyanazon a feladaton természetesen javítható a kiterjesztések száma, bár nem biztos, hogy ez minden esetben tényleges javulás lesz, hiszen másik heurisztika esetén a k értéke is változhat.
- ❑ A kiterjesztések száma ugyanis a kiterjesztett (zárt) csúcsok számához viszonyított szám
 - h_1 heurisztika mellett k_1 darab zárt csúcs, és 2^{k_1-1} kiterjesztés
 - h_2 heurisztika mellett k_2 darab zárt csúcs, és k_2 kiterjesztés
 - Mégis lehet, hogy $2^{k_1-1} < k_2$, ha $k_1 \ll k_2$.

Martelli példája



A probléma oka és csillapítása



- ❑ Egy csúcs – még akár egy árkon belül is – többször kiterjesztődhet.
- ❑ Használunk az árkokban egy másik, egy **másodlagos (belso)** kiértékelő függvényt! Bizonyítható, hogy ettől nem változik meg az **egy árokban kiterjesztett csúcsok halmaza**, csak **a csúcsok árkon belüli kiterjesztési sorrendje** lesz más, ennél fogva pedig a küszöbcsúcsok, azok sorrendje és értékei változatlanok maradnak. Ennél a belső kiértékelő függvény csak a futási időt (kiterjesztések számát) befolyásolja.

B algoritmus

- ❑ Martelli javasolta belső kiértékelő függvénynek a g költség függvényt.
- ❑ A *B algoritmust* az *A algoritmusból* kapjuk úgy, hogy bevezetjük az F aktuális küszöbértéket, majd
 - az 1. lépést kiegészítjük az $F := f(s)$ értékadással,
 - a 4. lépést pedig helyettesítjük az
if $\min_f(NYÍLT) < F$
then $n := \min_g(m \in NYÍLT \mid f(m) < F)$
else $n := \min_f(NYÍLT); F := f(n)$
endif elágazással.

B algoritmus futási ideje

- A *B algoritmus* ugyanúgy működik, mint az A^* , azzal a kivétellel, hogy egy árokhoz tartozó csúcsot csak egyszer terjeszt ki.
- *Futási idő elemzése:*
 - Legrosszabb esetben
 - minden zárt csúcs először küszöbcsúcsként terjesztődik ki. (Csökkenő kiértékelő függvény mellett egy csúcs csak egyszer, a legelső kiterjesztéskor lehet küszöb.)
 - Az i -dik árok legfeljebb az összes addigi $i-1$ darab küszöbcsúcsot tartalmazhatja (a start csúcs nélkül).
 - Így az összes kiterjeszték száma legfeljebb $\frac{1}{2} \cdot k^2$

Heurisztika szerepe

□ Milyen a jó heurisztika?

- megengedhető: $h(n) \leq h^*(n)$
 - Bár nincs mindenkor szükség optimális megoldásra.
- jól informált: $h(n) \sim h^*(n)$
- monoton megszorítás: $h(n) - h(m) \leq c(n,m)$
 - Ilyenkor nem érdemes *B algoritmust* használni

□ Változó heurisztikák:

- $f = g + \phi \cdot h$ ahol $\phi \sim d$
- B' algoritmus

B' algoritmus

```
if  $h(n) < \min_{m \in \Gamma(n)} (c(n,m) + h(m))$   
then  $h(n) := \min_{m \in \Gamma(n)} (c(n,m) + h(m))$   
else for  $\forall m \in \Gamma(n)$ -re loop  
    if  $h(n) - h(m) > c(n,m)$  then  $h(m) := h(n) - c(n,m)$   
endloop
```

- A h megengedhető marad
- A h nem csökken
- A mononton megszorításos élek száma nő

Mohó A algoritmus

- ❑ Nincs mindenkor szükség az optimális megoldásra.
 - Ilyenkor a mohó A^* algoritmus is használható, amely rögtön megáll, ha célcsúcs jelenik meg a $NYÍLT$ -ban.
- ❑ A mohó A^* algoritmus csak a megoldás megtalálását garantálja. De belátható
 - Ha h megengedhető és $\forall t \in T: \forall (n, t) \in A: h(n) + \alpha \geq c(n, t)$, akkor a talált megoldás költsége: $g(t) \leq h^*(s) + \alpha$
- ❑ A mohó A^* algoritmus megengedhető heurisztika mellett akkor garantálja az optimális megoldást is,
 - ha $\forall t \in T: \forall (n, t) \in A: h(n) = c(n, t)$ vagy
 - ha h monoton és $\exists \alpha \geq 0: \forall t \in T: \forall (n, t) \in A: h(n) + \alpha = c(n, t)$

V. Kétszemélyes játékok

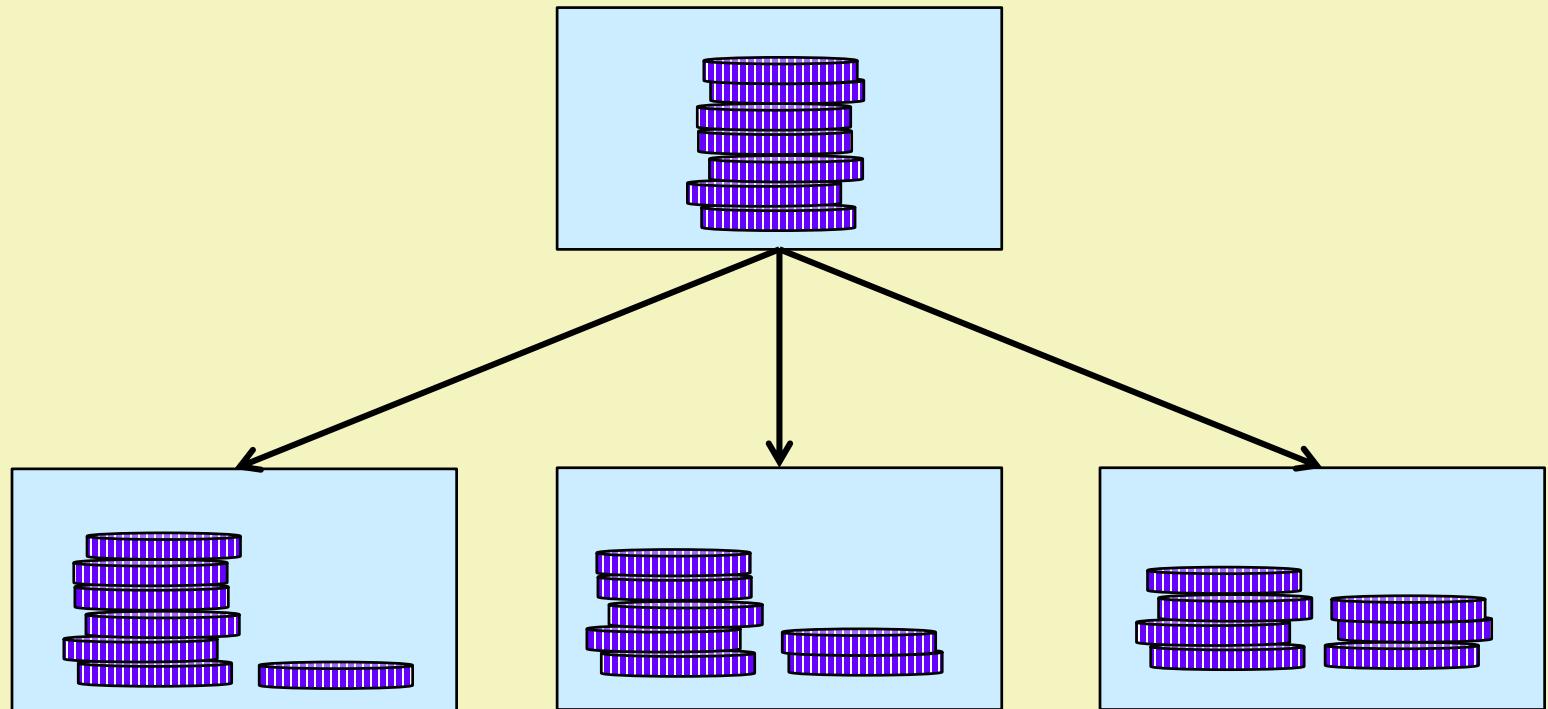
Kétszemélyes, teljes információjú, véges, determinisztikus, zéró összegű játékok

- ❑ Két játékos lép felváltva adott szabályok szerint, amíg a játszma véget nem ér.
- ❑ Mindkét játékos ismeri a maga és az ellenfele összes múltbeli és jövőbeli lépéseiit és lépési lehetőségeit, és azok következményeit.
- ❑ minden lépés véges számú lehetőség közül választható, és minden játszma véges lépésben véget ér. Egy lépés determinisztikus, a véletlennek nincs szerepe.
- ❑ Amennyit a játszma végén az egyik játékos nyer, annyit veszít a másik. (Legegyszerűbb változatban két esélyes: egyik nyer, másik veszít; vagy három esélyes: döntetlen is megengedett)

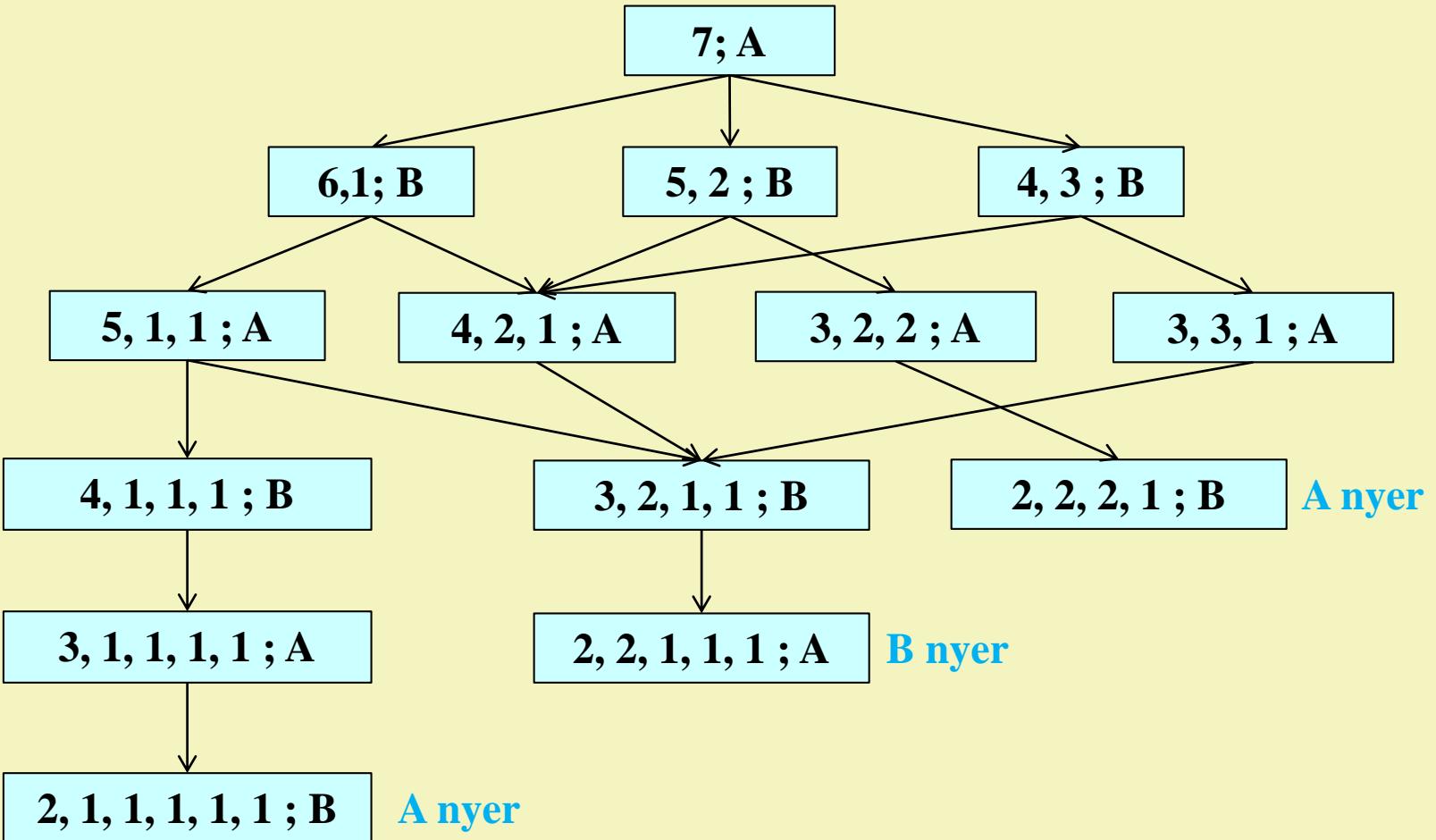
Állapottér-reprezentáció

- állapot – állás + soron következő játékos
- művelet – lépés
- kezdő állapot – kezdőállás + kezdő játékos
- végállapot – végállás + játékos
- + payoff függvény: $p_A, p_B : \text{végállapot} \rightarrow \mathbb{R}$ (játékosok: A, B)
 - Zéró összegű kétszemélyes játékban:
$$p_A(t) + p_B(t) = 0 \quad \text{ minden } t \text{ végállapotra}$$
 - Speciális esetben:
 - $p_A(t) = +1$ ha A nyer
 - $p_A(t) = -1$ ha A veszít
 - $p_A(t) = 0$ ha döntetlen

Grundy mama játéka



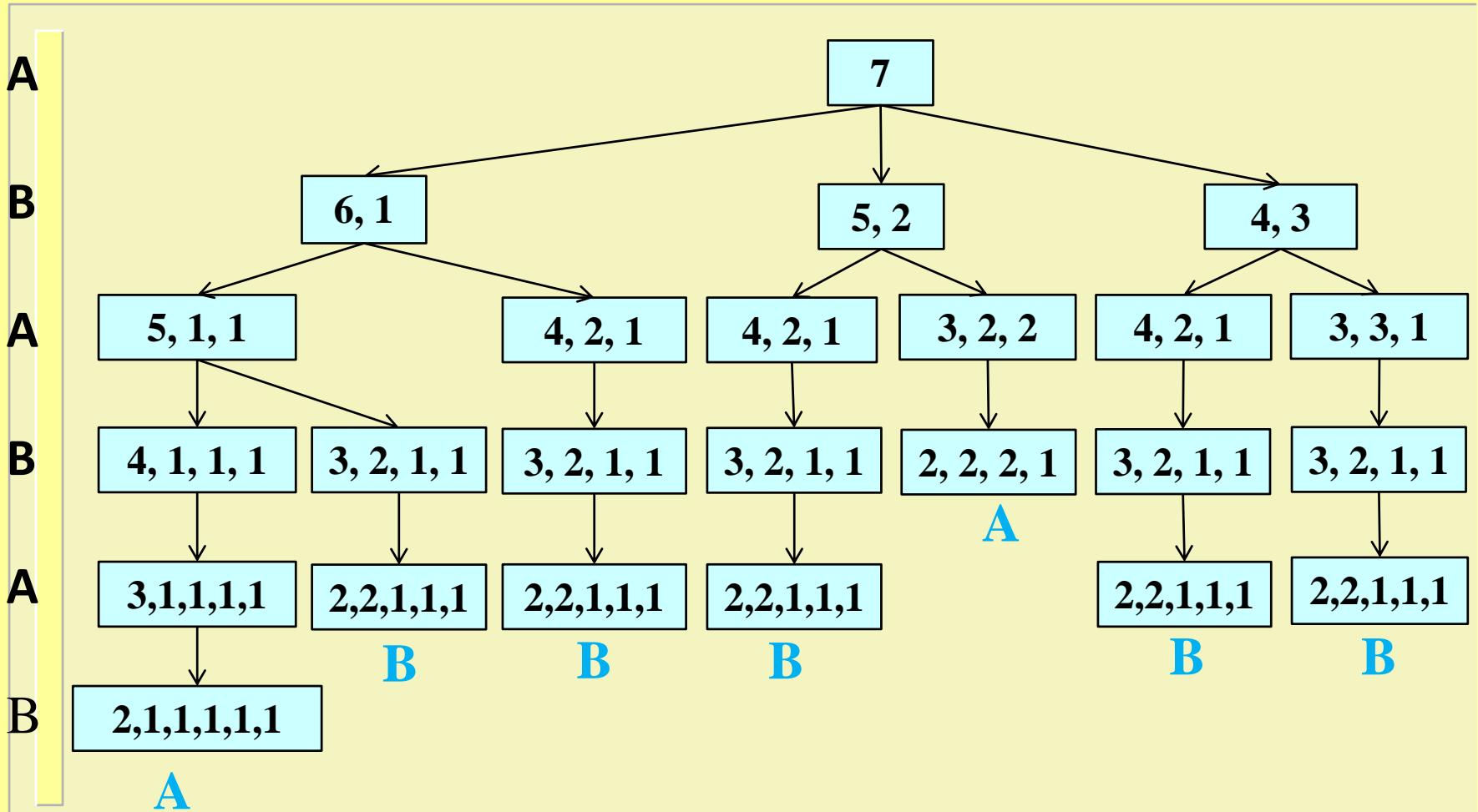
Grundy mama játékgráfja



Játékfa

- csúcs – állás (egy állás több csúcs is lehet)
- szint – játékos (felváltva az A és B szintjei)
- él – lépés (szintről szintre)
- gyökér – kezdőállás (kezdő játékos)
- levél – végállások
- ág – játszma

Grundy mama játékfája



Hogyan tud a **B** játékos biztosan nyerni?

A

B

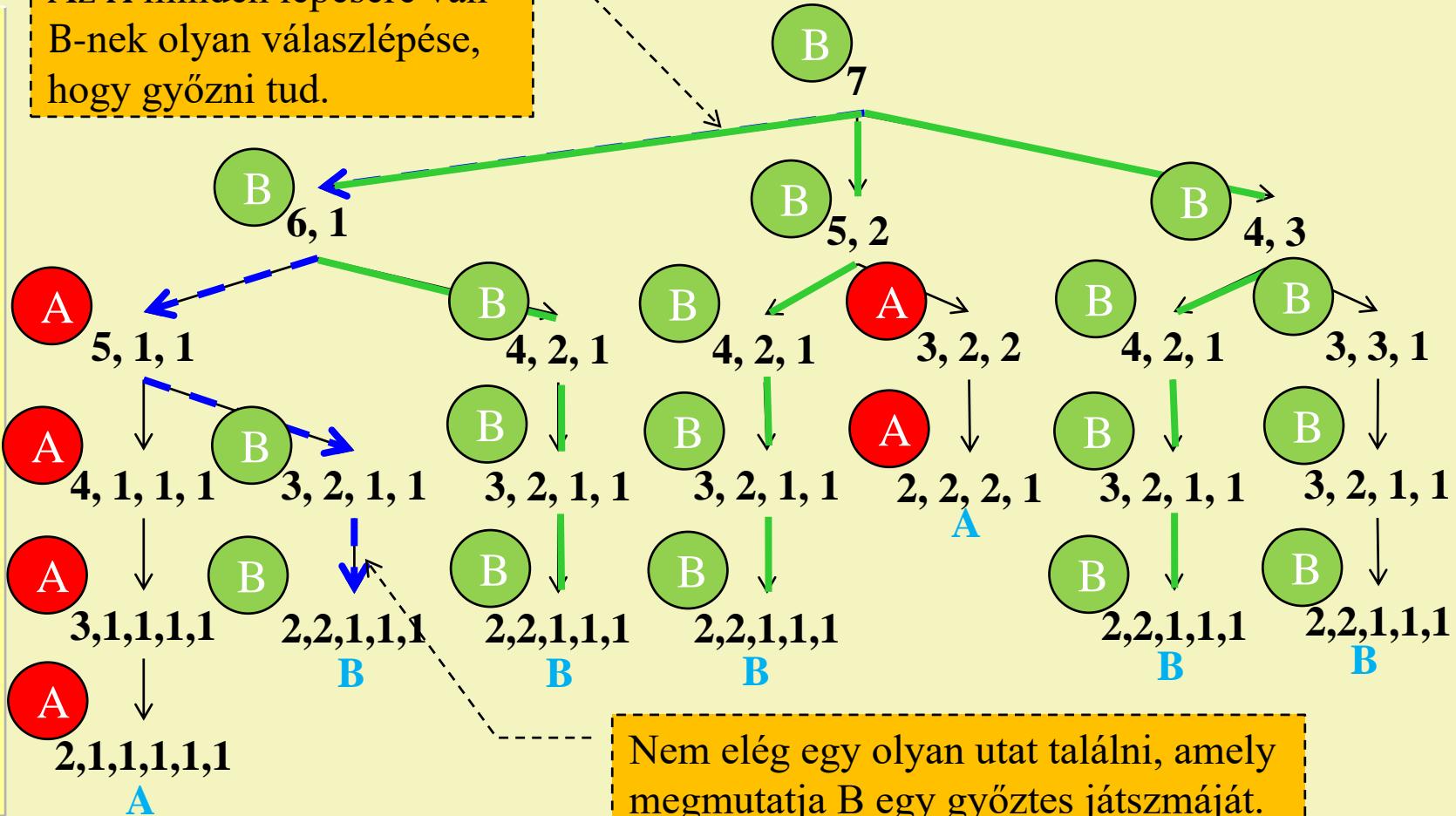
A

B

A

B

Az A minden lépésére van
B-nek olyan válaszlépése,
hogy győzni tud.



Nem elég egy olyan utat találni, amely
megmutatja B egy győztes játszmáját.

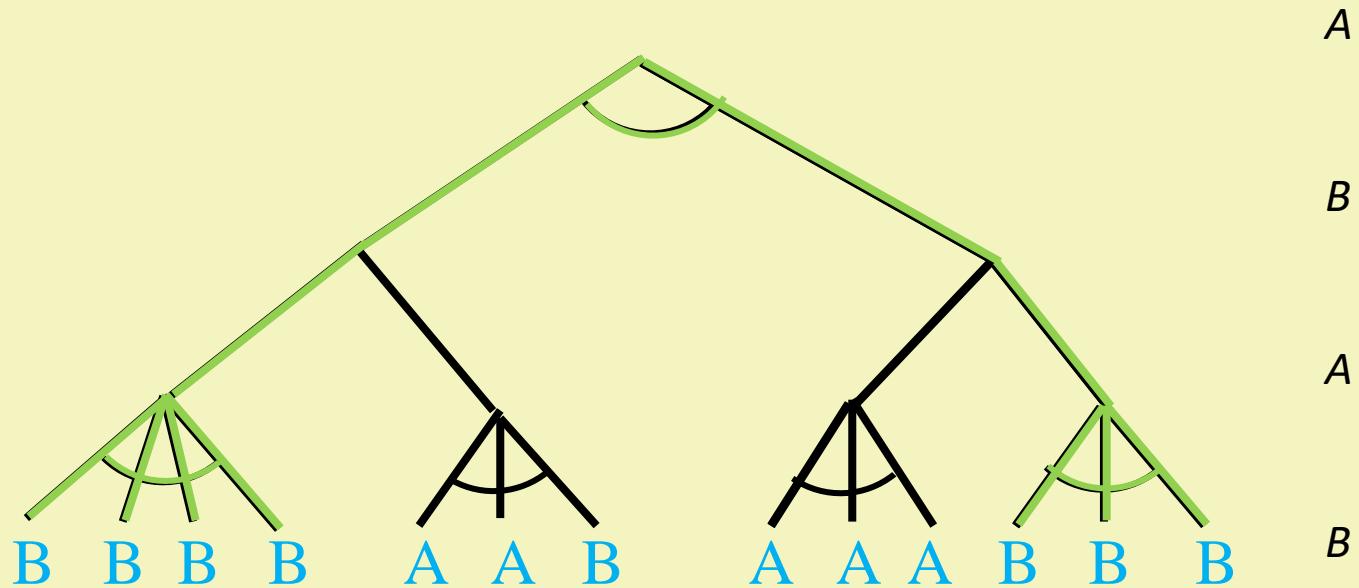
Nyerő stratégia

- Egy játékos nyerő stratégiája egy olyan elv, amelyet betartva az ellenfél minden lépésére tud olyan választ adni, hogy megnyerje a játékot.
- A nyerő stratégia NEM egyetlen győztes játszma, hanem olyan győztes játszmák összessége, amelyek közül az egyiket biztos végig tudja játszani az a játékos, aki rendelkezik a nyerő stratégiával.
- Hasznos lehet a **nem-vesztő stratégia** megtalálása is, ha döntetlent is megengedő játéknál nincs győztes stratégia.
- Általános zéró összegű játékoknál beszélhetünk **adott hasznosságot biztosító stratégiáról**.

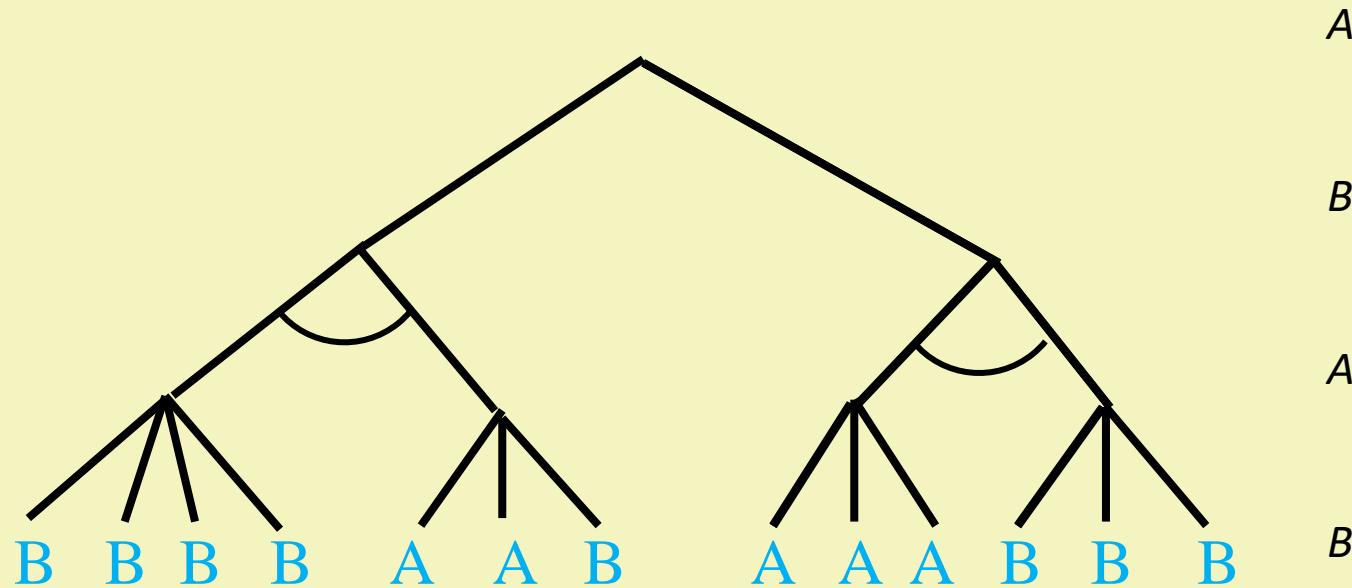
Megjegyzés

- ❑ A játék az egyik játékos szempontjából egy ÉS/VAGY fával ábrázolható.
 - saját szinten egy belső csúcs utódai között VAGY kapcsolat van
 - ellenfél szintjén egy belső csúcs utódai között ÉS kapcsolat van
- ❑ A nyerő (nem-vesztő) stratégiát az ÉS/VAGY játékfa azon hiper-útja mutatja, amely a kezdő állás csúcsából (startcsúcs, gyökércsúcs) vezet a nyerő (nem-vesztő) végállások csúcsaiba, és
 - saját szinten egy belső csúcsnak egyetlen utódját tartalmazza,
 - ellenfél szintjén egy belső csúcsnak minden utódát tartalmazza.
- ❑ A nyerő stratégia keresése tehát egy ÉS/VAGY fabeli hiper-út keresési probléma.

*Nyerő stratégia keresése a **B** játékos ÉS/VAGY fájában*



Nyerő stratégia keresése az A játékos ÉS/VAGY fájában

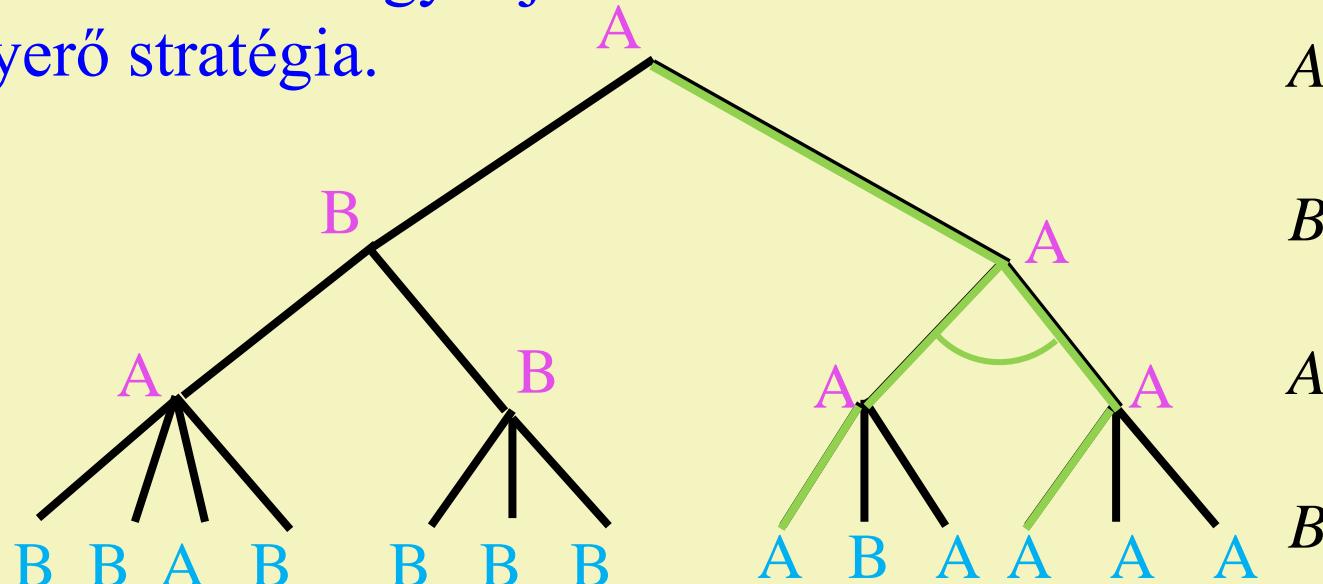


Nincs nyerő stratégia.

Csak az egyik játékosnak lehet nyerő stratégiája.

Téte

- A két esélyes (győzelem vagy vereség) teljes információjú véges determinisztikus kétszemélyes játékokban az egyik játékos számára biztosan létezik nyerő stratégia.



- A három esélyes játékokban (van döntetlen is) a nem vesztő stratégiát lehet biztosan garantálni.

Részleges játékfa-kiértékelés

- ❑ A nyerő vagy nem-vesztő stratégia megkeresése egy nagyobb játékfa esetében **reménytelen**.
- ❑ Az optimális lépés helyett a **soron következő jó lépést** keressük.
 - Legyen a bennünket képviselő játékos neve mostantól MAX, az ellenfél pedig MIN.
- ❑ Ehhez az aktuális állapotból indulva kell a játékfa **néhány szintjét felépíteni**, ezen a részfa leveleinek számunkra való hasznosságát **megbecsülni**, majd ez alapján a soron következő lépést **meghatározni**.

Kiértékelő függvény

- minden esetben szükségünk van egy olyan heurisztikára, amely a mi szempontunkból becsüli meg egy állás hasznosságát: $f: \text{Állások} \rightarrow [-1000, 1000]$ függvény.
- Példák:

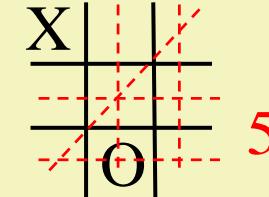
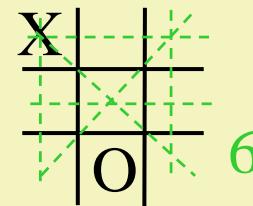
- Sakk: (kiértékelő függvény a fehérnek)

$$f(s) = (\text{fehér királynő száma}) - (\text{fekete királynő száma})$$

- Tic-tac-toe: $f(s) = M(s) - O(s)$

$M(s)$ = a saját lehetséges győztes vonalaink száma

$O(s)$ = az ellenfél lehetséges győztes vonalaink száma

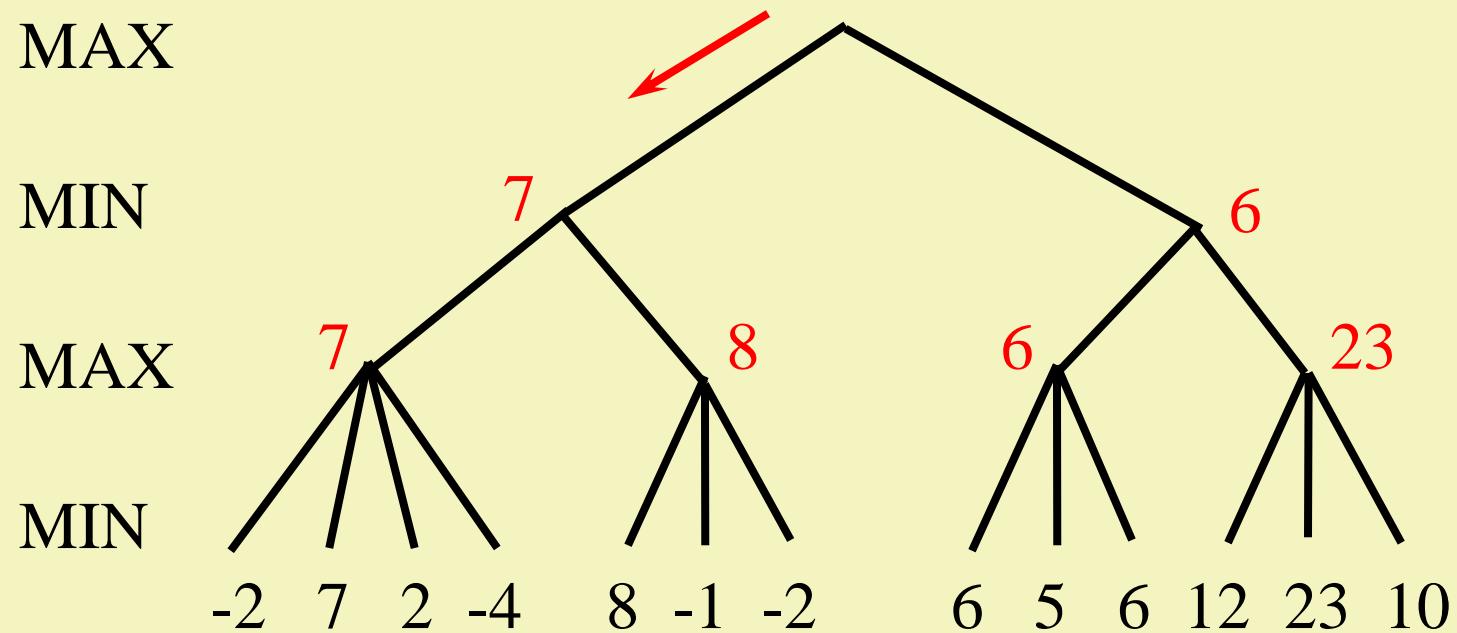


Minimax algoritmus

1. A játékfának az adott állás csúcsából leágazó részfáját felépítjük néhány szintig.
2. A részfa leveleit kiértékeljük a kiértékelő függvény segítségével.
3. Az értékeket felfuttatjuk a fában:
 - A saját (MAX) szintek csúcsaihoz azok gyermekéinek maximumát: $szülő := \max(gyerek_1, \dots, gyerek_k)$
 - Az ellenfél (MIN) csúcsaihoz azok gyermekéinek minimumát: $szülő := \min(gyerek_1, \dots, gyerek_k)$
4. Soron következő lépések ahhoz az álláshoz vezet, ahonnán a gyökérhez felkerült a legnagyobb érték.

Példa

Legyen a mi nevünk MAX, az ellenfélénk MIN.

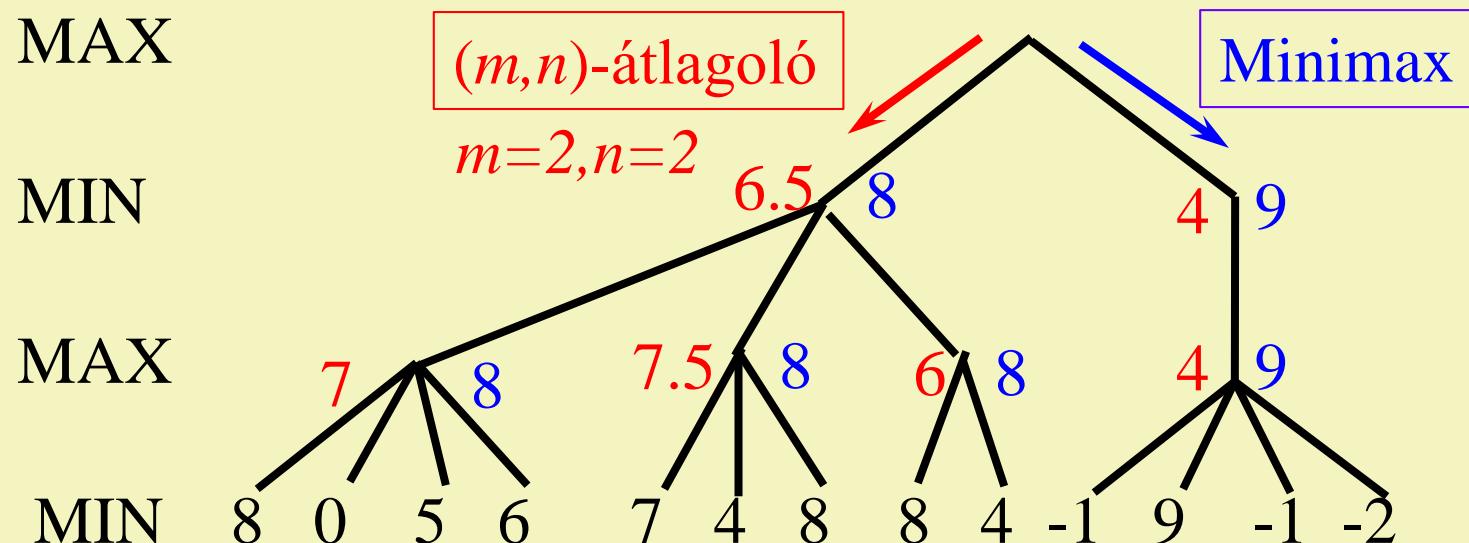


Megjegyzés

- Az algoritmust minden alkalommal, valahányszor mi következünk, megismételjük, hiszen lehet, hogy az ellenfél nem az általunk várt legerősebb lépésekkel válaszol, mert:
 - eltérő mélységű részfával dolgozik,
 - más kiértékelő függvényt használ,
 - nem minimax eljárást alkalmaz,
 - hibázik.

Átlagoló kiértékelés

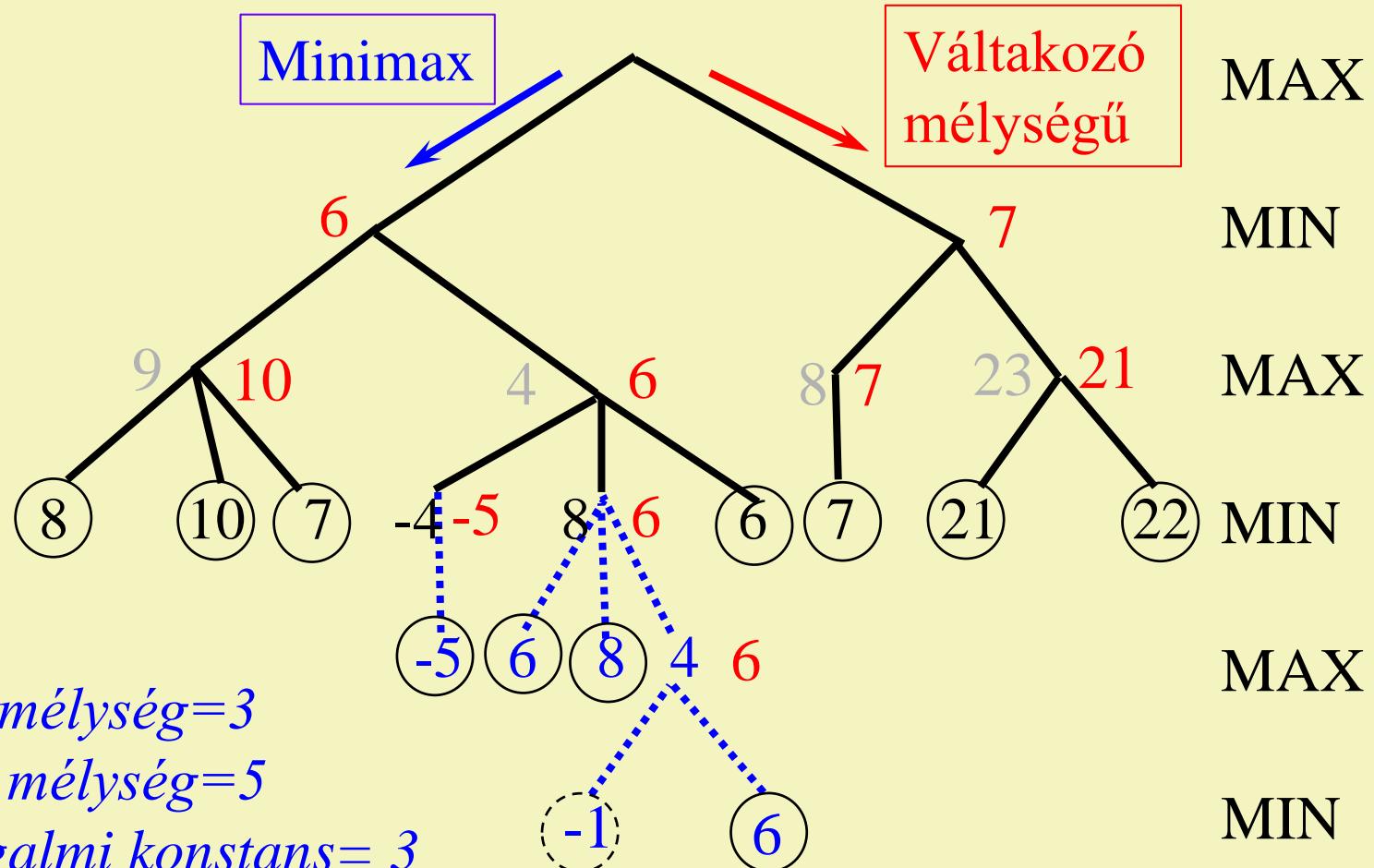
- Célja a kiértékelő függvény esetleges tévedéseinek simítása.
- MAX szintjeire az m darab legnagyobb értékű gyerek (\max_m) átlaga, a MIN-re az n darab legkisebb értékű gyerek (\min_n) átlaga kerül.



Váltakozó mélységű kiértékelés

- Célja, hogy a kiértékelő függvény minden ágon reális értéket mutasson. Megtévesztő lehet egy csúcsnál ez az érték ha annak szülőjénél a kiértékelő függvény lényegesen eltérő értéket mutat: a játék ezen szakasza nincs nyugalomban.
- Egy adott szintig (minimális mélység) mindenképpen felépítjük a részfát,
- majd ettől a szinttől kezdve egy adott szintig (maximális mélység) csak azon csúcsokat terjesztjük ki, amelyekre nem teljesül a
nyugalmi teszt: $|f(\text{szülő}) - f(\text{csúcs})| < K$,

Példa



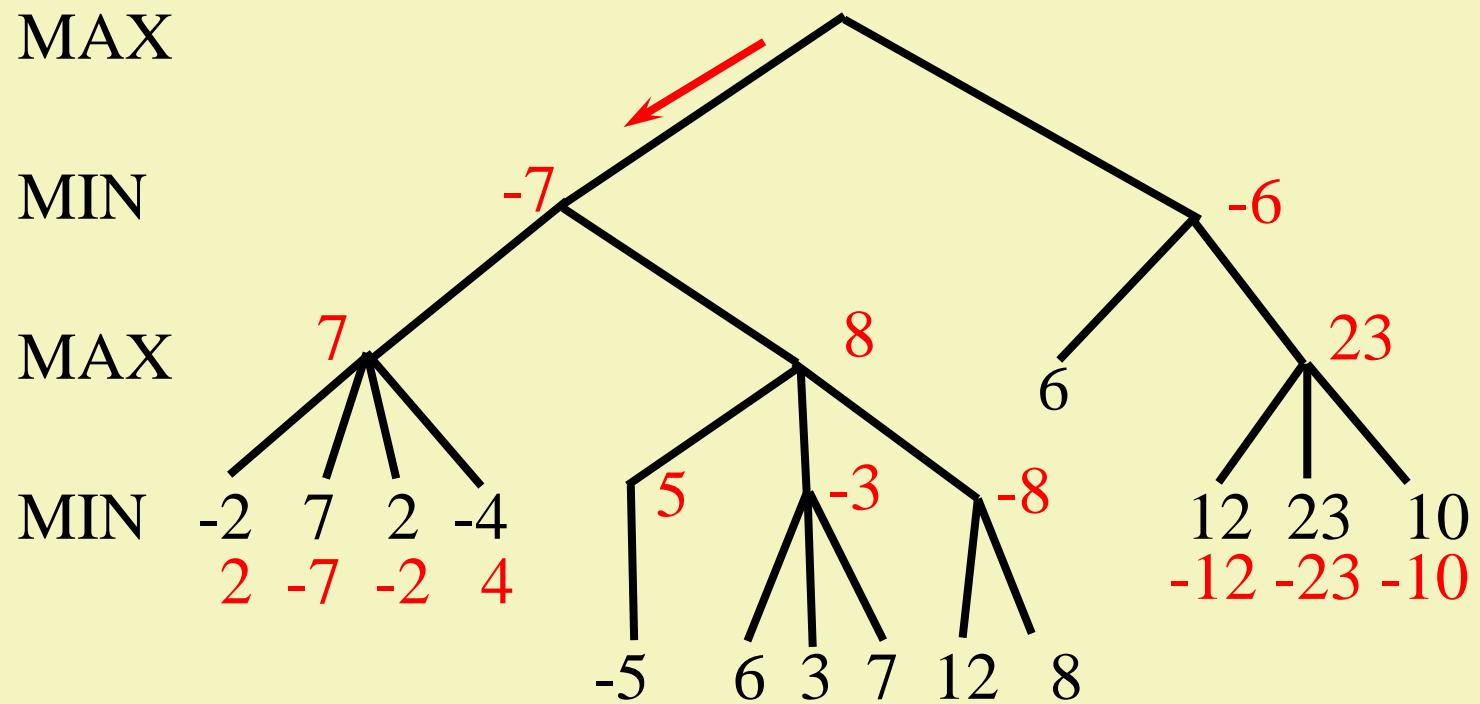
Szelektív kiértékelés

- ❑ Célja a **memória-igény** csökkentése.
- ❑ Elkülönítjük a lényeges és lényegtelen lépéseket, és csak a **lényeges lépéseknek** megfelelő részfát építjük fel.
- ❑ Ez a szétválasztás heurisztikus ismeretekre épül.

Negamax algoritmus

- Negamax eljárást **könnyebb implementálni**.
 - Kezdetben (-1) -gyel szorozzuk azon levélcsúcsok értékeit, amelyek az ellenfél (MIN) szintjein vannak, majd
 - Az értékek felfuttatásánál minden szinten az alábbi módon számoljuk a belső csúcsok értékeit:
$$\text{szülő} := \max(-\text{gyerek}_1, \dots, -\text{gyerek}_k)$$

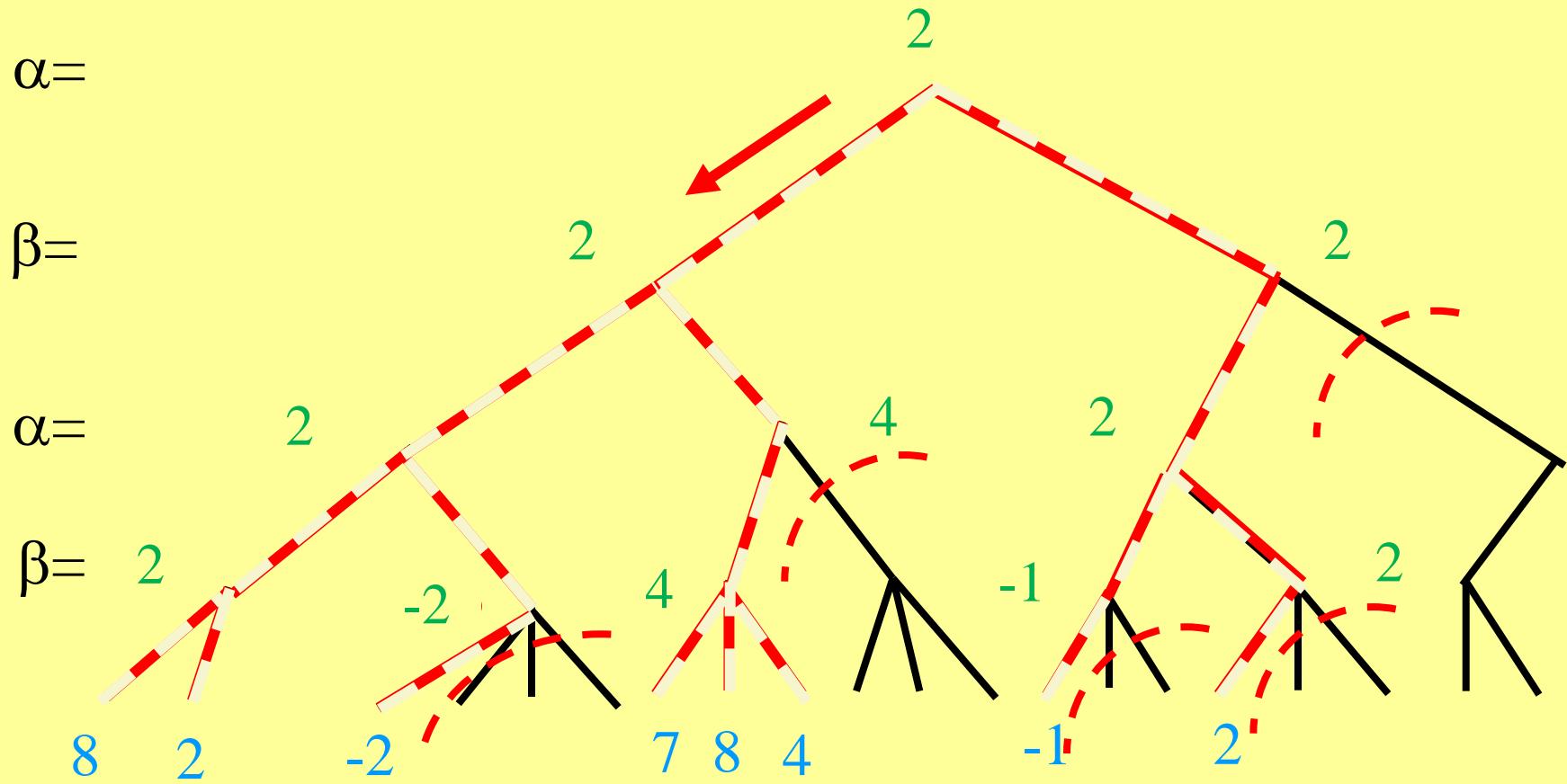
Példa



Alfa-béta algoritmus

- Visszalépéses algoritmus segítségével járjuk be a részfát (**olyan mélységi bejárás, amely mindenkor csak egy utat tárol**). Az aktuális úton fekvő csúcsok **ideiglenes értékei**:
 - a MAX szintjein α érték: ennél rosszabb értékű állásba innen már nem juthatunk
 - A MIN szintjein β érték: ennél jobb értékű állásba onnan már nem juthatunk
- **Lefelé haladva** a fában $\alpha := -\infty$, és $\beta := +\infty$.
- **Visszalépéskor** az éppen elhagyott (gyermek) csúcs értéke (felhozott érték) módosíthatja a szülő csúcs értékét:
 - a MAX szintjein: $\alpha := \max(\text{felhozott érték}, \alpha)$
 - a MIN szintjein: $\beta := \min(\text{felhozott érték}, \beta)$
- **Vágás**: ha az úton van olyan α és β , hogy $\alpha \geq \beta$.

Példa

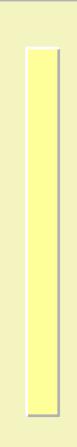


Elemzés

- ❑ Ugyanazt a kezdőlépést kapjuk eredményül, amit a minimax algoritmus talál. (Több egyforma kezdőirány esetén a „baloldalit” választjuk.)
- ❑ **Memória igény:** csak egy utat tárol.
- ❑ **Futási idő:** a vágások miatt sokkal jobb, mint a minimax módszeré.
 - Átlagos eset: egy csúcs alatt, két belőle kiinduló ág megvizsgálása után már vághatunk.
 - Optimális eset: egy d mélységű b elágazású fában kiértékelte levélcsúcsok száma: $\sqrt{b^d}$
 - Jó eset: A részfa megfelelő rendezésével érhető el.

Kétszemélyes játékot játszó program

- Váltakozó mélységű, szelektív, (m,n) átlagoló, negamax alfa-béta kiértékelést végez.
- Keretprogram, amely váltakozva fogadja a felhasználó lépésein, és generálja a számítógép lépésein.
- Kiegészítő funkciók (beállítások, útmutató, segítség, korábbi lépések tárolása, mentés stb.)
- Felhasználói felület, grafika
- Heurisztika megválasztása (kiértékelő függvény, szelekció, kiértékelés sorrendje)



VII. Evolúciós algoritmusok

Evolúció, mint kereső rendszer

- ❑ A problémára adható néhány lehetséges választ, azaz a problématér több **egyedét** tároljuk egyszerre. Ez a **populáció**.
- ❑ Kezdetben egy többnyire véletlen populációt választunk. A cél egy bizonyos célegyed vagy egy jó populáció előállítása.
- ❑ Az egyedekeket egy **rátermettségi függvény** alapján hasonlítjuk össze.
- ❑ A **populációt lépésről lépésre javítjuk** úgy, hogy a kevésbé rátermett egyedek egy részét a rátermettebbekhez hasonló egyedekre cseréljük le. Ez a változtatás visszavonhatatlan. Ez tehát egy **nem-módosítható stratégiájú keresés**.

Evolúciós operátorok és a terminálási feltétel

- **Szelekció:** Kiválasztunk néhány (lehetőleg rátermett) egyedet szülőnek.
- **Rekombináció (keresztezés):** Szülőkből utódok készülnek úgy, hogy a szülők tulajdonságait örököljék az utódok.
- **Mutáció:** Az utódok tulajdonságait kismértékben módosítjuk.
- **Visszahelyezés:** Új populációt alakítunk ki az utódokból és a régi populációból.
- **Terminálási feltétel:**
 - ha a célegyed megjelenik a populációban
 - ha a populáció egyesített rátermettségi függvény értéke egy ideje nem változik.

ADAT := kezdeti érték

while \neg terminálási feltétel(ADAT) **loop**

 SELECT SZ FROM alkalmazható szabályok

 ADAT := SZ(ADAT)

endloop

Evolúció alapalgoritmusa

Procedure EA

populáció := kezdeti populáció

while terminálási feltétel nem igaz **loop**

 szülők := szelekció(*populáció*)

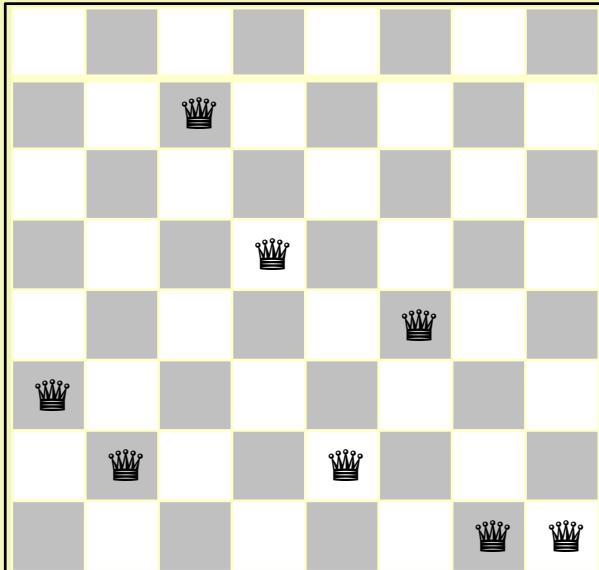
 utódok := rekombináció(szülők)

 utódok := mutáció(utódok)

populáció := visszahelyezés(*populáció*, utódok)

endloop

n-királynő probléma 1.



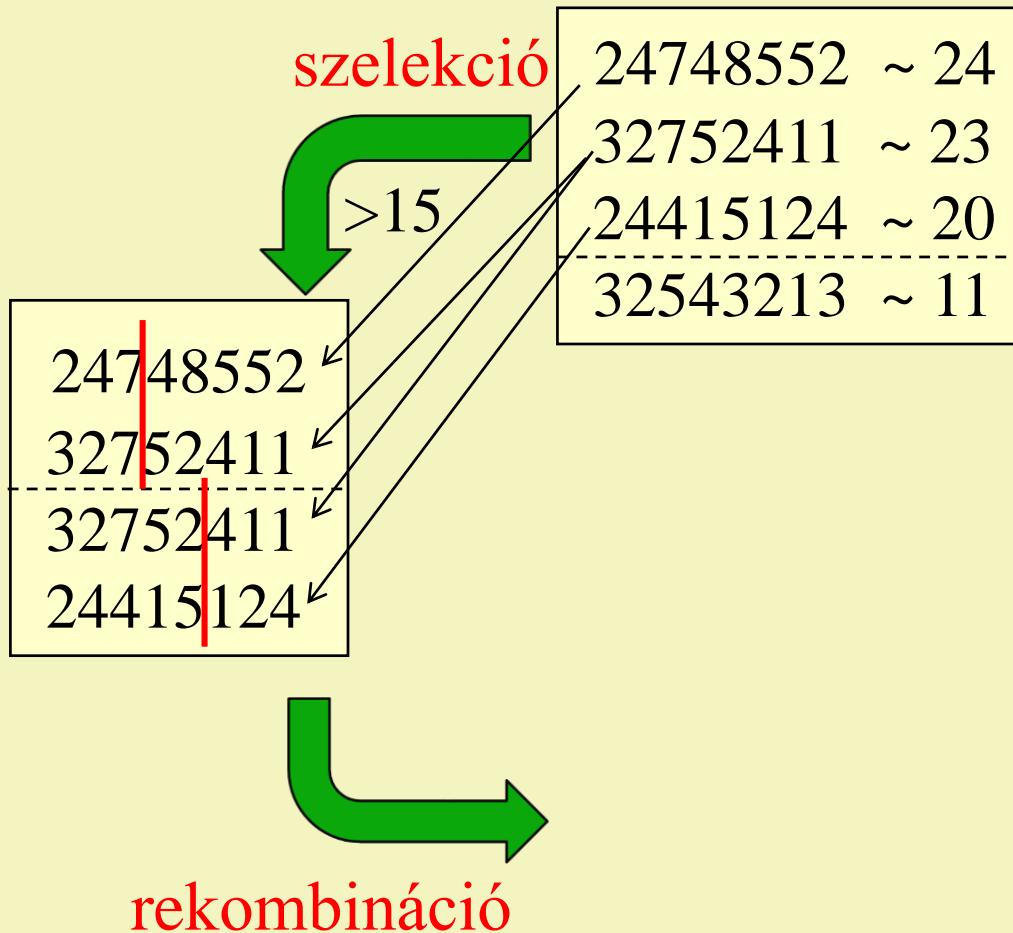
3 2 7 5 2 4 1 1

rátermettségi érték: 23

- Egyed: a királynők olyan elrendezése, ahol minden oszlop pontosan egy királynőt tartalmaz

- Reprezentáció: a királynők sor pozíciót tartalmazó sorozat
- Rátermettségi függvény: ütésekben nem levő királynő párok száma

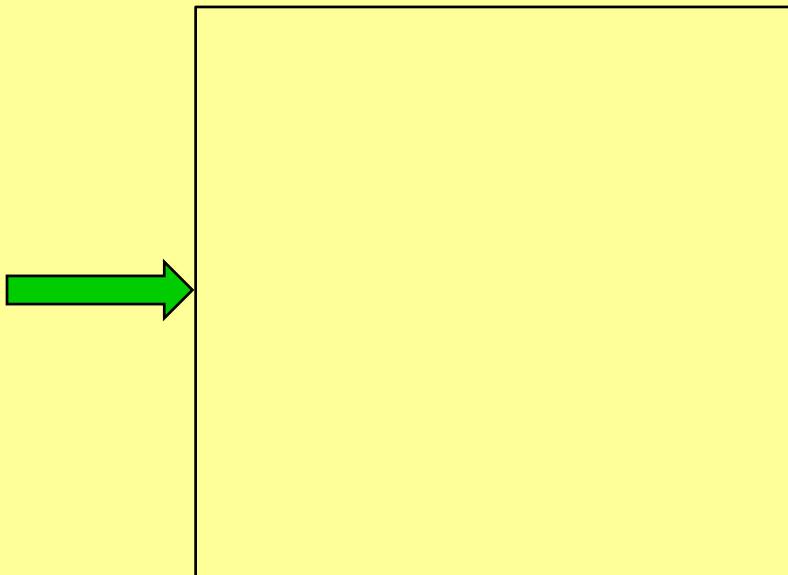
Evolúciós ciklus



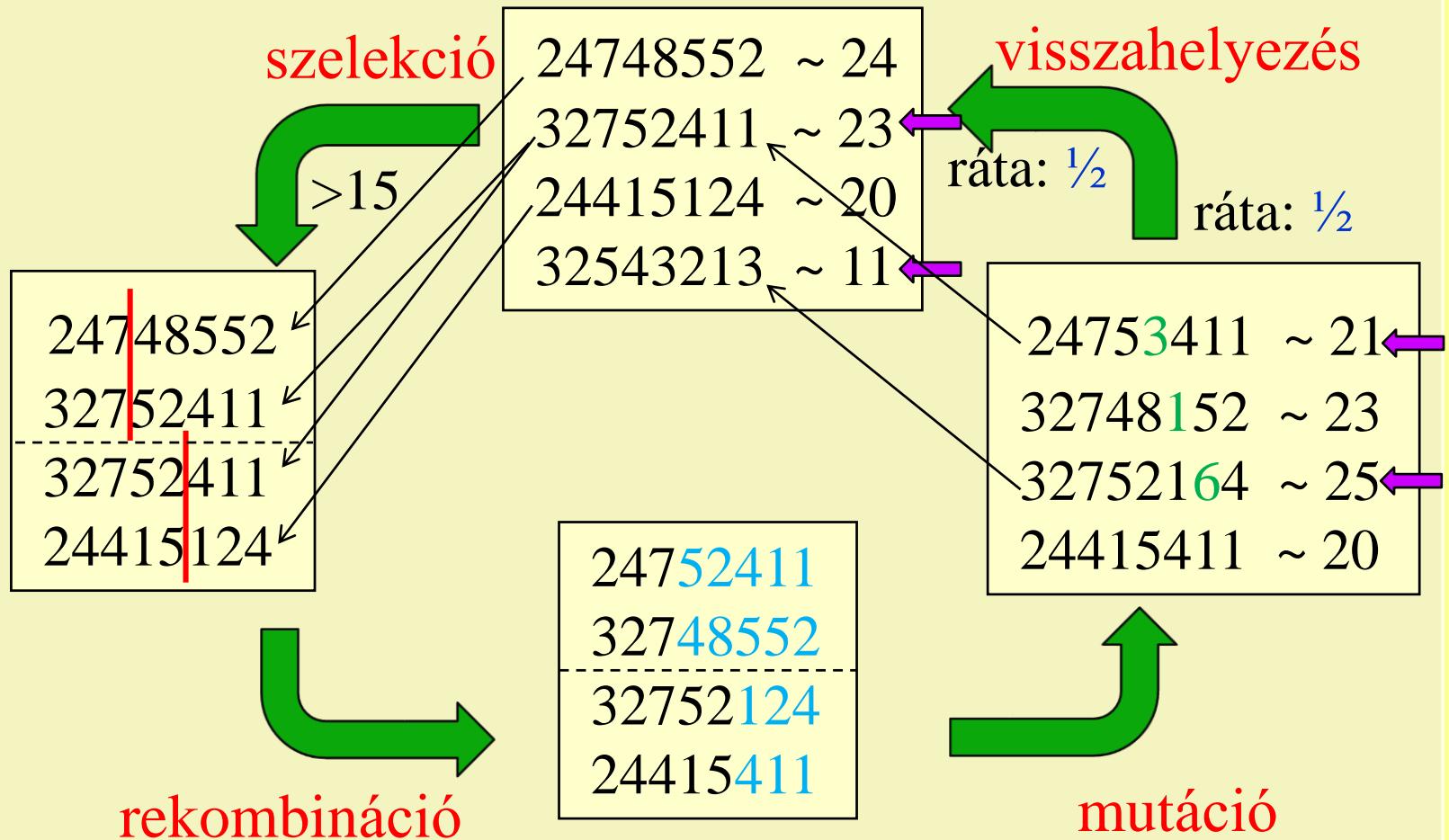
Keresztezés

3	2	7	5	2	4	1	1
---	---	---	---	---	---	---	---

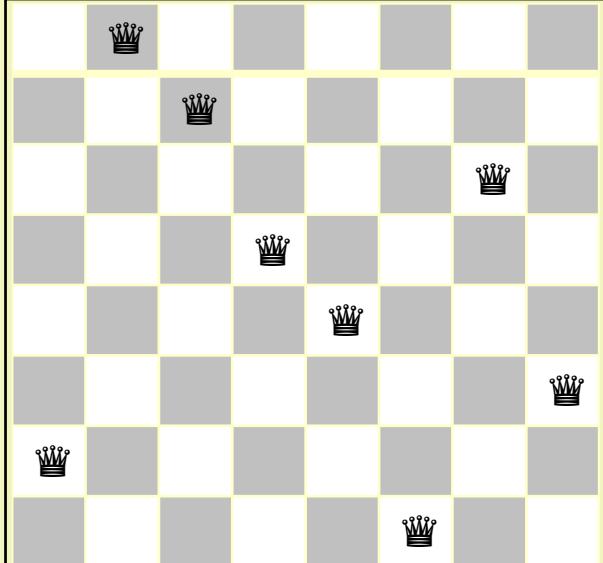
2	4	7	4	8	5	5	2
---	---	---	---	---	---	---	---



Evolúciós ciklus



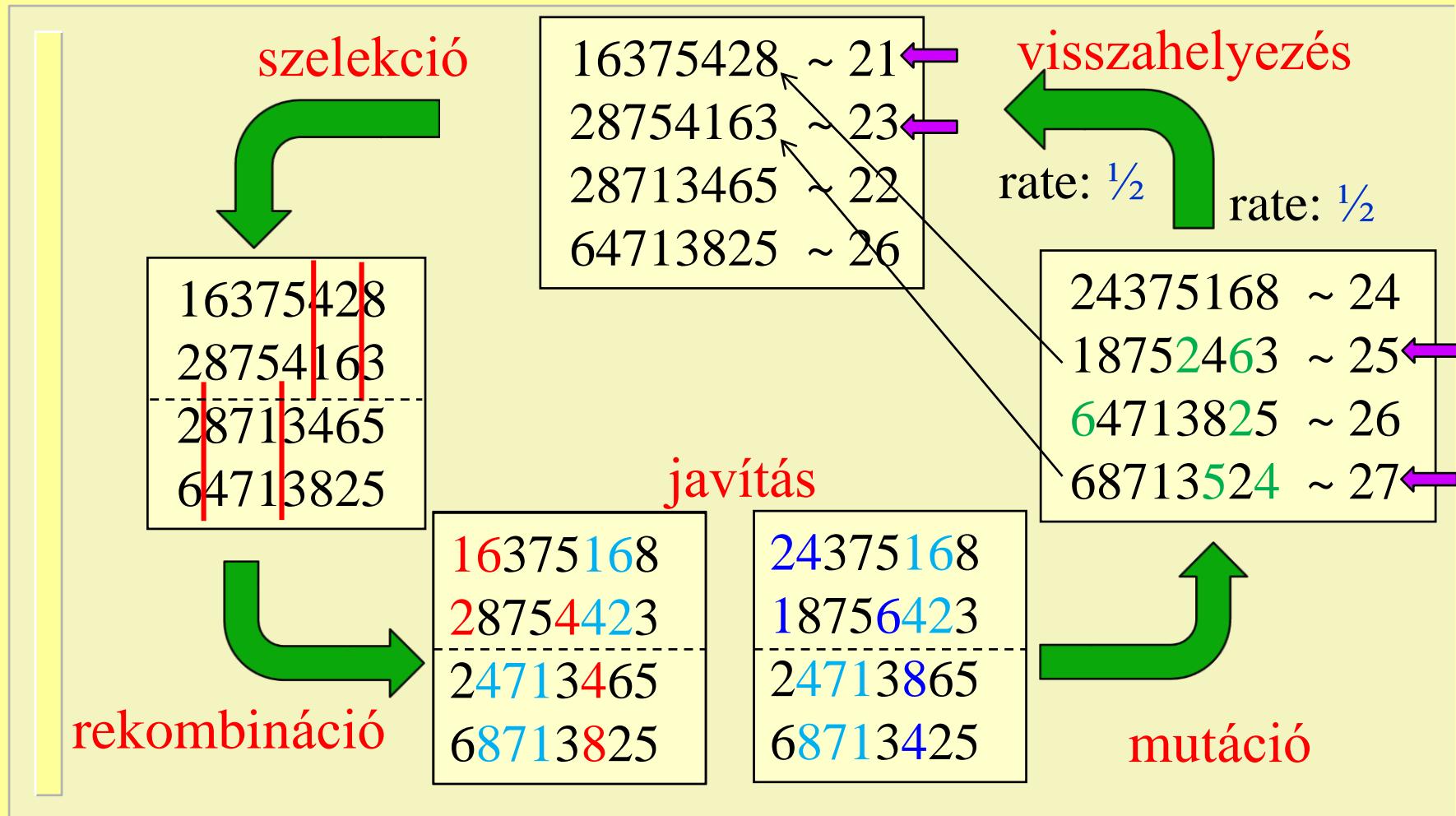
n-királynő probléma 2.



rátermettségi érték: 23

- Egyed: a királynők olyan elrendezése, ahol minden sor és oszlop pontosan egy királynőt tartalmaz
- Reprezentáció: a királynők sor pozíciót tartalmazó permutáció **permutáció invariáns**
- Rátermettségi függvény: ütésekben nem levő királynő párok száma

Evolúciós ciklus



Kielégíthetőségi probléma (SAT)

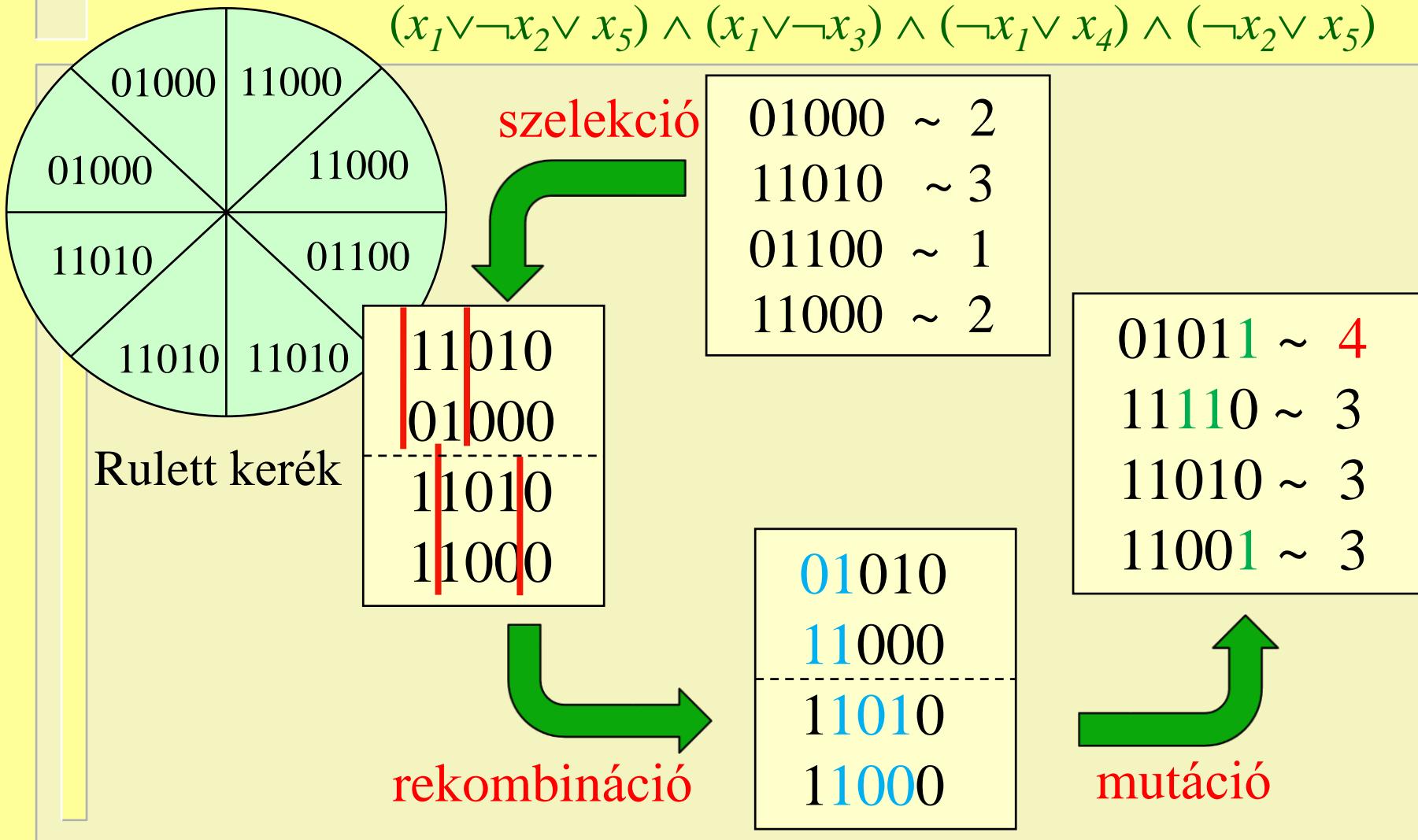
Adott egy n változós Boolean formula KNF alakban. A változók milyen igazság kiértékelése mellett lesz formula igaz?

E.g.: $(x_1 \vee \neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3) \wedge (\neg x_1 \vee x_4) \wedge (\neg x_2 \vee x_5)$

egy megoldás: $x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{false}, x_4 = \text{true}, x_5 = \text{true}$

- Egyed: egy lehetséges igazság kiértékelés
- Reprezentáció: logikai érték (bitek) sorozata
- Rátermettségi függvény: Az adott formula igazra értékeltek klózainak száma

Evolúciós ciklus



Evolúciós algoritmus elemei

- problématér egyedeinek reprezentációja: kódolás
- rátermettségi függvény (fitnesz függvény)
 - kapcsolat a kódolással és a céllal
- evolúciós operátorok
 - szelekció, rekombináció, mutáció, visszahelyezés
- kezdő populáció, megállási feltétel (cél)
- stratégiai paraméterek
 - populáció mérete, mutáció valószínűsége, utódképzési ráta, visszahelyezési ráta, stb.

Kódolás

- Egy egyedet egy **jelsorozattal** (kromoszómával) kódolunk. A jelsorozatnak néha ki kell elégítenie egy **kód-invariánst**.
- Az egyedekeket az őket reprezentáló kódjukon keresztül változtatjuk meg. Egy jel vagy jelcsoport, azaz a gén írja le az egyed egy tulajdonságát (attribútum-érték pájját).
 - Sokszor egy génnel a kódsorozatban elfoglalt pozíciója (lókusza) jelöli ki a gén által leírt attribútumot, amelynek értéke maga a gén (allél). A kód ekkor tulajdonságoknál **feldarabolható**: egy rövid kódszakasz megváltoztatása kis mértékben változtat az egyeden.
- Gyakori megoldások:
 - **Vektor**: valós vagy egész számok rögzített hosszú tömbje
 - **Bináris kód**: bitek rögzített hosszú tömbje
 - Véges sok elem **permutációja**

Gráf színezési probléma kódolása és rátermettségi függvénye

Adott egy véges egyszerű gráf, amelynek a csúcsait négy szín felhasználásával kell úgy kiszínezni, hogy a szomszédos csúcsok eltérő színűek legyenek.

Direkt kódolás



1. 2. 3. 4. 5. 6. 7.

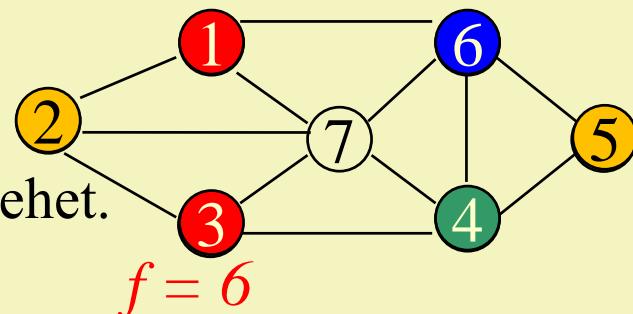
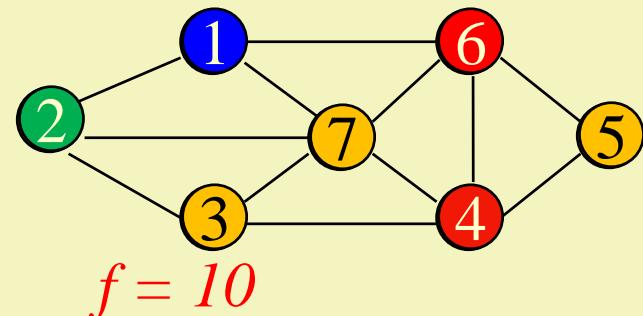
Az $x[i]$ az i -dik csúcs színe.

f a jó élek száma.

Indirekt kódolás



Az i -dik lépésben az $x[i]$ -dik csúcsot színezzük ki a lehető legvilágosabb színnel a szomszédjaihoz igazodva, ha lehet.
 f a kiszínezett csúcsok száma



A kő-papír-olló játék stratégiájának kódolása és rátermettségi függvénye

Alakítsunk ki jó stratégiát egy kő-papír-olló világbajnokságra!

- Olyan függvényre van szükségünk, amelyik a korábbi csaták kimenetele alapján javaslatot tesz a soron következő lépéinkre.
 - Például két korábbi csata alapján:

<i>Előzmény:</i>	<i>Én:</i>	K P	<i>Javaslat:</i>	K
<i>Ő:</i>	O O			
 - Ez még nem a teljes stratégia, mert nem csak a fenti előzményre, hanem az összes lehetséges előzményre kell soron következő lépést javasolni.

Kódolás

Egy stratégia (egyed) kódja: $\{0,1,2\}^{0..80}$
Az összes lehetséges stratégia száma: 3^{81}

<i>Signal</i>	<i>Előzmény (ÉnŐÉnŐ)</i>	<i>Válasz</i>
K ~ 0	KKKK ~ 0000 ~ 0	P ~ 1
P ~ 1	KKKP ~ 0001 ~ 1	O ~ 2
O ~ 2	KKKO ~ 0002 ~ 2	K ~ 0
	KKPK ~ 0010 ~ 3	P ~ 1

	OOOP ~ 2221 ~ 79	O ~ 2
	OOOO ~ 2222 ~ 80	K ~ 0

A stratégia: 1201 ... 20

Rátermettség kiértékelése

Stratégia: 1201 ... 120

Mintajáték:

Én: 0002221222011000
Ő: 0102112220001011

Jel

K ~ 0
P ~ 1
O ~ 2

Eset	→ Javaslat	Ellenfél	Érték
0001	→ 2	0	Vereség -1
0001	→ 2	1	Győzelem +1
0010	→ 1	1	Döntetlen 0
...			
2221	→ 2	1	Győzelem +1
2222	→ 0	0	Döntetlen 0

Szelekció

- **Célja:** a rátermett egyedek kiválasztása úgy, hogy a rosszabbak kiválasztása is kapjon esélyt.
 - **Rátermettség arányos** (rulett kerék algoritmus): minél jobb a rátermettségi függvényértéke egy elemnek, annál nagyobb valószínűsséggel választja ki
 - **Rangsorolásos**: rátermettség alapján sorba rendezett egyedek közül a kisebb sorszámuakat nagyobb valószínűsséggel választja ki
 - **Versengő**: véletlenül kiválasztott egyedcsoporthok (pl. párok) legjobb egyedét választja ki.
 - **Csonkolásos v. selejtezős**: a rátermettség szerint legjobb (adott küszöbérték feletti) valahány egyedből véletlenszerűen választ néhányat.

Rekombináció

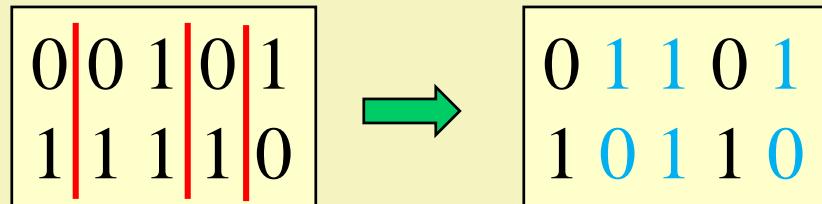
- A feladata az, hogy adott szülő-egyedekből olyan utódokat hozzon létre, amelyek a szüleik tulajdonságait "öröklik".
 - **Keresztezés**: véletlen kiválasztott pozíción jelcsoportok (gének) vagy jelek cseréje
 - **Rekombináció**: a szülő egyedek megfelelő jeleinek kombinálásával kapjuk az utód megfelelő jelét

Ügyelni kell a kód-invariáns megtartására: vizsgálni kell, hogy az új kód értelmes lesz-e (permutáció)

Keresztezés

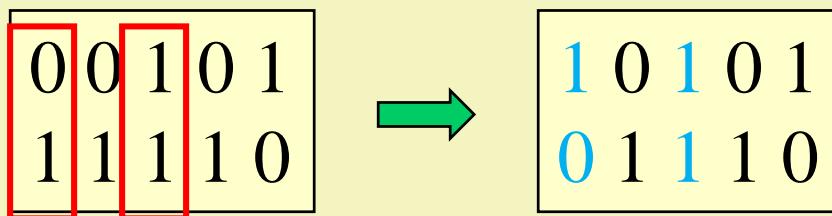
❑ Egy- illetve többpontos keresztezés

- Kódszakaszokat cserélünk



❑ Egyenletes keresztezés

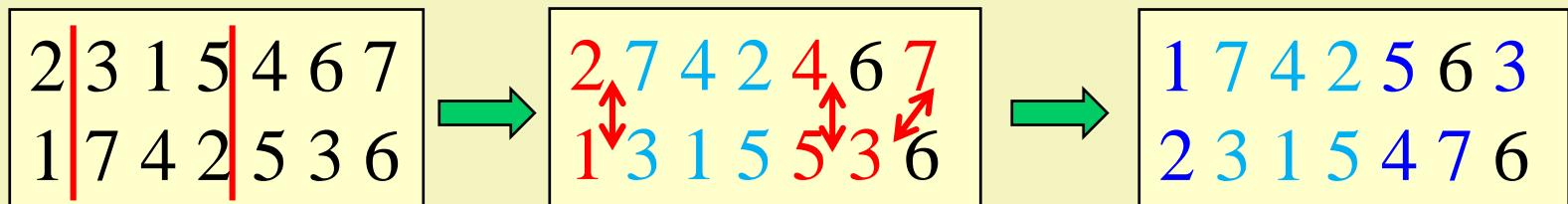
- Jeleket cserélünk



Permutációk keresztezése 1.

□ Parciálisan illesztett keresztezés

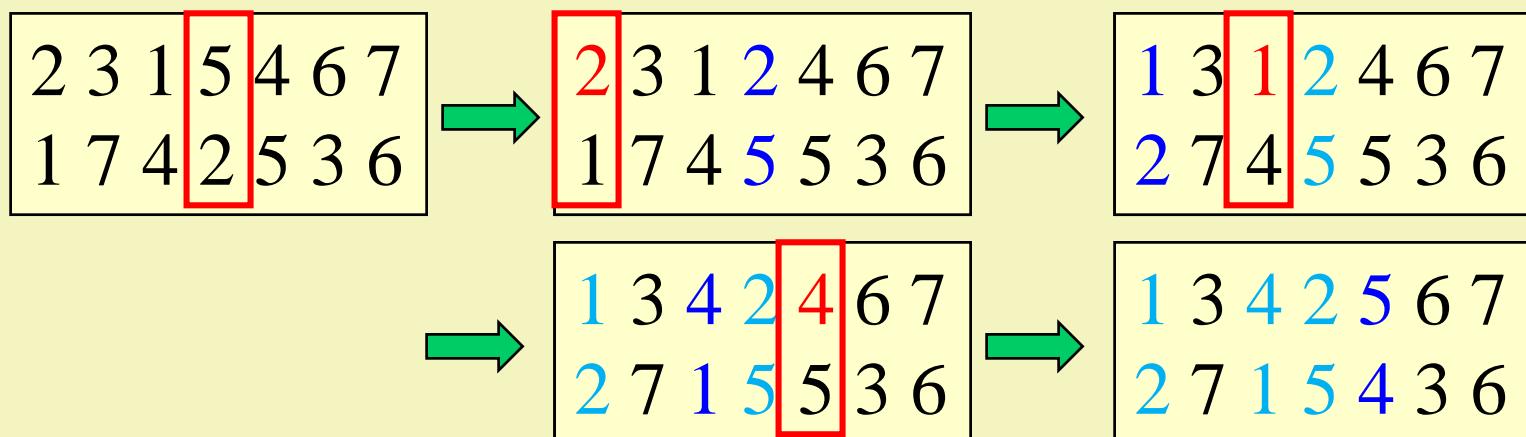
- Egy szakasz cseréje után párba állítja és kicseréli azokat a szakaszon kívüli elemeket, amelyek megsértik a permutáció tulajdonságát.



Permutációk keresztezése 2.

□ Ciklikus keresztezés

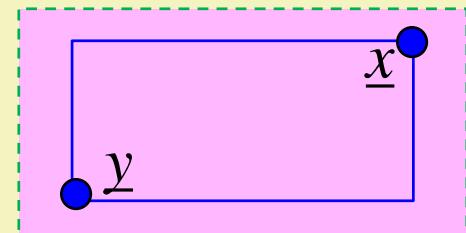
1. Választ egy véletlen $i \in [1..length]$ -t
2. $a_i \leftrightarrow b_i$
3. Keres olyan $j \in [1..length]$ -t ($j \neq i$), amelyre $a_j = a_i$,
4. Ha nem talál, akkor vége, különben $i := j$
5. goto 2.



Rekombináció vektorokra

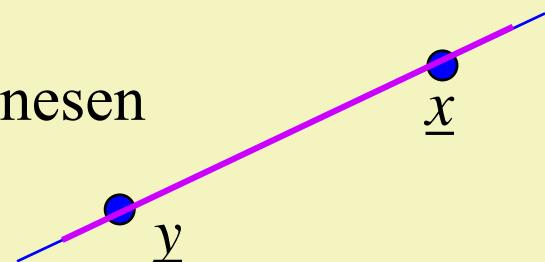
❑ Köztes rekombináció

- A szülők (\underline{x} , \underline{y}) által kifeszített hipertéglakörnyezetében lesz az utód (\underline{u}).
- $\forall i=1 \dots n : u_i = a_i x_i + (1-a_i) y_i \quad a_i \in [-h, 1+h]$ véletlen



❑ Lineáris rekombináció

- A szülők (\underline{x} , \underline{y}) által kifeszített egyenesen a szülők környezetében vagy a szülők között lesz az utód (\underline{u}).
- $\forall i=1 \dots n : u_i = a x_i + (1-a) y_i \quad a \in [-h, 1+h]$ véletlen



Mutáció

- ❑ A mutáció egy egyed (utód) kis mértékű véletlen változtatását végzi.
- ❑ Valós tömbbel való kódolásnál kis p valószínűsséggel:
 - $\forall i=1 \dots n : z_i = x_i \pm range_i^* (1-2*p)$
- ❑ Bináris tömbbel való kódolásnál kis p valószínűsséggel:
 - $\forall i=1 \dots n : z_i = 1 - x_i$ if $random[0..1] < p$
- ❑ Permutáció esetén
 - egy jelpár cseréje
 - egy kódszakaszban a jelek ciklikus léptetése vagy megfordítása vagy átrendezése.

Visszahelyezés

- A visszahelyezés a populációnak az utódokkal történő frissítése: Kiválasztja a populációnak a lecserélendő egyedeit, és azok helyére a kiválasztott utódokat teszi.

$$\text{utódképzési ráta (u)} = \frac{\text{utódok száma}}{\text{populáció száma}}$$

két szelekció is kell

$$\text{visszahelyezési ráta (v)} = \frac{\text{lecserélendő egyedek száma}}{\text{populáció száma}}$$

- ha $u=v$, akkor feltétlen cseréről van szó
- ha $u < v$, akkor egy utód több példánya is bekerülhet
 - további szelekció
- ha $u > v$, akkor az utódok közül szelektál
 - további szelekció

Gépi Tanulás Előadás 1

Felügyelt Tanulás: Bevezetés, K Legközelebbi Szomszédok, Véletlen Erdők

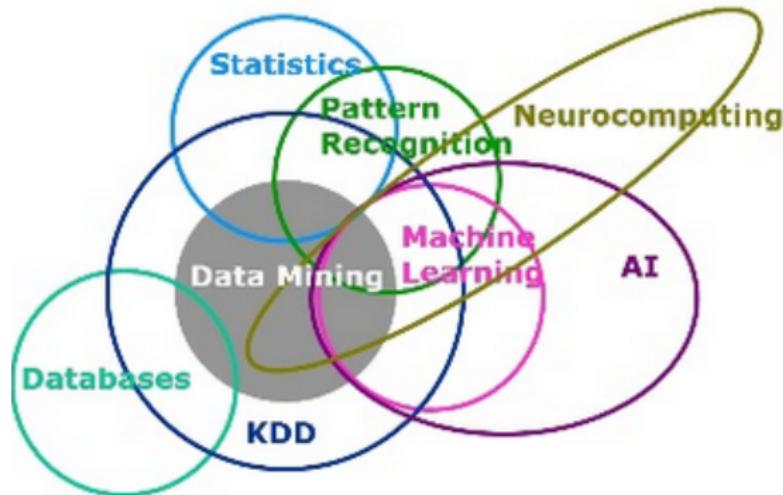
Milacski Zoltán Ádám¹

¹Eötvös Loránd Tudományegyetem
Programozáselmélet és Szoftvertechnológiai Tanszék
srph25@gmail.com

2018. május 3.



MACHINE LEARNING IS...



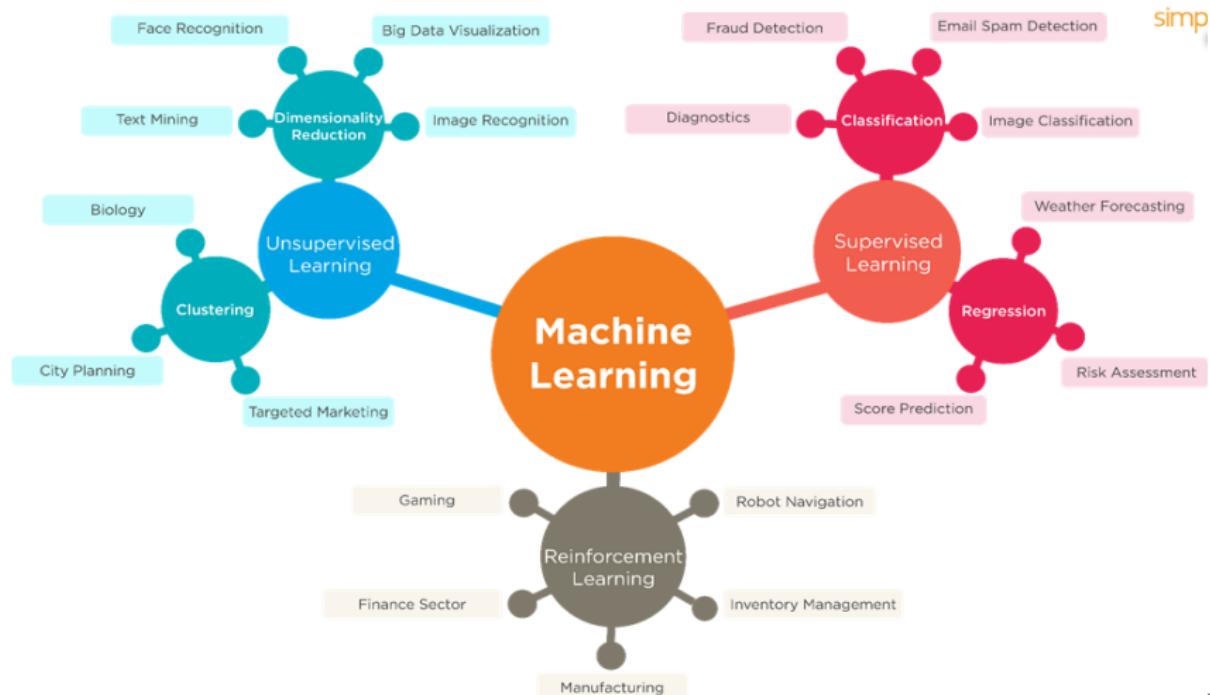
“Field of study that gives computers the ability to learn **without being explicitly programmed.**”

~ Arthur Samuel, 1959

- AI: Intelligens gépek, amik emberszerűen gondolkodnak és cselekednek.
- ML: Rendszerek amik előre programozás nélkül tanulnak.
YCombinator szerinti következő forradalmi technológia (internet, mobil után).

Gépi Tanulás

Mi ez és mi nem



ML módszerek tipikusan jól skálázódnak a feladat méretében (zavarbaejtően párhuzamos tenzor műveletek GPU-n).



Gépi Tanulás

Mi ez és mi nem

► Pure Reinforcement Learning (cherry)

- The machine predicts a scalar reward given once in a while.

► A few bits for some samples

► Supervised Learning (icing)

- The machine predicts a category or a few numbers for each input
- 10→10,000 bits per sample

► Unsupervised/Predictive Learning (génoise)

- The machine predicts any part of its input for any observed part.
- Predicts future frames in videos
- Millions of bits per sample



► Unsupervised Learning is the Dark Matter (or Dark Energy) of AI

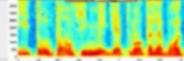
- RL: Megy, ha a jutalom (szám) adott tanuláshoz. Humán tudást reprodukál.
- SL: Megy, ha az output (vektor) adott tanuláshoz. Humán tudást reprodukál.
- UL: Megoldatlan, sok nyitott kutatási kérdés (tenzor). Ember nélkül tanul.



Felügyelt Tanulás

Szemléletesen

Felügyelt Tanulás: tanuljuk meg az input-ból output-ba menő leképezést, $x \mapsto y$

Input	Output
Pixels: 	"lion"
Audio: 	"see at tuhl res taur aun ts"
<query, doc>	P(click on doc)
"Hello, how are you?"	"Bonjour, comment allez-vous?"
Pixels: 	"A close up of a small child holding a stuffed animal"

Véges számú tanító példapárból: $(x_n, y_n), n = 1, \dots, N$.

Úgy, hogy a leképezés pontos legyen nem látott párokra is!



Felügyelt Tanulás

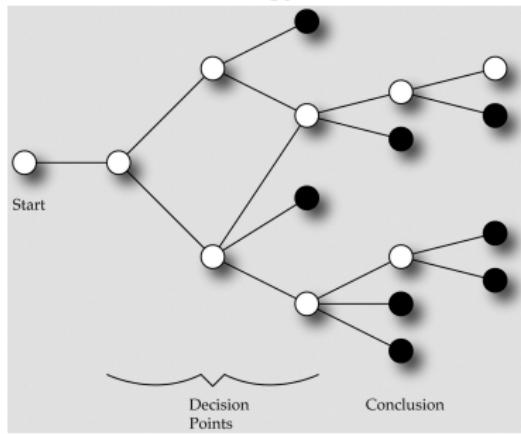
Szemléletesen

- Első gondolat: csinálunk leképezést elágazással, azaz küszöböljünk le egy változót:

if feature>threshold then .. else ..

if $\varphi(x) > t$ then return \hat{y}_1 else return \hat{y}_2

- Egymásba ágyazott elágazások több változóval: esetek száma exponenciálisan nő, nagyon nehéz kézzel megtervezni és leprogramozni.



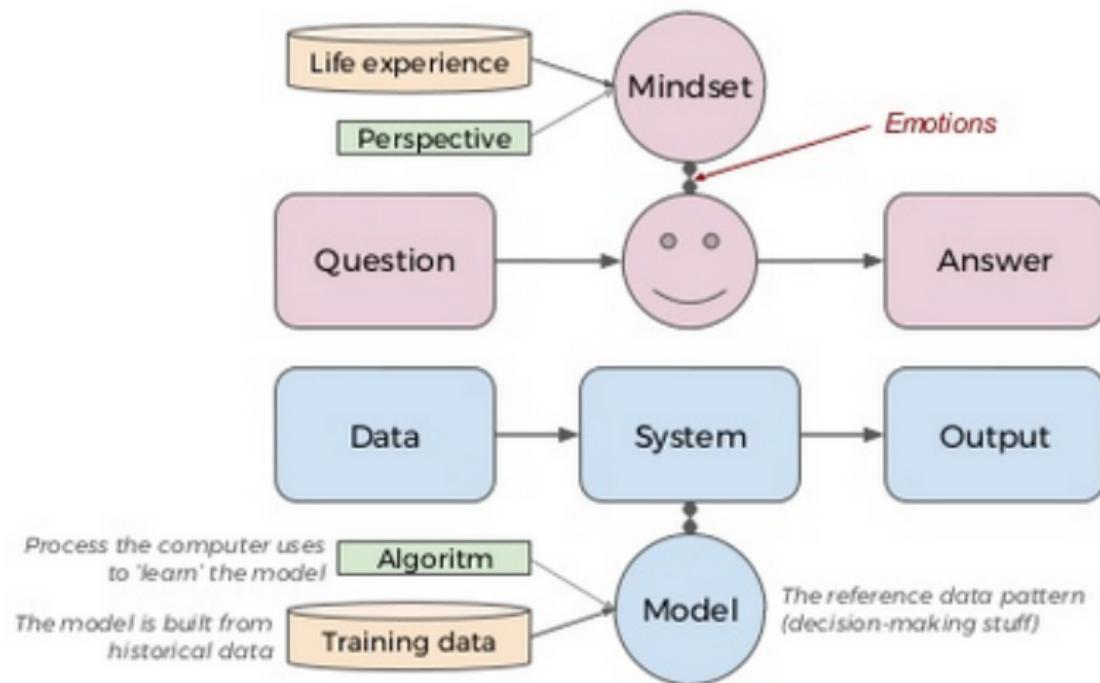
Her: I develop artificial intelligence for chatbots

Me: [trying to think of something to impress her] I write nested if blocks too



- Megoldás: tanuljuk meg a $\varphi(x)$ változókat és a t küszöböket az adatból!
 - Emlékeztetőül: ML egy szoftver, ami önmagát írja...

SIMILAR TO HOW WE LEARN



Felügyelt Tanulás

Szemléletesen

Algorithm: Learn Something

Input: Data, Mental Model

Output: Updated Mental Model

1 while *Mental Model Makes Bad Predictions* do

2 make a guess at answer

3 measure error

4 if *error is acceptably small* then

5 break

6 else

7 propose model adjustment

8 Mental Model \leftarrow Mental Model + adjustment

Algorithm: Gradient Descent

Input: Y, Θ, X, α , tolerance, max iterations

Output: Θ

1 for $i = 0; i < \text{max iterations}; i++$ do

2 current cost = $\text{Cost}(Y, X, \Theta)$

3 if *current cost < tolerance* then

4 break

5 else

6 gradient = $\text{Gradient}(Y, X, \Theta)$

7 $\theta_j \leftarrow \theta_j - \alpha \cdot \text{gradient}$



Felügyelt Tanulás

Formálisan: Optimalizációs Feladat

- Adottak az $(x_n, y_n), n = 1, \dots, N$ tanítópárok, keressük az optimális θ^* paraméterbeállítást az $f(\theta, \cdot)$ paraméteres leképezéshez úgy, hogy $f(\theta^*, x_n) \approx y_n$ (közelítsük az $x_n \mapsto y_n$ leképezést):

$$\min_{\theta} L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N \ell\left(\underbrace{f(\theta, x_n)}_{\hat{y}_n}, y_n\right),$$

ahol $\ell(\hat{y}_n, y_n)$ a hibafüggvény és θ a paramétervektor.

Becslés ($f(\theta^*, x_n)$ kiszámítása adott θ^* -ra) nagyon gyors!

Tanítás (minimalizáló θ^* megkeresése) nagyon lassú!

Általában: nemkonvex optimalizálási feladat, NP-nehéz a θ^* globális optimumot megtalálni.

- Jó hírek: működik, de csak ha...

- ... N elég nagy! Az y_n -ek összegyűjtése drága humán munka... (UL?),
- ... $\ell(\hat{y}_n, y_n)$, $f(\theta, x_n)$ és az optimalizálási módszer megfelelően vannak megválasztva! Nehéz humán munka, sok kísérlet... (UL?),
- ... θ közel van θ^* -hoz! Egyelőre nem tudjuk bizonyítani, de megy, így feltehetően igaz... (θ^* egyébként is túl mohó és túltanulásra vezet...).



Felügyelt Tanulás

Formálisan: Hibafüggvény

Hibafüggvény: $\ell: \mathbb{R}^m \times \mathbb{R}^m \rightarrow \mathbb{R}$

- Felügyelt:

- Regresszió ($y \in \mathbb{R}^m$):

négyzetes hiba: $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_2^2$

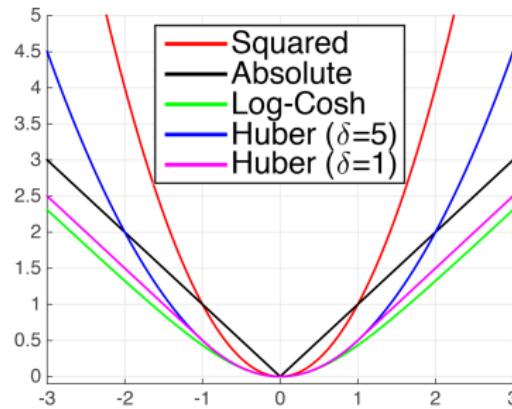
abszolút hiba: $\ell(\hat{y}_n, y_n) = \|\hat{y}_n - y_n\|_1$

- Osztályozás ($y \in \{0, 1\}^m$):

kategorikus kereszt-entrópia: $\ell(\hat{y}_n, y_n) = - \sum_{i=1}^m y_{n,i} \log \hat{y}_{n,i}$

- Felügyeletlen: amikor y_n -t nem adja meg az ember, jobb híján legyen $y_n = x_n$

- $\ell(\hat{x}_n, x_n) = \|\hat{x}_n - x_n\|_2^2$



Felügyelt Tanulás

Formálisan: Paraméteres leképezés és Optimalizálási módszer

Paraméteres leképezés: hármat fogunk látni hamarosan.

- K Legközelebbi Szomszédok: részletesen **ma**,
- Véletlen Erdők (Döntési Fák): részletesen **ma**,
- Mély Neurális Hálózatok: részletesen **jövő héten**.

Optimalizálási módszer: paraméteres leképezésenként más, annak jellegéből adódik (pl. ha diszkrét, akkor mohó; ha folytonos, akkor gradiens-módszer). De általában így néz ki a tanítás:

```
def train_and_val(hyp, train_set, val_set):  
    model = build_model(hyp)  
    for _ in range(300):  
        train_losses.append(model.train(train_set))  
        val_losses.append(model.test(val_set))  
        plot_learning_curves(train_losses, val_losses)  
        if val_losses[-1] < best_val_loss:  
            best_val_loss = val_losses[-1]  
            best_weights = weights  
    return best_val_loss, best_weights
```

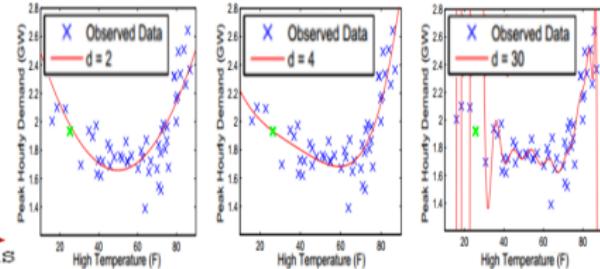
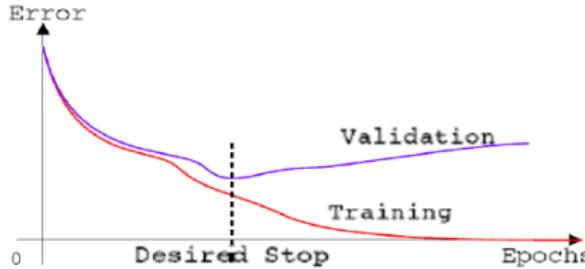


Felügyelt Tanulás

Túltanulás

A betanított leképezés nem általánosít jó! Mohó módon csak az $(x_n, y_n), n = 1, \dots, N$ párokra optimalizáltunk, megtanult olyat is, ami csak ezekre igaz, általában viszont nem! Ez a **túltanulás**.

- Korai Leállás: Monitorozzuk a hibát egy tanító halmaztól független validációs halmazon (bal: lila görbe, jobb: zöld x), álljunk le amikor nőni kezd.



- Regularizáció: Adjunk hozzá $\lambda \|\theta\|_2^2$ -t $L(\theta, x_n, y_n)$ -hoz, hogy egyszerű modellt kényszerítsünk (Occam borotvája: egyszerűbb modell jobban általánosít).
- Zajtalanítás: Zajosítuk a mintát tanításkor (N nő), a leképezésnek meg kell tanulnia a zajt kiküszöbölni.
- Augmentáció: Új tanító példák generálása (N nő), pl. képeknél kivágás, tükrözés, forgatás, színcsere (ami nem változtatja y -t).
- Finomhangolás: Indítsuk θ -t nagy adatbázison optimalizált θ^* -ból.

Felügyelt Tanulás

Hiperparaméterek

Az ML-ben sok hiperparaméter van: leképezés, hibafüggvény, optimalizációs módszer kiválasztása, regularizáció λ -ja, zajszint, stb. Hogyan állítsuk be ezeket?

- Másoljuk le másnak a beállításait. Nem lesz a mi feladatunkon is optimális.
- Csinálunk Véletlen Keresést vagy Bayes-Optimalizációt*: tanítsunk sok hiperparaméter-beállítással a tanító halmazon, mohón válasszuk a legjobbat a validációs halmazon (sok lila görbe közül a legalsó). Sajnos ez többszörözi a számítási igényt, de egyelőre nem tudunk jobbat...

* BO is felügyelt: x = hiperparaméter, y = validációs halmazon vett hiba.

```
train_set , val_set, test_set = load_dataset()
for _ in range(100):
    hyps.append(pick_hyperparams(hyps, val_losses))
    val_losses.append(train_and_val(hyps[-1], train_set,
val_set))
    if val_losses[-1] < best_val_loss:
        best_val_loss = val_losses[-1]
        best_hyp = hyps[-1]
train_and_val(best_hyp, train_set + val_set, test_set)
```



K Legközelebbi Szomszédok

Paraméteres leképezés és Optimalizálási módszer

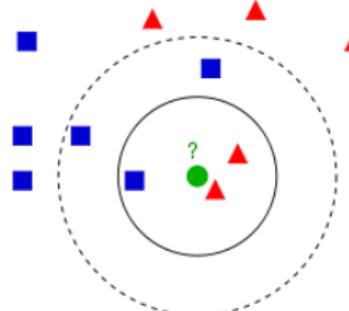
- **K Legközelebbi Szomszédok (KNN):** (Nem-)paraméteres leképezés, fix K -ra becsüljünk a K legközelebbi x_n -ekhez tartozó y -ok átlagával:

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\frac{\mathbb{I}(x_n \text{ benne van } x \text{ K legközelebbi szomszédjában})}{K}}_{w_n} y_n.$$

Speciálisan $\theta = \{(x_n, y_n) \mid n = 1, \dots, N\}$, nem kell optimalizálni (de lehetne).

Le kell tárolni az összes tanítópárt, nem kompresszálja x_n -ekben rejlő információt. Legközelebbi szomszédok: $\text{argsort}_n \|x_n - x\|_2^2$.

Előny: Egyszerű leprogramozni. Hátrány: Ha N nagy, tárolni és sorbarendezni erőforrásigényes.



K Legközelebbi Szomszédok

Szoftver

Kód elérhető az sklearn csomagban (Python):
sklearn.neighbors.KNeighborsClassifier

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[ 0.66666667  0.33333333]]
```

sklearn.neighbors.KNeighborsRegressor

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsRegressor
>>> neigh = KNeighborsRegressor(n_neighbors=2)
>>> neigh.fit(X, y)
KNeighborsRegressor(...)
>>> print(neigh.predict([[1.5]]))
[ 0.5]
```



Véletlen Erdők

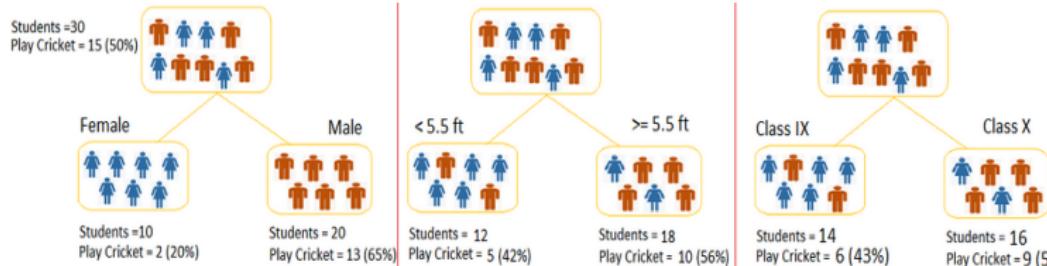
Paraméteres leképezés és Optimalizálási módszer

- Döntési Fa (DT): Paraméteres leképezés x komponenseinek (változók) sorozata, amik a tanító halmaz legjobb vágásait adják y szerint minden lépésben (Kereszt-entrópia vagy Négyzetes hiba szerint). Gyakorlatilag if-then-else fastruktúrában, paraméterek a változók és a küszöök.

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\frac{\mathbb{I}(x_n \text{ az } x \text{ levelében levő } K' \text{ input egyike})}{K \cdot K'}}_{w_n} y_n.$$

Optimalizálás: faépítés mohó algoritmussal a legjobb vágásokra.

Pl. krikettezésre egy szinten 3 lehetséges vágás közül a Nő/Férfi a legjobb:



Előny: értelmezhető, olcsó használat, a párokat nem kell tárolni. Hátrány: faépítés NP-teljes, a mohó lokálisan optimális, de globálisan nem optimális.

Véletlen Erdők

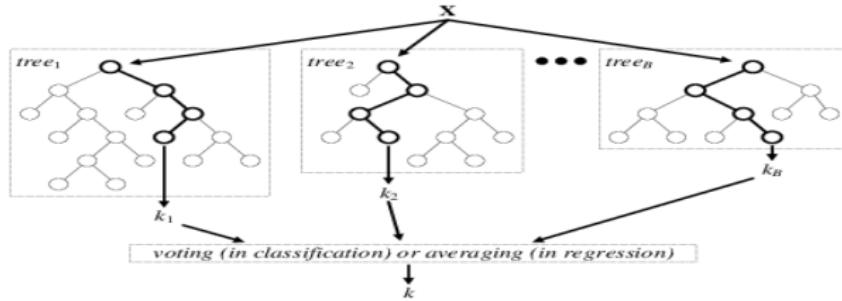
Paraméteres leképezés és Optimalizálási módszer

- **Véletlen Erdő (RF)**: Paraméteres leképezés K db DT, minden egyik véletlen változó- és példa-részhalmazzal, ezek eredményeinek átlaga a végső becslés:

$$f(\theta, x) = \sum_{n=1}^N \underbrace{\sum_{k=1}^K \frac{\mathbb{I}(x_n \text{ az } x \text{ levelében levő } K'_k \text{ input egyike } DT_k\text{-ban})}{K \cdot K'_k}}_{w_n} y_n.$$

A véletlen generálás (zaj) elkerüli a túltanulást, kevésbé mohó, viszont kevésbé értelmezhető.

Előny: **párhuzamos, olcsó** használat. Hátrány: szintén **NP-teljes**.



Véletlen Erdők

Szoftver

Kód elérhető az sklearn csomagban (Python):
sklearn.ensemble.RandomForestClassifier

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>>
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                             n_informative=2, n_redundant=0,
...                             random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=2, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=0, verbose=0, warm_start=False)
>>> print(clf.feature_importances_)
[ 0.17287856  0.806008704  0.01884792  0.00218648]
>>> print(clf.predict([[0, 0, 0, 0]]))
[1]
```

sklearn.ensemble.RandomForestRegressor

```
>>> from sklearn.ensemble import RandomForestRegressor
>>> from sklearn.datasets import make_regression
>>>
>>> X, y = make_regression(n_features=4, n_informative=2,
...                         random_state=0, shuffle=False)
>>> regressor = RandomForestRegressor(max_depth=2, random_state=0)
>>> regressor.fit(X, y)
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                      oob_score=False, random_state=0, verbose=0, warm_start=False)
>>> print(regressor.feature_importances_)
[ 0.17339552  0.81594114  0.          0.01066333]
>>> print(regressor.predict([[0, 0, 0, 0]]))
[-2.50699856]
```

Gépi Tanulás Előadás 2

Felügyelt Tanulás: Mesterséges Mély Neurális Hálózatok

Milacski Zoltán Ádám¹

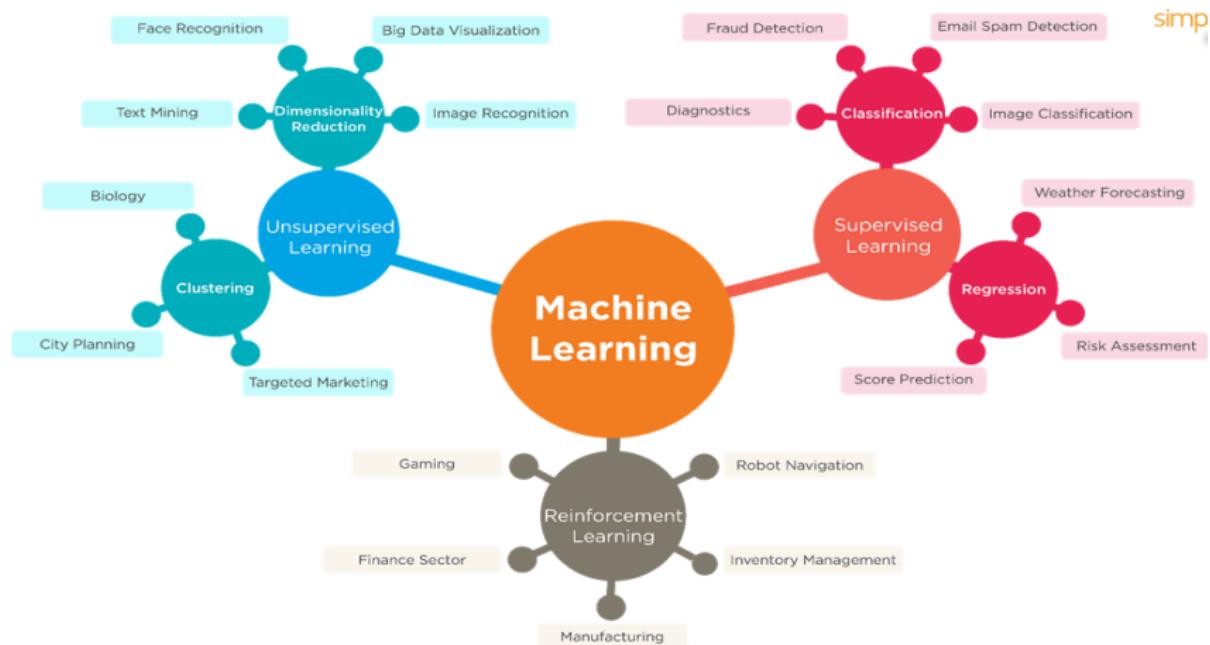
¹Eötvös Loránd Tudományegyetem
Programozáselmélet és Szoftvertechnológiai Tanszék
srph25@gmail.com

2018. május 3.



Gépi Tanulás

Felügyelt, Felügyeletlen és Megerősítéses



- Múlt órán: Felügyelt, KNN, RF;
- Ma: Felügyelt, Mesterséges Mély Neurális Hálózatok (ANN, DNN);
- Jövő órán: Felügyeletlen, Megerősítéses.



Felügyelt Tanulás

Formálisan: Optimalizációs Feladat

- Adottak az $(x_n, y_n), n = 1, \dots, N$ tanítópárok, keressük az optimális θ^* paraméterbeállítást az $f(\theta, \cdot)$ paraméteres leképezéshez úgy, hogy $f(\theta^*, x_n) \approx y_n$ (közelítsük az $x_n \mapsto y_n$ leképezést):

$$\min_{\theta} L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N \ell\left(\underbrace{f(\theta, x_n)}_{\hat{y}_n}, y_n\right),$$

ahol $\ell(\hat{y}_n, y_n)$ a hibafüggvény és θ a paramétervektor.

Becslés ($f(\theta^*, x_n)$ kiszámítása adott θ^* -ra) nagyon gyors!

Tanítás (minimalizáló θ^* megkeresése) nagyon lassú!

Általában: nemkonvex optimalizálási feladat, NP-nehéz a θ^* globális optimumot megtalálni.

- Jó hírek: működik, de csak ha...

- ... N elég nagy! Az y_n -ek összegyűjtése drága humán munka... (UL?),
- ... $\ell(\hat{y}_n, y_n)$, $f(\theta, x_n)$ és az optimalizálási módszer megfelelően vannak megválasztva! Nehéz humán munka, sok kísérlet... (UL?),
- ... θ közel van θ^* -hoz! Egyelőre nem tudjuk bizonyítani, de megy, így feltehetően igaz... (θ^* egyébként is túl mohó és túltanulásra vezet...).



Paraméteres leképezés és Optimalizálási módszer

Összehasonlítás

Paraméteres leképezés és Optimalizálási módszer szorosan összetartozik.

Összehasonlítás:

Módszer	θ	$f(\theta, x_n)$	\min_{θ}	Megjegyzés
KNN	$(x_n, y_n),$ $n = 1, \dots, N$	$\sum_{n=1}^N w_n^{KNN} y_n$	nincs, θ^* ismert	nem tömörít, lassú, glob. opt. θ^* , K -ra érzékeny
RF	legj. vágó vált., t küszöbök	$\sum_{n=1}^N w_n^{RF} y_n$	mohó, faépítés	tömörít, gyors, lok. opt. θ^*
DNN	w_j súlyok	$h(\sum_{j=1}^J w_j x_{n,j} + b)$ többsz. össz. fv.	gradiens- módszer	tömörít, gyors, glob. opt. θ^*



Mesterséges Mély Neurális Hálózatok

- Mesterséges Mély Neurális Hálózat (DNN): Összetett függvény alakú paraméteres leképezés:

$$f(\theta, x_n) = g_K(\theta_K, g_{K-1}(\dots g_2(\theta_2, g_1(\theta_1, x_n)) \dots))$$

ahol $\theta = \{\theta_1, \dots, \theta_K\}$ a parameterek (súlyok). Nemkonvex, mert θ_i és θ_j ($i \neq j$) szorzata megjelenik $f(\theta, x_n)$ -ben.

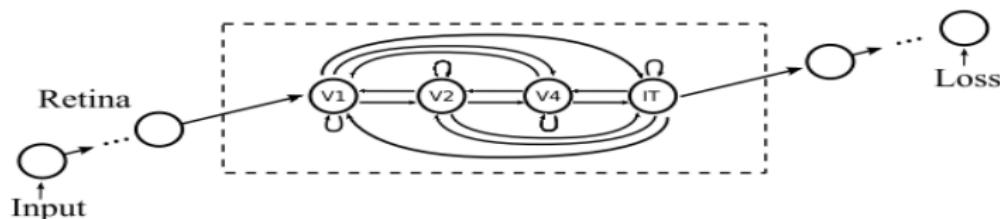
Réteg: $x_n^{(k)} = g_k(\theta_k, x_n^{(k-1)})$

Hálózat (előreterjesztés): $f(\theta, x_n)$.

A rétegek alacsonyabb szintű leíró változókat kombinálnak össze magasabb szintűekké az összetett függvény alak miatt.

A főemlősök látórendszerét utánozza:

Pre-net



Post-net

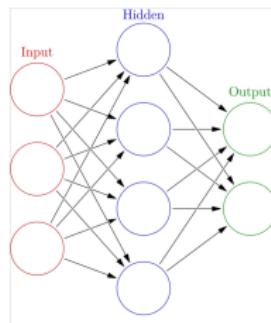


Mesterséges Mély Neurális Hálózatok

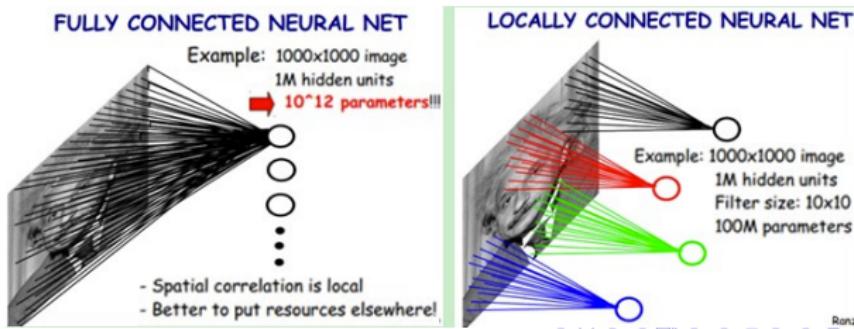
Rétegek és Nemlineáritások

- Rétegek:

- Sűrű (Teljes Konnektivitású, 2D vektorok): $x_n^{(k)} = h_k(W_k x_n^{(k-1)} + b_k)$



- Konvolúciós (Lokális Konnektivitású, 4D képek): $x_n^{(k)} = h_k(W_k * x_n^{(k-1)} + b_k)$

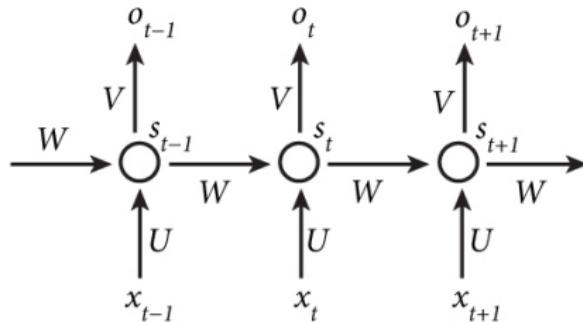


Mesterséges Mély Neurális Hálózatok

Rétegek és Nemlinearitások

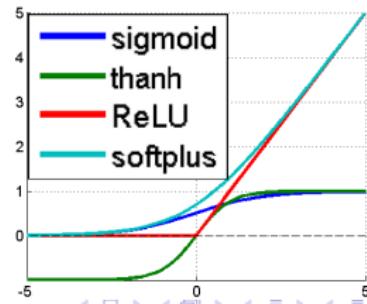
- Rétegek:

- Rekurrens (3D idősorok): $x_{n,t}^{(k)} = h_k(W_k^{hh}x_{n,t-1}^{(k)} + W_k^{xh}x_{n,t}^{(k-1)} + b_k)$



- Nemlineáritások: h_k elemenkénti nemlineáris függvény a rétegekben.

- sigmoid: $h_k(z) = \frac{1}{1+e^{-z}}$,
- tanh: $h_k(z) = \tanh(z)$,
- relu: $h_k(z) = \max(0, z)$,
- softmax: $h_k(z)_j = \frac{e^{z_j}}{\sum_{i=1}^l e^{z_i}}$.



Mesterséges Mély Neurális Hálózatok

Optimalizálási Módszer: Gradiens-módszer

Keressük θ^* -ot! $f(\theta, x_n)$ deriválható θ szerint, így $L(\theta, x_n, y_n)$ is.

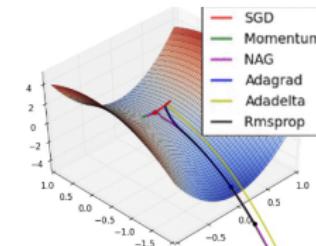
- Sztochasztikus Gradiens-módszer (SGD): Negatív gradiens a lokális optimum fele mutat, lépjünk pici ebbe az irányba! Legyen $\theta_0 \sim \mathcal{N}(0, 0.001^2)$, ezután:

$$\theta_{I+1} := \theta_I - \alpha \frac{\partial L(\theta, x_n, y_n)}{\partial \theta} \Big|_{\theta=\theta_I}.$$

Tanulási ráta (lépésköz): α , pici szám, pl. 0.001.

Visszaterjesztés: $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$, automatikus gépi deriválással (láncszabály...).

Minibatch: (x_n, y_n) párok véletlen részhalmaza minden lépésben.



Elakadhat nyeregpontokban!

- Kvázi-Newton módszerek: ki tudnak mozdulni nyeregpontokból adaptív tanulási rátákkal, pl. RMSProp, Adadelta, Adam.

Mesterséges Mély Neurális Hálózatok

Szoftvereszközök

Használunk **GPU**-t gyorsasághoz: a tenzorműveletek zavarbaejtően párhuzamosak. SGD megfelelő kevés memóriához.

Probléma: GPU programozás túl alacsony szintű (lassú és hibákkal teli fejlesztés)...

Megoldás: kódolunk magas szinten (pl. Python-ban) és fordítsuk le alacsony szintű GPU kódra!

Ehhez speciális szoftvereszközök szükségesek:

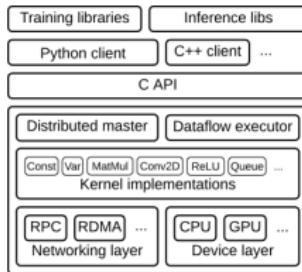
- CUDA, OpenCL: Alacsony szintű, GPU kód
- **Tensorflow**, Theano, CNTK, PyTorch: Közepes szintű (Back-end), szimbolikus előreterjesztés, automatikus szimbolikus deriválás, fordítás GPU kódra és meghívás konkrét numerikus értékekkel
- **Keras**: Magas szintű (Front-end), nemlineáritások, rétegek, hibafüggvények, gradiens-módszerek
- OpenAI Gym: Megerősítéses Tanulás framework
- Hyperopt: Hiperparaméter keresés
- Sacred: Kísérletek logolása és reprodukciója
- Elasticcluster: Elosztott számítások felhőben



Mesterséges Mély Neurális Hálózatok

GPU programozás Tensorflow-ban

- Fejezzük ki algoritmusunkat szimbolikus formában, **számítási gráf** felépítésével
- Építés fázis:
 - Tensor: típusos többdimenziós tömb (statikus típus, dimenzió, méret).
 - Operation (op): kap nulla vagy néhány tenzort, számol velük, majd visszaad nulla vagy néhány tenzort (szimbolikus gradiense implementálva van).
 - Variable: állapotok tárolása több hívás (végrehajtás) között (tf.assign op).
 - Placeholder: bemenetek, kijelölik a 'feed' műveleteket.
- Végrehajtási fázis:
 - Session: műveleteket eszközre (CPU vagy GPU) helyezi, metódusokat ad a végrehajtásukhoz, tenzorokat ad vissza numpy ndarray-ként.
 - Fetch: műveletek kimeneteinek kinyerése, gráf végrehajtása.
 - Feed: művelet kimeneteinek lecserélése konkrét tenzor értékre.



```
input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)
output = tf.mul(input1, input2)

with tf.Session() as sess:
    print(sess.run([output], feed_dict={input1:[7.], input2:[2.]}))

# output:
# [array([ 14.], dtype=float32)]
```



Mesterséges Mély Neurális Hálózatok

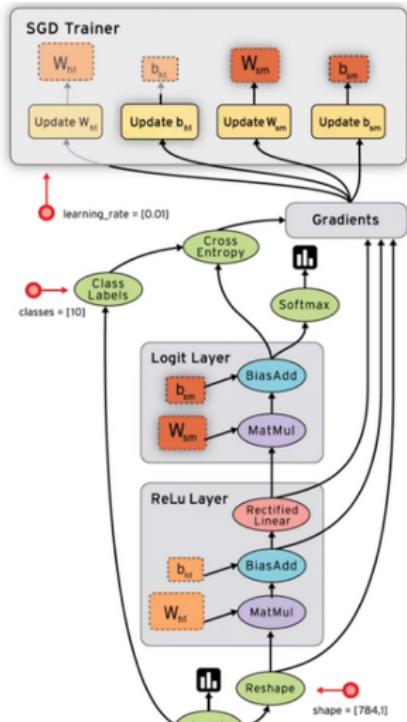
Mély Hálók implementációja Tensorflow-ban

Hogyan csinálunk Mesterséges Mély Neurális Hálózatot Tensorflow-ban?

Name	Math	Tensorflow
Bemenet, kimenet	$x_n, y_n, n = 1, \dots, N$	
Paraméter	θ	tf.placeholder
Előreterjesztés	$f(\theta, x_n)$	tf.Variable
Hibafüggvényion	$L(\theta, x_n, y_n) = \frac{1}{N} \sum_{n=1}^N l(f(\theta, x_n), y_n)$	tf.losses ops
Szimbolikus gradiens	$\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$	tf.gradients op
Inicializáció	θ_0	tf.random_normal_initializer
Gradiens-módszer	$\theta_{l+1} := \theta_l - \alpha \frac{\partial L(\theta, x_n, y_n)}{\partial \theta} \Big _{\theta=\theta_l}$	tf.train.Optimizer
Tanít, tesztel, becsül	$x_n := X_n, y_n := Y_n$	tf.Session.run(), fetch, feed

Előny: Az $\frac{\partial L(\theta, x_n, y_n)}{\partial \theta}$ szimbolikus gradiens automatikusan számolható a tf.gradients op által (láncszabály többszöri ismétlésével a számítási gráfon, ahol minden op-nak ismert a gradiense; ezt rendkívül nehéz lenne papíron levezetni). Ez nagyban egyszerűsíti a kísérletezgetést (csak az Előreterjesztést kell cserélgetni, ami könnyű!).

Hátrány: Nehéz debug-olni.



Mesterséges Mély Neurális Hálózatok

Mély Hálók implementációja Keras-ban

Probléma: Tensorflow-ban rendre ugyanazokat a számítási részgráfokat (rétegek, nemlineáritások) kell megadni, újra implementálhatni őket felesleges. Ezek magasabb absztrakciós szintet képviselnek.

Megoldás: Keras magasabb szinten rendszerezi őket objektum-orientáltsággal (osztályok és öröklődés)!

- Könnyű és gyors prototípus gyártás (felhasználóbarát, moduláris, bővíthető).
- Sok beépített réteg osztály, amik kombinálhatóak is.
- Használhat TensorFlow-t, CNTK-t vagy Theano-t a háttérben (keras.backend).
- Csak Python (nincs szükség egyéb konfigurációs fájlokra, így bővíthető).
- Egyszerű keras.models.Model metódusok tanításra, tesztelésre, becslésre: fit(), evaluate(), predict().

```
from keras.models import Sequential
model = Sequential()
from keras.layers import Dense, Activation
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn API.
model.fit(x_train, y_train, epochs=5, batch_size=32)
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)
classes = model.predict(x_test, batch_size=128)
```



Mesterséges Mély Neurális Hálózatok

Megoldatlan Kérdések



Algorithm 1: The layered thresholding algorithm.

Assuming two layers for simplicity, Algorithm 1 can be summarized in the following equation

$$\mathbf{f}_2 = \mathcal{P}_{\beta_2} \left(\mathbf{D}_2^T \mathcal{P}_{\beta_1} \left(\mathbf{D}_1^T \mathbf{X} \right) \right).$$

Comparing the above with Equation (1), given by

$$f(\mathbf{X}, \{\mathbf{W}_i\}_{i=1}^2, \{\mathbf{b}_i\}_{i=1}^2) = \text{ReLU} \left(\mathbf{W}_2^T \cdot \text{ReLU} \left(\mathbf{W}_1^T \mathbf{X} + \mathbf{b}_1 \right) + \mathbf{b}_2 \right),$$

one can notice a striking similarity between the two.

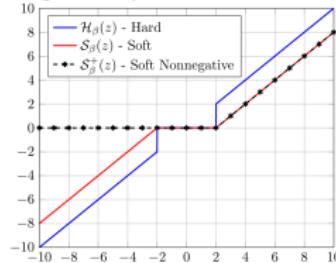


Figure 3: The thresholding operators for a constant $\beta = 2$.

- **Tételek:** ekvivalencia konvolúciós ReLU hálók és ritka reprezentáció között (utóbbira sok bizonyított téTEL van, pl. globálisan optimális θ^* -ra). De lesz több téTEL is hamarosan... .
- Jobb rétegek: Kapszulák (Hinton) magasabb rendű invariáns változókkal.
- Jobb fejlesztői eszközök: imperatív programozás, debug-olás... .

Ajánlott Irodalom

- Online Kurzusok

- Vincent Vanhoucke kurzusa: Tensorflow
- Andrew Ng kurzusa: Gép Tanulás Bevezető
- NVIDIA kurzusa: Szoftvereszközök
- Geoffrey Hinton kurzusa: Elmélet
- Andrej Karpathy kurzusa: Konvolúciós Hálók
- Stephen Boyd kurzusa: Konvex Optimalizálás
- Georgia Tech kurzusa: Megerősítéses Tanulás

- Forráskódok

- Keras útmutató
- Keras példák
- Keras GitHub (haladó)

- Tudományos Cikkek

- Google Scholar (haladó): Yann LeCun, Joshua Bengio, Geoffrey Hinton, Ilya Sutskever, Christian Szegedy, Alex Krizhevsky, Andrew Ng, Quoc Le, Vincent Vanhoucke, Diederik Kingma



Alkalmazások

Youtube videók



Nem felügyelt tanulás

Pintér Balázs

2018-05-17

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Nem felügyelt tanulás

- Felügyelt tanulás: címkézett adatokból tanulunk valamilyen függvényt
 - Más megközelítések
 - 1 **Nem felügyelt tanulás**
 - 2 Semi-supervised learning
 - 3 Megerősítéses tanulás
 - 4 Evolúciós algoritmusok
 - 5 Neuroevolúció
- <http://www.youtube.com/watch?v=qv6UV0Q0F44>

Tartalom

1 Nem felügyelet tanulás

2 Klaszterezés

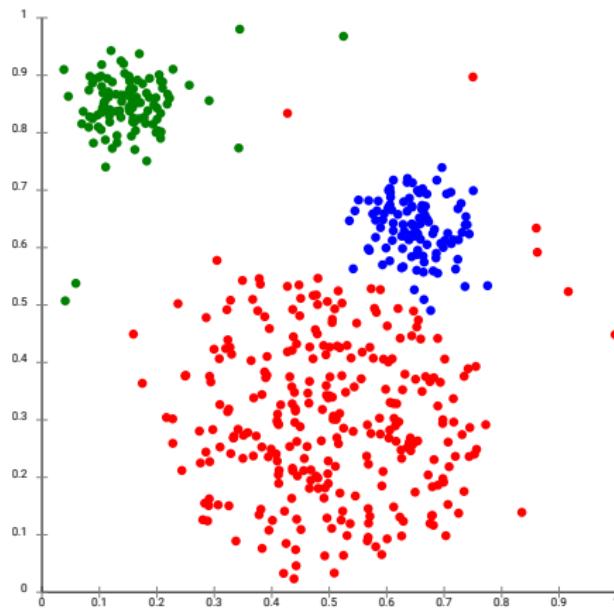
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

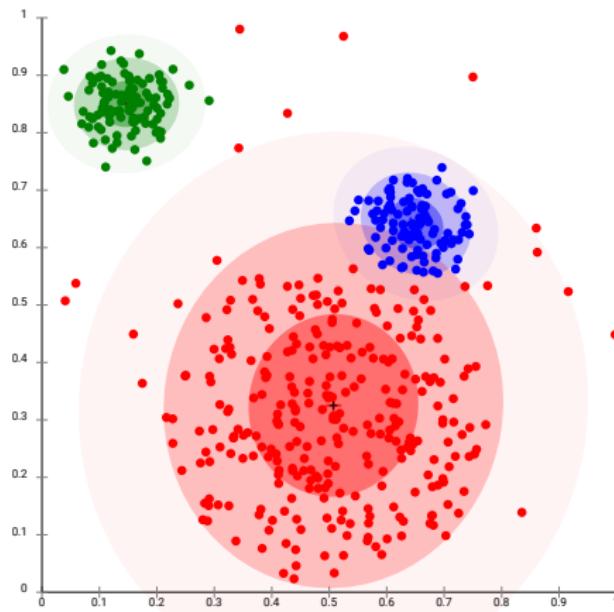
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Példa – sűrűség alapú klaszterezés



Példa – eloszlás alapú klaszterezés



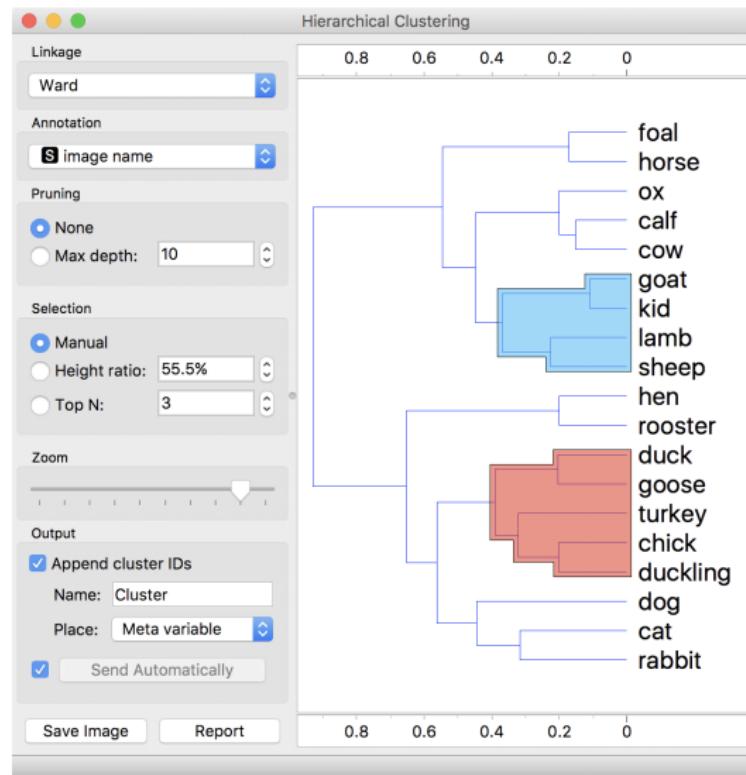
Feladat

- Úgy csoportosítunk dolgokat, hogy a hasonlóak egy csoportba kerüljenek
 - Klaszteren belül minél hasonlóbbak
 - Klaszterek között minél kevésbé hasonlóak
- A dolgok általában \mathbb{R}^n -beli (vagy gráfbeli) pontok, pl.:
 - Ügyféladatok piacssegmentáláshoz
 - Dokumentumok szózsákkal modellezve, téma-k meghatározásához, keresési találatok összegzésére
 - Szavak kontextusai, jelentések indukálásához
 - Szerverek adatai (melyikek aktívak általában együtt)
- Egy csoportot egy *klaszternek* hívunk

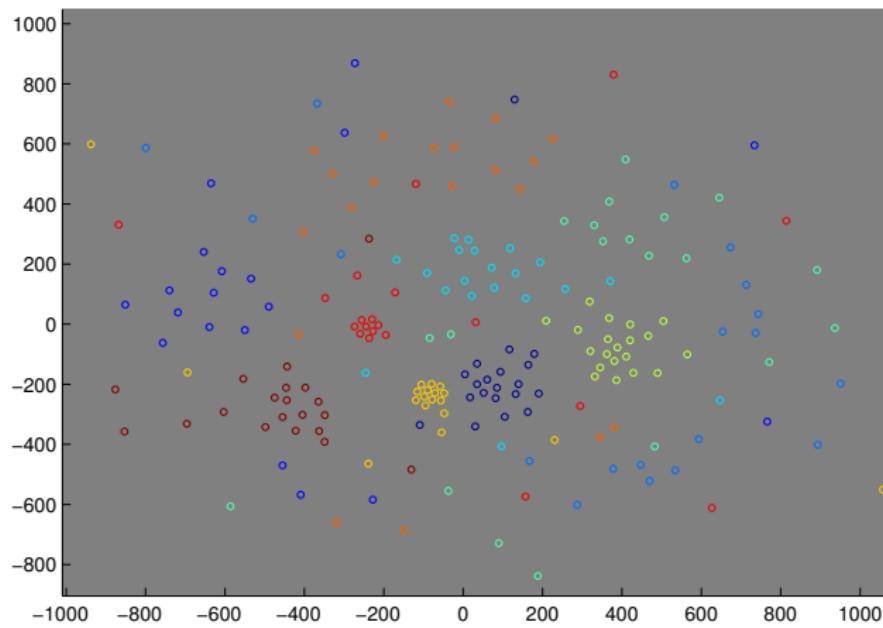
Fajtái

- Lehet hard vagy soft clustering
 - Hard clustering: egy adatpont csak egy klaszterben szerepelhet
 - Soft clustering: minden adatpontra megvan, hogy mennyire tartozik az egyes klaszterekbe
- Átmenetek
 - Átfedő klaszterezés: egy elem több klaszterbe is tartozhat, de vagy beletartozik, vagy nem
 - Hierarchikus klaszterezés: a klasztereket hierarchiába szervezzük, a gyerek klaszterbe tartozó elemek a szülőbe is beletartoznak

Hierarchikus klaszterezés



Példa természetes klasztereződésre – azonos értelmű szavak jelentései (t-SNE)



Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

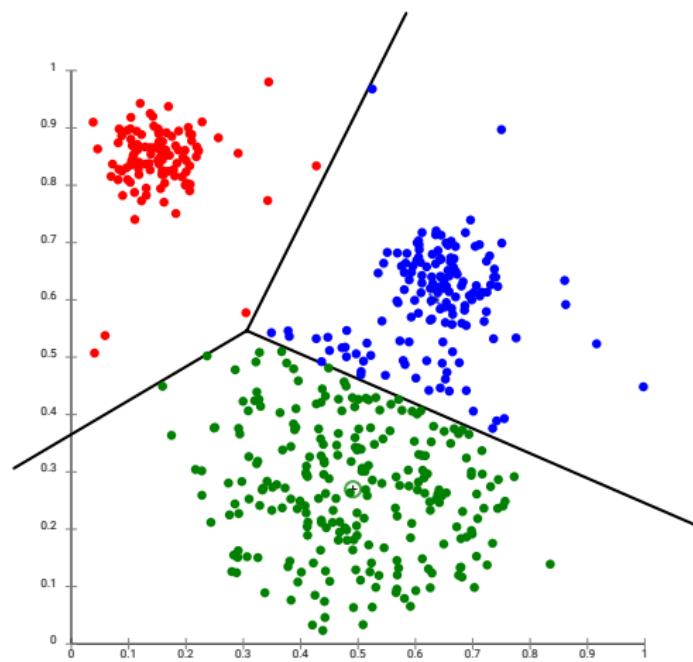
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Példa



Feladat

- Adott: k , a klaszterek száma
- minden klasztert a középpontjával reprezentálunk
 - Centroid, a klaszter pontjainak átlaga
- A feladat: keressük meg a k klaszter középpontot és az adatpontokat rendeljük ezekhez hozzá úgy, hogy a klaszteren belüli, középponttól számított távolságnégyzeteket minimalizáljuk
 - Ekvivalens a páronkénti távolságnégyzetek minimalizálásával
 - NP-nehéz, így approximáljuk
 - Csak lokális optimumot találunk
 - Többször futtathatjuk különböző véletlen inicializációkkal

k-means feladat

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x} \in S_i} \|\mathbf{x} - \boldsymbol{\mu}_i\|^2 = \arg \min_{\mathbf{S}} \sum_{i=1}^k \frac{1}{2|S_i|} \sum_{\mathbf{x}, \mathbf{y} \in S_i} \|\mathbf{x} - \mathbf{y}\|^2$$



Algoritmus

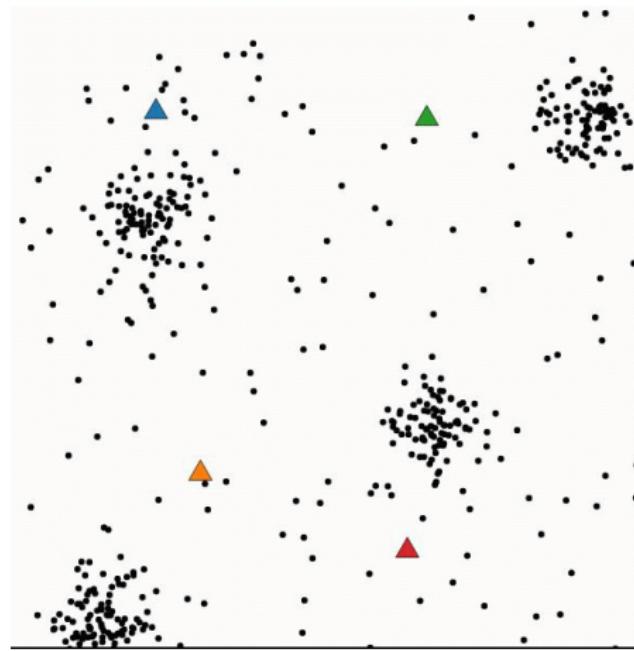
- Kezdetben adott k és az adatpontok \mathbb{R}^n -ben
- Inicializáljuk az $m_1^{(1)}, m_2^{(1)}, \dots, m_k^{(1)}$ centroidokat
 - Véletlenszerűen kiválasztunk k adatpontot, vagy
 - minden adatpontot véletlenszerűen egy klaszterbe sorolunk és kiszámoljuk a centroidokat
- Váltogatjuk a következő két lépést, amíg nem konvergálunk
 - 1 minden adatpontot a legközelebbi centroidhoz rendelünk:

$$S_i^{(t)} = \{x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k\}$$

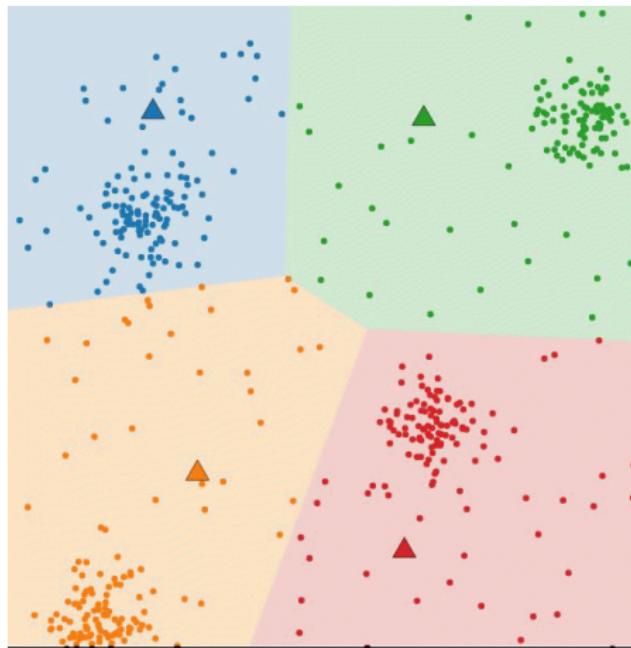
- 2 Kiszámítjuk az új centroidokat

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

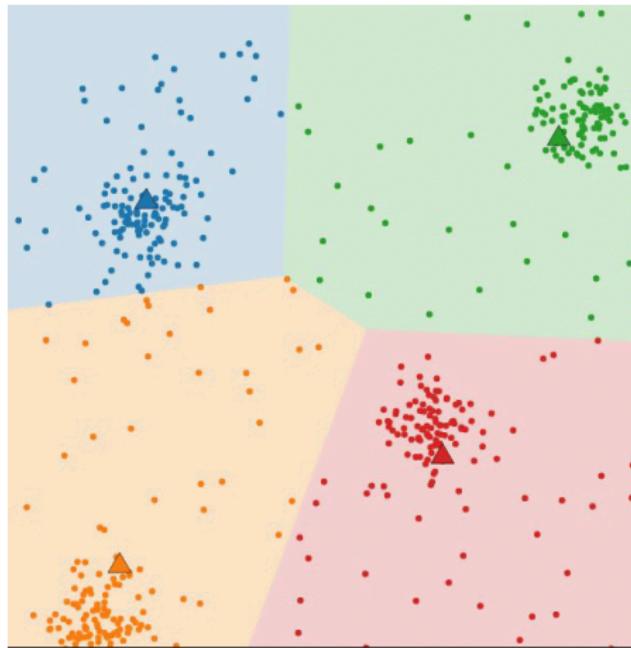
Algorithmus



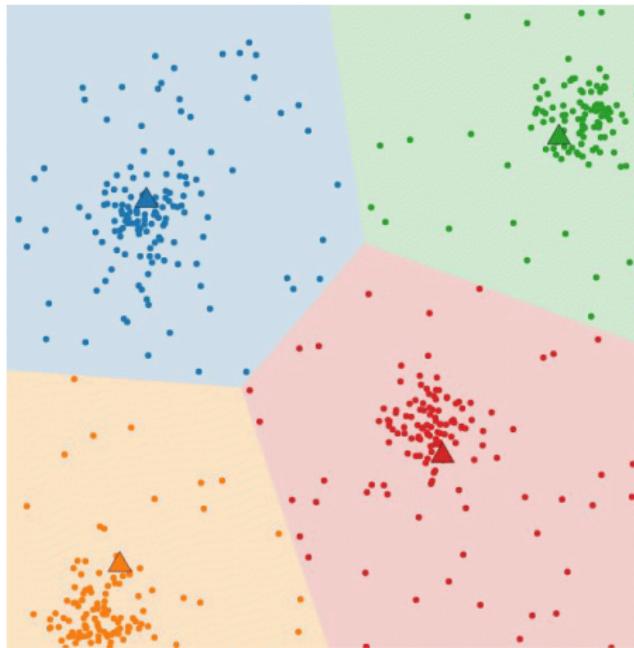
Algoritmus



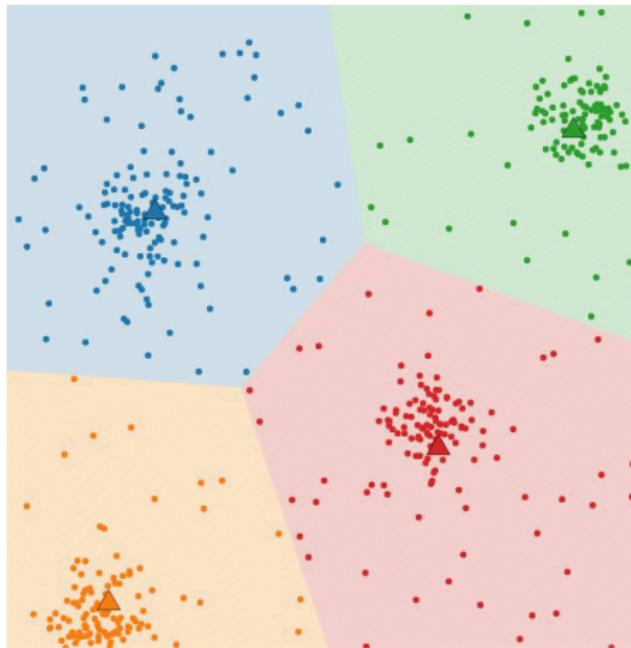
Algoritmus



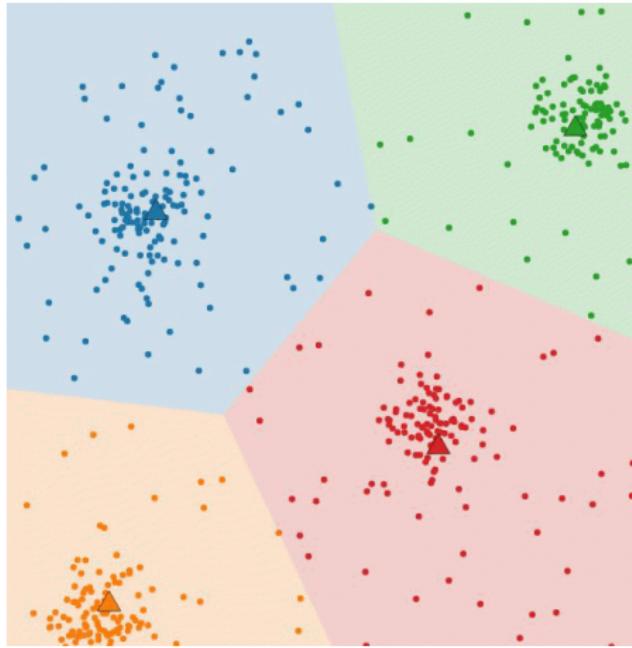
Algoritmus



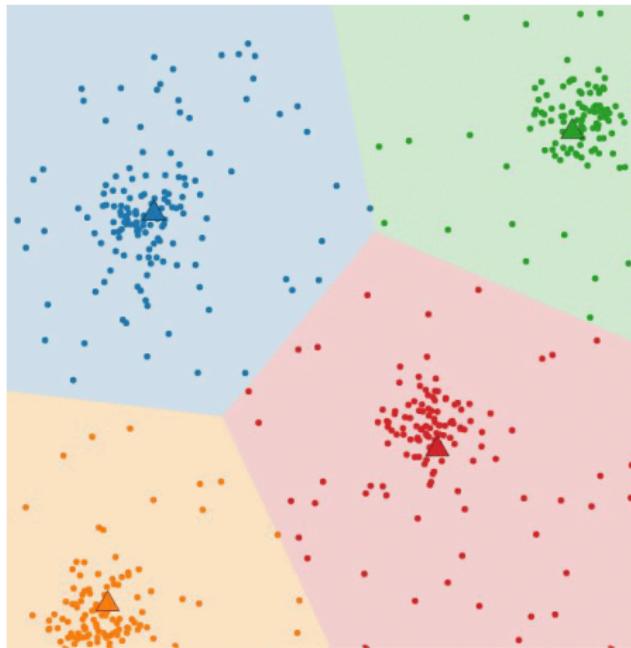
Algoritmus



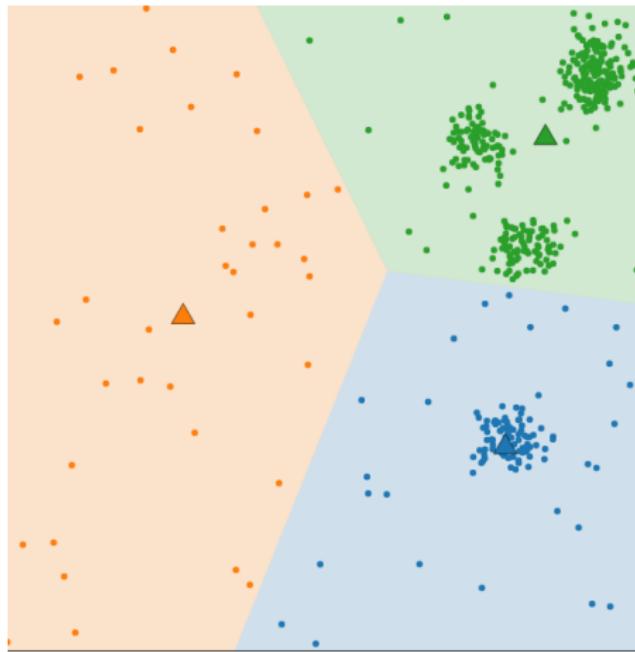
Algoritmus



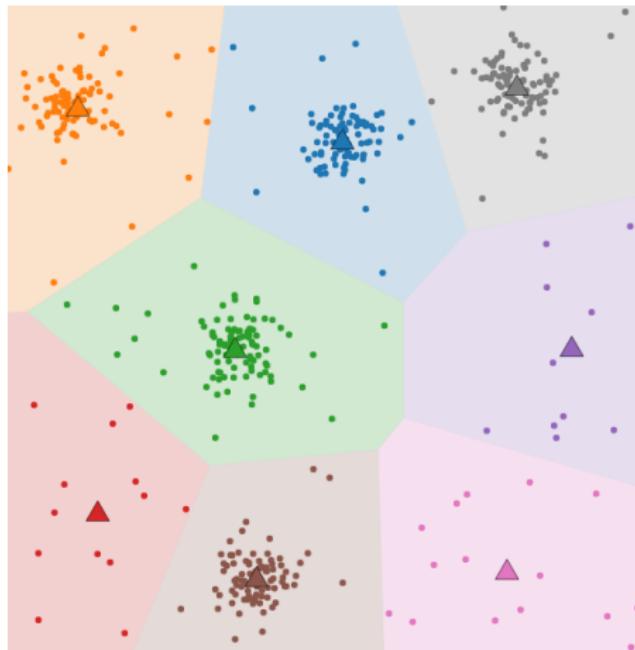
Algoritmus



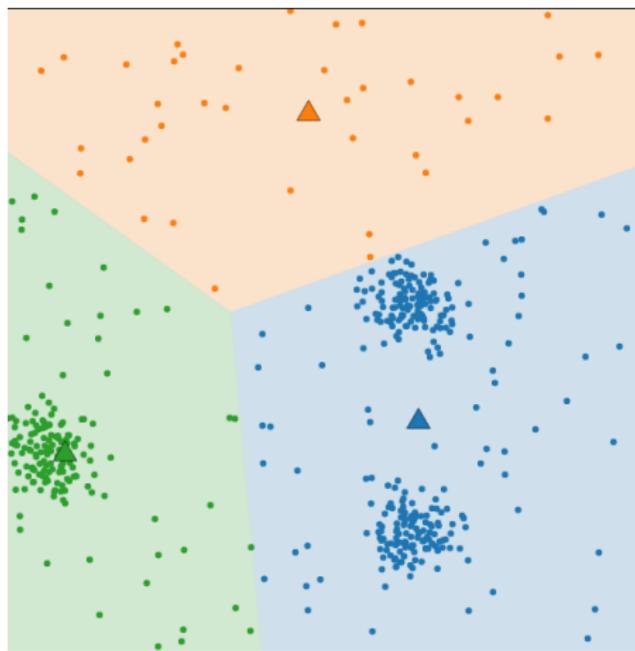
Problémák – túl kicsi k-t adunk meg



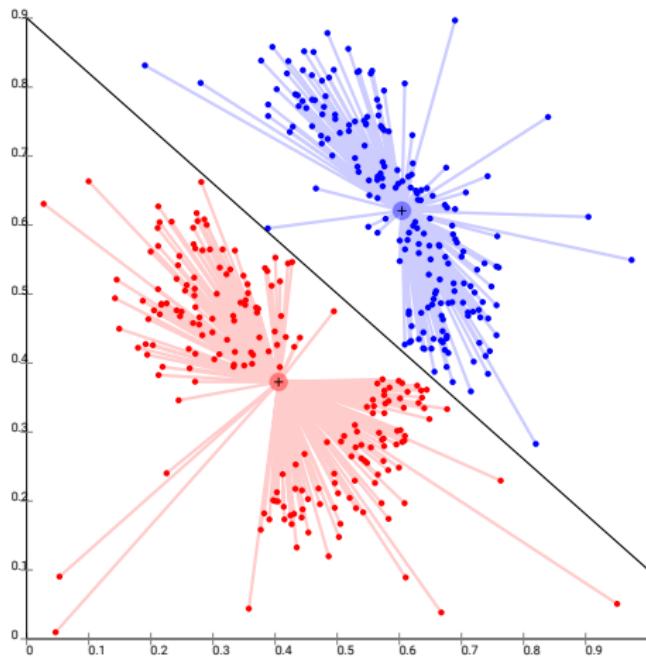
Problémák – túl nagy k-t adunk meg



Problémák – rossz inicializáció



Problémák – sűrűség alapúak a klaszterek



Python példák

- Kép kvantálás: http://scikit-learn.org/stable/auto_examples/cluster/plot_color_quantization.html
- Dokumentumok klaszterezése:
http://scikit-learn.org/stable/auto_examples/text/document_clustering.html

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

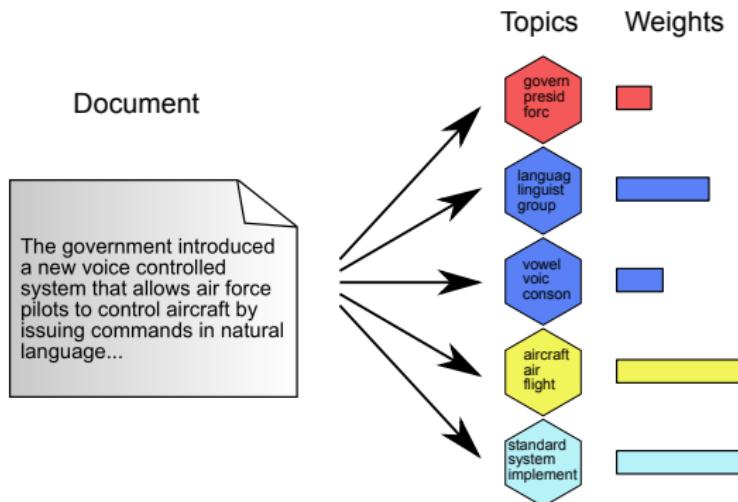
3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

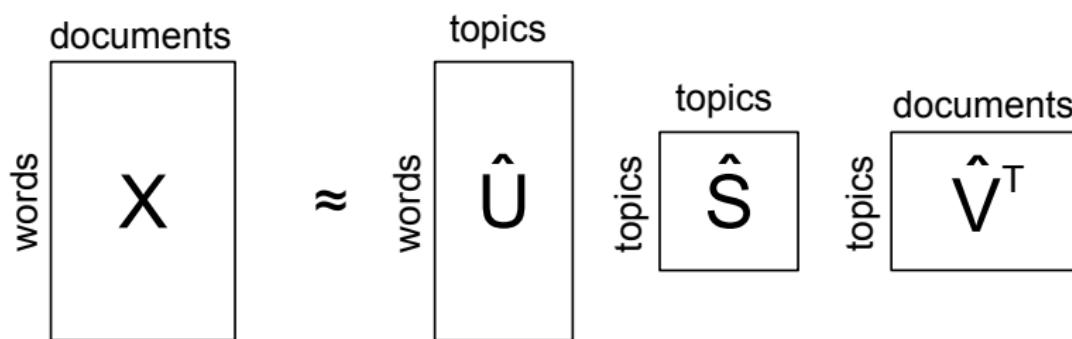
4 Autoenkóderek

Témamodellek

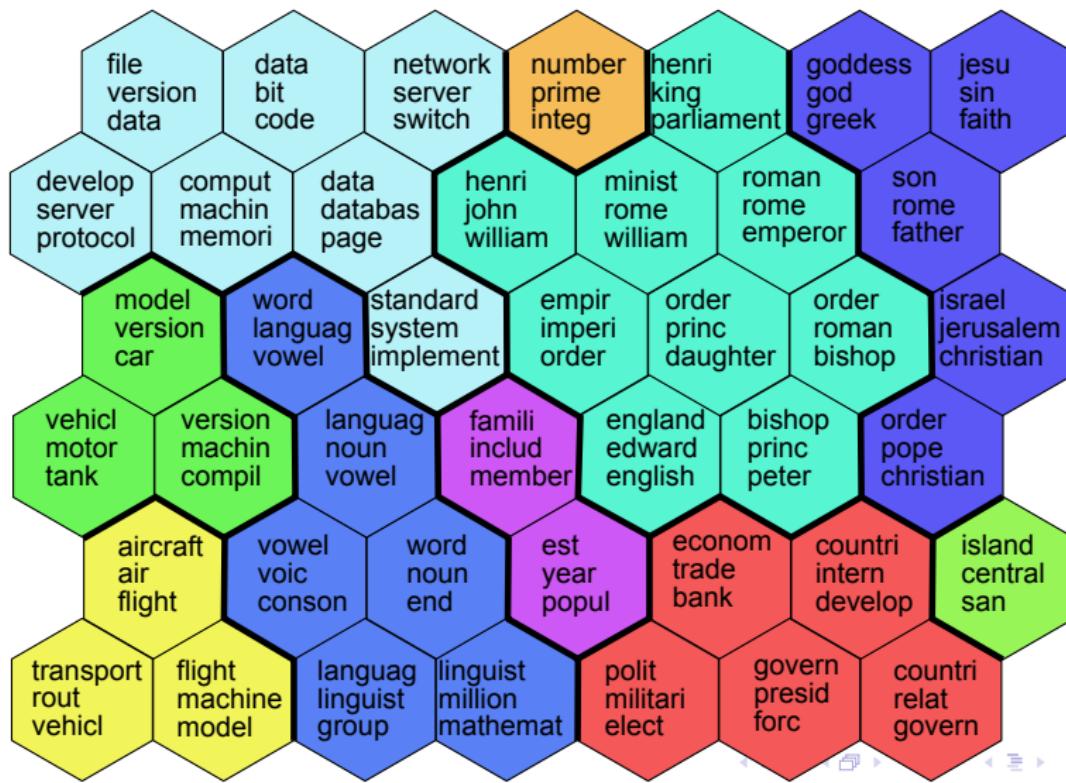
- Soft clusteringre példa: témamodellek
 - Egy dokumentum mennyire szól az egyes témákról
 - Egy témába milyen szavak tartoznak?
 - Pl. Latent Semantic Analysis (SVD)



Latent Semantic Analysis



Kiterjesztés csoportritka regularizációval



Példa: kakukktojás játék

Egybetartozó szavak				Kakukktojás
cao	wei	liu	emperor	king
superman	clark	luthor	kryptonite	batman
devil	demon	hell	soul	body
egypt	egyptian	alexandria	pharaoh	bishop
singh	guru	sikh	saini	delhi
language	dialect	linguistic	spoken	sound
mass	force	motion	velocity	orbit
voice	speech	hearing	sound	view
athens	athenian	pericles	corinth	ancient
data	file	format	compression	image
function	problems	polynomial	equation	physical

Tartalom

1 Nem felügyelet tanulás

2 Klaszterezés

- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

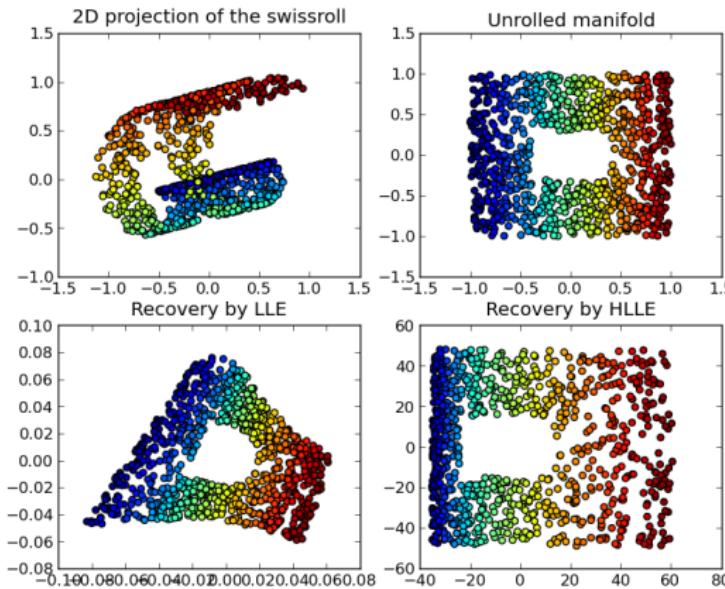
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

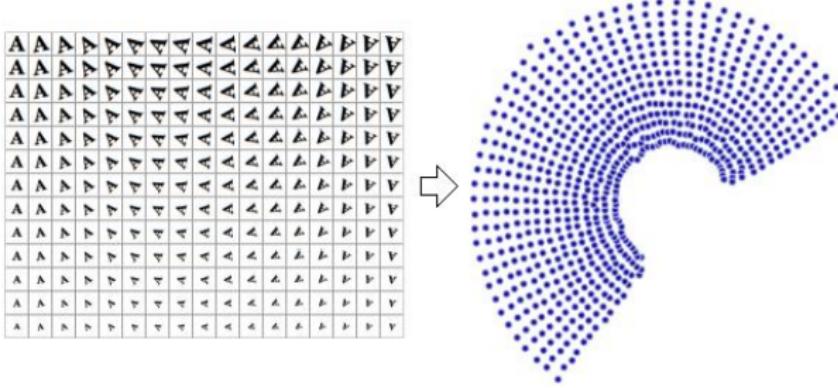
Miért dimenziócsökkentünk?

- Az adatok valójában alacsonyabb dimenziósak, csak magasabb dimenziós térben vannak
- Láttatjuk az adatokat
- Eltüntetjük a zajt
- Csökkentjük a tanulási feladat bonyolultságát (jobb eredmények, kisebb futási idő, ...)
- A csökkentett dimenziójú adatokon új törvenyszerűségeket, sejtéseket láthatunk meg
- A probléma megoldásához kisebb dimenziós és/vagy sűrű reprezentációra van szükségünk

Példa: Swiss roll



Példa: A betű forgatása



Nem felügyelt tanulás

└ Dimenziócsökkentés

└ Kovariancia, korreláció

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

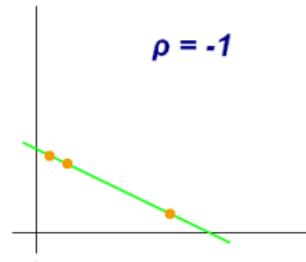
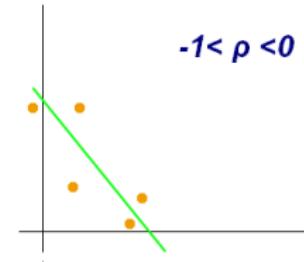
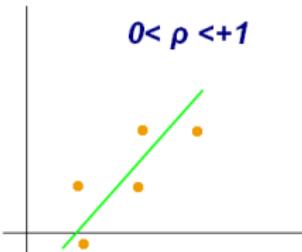
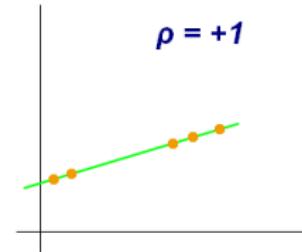
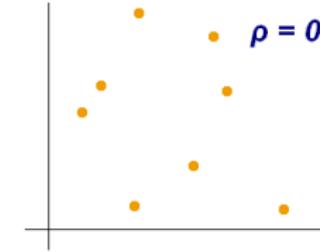
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

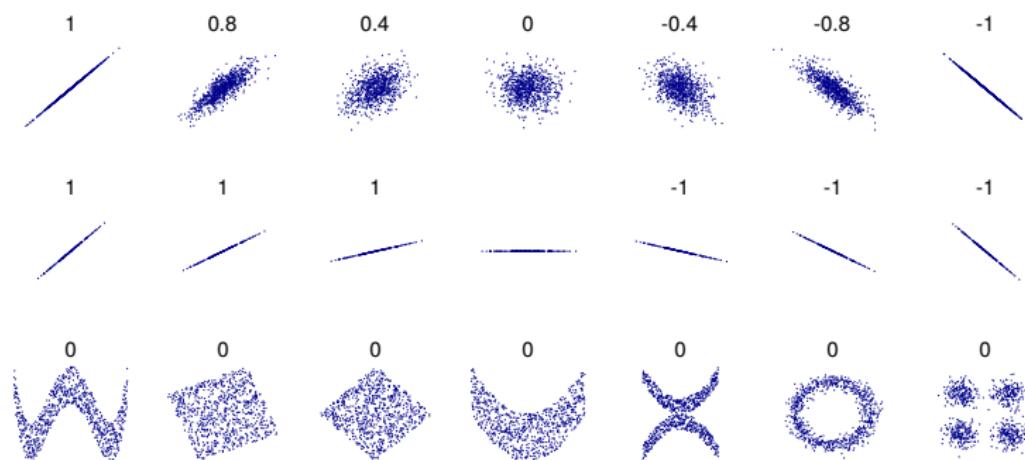
- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Kovariancia, korreláció

 $\rho = -1$  $-1 < \rho < 0$  $0 < \rho < +1$  $\rho = +1$  $\bullet \rho = 0$ 

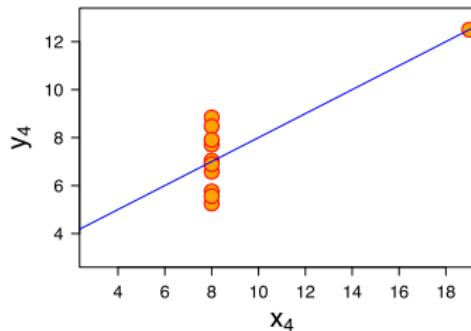
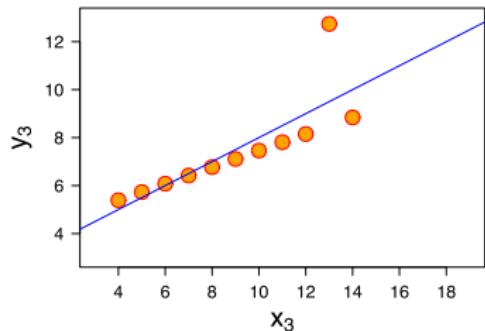
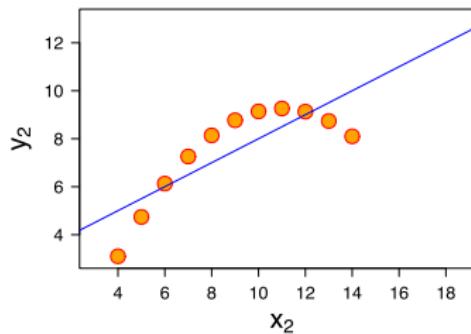
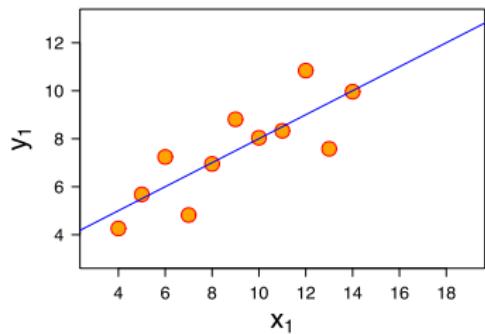
Kovariancia, korreláció



Kovariancia, (Pearson) korreláció

- Azt mérik, hogy X , Y val. változók mennyire mozognak együtt
- Lineáris kapcsolatot mutatnak
- $\text{Cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$
- Pl.: Pozitív: Ha $X > E(X)$, akkor $Y > E(Y)$, ha $X < E(X)$, akkor $Y < E(Y)$
- $\text{Cov}(X, Y) = E[XY] - E[X]E[Y]$
- Korreláció: „Normalizált” kovariancia, -1 és 1 között
- $\rho_{X,Y} = \text{corr}(X, Y) = \frac{\text{cov}(X, Y)}{\sigma_X \sigma_Y} = \frac{E[(X - \mu_X)(Y - \mu_Y)]}{\sigma_X \sigma_Y}$
- $r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}}$

Mind a négy adathalmaz korrelációs együtthatója 0.816



Kovariancia mátrix

- \mathbf{X} egy vektor, aminek az elemei val. változók
- A kovariancia mátrix elemei X_i, X_j közti kovarianciák
- $\Sigma_{ij} = \text{cov}(X_i, X_j) = E[(X_i - \mu_i)(X_j - \mu_j)]$
- $\mu_i = E(X_i)$

-
- $$\begin{bmatrix} E[(X_1 - \mu_1)(X_1 - \mu_1)] & E[(X_1 - \mu_1)(X_2 - \mu_2)] & \cdots & E[(X_1 - \mu_1)(X_n - \mu_n)] \\ E[(X_2 - \mu_2)(X_1 - \mu_1)] & E[(X_2 - \mu_2)(X_2 - \mu_2)] & \cdots & E[(X_2 - \mu_2)(X_n - \mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(X_n - \mu_n)(X_1 - \mu_1)] & E[(X_n - \mu_n)(X_2 - \mu_2)] & \cdots & E[(X_n - \mu_n)(X_n - \mu_n)] \end{bmatrix}$$
- A főátlóban a szórások vannak.
 - Ekvivalens: $\Sigma = E(\mathbf{X}^\top \mathbf{X}) - \mu^\top \mu$

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

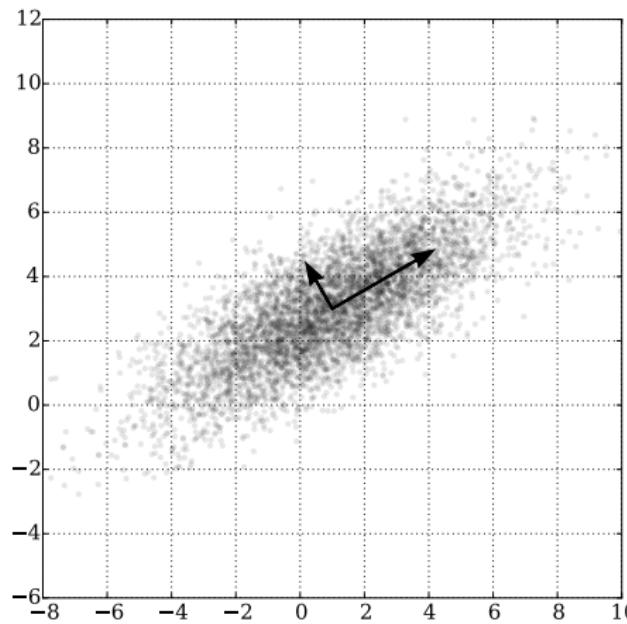
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Példa - 2d normáleloszlás



Főkomponens analízis

- Principal component analysis (PCA)
- Demo:
<http://setosa.io/ev/principal-component-analysis/>
- Az adathalmazt egy új koordinátarendszerben ábrázoljuk, a tengelyek merőlegesek
- Az adathalmaz vetítései közül a legnagyobb szórású az első tengelyen (főkomponensen) van
- A második legnagyobb szórású a második főkomponensen, ...
- Új változók/adatok: a főkomponensekre vetítjük le az eredeti változókat. Ezek már korrelálatlanok
- Dimenziócsökkentés: eldobjuk azokat a tengelyeket (és koordinátákat), amiken kicsi a szórás

Főkomponens analízis

- $\mathbf{X} \in \mathbb{R}^{n \times p}$: adathalmaz, egy sor egy adatpont
- $\mathbf{t}_{(i)} = (t_1, \dots, t_l)_{(i)}$: az adatpontok az új koordinátarendszerbe transzformálva $\mathbf{w}_{(k)} = (w_1, \dots, w_p)_{(k)}$ -val

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(k)} \quad \text{for } i = 1, \dots, n \quad k = 1, \dots, l$$

- Szórás maximalizálása

$$\mathbf{w}_{(1)} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \sum_i (t_1)_{(i)}^2 \right\} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \sum_i (\mathbf{x}_{(i)} \cdot \mathbf{w})^2 \right\}$$

Főkomponens analízis

- Ugyanez mátrixosan:

$$\mathbf{w}_{(1)} = \arg \max_{\|\mathbf{w}\|=1} \{\|\mathbf{Xw}\|^2\} = \arg \max_{\|\mathbf{w}\|=1} \left\{ \mathbf{w}^T \mathbf{X}^T \mathbf{Xw} \right\}$$

- Mivel \mathbf{w} egységvektor:

$$\mathbf{w}_{(1)} = \arg \max \left\{ \frac{\mathbf{w}^T \mathbf{X}^T \mathbf{Xw}}{\mathbf{w}^T \mathbf{w}} \right\}$$

- Ez a Rayleigh-hányados, a legnagyobb lehetséges érték az $\mathbf{X}^T \mathbf{X}$ legnagyobb sajátértéke lesz, ahol \mathbf{w} a hozzá tartozó sajátvektor
- A többi komponensre is így van → a főkomponensek az $\mathbf{X}^T \mathbf{X}$ sajátvektorai

Főkomponens analízis – algoritmus

- Az \mathbf{X} mátrixban vannak az adataink
- Nulla átlagúra hozzuk az adatokat (kivonjuk az átlagot)
- Kiszámoljuk a $\mathbf{Q} = \mathbf{X}^T \mathbf{X}$ kovariancia mátrixot
- Meghatározzuk ennek a mátrixnak a sajátértékeit, és a sajátvektorait
- A sajátvektorok a főkomponensek, a belőlük álló bázis az új koordinátarendszer
- A legnagyobb sajátértékhez tartozó főkomponens a legnagyobb szórású, és így tovább
- Dimenziócsökkentés: csak a k legnagyobb sajátértékű főkomponenseket tartjuk meg

PCA és SVD

SVD

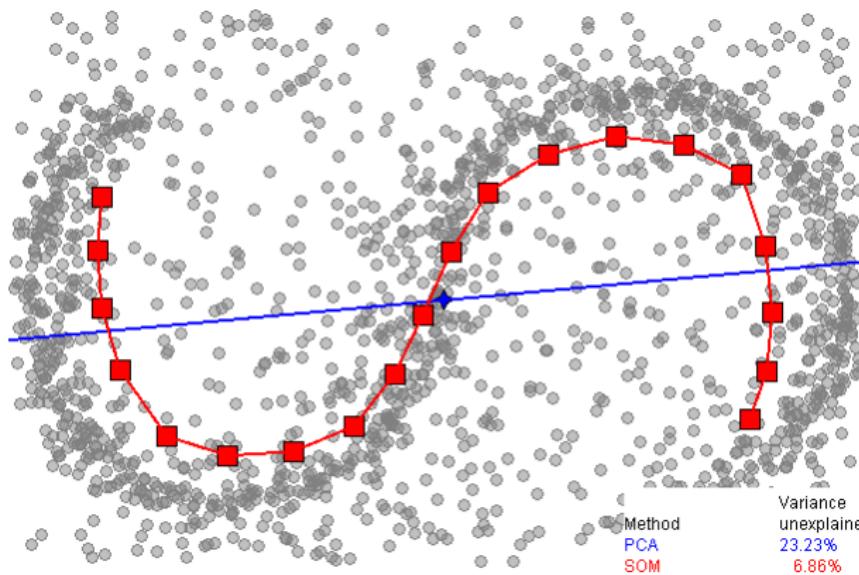
$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{W}^T$$

PCA SVD-vel

$$\begin{aligned}\mathbf{X}^T\mathbf{X} &= \mathbf{W}\Sigma^T\mathbf{U}^T\mathbf{U}\Sigma\mathbf{W}^T \\ &= \mathbf{W}\Sigma^T\Sigma\mathbf{W}^T \\ &= \mathbf{W}\hat{\Sigma}^2\mathbf{W}^T\end{aligned}$$

- \mathbf{W} -ben már $\mathbf{X}^T\mathbf{X}$ sajátvektorai vannak. A szinguláris értékek a sajátértékek négyzetgyökei.

A PCA is lineáris



Python példák

- A feature scaling fontossága:

http://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

Tartalom

1 Nem felügyelt tanulás

2 Klaszterezés

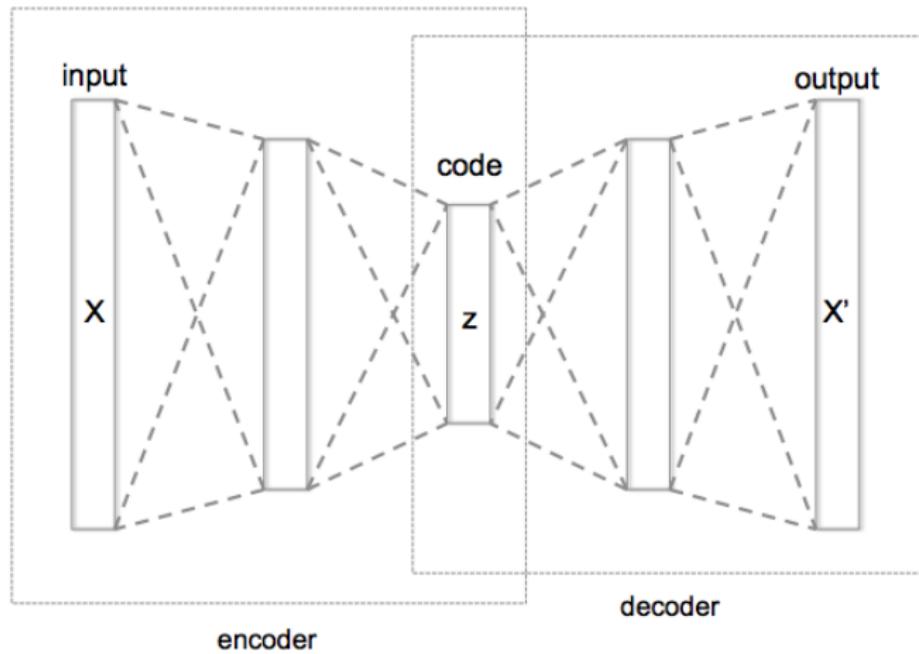
- Hard clustering – k-means
- Soft clustering – témamodellek

3 Dimenziócsökkentés

- Kovariancia, korreláció
- Főkomponens analízis

4 Autoenkóderek

Autoenkóderek



Autóenkóderek

Egyszerű autóenkóder

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

- Ez az egyszerű autoenkóder a PCA alterébe projektál
- Flexibilis, sokféle variáció létezik
 - Denoising autoencoder: zajos inputból kell zajtalan outputot előállítani
 - Sparse autoencoder: csak néhány egység lehet aktív a rejtett reprezentációban
 - VAE: Egy valószínűségi modellt feltételez, a poszterior eloszlást approximálja
- Sokszor fontosak egy felügyelt mély háló előtanításában

Köszönöm a figyelmet!

Köszönöm a figyelmet!