Keresések

Kereső rendszer (KR)

Procedure *KR*
1. ADAT := *kezdeti érték*
2. **while** ¬*terminálási feltétel*(ADAT) **loop**
3. SELECT SZ FROM *alkalmazható szabályok*
4. ADAT := SZ(ADAT)
5. **endloop**

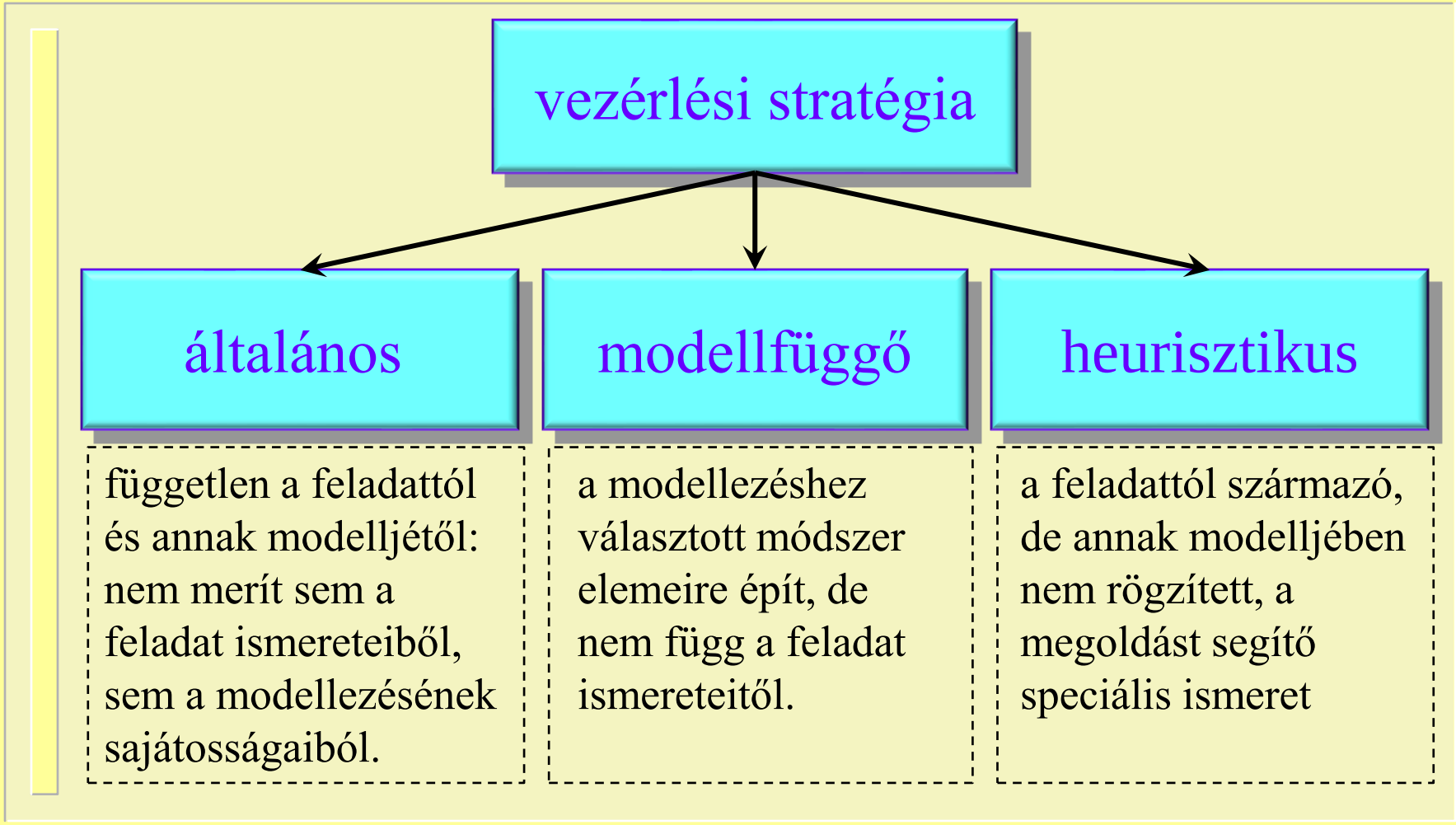
end

globális munkaterület
tárolja a keresés során megszerzett és megőrzött ismeretet (egy részgráfot)
(*kezdeti érték ~ start csúcs, terminálási feltétel ~ célcsúcs*)

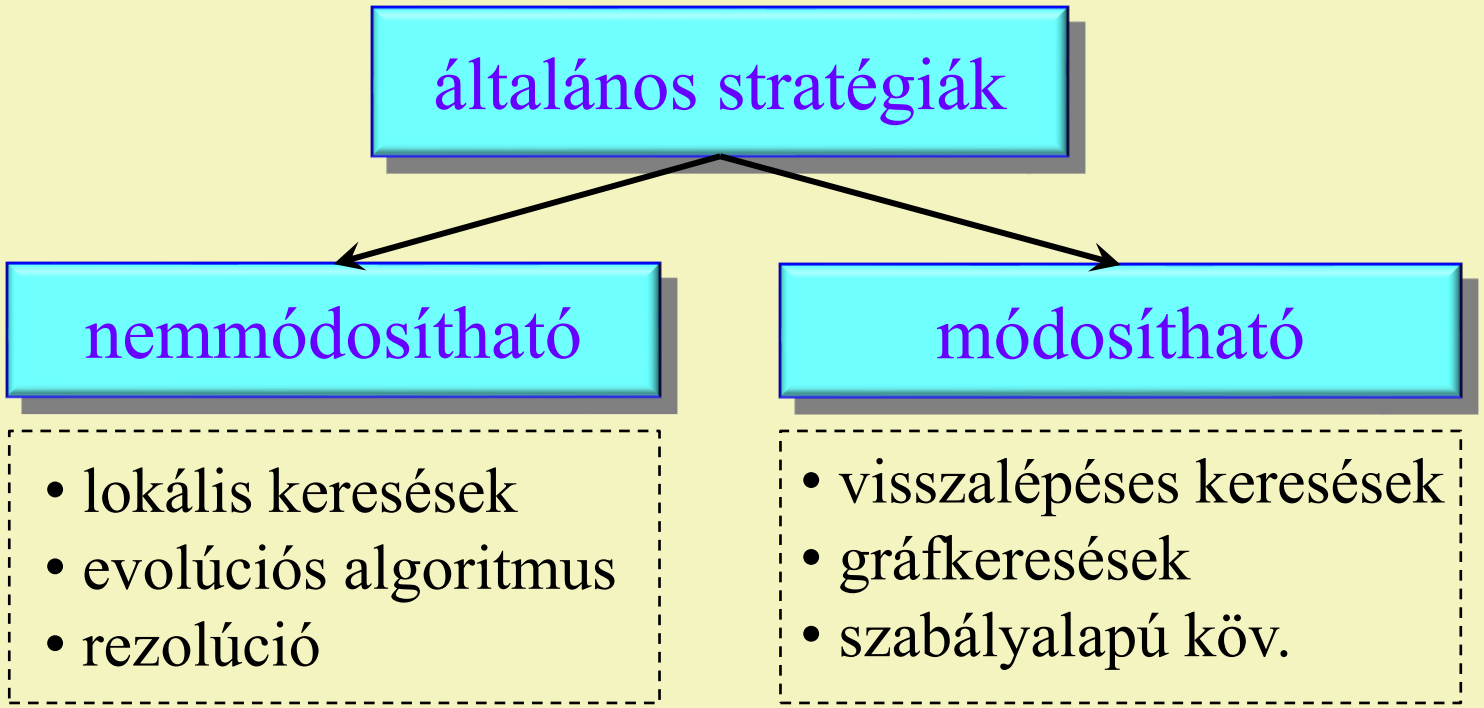
keresési szabályok
megváltoztatják a globális munkaterület tartalmát
(*előfeltétel, hatás*)

vezérlési stratégia
alkalmazható szabályok közül kiválaszt egy „megfelelőt”
(*általános elv + heurisztika*)

KR vezérlési szintjei



Általános vezérlési stratégiák



1. Lokális keresések

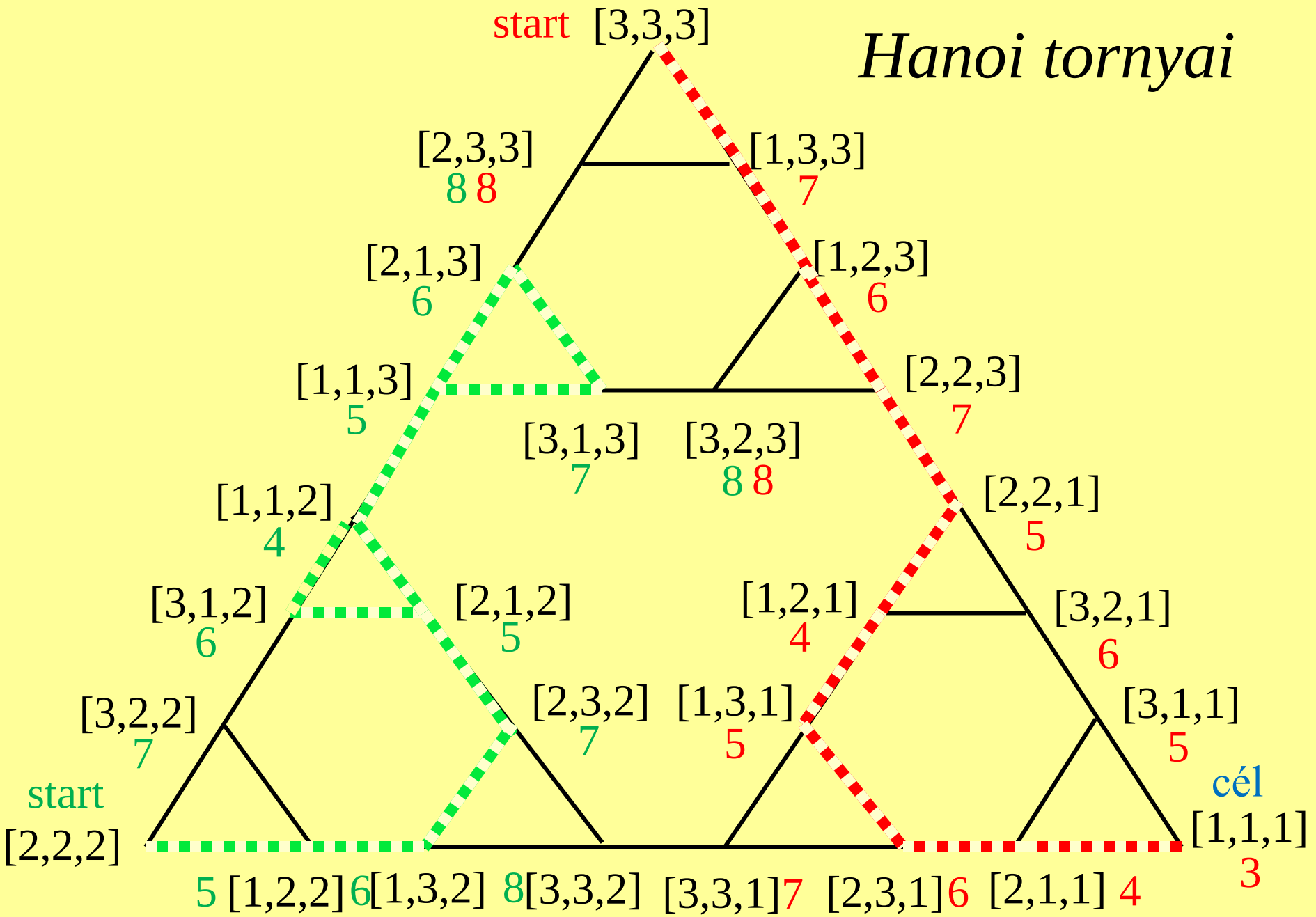
- A lokális keresés olyan KR, amely a probléma reprezentációs gráfjának egy kis részét tárolja (a globális munkaterületén).
  - Kezdetben a startcsúcsot ismeri, és
  - akkor áll le, ha a célcsúcs megjelenik a látókörében, vagy valamilyen okból nem tud tovább keresni.
- Az eltárolt részgráf csúcsait a részgráf szűk környezetéből vett „jobb” csúcsokra cseréli le (keresési szabály).
- A „jobbság” eldöntéséhez (vezérlési stratégia) egy kiértékelő függvényt (cél-, rátermettségi-, heurisztikus függvényt) használ, amely reményeink szerint annál jobb értéket ad egy csúcsra, minél közelebb esik az a célhoz.

Hegymászó módszer

- A globális munkaterület egy aktuális csúcsot (*akt*), és annak azt a szülőjét ($\pi(akt)$) tárolja el, amely a megelőző aktuális csúcs volt.
  - Kezdetben a startcsúcs lesz az aktuális csúcs.
  - Terminál, ha az aktuális csúcs célcsúcs vagy zsákutca.
- Egy keresési szabály az aktuális csúcsot cseréli le annak egy gyerekére ($\Gamma(akt)$).
- A vezérlési stratégia mindig azt a szabályt választja, amelyik az aktuális csúcs legjobb – de lehetőleg nem a szülőcsúcsként nyilvántartott – gyerekére lép.

Megjegyzés: Egy másik változata a hegymászó algoritmusnak nem engedi meg, hogy az aktuális csúcsot egy rosszabb értékű csúcsra cseréljük (ilyenkor a keresés inkább leáll).

Hanoi tornyai



ADAT := *kezdeti érték*
while ¬*terminálási feltétel*(ADAT) **loop**
SELECT SZ FROM *alkalmazható szabályok*
ADAT := SZ(ADAT)
endloop

Hegymászó módszer algoritmusa

1. *akt* := *start*
2. **while** $akt \notin T$ **loop**
3. $akt := \mathbf{arg\ opt}_f(\Gamma(akt) - \pi(akt))$
4. **endloop**
5. **return** *akt*

$\Gamma(akt)$ ~ *akt* gyermekei
$\pi(akt)$ ~ *akt* egy szülője

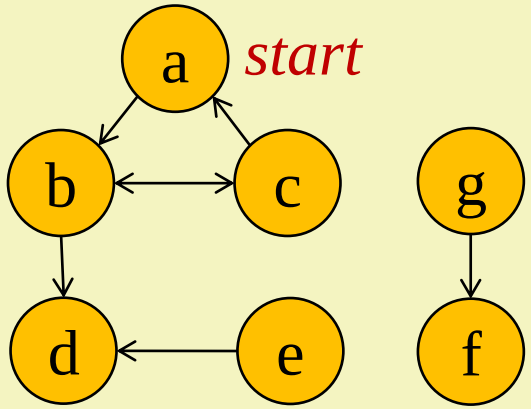
if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
if $\Gamma(akt) = \{\pi(akt)\}$ **then** $akt := \pi(akt)$
else $akt := \mathbf{arg\ opt}_f(\Gamma(akt) - \pi(akt))$

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

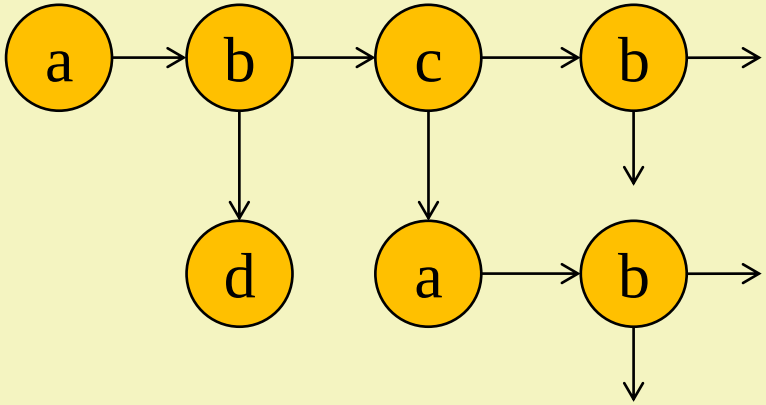
Hogyan „látja” egy keresés a reprezentációs gráfot?

- Egy keresés fokozatosan fedezi fel a reprezentációs gráfot: bizonyos részeihez soha nem jut el, de a felfedezett részt sem feltétlenül tárolja el teljesen, sőt, sokszor torzultan „látja” azt: ha például egy csúcshoz érve nem vizsgálja meg, hogy ezt korábban már felfedezte-e, hanem új csúcsként regisztrálja, akkor az eredeti gráf helyett egy fát fog „látni”.

eredeti gráf:



keresés által látott gráf:



Reprezentációs gráf „fává egyenesítése”

- Ha a keresés nem vizsgálja meg, hogy egy csúcsot korábban már felfedezett-e, akkor valójában a reprezentációs gráfnak a 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 drasztikusan megnövelik a kiegyenesítéssel kapott fa méretét. Olcsóbb, ha mindig eltároljuk egy csúcsnak azt a szülőcsúcsát, amelyik felől a csúcsot elértük. Így egy csúcsból a szülőjébe visszavezető él könnyen felismerhető és figyelmen kívül hagyható.

Hegymászó módszer értékelése

- Előny: könnyű implementálni
- Hátrányok:
  - Csak erős heurisztika esetén lesz sikeres: különben „eltéved” (nem talál megoldást), sőt zsákutcában „beragad” (leáll). Segíthet, ha:
    - véletlenül választott startcsúcsból újra- és újra elindítjuk → random restart local search
    - *k* darab aktuális csúcs legjobb *k* darab gyerekére lépünk → local beam search
    - gyengítjük és véletlenítjük a mohó stratégiáját → simulated annealing
  - Lokális optimum hely körül vagy ekvidisztans felületen (azonos értékű szomszédos csúcsok között) található körön, végtelen működésbe eshet. Segíthet, ha:
    - növeljük a memóriát → tabu search

Tabu keresés

- A globális munkaterület eltárolja az aktuális csúcsot (*akt*), a keresés során utoljára érintett néhány csúcsot (*Tabu*), és az eddigi legjobb csúcsot (*opt*).
  - Kezdetben az *akt* és az *opt* a startcsúcs, a *Tabu* pedig üres.
  - Terminál, ha az *opt* célcsúcs vagy régóta nem változik, illetve az *akt* egy zsákutca.
- Egy keresési szabály az aktuális csúcsot cseréli le a legjobb gyerekére, aktualizálja a *Tabu* halmazt (a lecserélt aktuális csúcsot elhelyezi benne: a Tabu egy „sor adatszerkezet”), és ha *akt* jobb, mint az *opt*, akkor *opt* új értéke az *akt* lesz.
- A vezérlési stratégia mindig azt a szabályt választja, amelyik az aktuális csúcsnak a legjobb – de a *Tabu* halmazban nem tárolt – gyerekére lép.

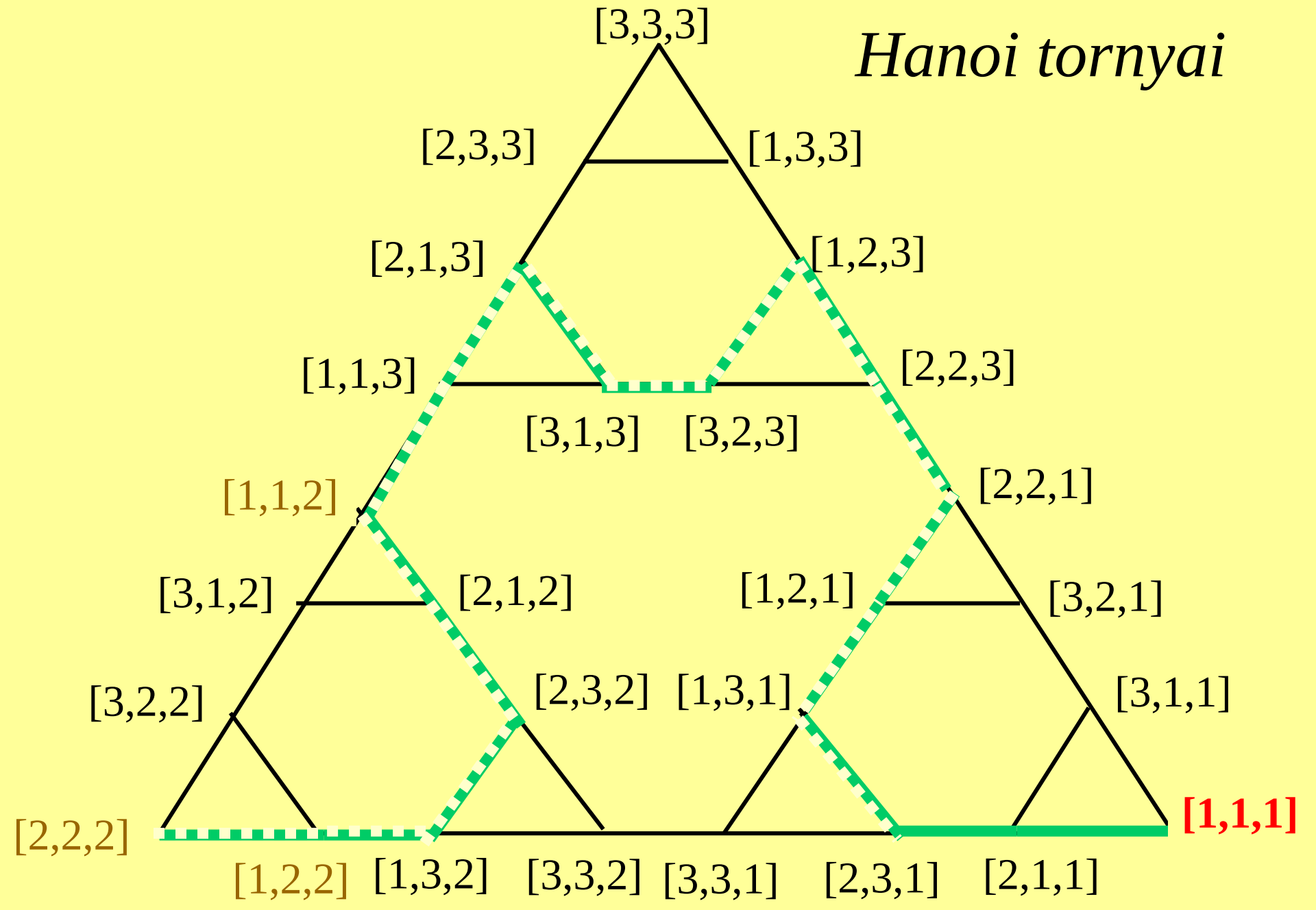
ADAT := *kezdeti érték*
while ¬*terminálási feltétel*(ADAT) **loop**
SELECT SZ FROM *alkalmazható szabályok*
ADAT := SZ(ADAT)
endloop

Tabu keresés algoritmusa

1. *akt*, *opt*, *Tabu* := *start*, *start*, ∅
2. **while not** (*opt*∈*T* **or** *opt* régóta nem változik) **loop**
3. *akt* := ***arg opt***$_f$(Γ(*akt*)−*Tabu*)
4. *Tabu* := Módosít(*akt*,*Tabu*)
5. **if** *f*(*akt*) jobb, mint *f*(*opt*) **then** *opt* := *akt*
6. **endloop**
7. **return** *akt*

if Γ(*akt*) = ∅ **then return** nem talált megoldást
else if Γ(*akt*)⊆*Tabu*) **then** *akt* := **arg opt**$_f$(Γ(*akt*))
else *akt* := **arg opt**$_f$(Γ(*akt*)− *Tabu*)

Hanoi tornyai



Tabu keresés értékelése

- 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

- Csak a vezérlési stratégiája tér el a hegymászó módszertől: az aktuális csúcs (*akt*) gyermekei közül véletlenszerűen választ új csúcsot (*új*), és azt akkor fogadja el aktuális csúcsnak, ha a függvény-értéke viszonylag jó :
  - ha az *új* csúcs kiértékelő függvény-értéke nem rosszabb, mint az *akt* csúcsé ($f(új) \leq f(akt)$), akkor elfogadja.
  - ha az *új* csúcs függvényértéke rosszabb ($f(új) > f(akt)$), akkor az *új* csúcs elfogadásának valószínűsége fordítottan arányos az $|f(akt) - f(új)|$ különbséggel:

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

Hűtési ütemterv

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

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

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

f(új)=120, f(akt)=107

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 ¬*terminálási feltétel*(ADAT) **loop**
SELECT SZ FROM *alkalmazható szabályok*
ADAT := SZ(ADAT)
endloop

Szimulált hűtés algoritmusa

1. *akt* := *start* ; *k* := 1 ; *i* := 1
2. **while not**(*akt*∈*T* **or** *f*(*akt*) régóta nem változik) **loop**
3. **if** $i > L_k$ **then** *k* := *k*+1; *i* := 1
4. *új* := ***select***(Γ(*akt*)−π(*akt*))
5. **if** *f*(*új*)≤*f*(*akt*) **or**
   f(*új*)>*f*(*akt*) **and** $e^{\frac{f(akt)-f(új)}{T_k}} > rand[0,1]$
6. **then** *akt* := *új*
7. *i* := *i*+1
8. **endloop**
9. **return** *akt*

if Γ(*akt*) = ∅ **then return** nem talált megoldást
if Γ(*akt*) ={π(*akt*)} **then** *új* := π(*akt*)
else *új* := ***select***(Γ(*akt*)−π(*akt*))

Gráfszínezés

- Feladat:
  - Adott egy véges egyszerű gráf, amelynek a csúcsait a lehető legkevesebb szín felhasználásával úgy kell kiszínezni, hogy a szomszédos csúcsok eltérő színűek legyenek.
- Cél:
  - A gráf csúcsainak olyan minimális osztályból álló osztályozását keressük, ahol egy osztályba tartozó csúcsok között nem vezet él.
  - Majd az egyes osztályokba sorolt csúcsokat lehet ugyanolyan színűre színezni, a felhasznált színek száma pedig az osztályok száma lesz.

Gráfszínezés állapottér modellje

- *Állapot*: a csúcsoknak egy „gyengített” osztályozása, ahol
  - egy osztályhoz tartozó csúcsok között lehetnek élek
  - a gráf maximális fokszámánál több osztály van, de lehet egy osztály üres is
- *Művelet*: Egy osztályból egy csúcsot egy másik osztályba helyez át.
- *Kezdő állapot*: tetszőleges
- *Célállapot*: a legjobb osztályozás
  (minél kevesebb első néhány osztályban legyenek csak csúcsok, amelyek között ne legyen él)
- *Állapot-gráf*: Exponenciális méretű az eredeti gráf csúcsszámához mérve.

Gráfszínezés kiértékelő függvénye

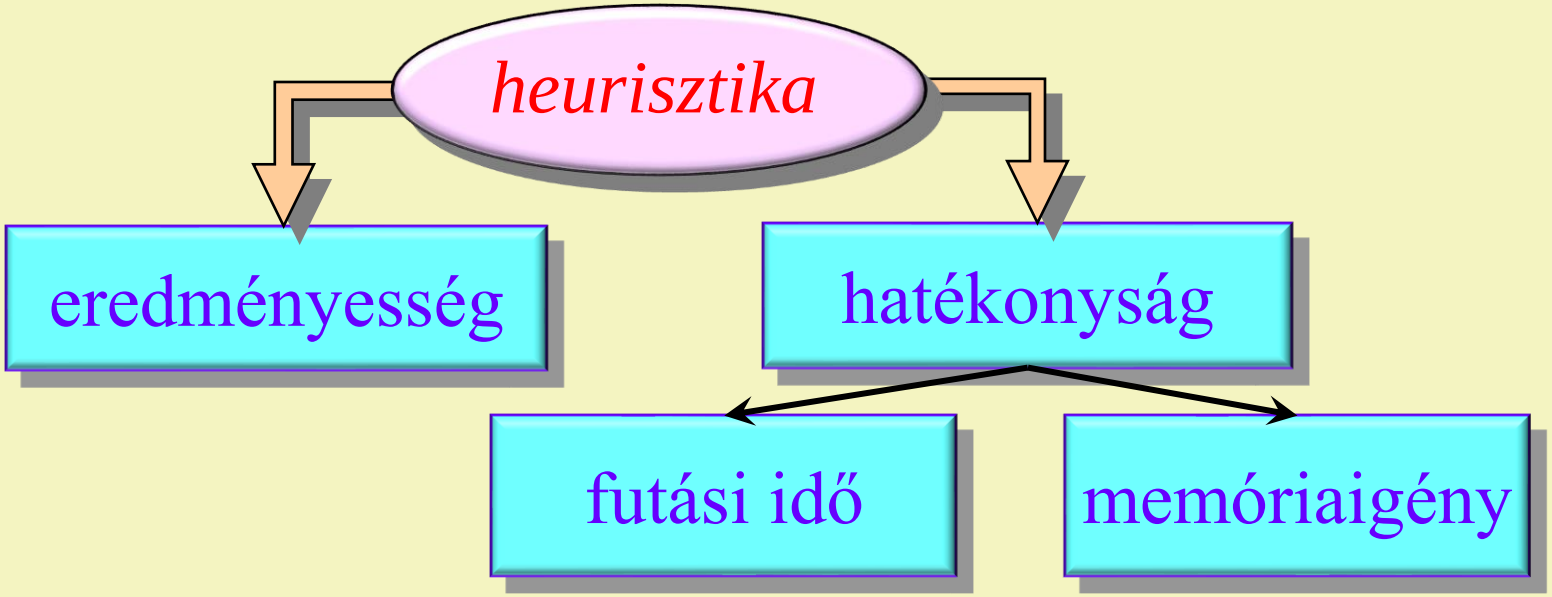
- Annál jobb egy $(O_1, \ldots, O_k)$ osztályozás,
  - minél több csúcs van az első néhány osztályában (ezáltal minél több üres osztálya van), és
  - minél kevesebb egy osztályon belül vezető élek száma.
- $f(n)=\Sigma_j w_j(\lambda |A(O_j)| - |O_j|)$
  - ahol $A(O_j)$ az O_j osztálybeli élek halmaza
  - a $w_j>0$ számok szigorúan növő sorozatot alkotnak.
- Könnyű a „szomszédos” osztályozás kiértékelő függvény értékét kiszámolni.

Lokális kereséssel megoldható feladatok

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

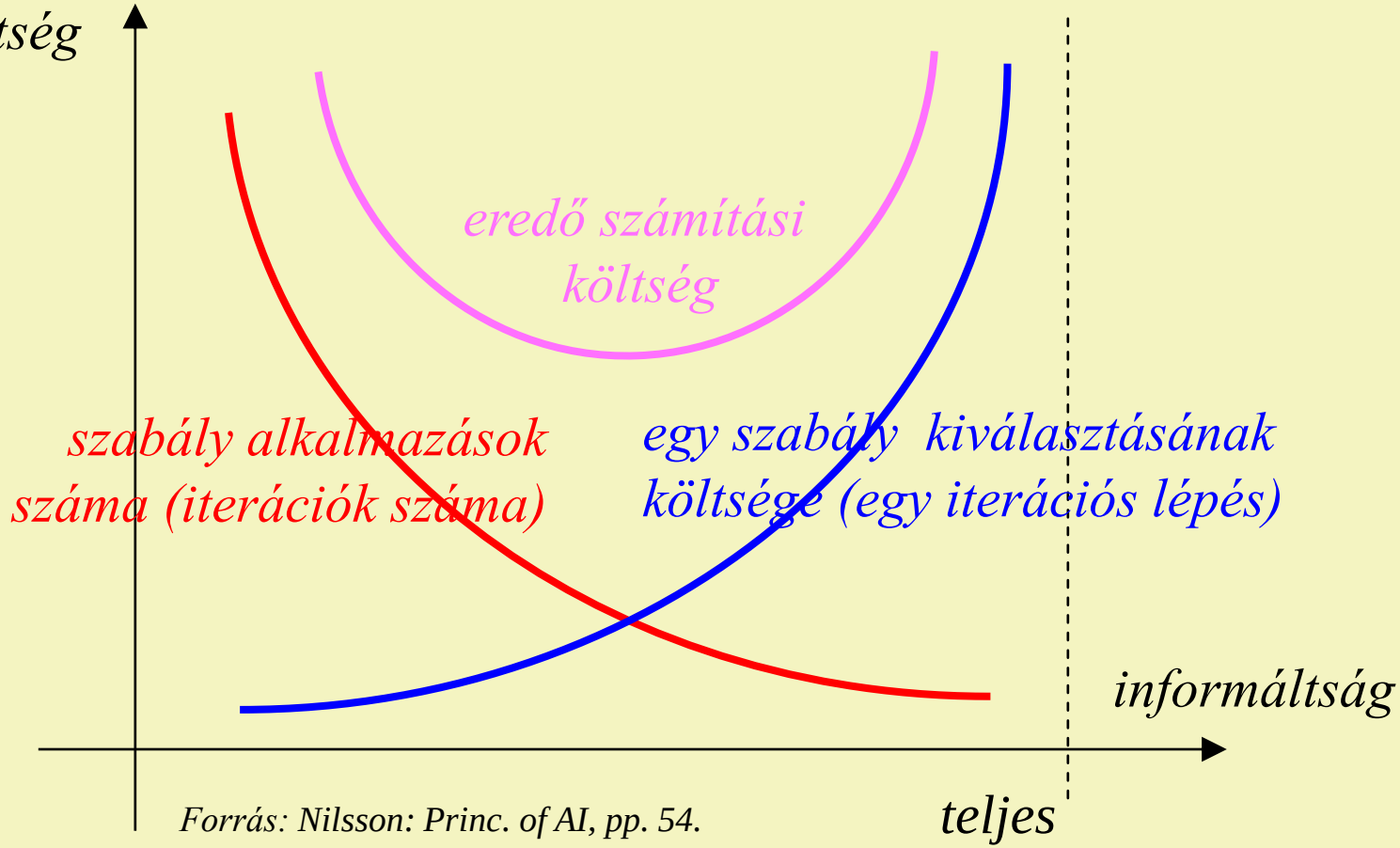
A heurisztika hatása a KR működésére

- A heurisztika olyan, a feladathoz kapcsolódó ötlet, amelyet közvetlenül építünk be egy algoritmusba azért, hogy annak eredményessége és hatékonysága javuljon (egyszerre képes javítani a futási időt és a memóriaigényt), habár erre általában nem ad garanciát.

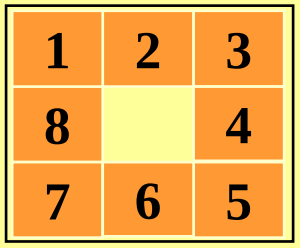


```
ADAT := kezdeti érték
while ¬terminálási feltétel(ADAT) loop
    SELECT SZ FROM alkalmazható szabályok
    ADAT := SZ(ADAT)
endloop
```

Heurisztika és KR hatékonysága

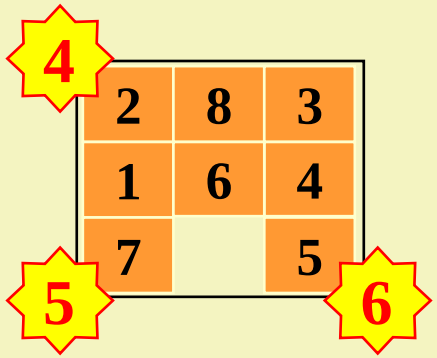


Forrás: Nilsson: Princ. of AI, pp. 54.



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

- Rossz helyen levő lapkák száma (W): azon lapkák szána, amelyek nem a célbeli helyükön vannak.

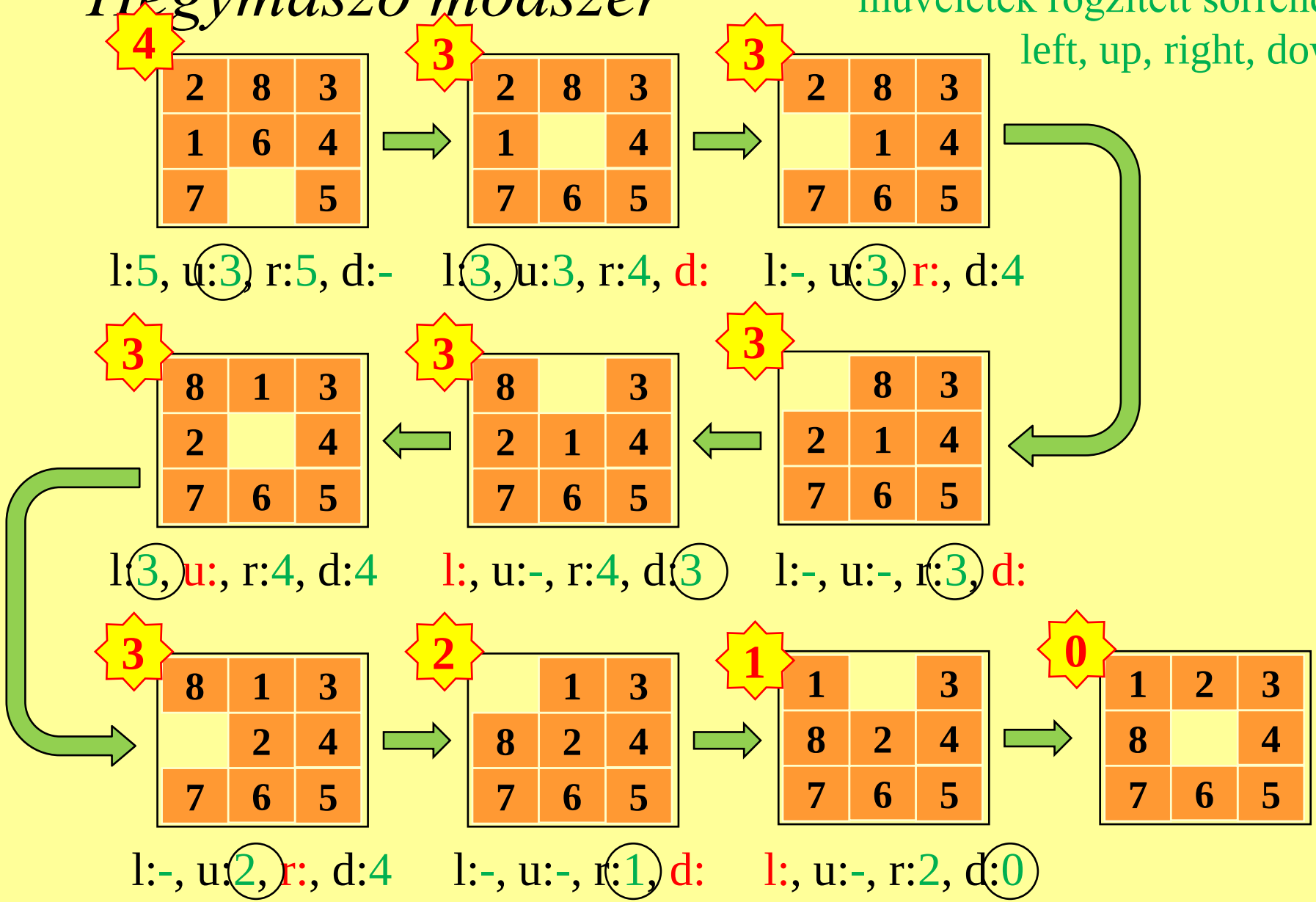


- Manhattan (P): a lapkák célbeli helyüktől vett minimális távolságainak (függőleges és vízszintes mozgatásai számának) összege

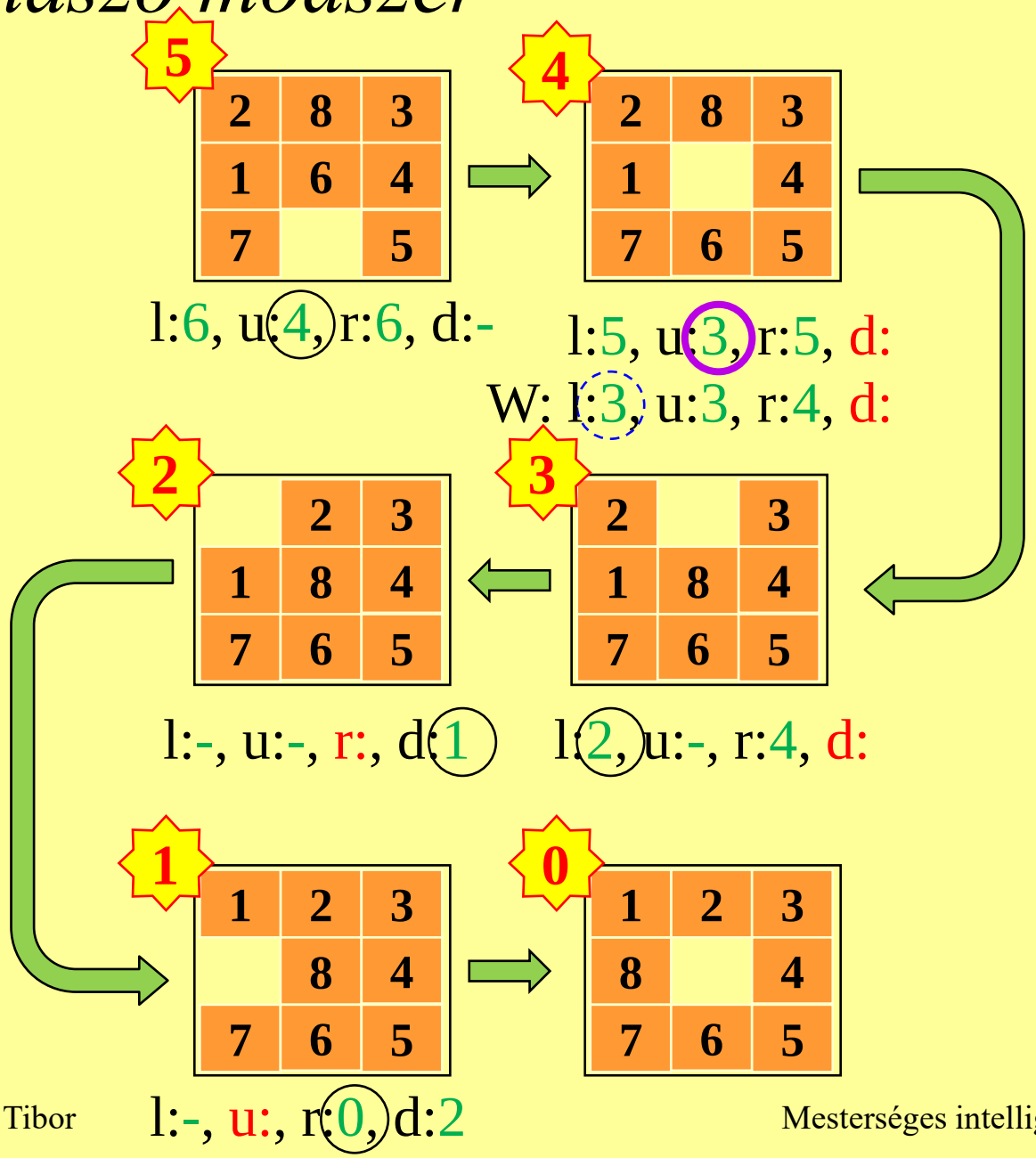
- Keret (F): büntető pontokat ad
  - +1 minden olyan lapkára a szélen, amelyet nem a célbeli szomszédja követ az óra járásával megegyező irányban,
  - +2 minden olyan sarokra, ahol nem a cél szerinti lapka áll.

Hegymászó módszer

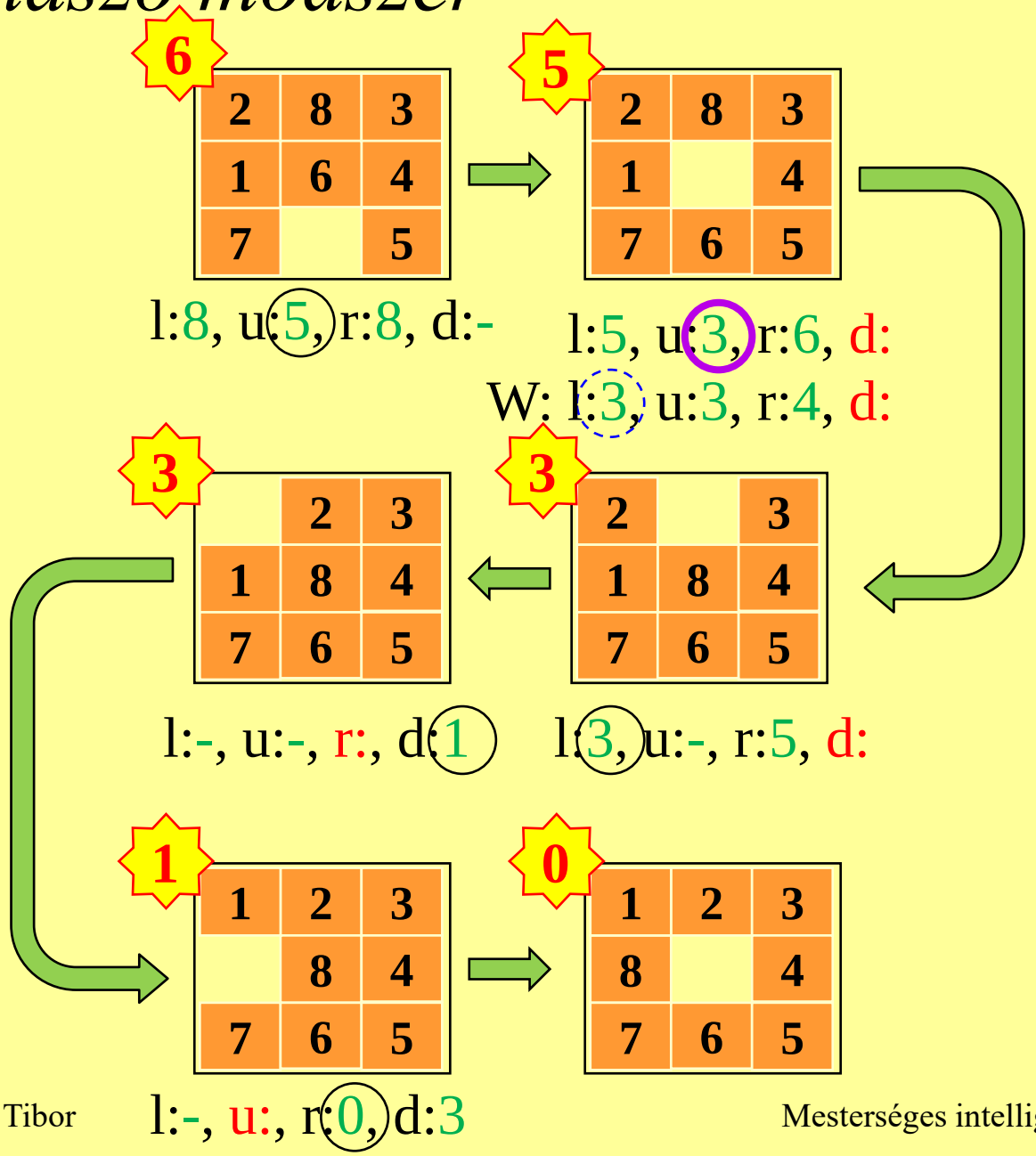
modell függő vezérlés:
műveletek rögzített sorrendje
left, up, right, down



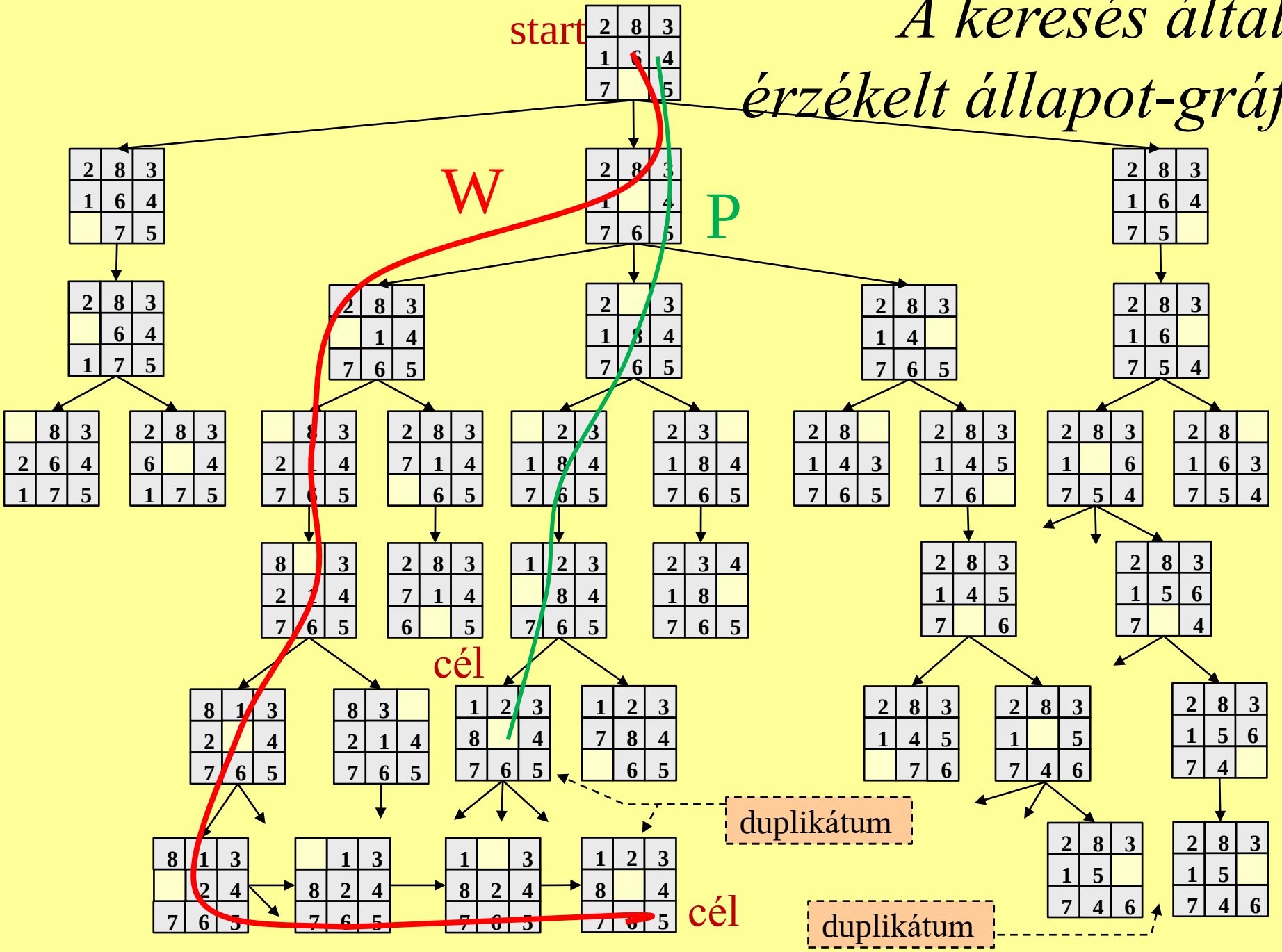
Hegymászó módszer

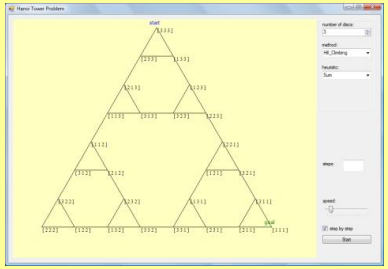


Hegymászó módszer



A keresés által érzékelt állapot-gráf

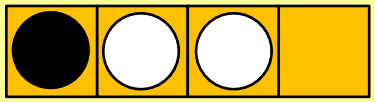




Heurisztikák a Hanoi tornyai problémára

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

Fekete-fehér kirakó



Egy n+m+1 hosszú sínen n fekete és m fehér lapka és egy üres hely van. Egy lapkát szomszédos üres helyre tolhatunk vagy a szomszéd felett üres helyre ugrathatunk. Kezdetben a feketék után jönnek a fehérek, majd az üres hely. Kerüljenek a fehérek a feketék elé!

Állapottér: **AT** = $rec(v : \{B, W, \_\}^{n+m+1}, üres : [1..\ n+m+1])$

invariáns: n darab B, m darab W, 1 üres hely, $üres$ az üres hely indexe

Műveletek: **TolBal, TolJobb, UgrikBal, UgrikJobb**: **AT** → **AT**

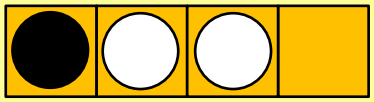
TolBal : HA $this.üres \neq 1$ (*this* : **AT**)

AKKOR $this.v[this.üres-1] \leftrightarrow this.v[this.üres]$

$this.üres := this.üres-1$

Kezdőállapot: [*B, … , B, W, … , W, _*]

Végállapot: $\forall i,j \in [1..\ n+m+1],\ i<j : \neg(this.v[i]=B \wedge this.v[j]=W)$



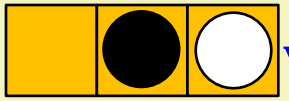
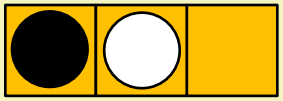
Heurisztikák a Fekete-fehér kirakóra

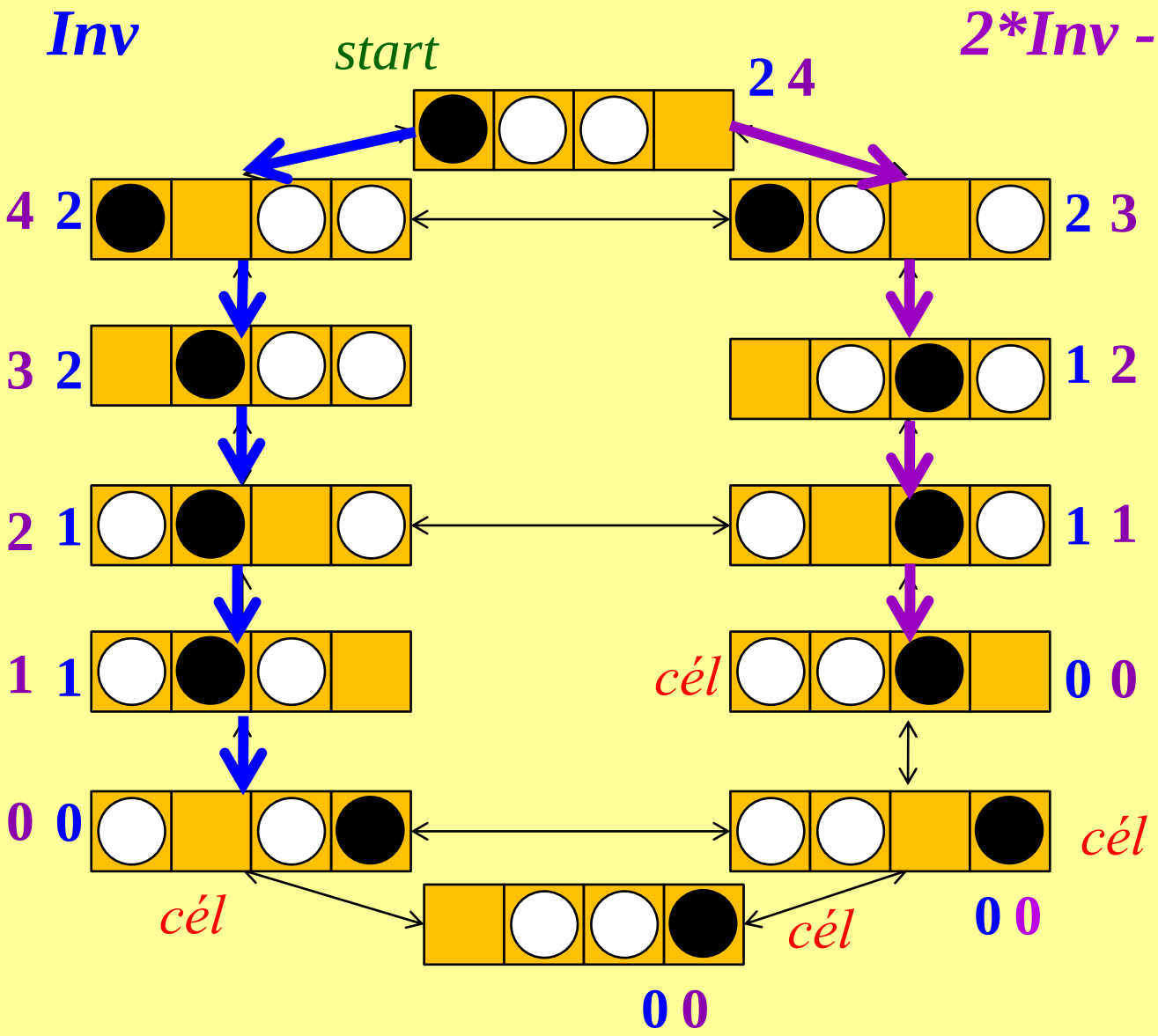
- Inverziószám:

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

- Módosított inverziószám:

  M(this)= 2 · I(this) –

  – (1, **ha** *this*-nek része  **vagy**  *)*


Inv
start
2*Inv - ?1
2 4
4 2
2 3
3 2
1 2
2 1
1 1
1 1
cél
0 0
0 0
cél
cél
cél
0 0
0 0