III. Keresések

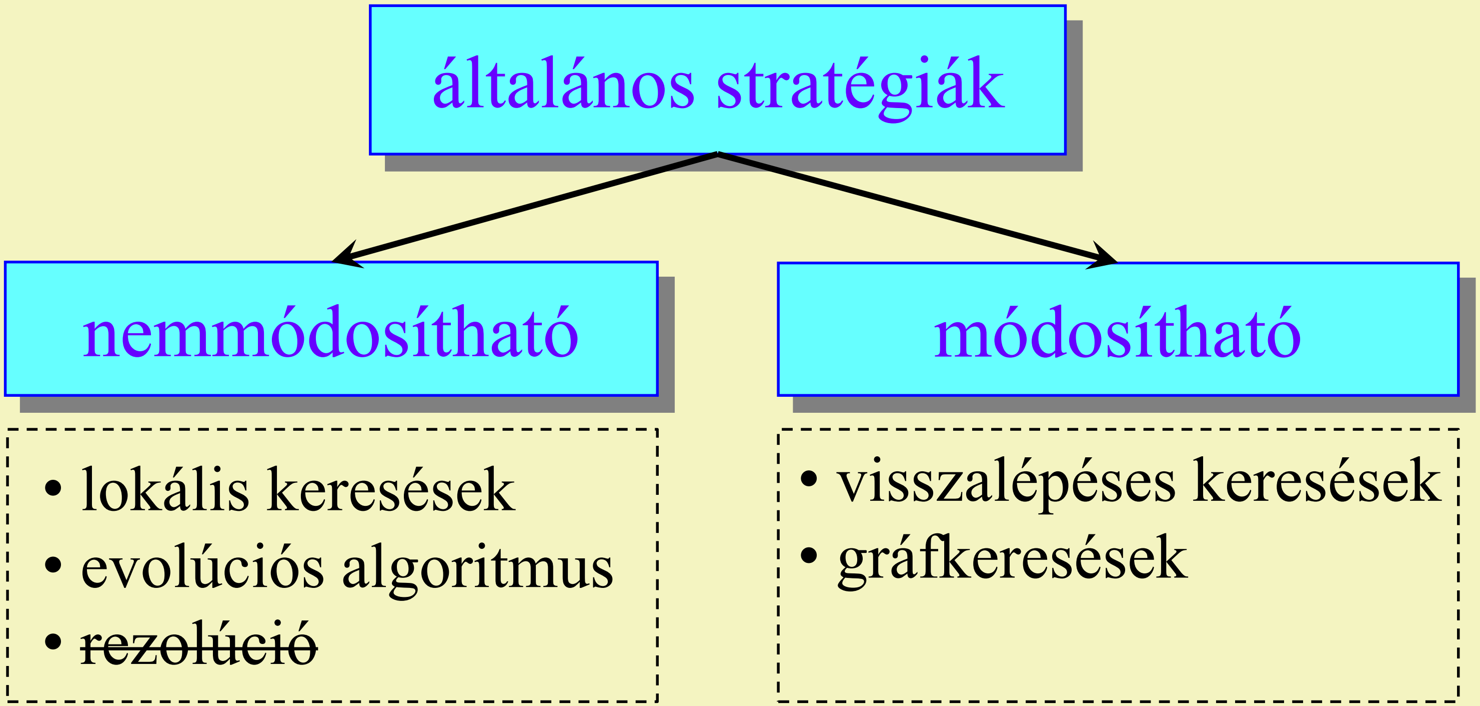
ADAT := *kezdeti érték*
while ¬*terminálási feltétel*(ADAT) **loop**
    SELECT SZ FROM *alkalmazható szabályok*
    ADAT := SZ(ADAT)
endloop

KR vezérlési szintjei

vezérlési stratégia

- **általános**: független a feladattól és annak modelljétől: nem merít sem a feladat ismereteiből, sem a modell sajátosságaiból.
- **modell függő**: nem függ a feladat ismereteitől, de épít a feladat modelljének általános elemeire.
- **heurisztikus**: a feladattól származó, annak modelljében nem rögzített, a megoldást segítő speciális ismeret

Általános vezérlési stratégiák



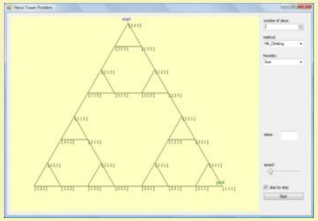
1. Lokális keresések

- 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 célhoz.

ADAT := *kezdeti érték*
while ¬*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.



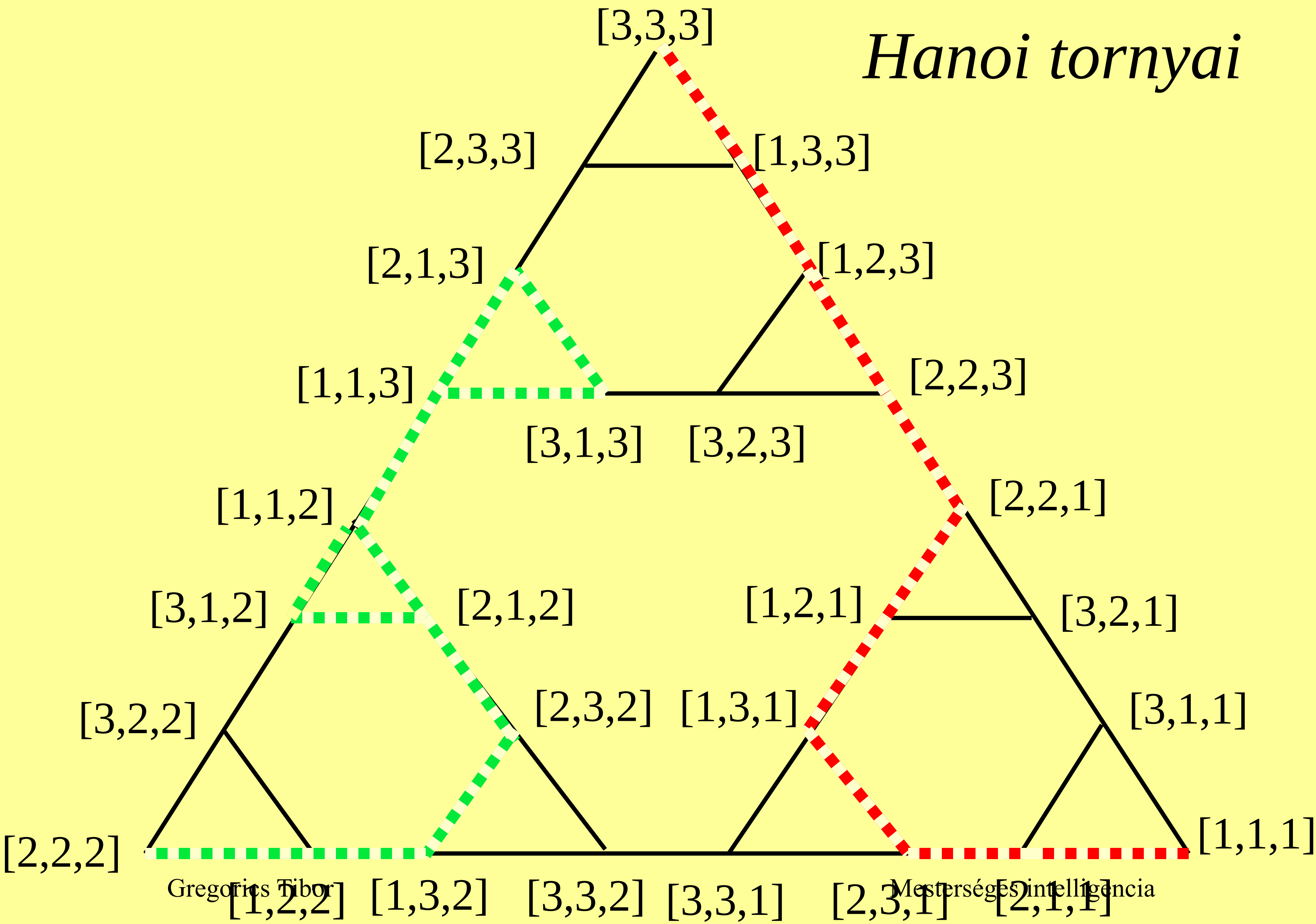
1. $akt := start$
2. **while** $akt \notin T$ **loop**
3. $\quad akt :=$ **opt**$_f(\Gamma(akt) - \pi(akt))$
4. **endloop**
5. **return** *akt*

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

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

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

Hanoi tornyai



Hátrányok

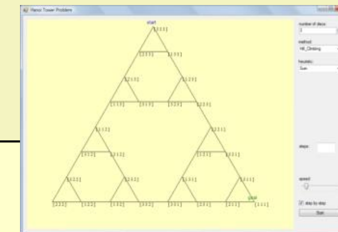
- Csak erős heurisztika esetén lesz sikeres: különben „eltéved" (nem talál megoldást), sőt zsákutcába kerülve „beragad". Segíthet, ha:
  - véletlenül választott startcsúcsból újra- és újraindítjuk
  - *k* aktuális csúcsnak a legjobb *k* gyerekére lépünk
  - gyengítjük a mohó stratégiát ⟹ szimulált hűtés
- 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 célcsúcs
  - ha az *opt* sokáig nem változik.

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

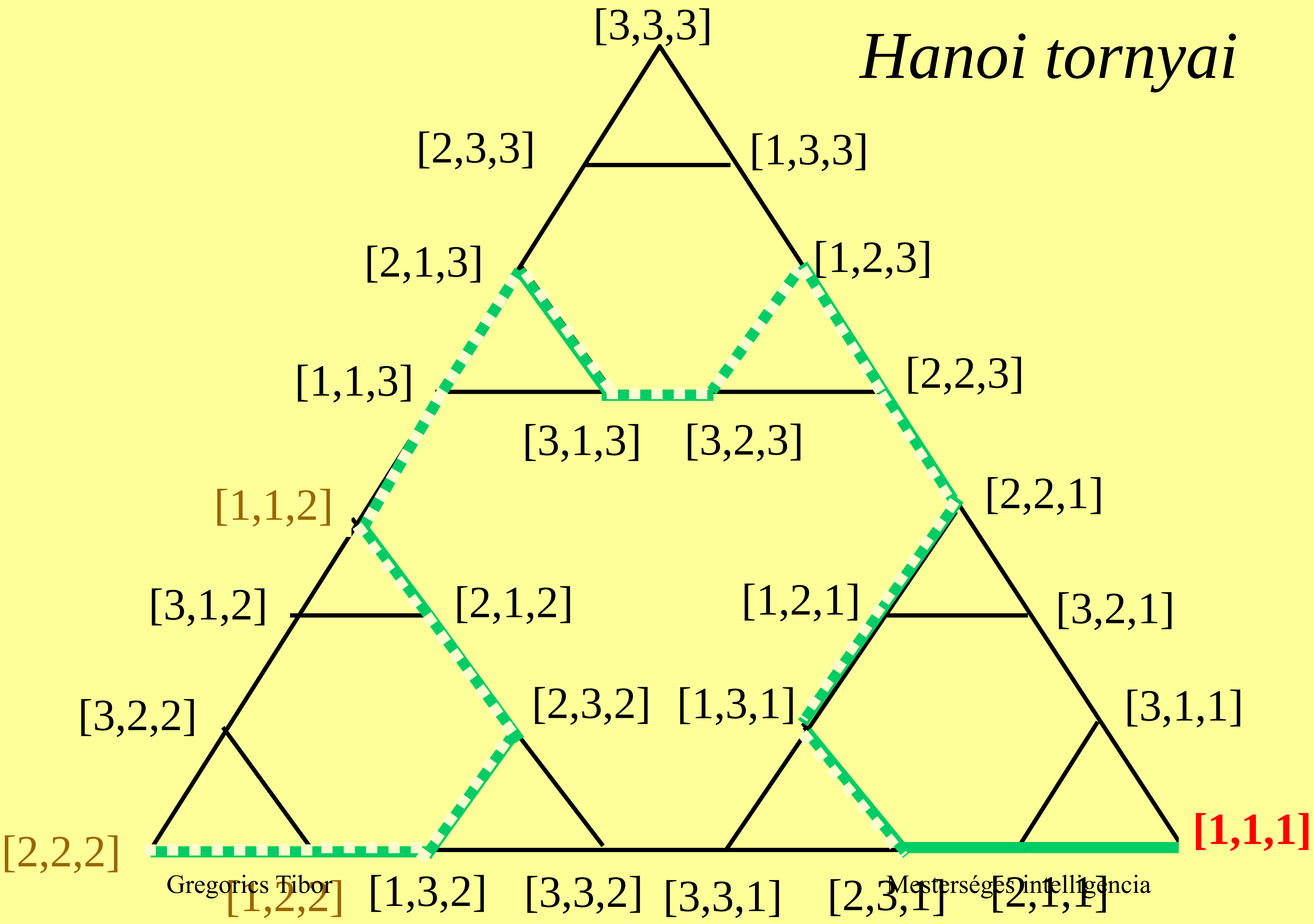
Tabu keresés algoritmusa



1. $akt, opt, Tabu :=$ $start, start, \emptyset$
2. **while not** ($opt \in T$ **or** *opt* régóta nem változik) **loop**
3. $\quad akt :=$ $\mathbf{opt}_f(\, \Gamma(akt) - Tabu \,)$
4. $\quad Tabu :=$ Módosít($akt, Tabu$)
5. $\quad$ **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** $akt := \mathbf{opt}_f(\Gamma(akt))$
else $akt := \mathbf{opt}_f(\Gamma(akt) - Tabu)$

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 $f(új) \leq f(akt)$), akkor elfogadjuk aktuális csúcsnak
- Ha az *új* csúcs függvényértéke rosszabb (itt $f(új) > f(akt)$), akkor egy olyan véletlenített módszert alkalmazunk, ahol 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. Ezt egy (T_k , L_k) $k=1,2,...$ ütemterv módosítja, amely L_1 lépésen keresztül T_1, majd L_2 lépésen keresztül T_2, stb. lesz.

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

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

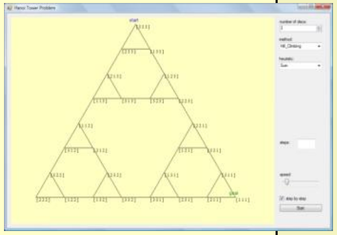
$f(új)=120, f(akt)=107$

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**($akt \in T$ **or** $f(akt)$ régóta nem változik) **loop**
3.   **if** $i > L_k$ **then** $k := k+1;\; i := 1$
4. $új :=$ **select**$(\,\Gamma(akt) - \pi(akt)\,)$
5.   **if** $f(új) \leq f(akt)$ **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*



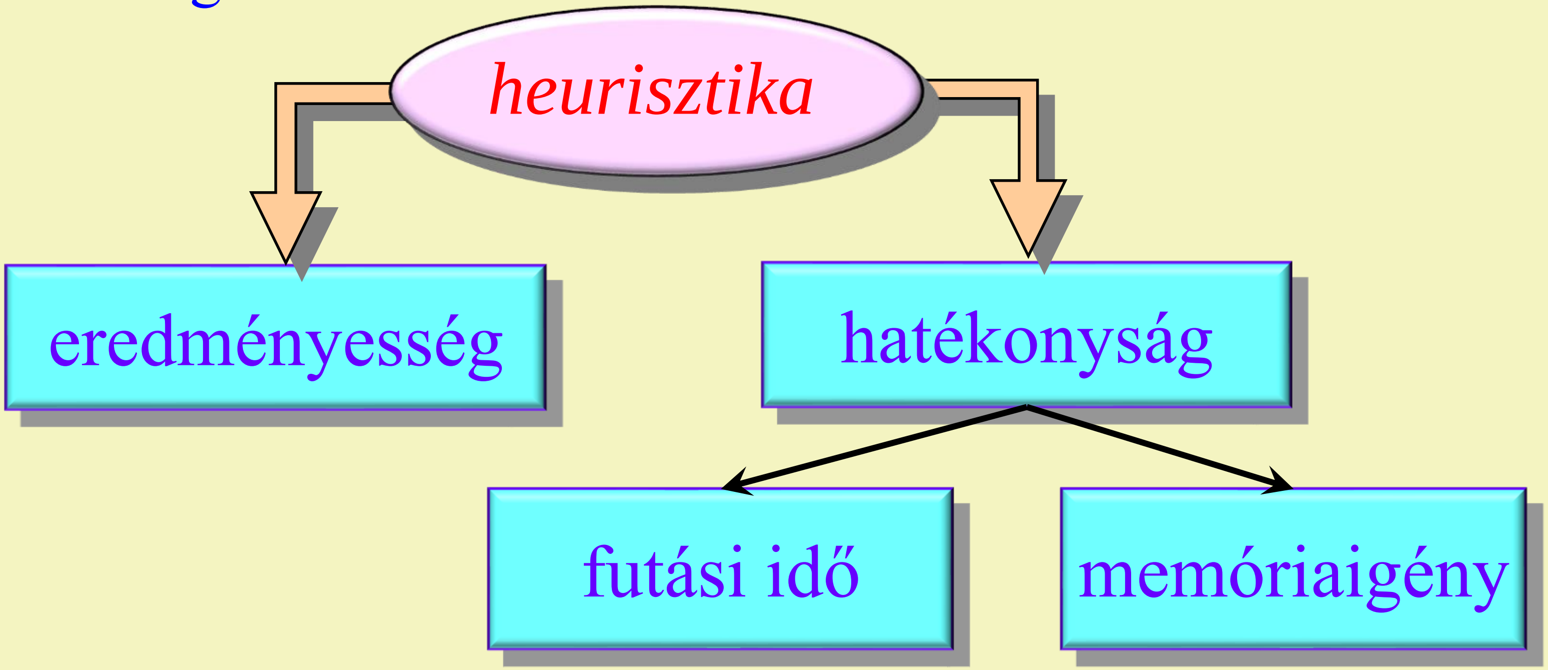
if $\Gamma(akt) = \emptyset$ **then return** nem talált megoldást
if $\Gamma(akt) - \pi(akt) = \emptyset$ **then** $új := \pi(akt)$
else $új :=$ **select**$(\Gamma(akt) - \pi(akt))$

Lokális kereséssel megoldható feladatok

- Erős heurisztika nélkül nincs sok esély a cél megtalálására.
  - Jó heurisztikára épített kiértékelő függvénnyel 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.)

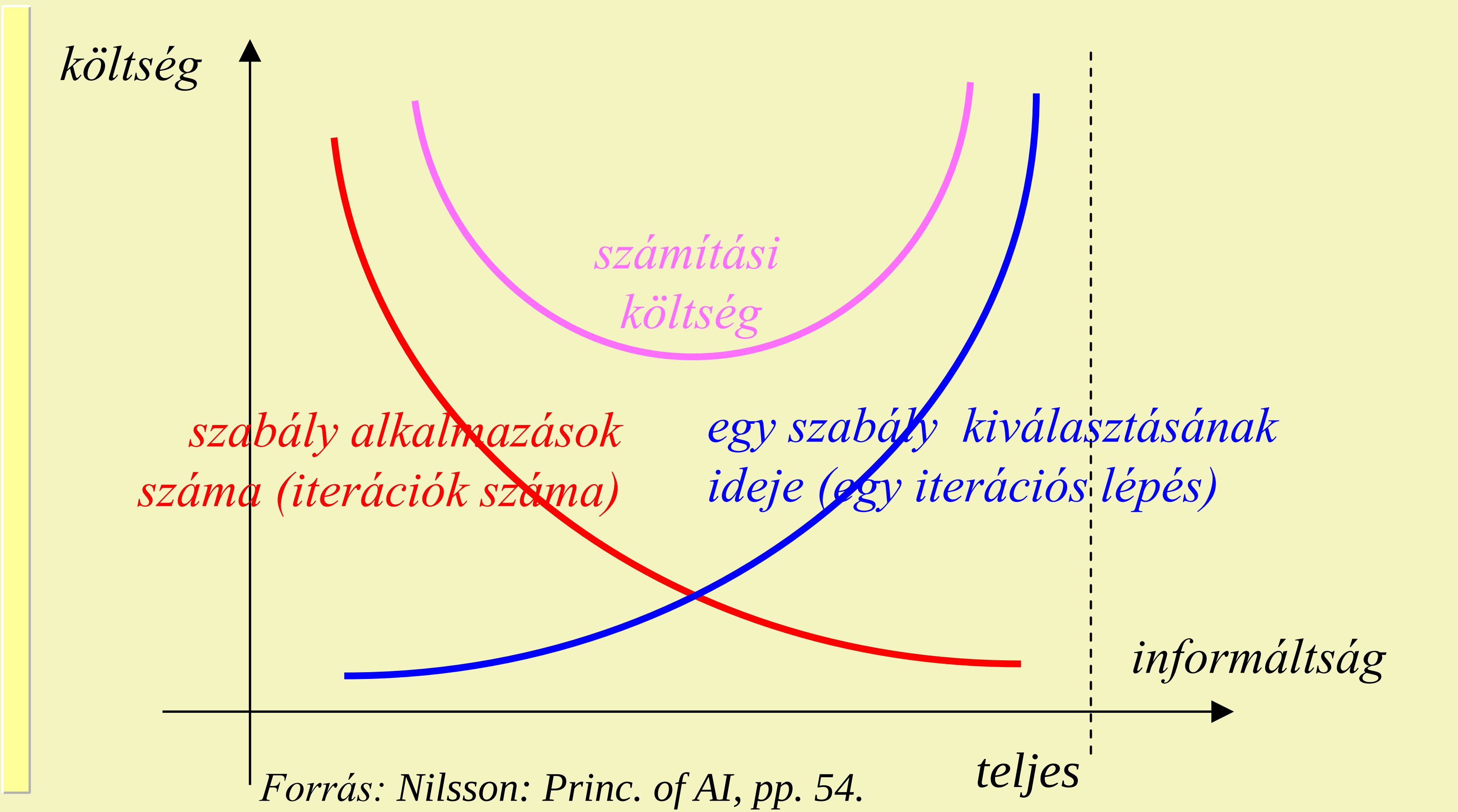
A heurisztika hatása a KR működésére

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

KR hatékonysága



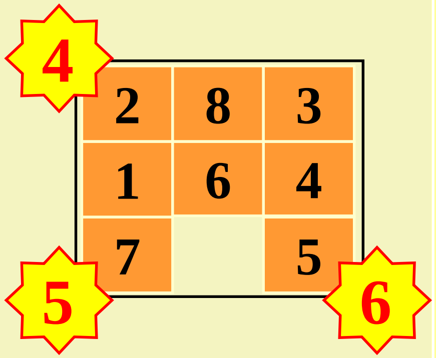
Forrás: Nilsson: Princ. of AI, pp. 54.

1	2	3
8		4
7	6	5

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

- Rossz helyen levő lapkák száma:

$$W(this) = \sum_{\substack{i,j \\ this[i,j]\neq 0 \,\wedge\, this[i,j]\neq cél[i,j]}} 1$$



- 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

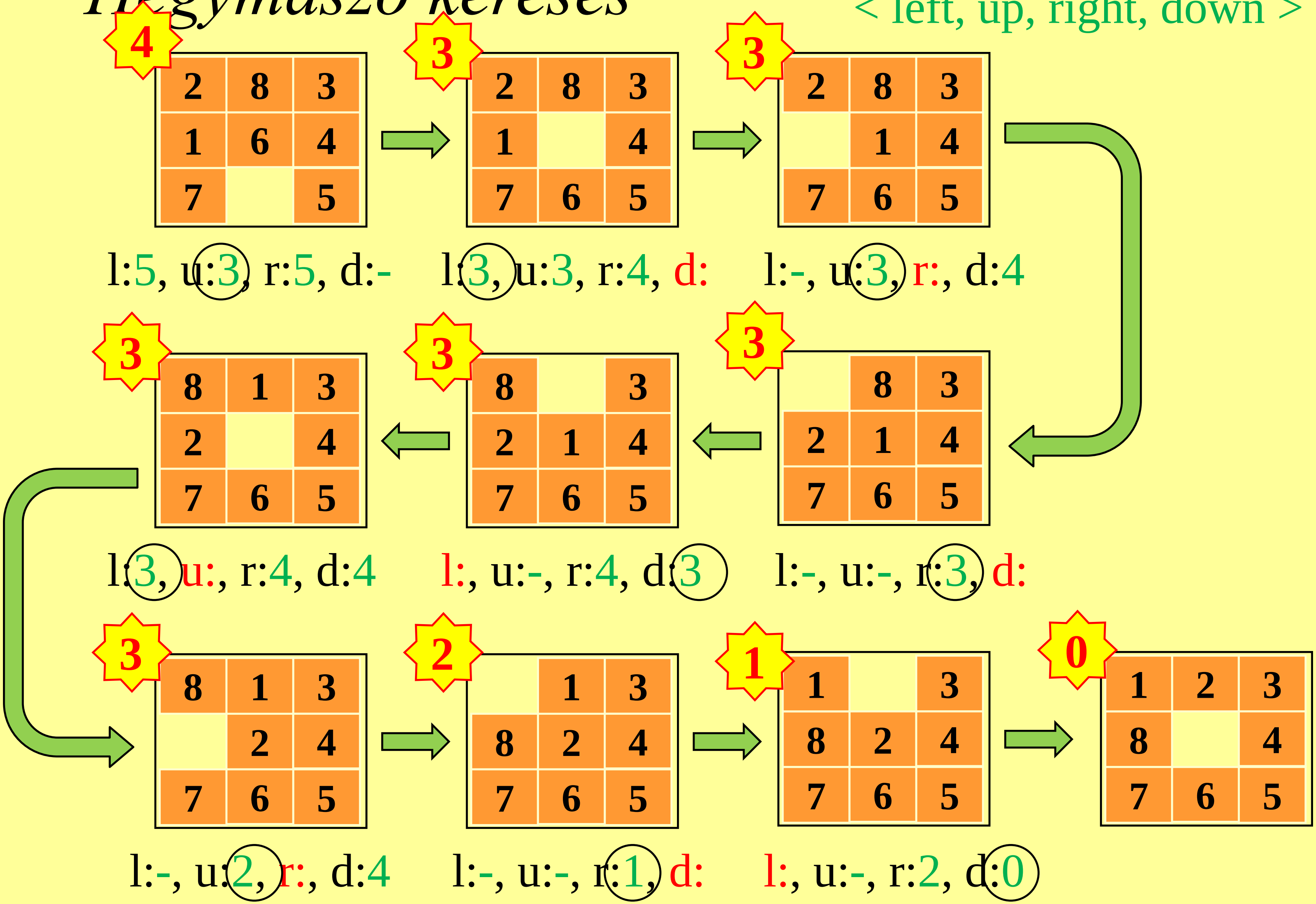
célbelioszlop(*this*[*i*,*j*]) ~ a *this*[*i*,*j*] célállapotbeli helyének oszlopa

- „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 (×2) nincs még a cél szerinti lapka?

W

Hegymászó keresés

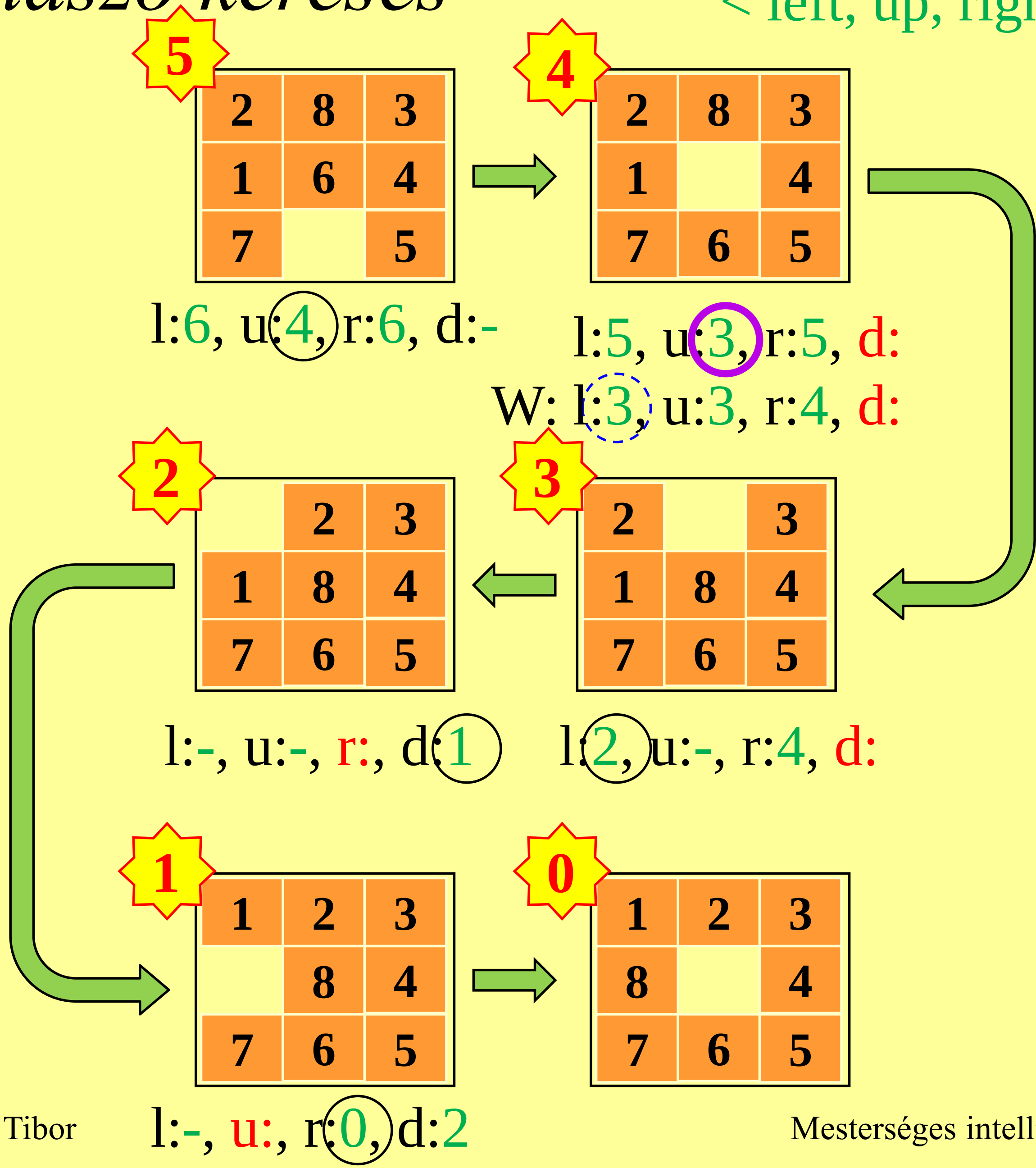
Csúcs utódainak sorrendje:
< left, up, right, down >



P

Hegymászó keresés

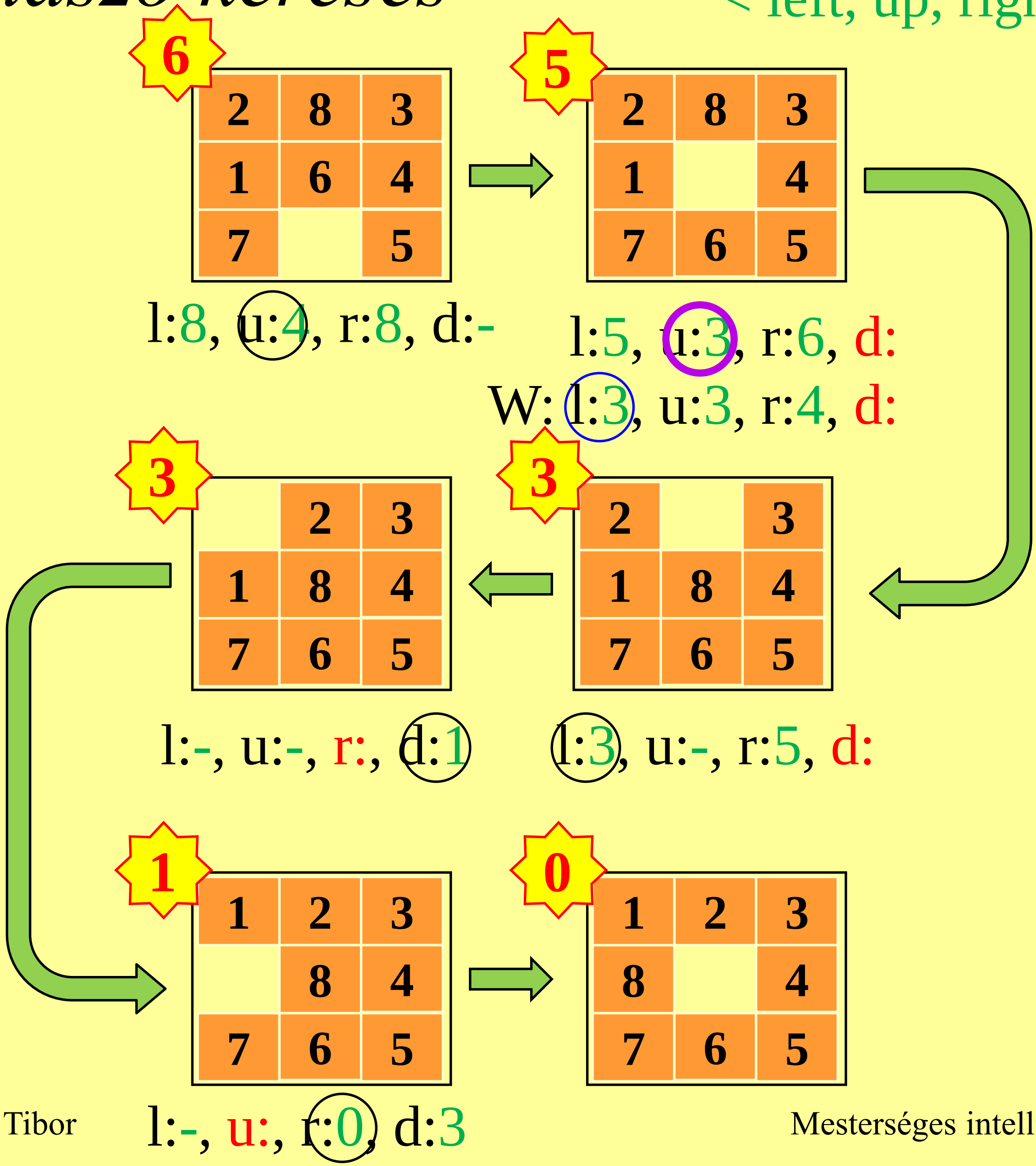
Csúcs utódainak sorrendje:
< left, up, right, down >

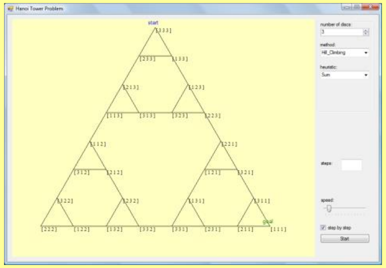


F

Hegymászó keresés

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

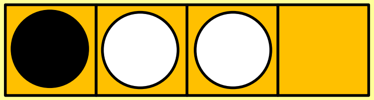




Heurisztikák a Hanoi tornyai problémára

- 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: $EWS(this) = WS(this) -$

$$- \sum_{\substack{i=2..n \\ this[i-1]>this[i]}} 1 \; + \sum_{\substack{i=2..n-1 \\ this[i-1]=this[i+1] \,\wedge\, this[i]\neq this[i-1]}} 2$$

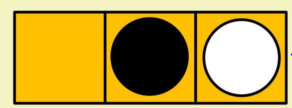
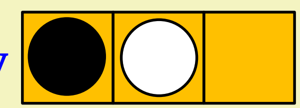


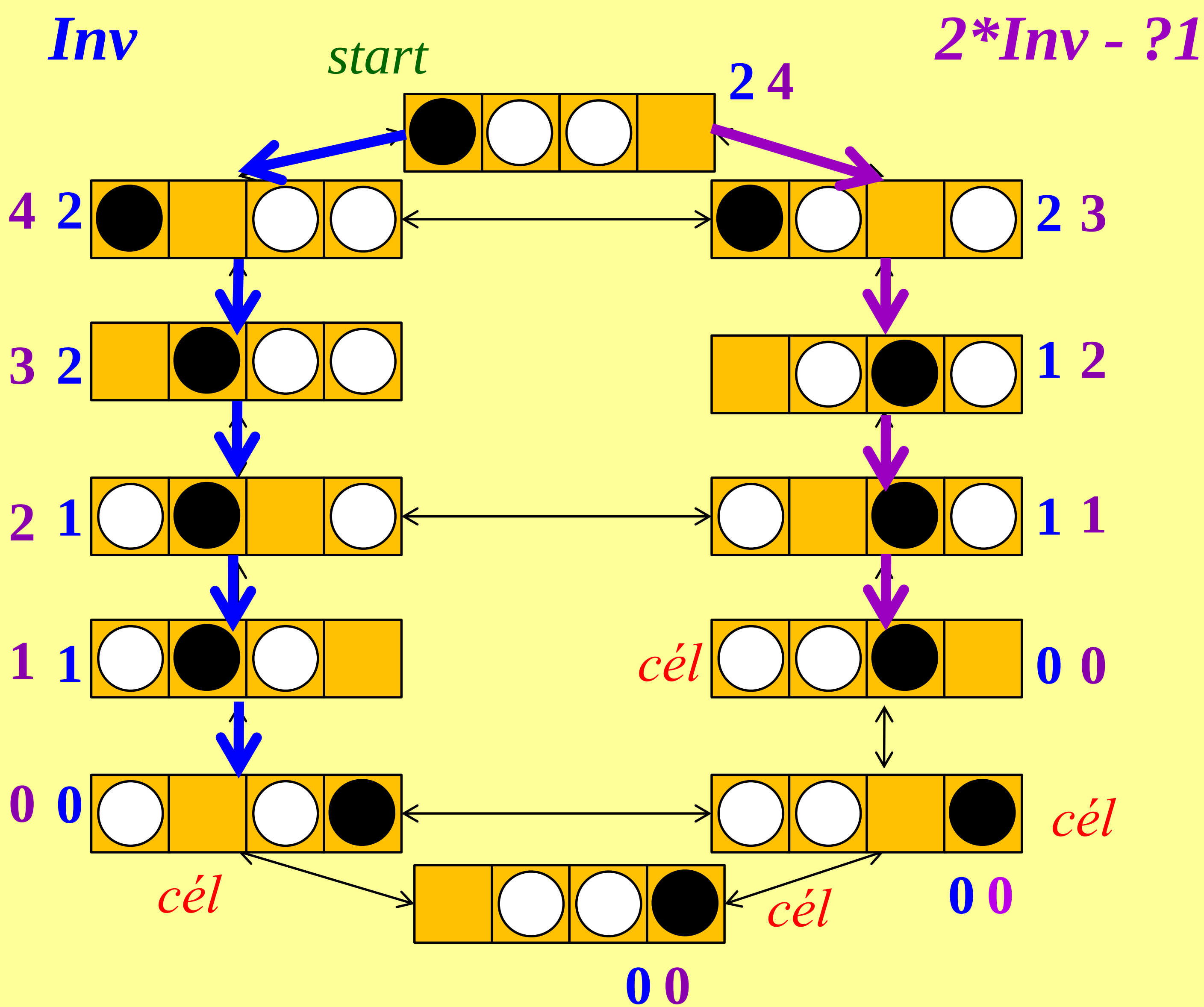
Heurisztikák a Fekete-fehér kirakóra

- Inverziószám:

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

- Módosított inverziószám:

  $M(this) = 2 \cdot I(this) -$
  – (*1*, **ha** *this*-nek része  **vagy** )



Inv
2*Inv - ?1
start
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