# OSNOVE UMETNE INTELIGENCE

## 2021/22

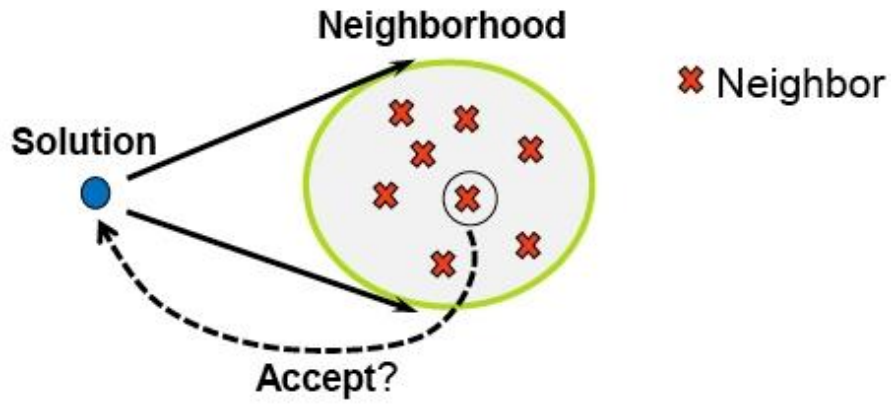
*lokalno preiskovanje*

# Pridobljeno znanje s prejšnjih predavanj

- **informirano (hevristično) preiskovanje**
  - **požrešno iskanje**
    - vedno razvijemo najbolj obetavno vozlišče glede na hevristično oceno
    - vozlišča urejena v prioritetni vrsti
    - $f(n) = h(n)$
    - nepopoln, neoptimalen, možnost ciklanja
  - **A***
    - $f(n) = g(n) + h(n)$
    - vozlišča urejena v prioritetni vrsti
    - popoln in optimalen, če je hevristika dopustna (ne precenjuje cene do cilja)
  - **IDA***
    - podobno iterativnem poglabljanju, vendar povečujemo mejo funkcije $f(n)$
    - neučinkovito, če prevladujejo vozlišča z raznolikimi vrednostmi funkcije $f(n)$
    - IDA* razvija vozlišča v prioritetnem vrstnem redu, če je hevristika monotona
  - **kakovost hevrističnih funkcij**
    - vpliva na efektivno vejanje in število generiranih vozlišč
    - želimo dopustne hevristike s čim višjimi vrednostmi in s sprejemljivim časom izračuna
    - hevristike pridobimo: s poenostavljanjem problema, z vzorci podproblemov, z uteževanjem kriterijev

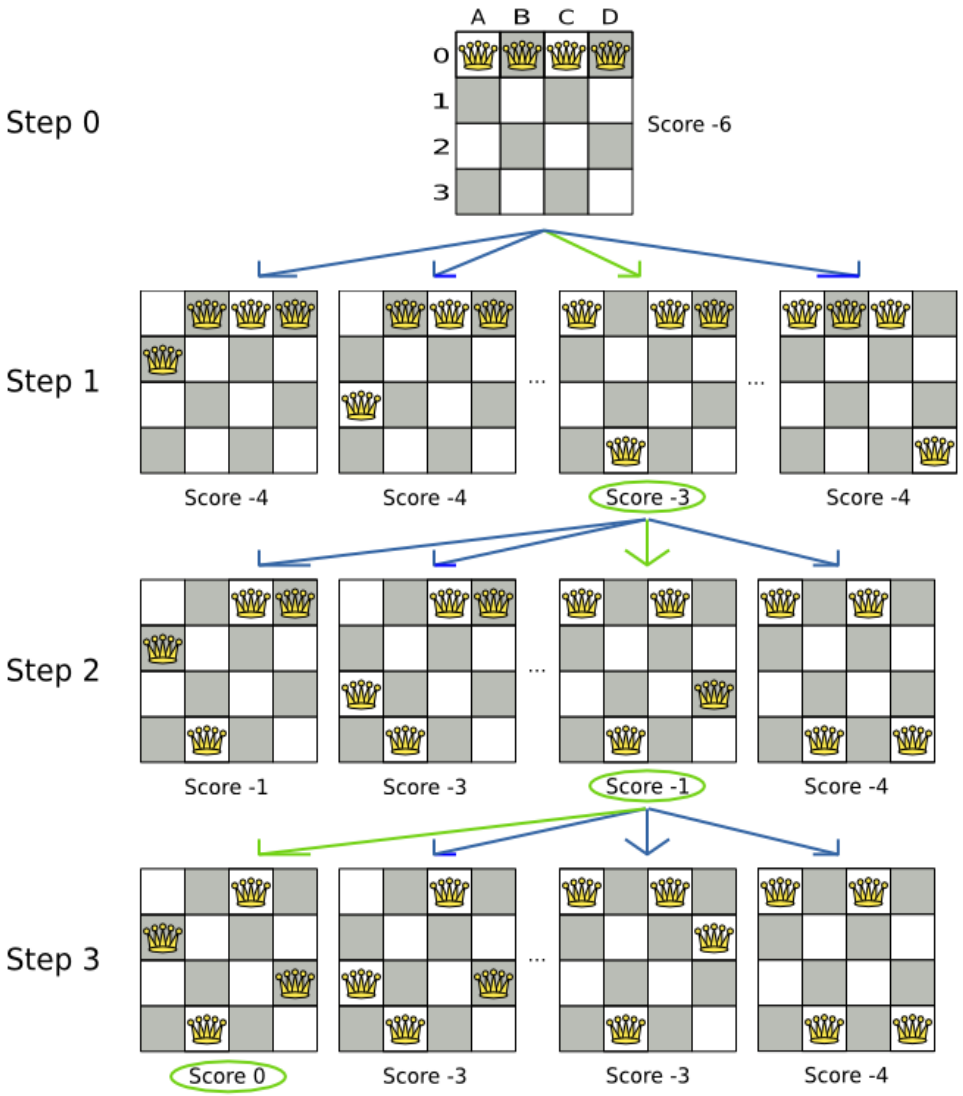
# Lokalni preiskovalni algoritmi

- namesto sistematičnega preiskovanja možnih poti od začetka do cilja izvajajo **iterativno ocenjevanje** in spreminjanje podanih stanj
  - izberi **začetno** množico stanj (eno ali več njih)
  - poišči sosednja stanja od trenutnega, pri tem **ni nujno ohranjati poti**
  - ponavljaj do **ustavitvenega pogoja**
- koristni v primerih:
  - če nas primarno zanima samo kakovost rešitve (stanja) in ne nujno tudi pot do cilja (primer: pri igri 8 ploščic je pot pomembna, pri problemu 8 kraljic pa ne)
  - za reševanje optimizacijskih problemov, kjer je podana **kriterijska funkcija** za oceno kakovosti rešitve
- prednosti:
  - **majhna poraba** prostora
  - v praksi najdejo **dober približek** rešitve v prostorih, ki so s sistematičnimi preiskovalnimi algoritmi neobvladljivi



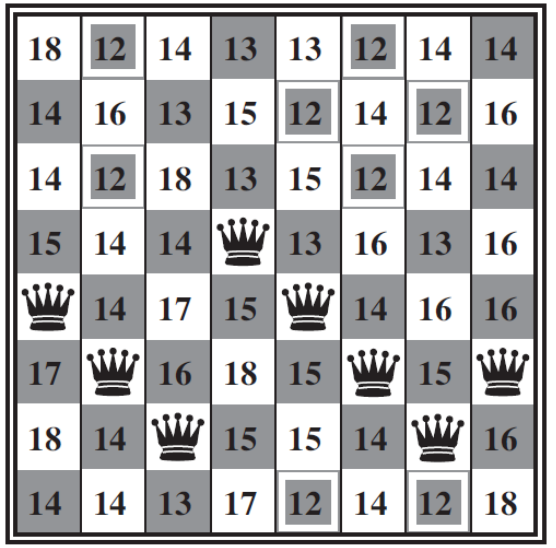
# Lokalni preiskovalni algoritmi

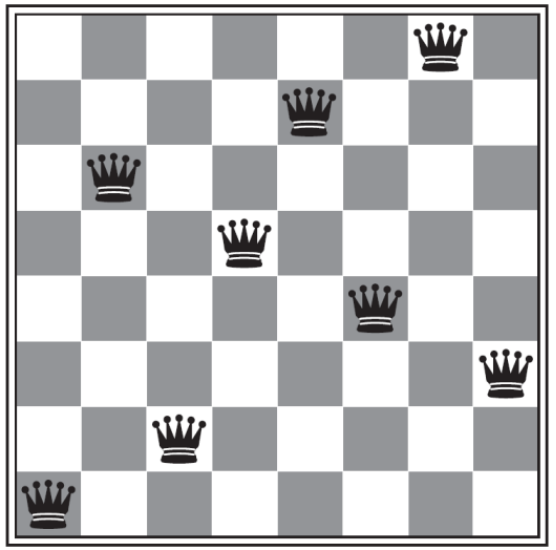
- **primer**: 4 kraljice na šahovnici
- **kriterijska funkcija**: maksimiziramo – (minus) število kraljic, ki se medsebojno napadajo
- če želimo ohraniti zaporedje korakov, ki vodijo do rešitve, je najdena **iskalna pot** pomembna, sicer pa ne; najdena iskalna pot za desni problem:
  - B0 na B3
  - D0 na D2
  - A0 na A1



# Plezanje na hrib

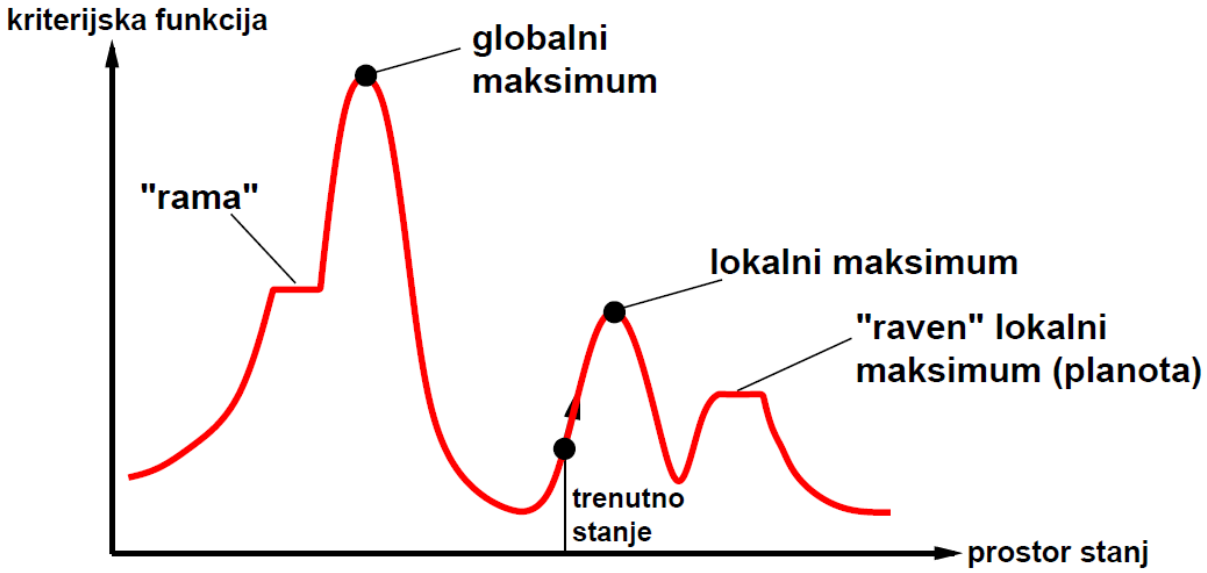
- angl. *hill-climbing search* (ali: *greedy local search*)
- premikaj se po prostoru stanj v smeri najboljše izboljšave kriterijske funkcije
- primer problema 8 kraljic:
  - kot "soseda" trenutnega stanja definiramo stanje, kjer 1 kraljico premaknemo na drugo polje znotraj istega stolpca
  - skupaj: 8×7=56 sosednih stanj
  - kriterijska funkcija: minimiziramo število kraljic, ki se napadajo (na zgornji sliki je prikazana vrednost kriterijske funkcije, če kraljico iz vsakega stolpca premaknemo na izbrano mesto znotraj stolpca)
  - lokalno iskanje lahko "obtiči" v lokalnem optimumu (primer na spodnji sliki: h=1, vendar ima vsak sosed stanja višjo vrednost funkcije)
  - v 14% lokalno iskanje najde rešitev v 4 korakih, v 86% obtiči v lokalnem maksimumu po 3 korakih (vseh stanj je 17 milijonov)

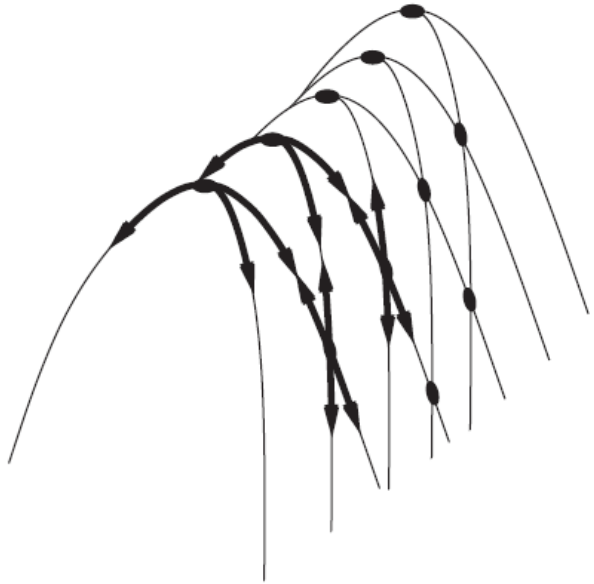


# Lokalni preiskovalni algoritmi

- preiskujejo prostor stanj z **namenom** najti **globalni maksimum** glede na vrednost kriterijske funkcije
  - optimalni algoritem: najde globalni maksimum
- težave:
  - **lokalni maksimumi** ("vrhovi")
  - področja, kjer ima kriterijska funkcija **konstantno vrednost** (rame, planote)
  - **grebeni** (za plezanje navzgor je najprej potreben sestop po pobočju grebena)



# Plezanje na hrib

- možnost obtičanja v lokalnem maksimumu (najden cikel pri preiskovanju)


https://docs.optaplanner.org/6.5.0.CR1/optaplanner-docs/html/ch10.html#hillClimbing

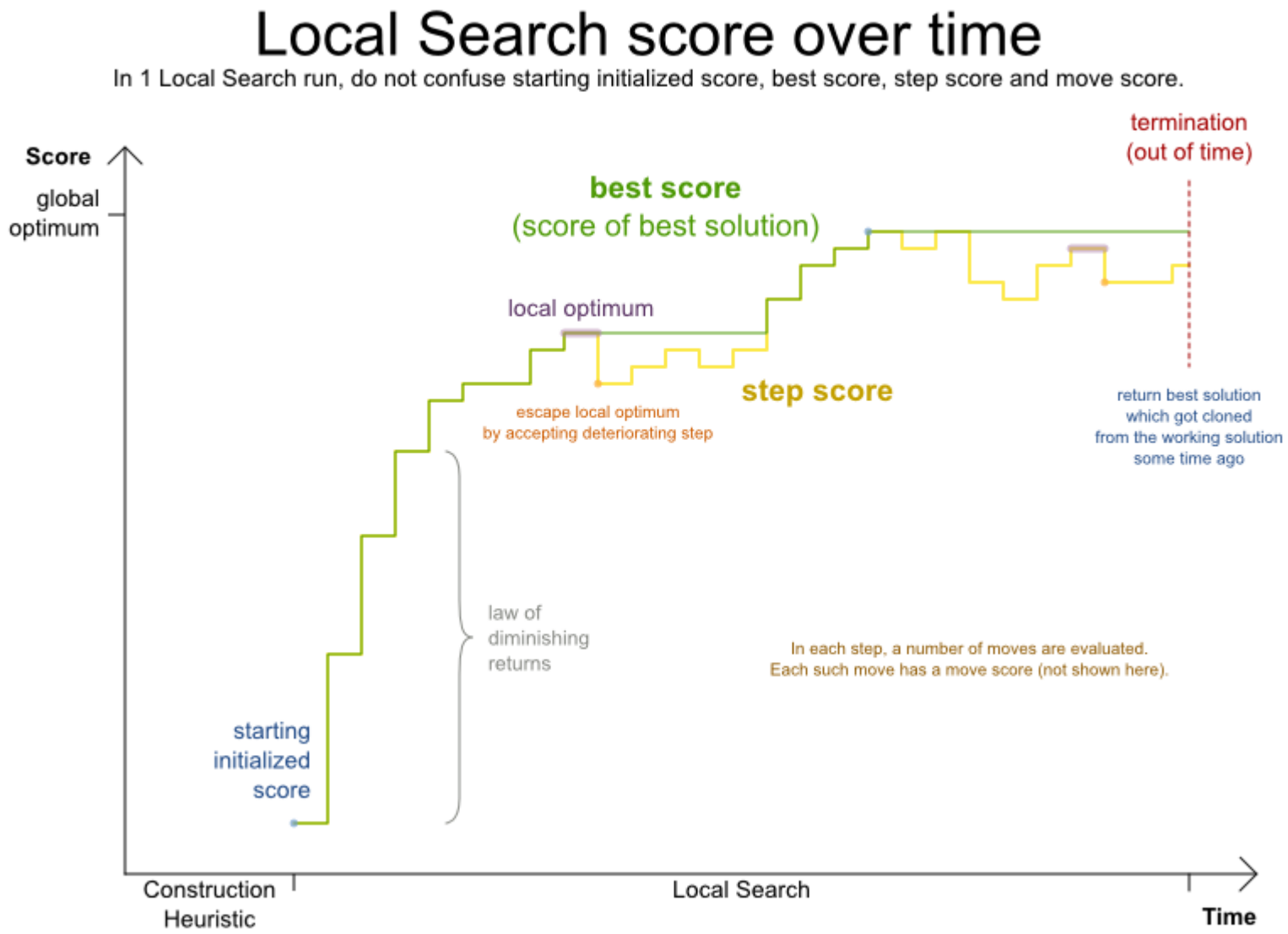
# Reševanje iz lokalnih maksimumov

- **koraki vstran**: če ima naslednje stanje isto vrednost kriterijske funkcije, dovolimo premik v to stanje (korak "vstran", angl. *sideways move)*
  - upamo, da smo na "rami" in ne na "planoti"
  - smiselno je omejiti število dovoljenih korakov vstran
  - primer: pri problemu 8 kraljic verjetnost uspeha naraste s 14% na 94%

- **stohastično** plezanje na hrib: iz množice boljših stanj lahko z določeno verjetnostjo izberemo tudi slabše stanje (pri čemer upoštevamo, da imajo boljša stanja večjo verjetnost izbora)

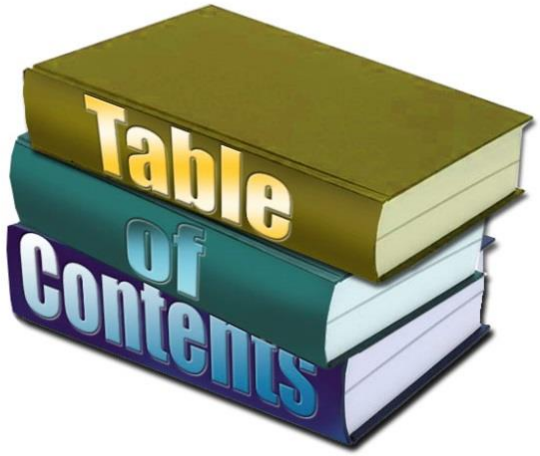
- **naključni ponovni zagon**: večkrat poženi plezanje na hrib iz naključnih začetnih stanj, dokler ne najdeš rešitve
  - če je verjetnost uspeha enega zagona $p$, je v povprečju potrebnih $1/p$ zagonov
  - za problem 8 kraljic:
    - $p = 0.14 \Rightarrow$ potrebnih je 7 zagonov (skupaj približno 22 korakov)
    - če dovolimo tudi korake vstran ($p = 0{,}94$), je potrebnih $1/0.94 \approx 1{,}06$ zagonov

# Reševanje iz lokalnih maksimumov




https://docs.optaplanner.org/6.5.0.CR1/optaplanner-docs/html/ch10.html#hillClimbing

# Pregled




- **preiskovanje prostora stanj**
  - neinformirani preiskovalni algoritmi
  - informirani preiskovalni algoritmi

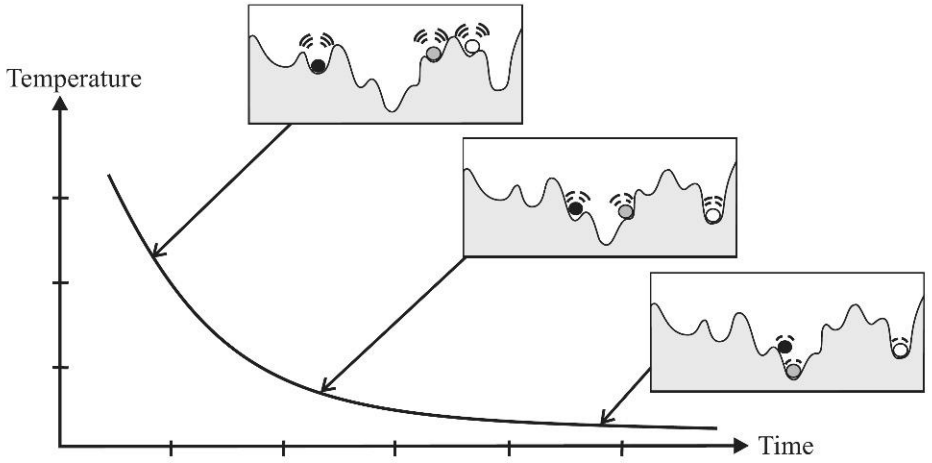
  - lokalni preiskovalni algoritmi
    - plezanje na hrib (hill-climbing)
    - simulirano ohlajanje
    - lokalno iskanje v snopu

  - preiskovanje grafov AND/OR
    - predstavitev problemov z grafi AND/OR
    - algoritem AO*

# Simulirano ohlajanje

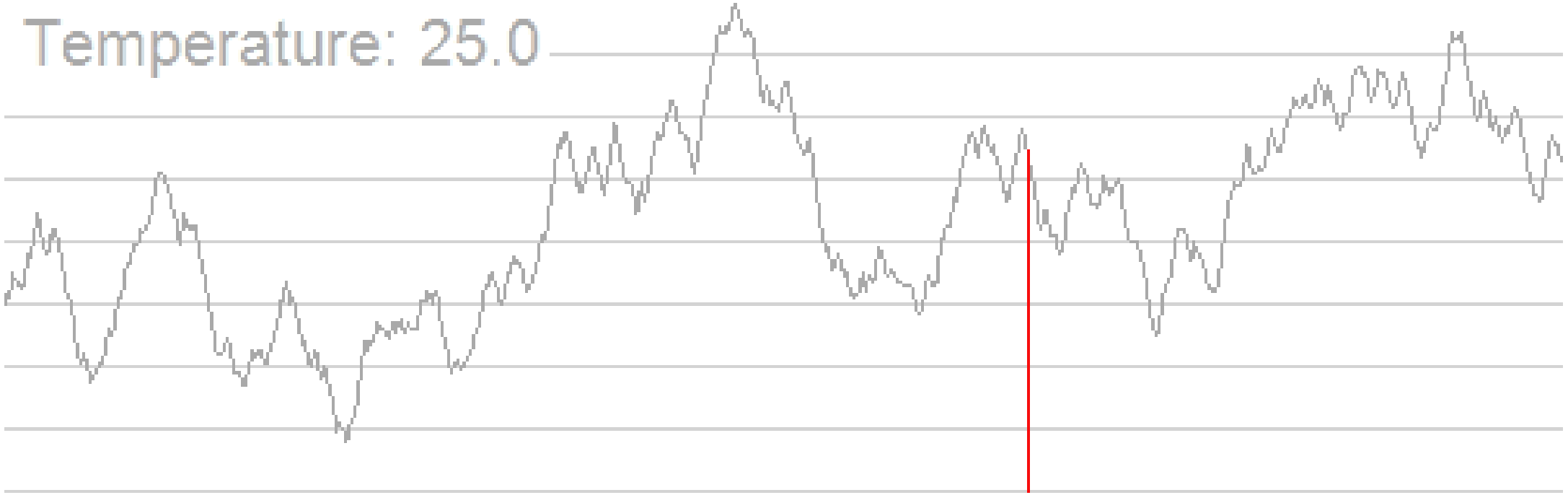
- angl. *simulated annealing*
- optimizacijski algoritem, ki izvira iz fizikalnih lastnosti v metalurgiji (ko je jeklo tekoče, so molekule v njem bolj gibljive; ko se ohlaja, se strjuje in molekule se umirjajo)
- analogija:
  - generiramo **naključne sosede** trenutnega stanja
  - če najdemo **boljše stanje, ga vedno izberemo**
  - če najdemo **slabše stanje, ga izberemo z določeno verjetnostjo**
  - verjetnost izbire neoptimalnega stanja s časom pada (nižanje temperature)
- analogija: zibanje igralne površine, da žogice skočijo iz lokalnih optimumov (na sliki: iščemo lokalne minimume)

```
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
    ΔE ← VALUE[next] – VALUE[current]
    if ΔE > 0 then current ← next
    else current ← next only with probability e^(Δ E/T)
```



# Simulirano ohlajanje - demo

- iskanje globalnega maksimuma

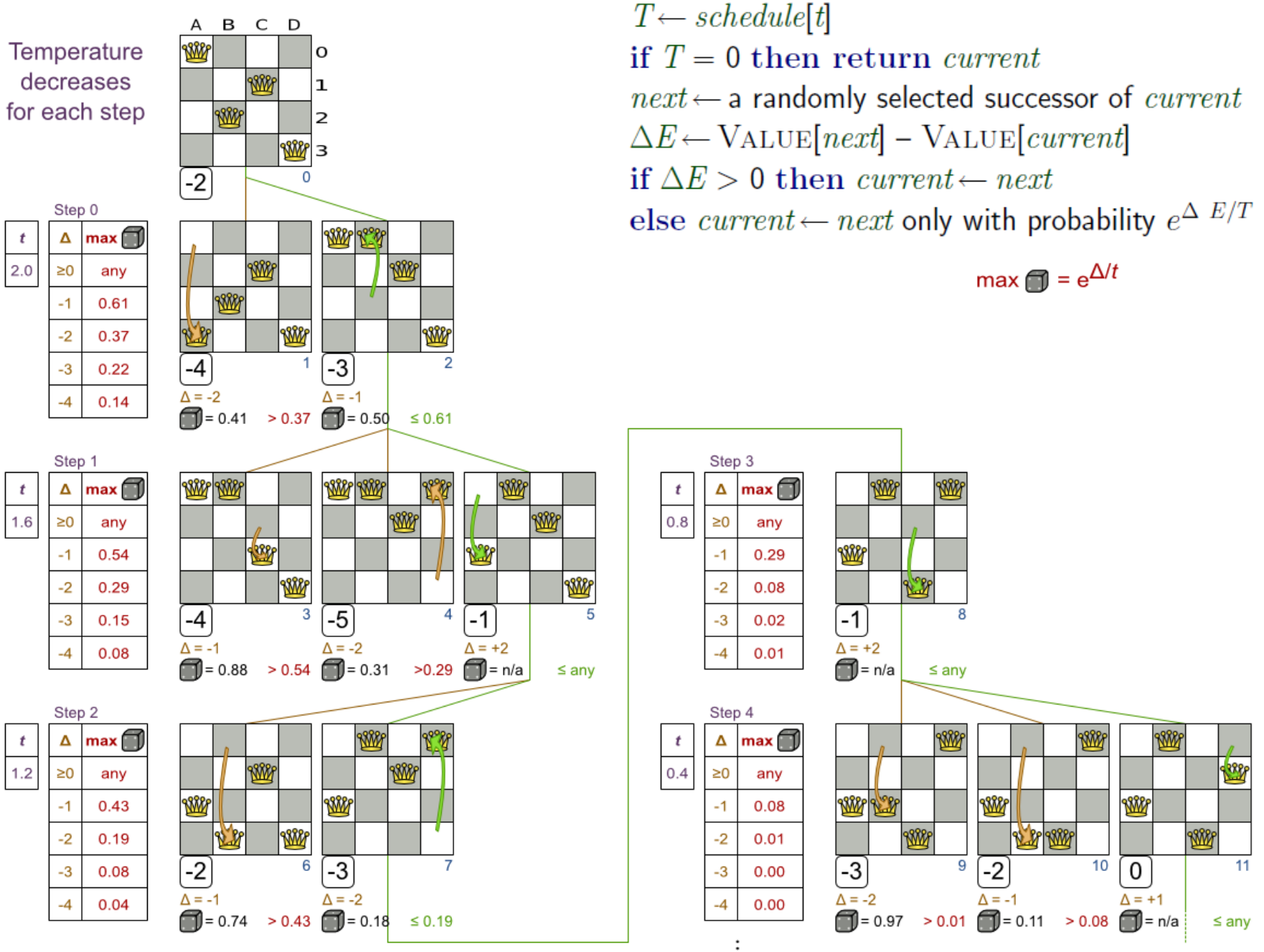



- druga simulacija

https://medium.com/@osanseviero/one-month-in-ai-60b340e83cf3

# Simulirano ohlajanje

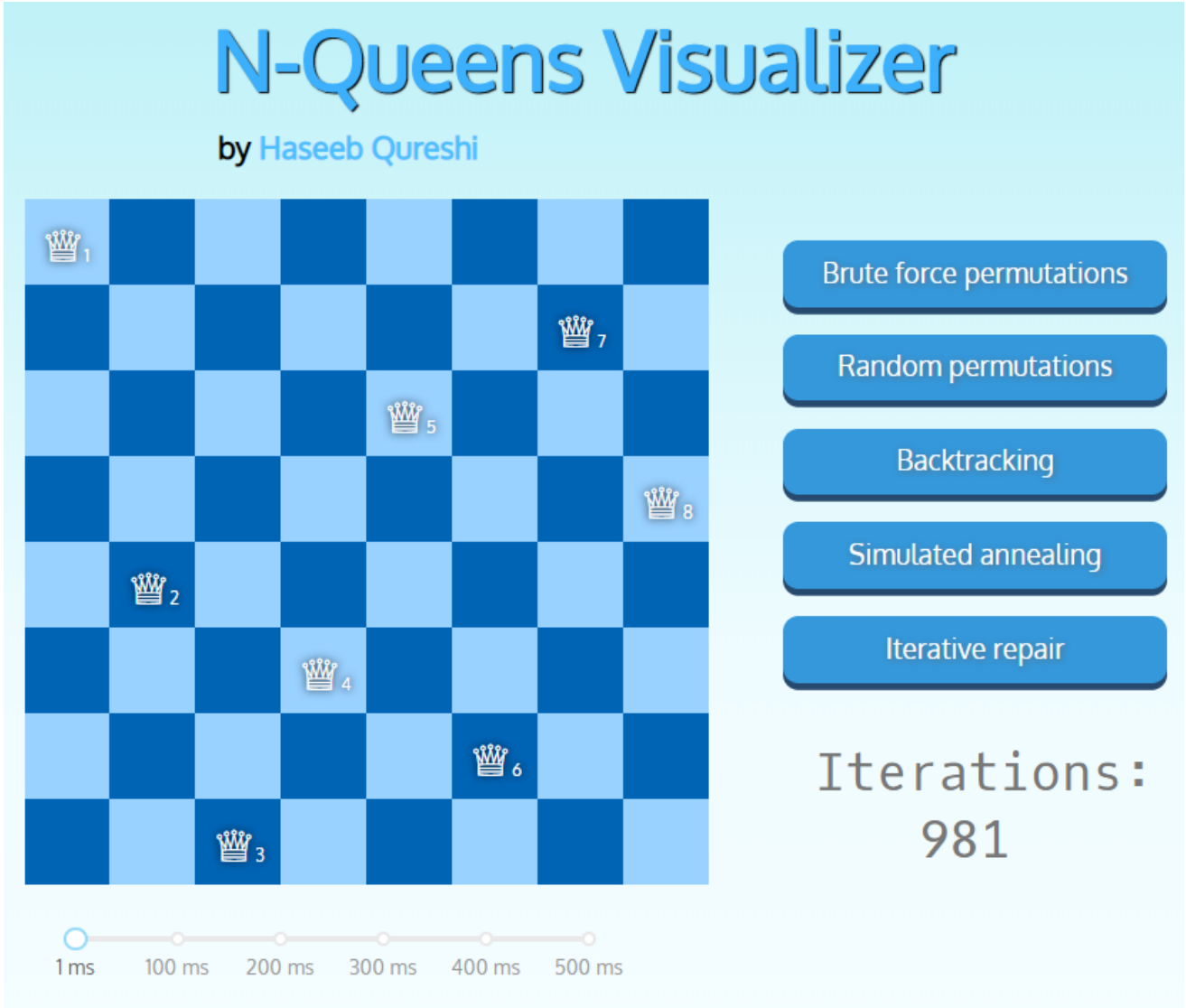
- za vsako slabše vozlišče naključno odločimo, ali ga izberemo kot naslednika
- če je vozlišče boljše, ga vedno izberemo za naslednika
- sčasoma (s padcem temperature) simulirano ohlajanje preide v navadno plezanje na hrib




https://docs.optaplanner.org/6.5.0.CR1/optaplanner-docs/html/ch10.html#hillClimbing

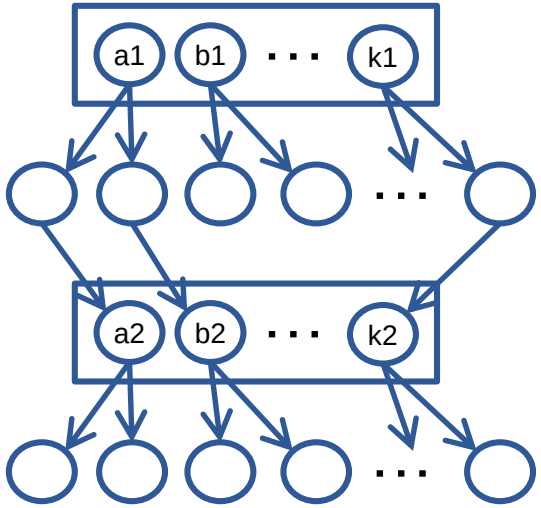
# Demo

- https://haseebq.com/n-queens-visualizer/



# Lokalno iskanje v snopu

- algoritem:
  - v spominu hrani ***k* aktualnih stanj** namesto enega
  - izberi ***k* najboljših stanj med sosedi** množice aktualnih stanj
  - ponavljaj do ustavitvenega pogoja
- ni enako kot *k* vzporednih iskanj, ker ocenjujemo **kakovost cele generacije** iskanj (ne neodvisnih vzporednih iskanj)
- problemi:
  - celoten snop k iskanj obtiči v lokalnih maksimumih
  - rešitev: stohastično iskanje v snopu: izberi naključne naslednike z verjetnostjo, ki je sorazmerna njihovi kakovosti




generiraj *k* naključnih začetnih stanj

generiraj sosede

izberi *k* najboljših naslednikov

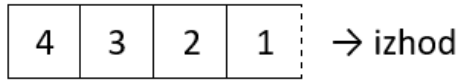
ponavljaj…

# Primer izpitne naloge

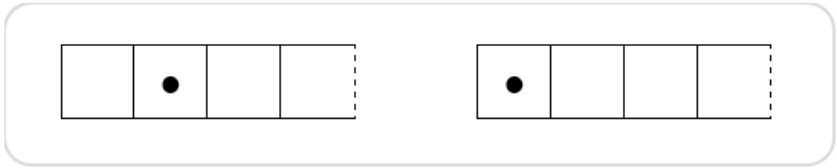
- 3. izpit, 7. 9. 2018

**3. NALOGA (25t):**

Rešujemo problem iskanja izhoda iz hodnika, ki ga sestavljajo štiri sobe (prikazano na desni sliki). V eni izmed sob se nahaja robot, ki se lahko na vsakem koraku lahko premakne za 1 sobo v levo (L) ali v desno (D).




Problem želimo rešiti s preiskovanjem v snopu, ki na vsakem koraku hrani 2 aktualni stanji (k=2). Preiskovanje začnemo s stanjema, ki sta prikazani na spodnji sliki (ti dve stanji smo izbrali naključno, pika prikazuje lokacijo robota). Kot kriterijsko funkcijo najdene rešitve uporabljamo razdaljo do izhoda (vrednost kriterijske funkcije za robota v vsaki posamezni sobi je prikazana na zgornji skici).



Naloge:

a) **(10t) Nariši zaporedje stanj pri preiskovanju, ki ga nadaljuj vse dokler ne najdemo končnega stanja (robot se premakne skozi desno (črtkano) stranico labirinta).**
b) **(10t) Denimo, da namesto "navadnega" iskanja v snopu uporabljamo stohastično iskanje v snopu. Za generirane rešitve v 2. (nasledniki začetnega vozlišča) in 3. iteraciji preiskovanja izračunaj verjetnosti, da bo sosed izbran za naslednjo iteracijo.**
c) **(5t) V katero družino algoritmov spada iskanje v snopu? Naštej še tri druge algoritme iz iste družine, ki jih poznaš.**

# Preiskovanje AND/OR grafov, igranje iger