OSNOVE UMETNE INTELIGENCE

2022/23

informirani preiskovalni algoritmi (IDA)*

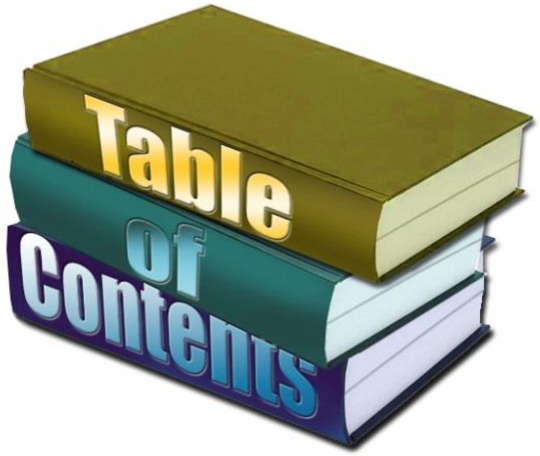
lokalni preiskovalni algoritmi

© Zoran Bosnić

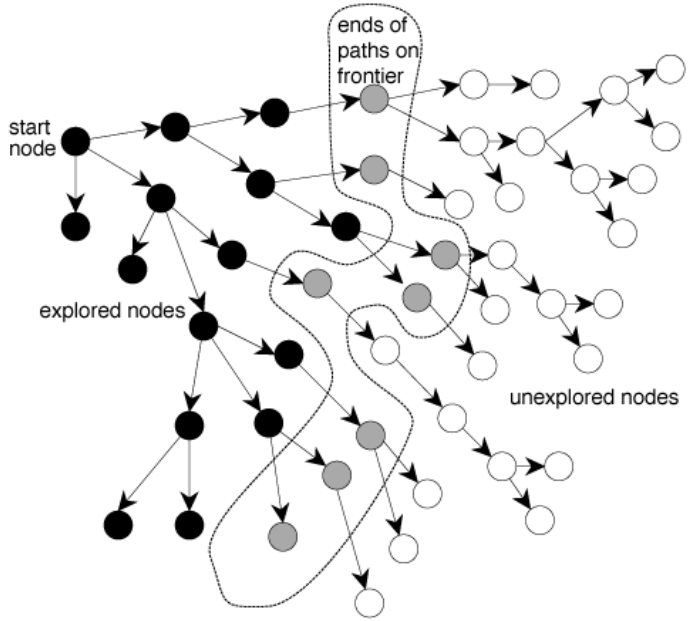
Pridobljeno znanje s prejšnjih predavanj

- **neinformirano preiskovanje**
  - **iterativno poglabljanje**
    - povečevanje omejitve globine
    - popoln, optimalen, časovna zahtevnost $O(b^d)$, prostorska zahtevnost $O(bd)$
    - kombinira prednosti iskanja v širino in iskanja v globino
  - **dvosmerno iskanje**
    - potrebno poznavanje končnega stanja
    - časovna zahtevnost $O(b^{d/2})$
  - **cenovno-optimalno iskanje**
    - razvija vozlišče, ki ima najmanjšo skupno ceno dosedanje poti – $g(n)$, fronta urejena kot prioritetna vrsta
    - popoln, optimalen, časovna in prostorska zahtevnost $O(b^{1+\lfloor C^*/\epsilon \rfloor})$
    - potrebna detekcija ciljnega vozlišča šele ob njegovem razvijanju
- **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)

Pregled



- preiskovanje prostora stanj
  - informirani preiskovalni algoritmi
    - hevristično preiskovanje (primer)
    - požrešno preiskovanje
    - A*
    - IDA*
    - kakovost hevrističnih funkcij
  - lokalni preiskovalni algoritmi in optimizacijski problemi
    - plezanje na hrib
    - simulirano ohlajanje
    - lokalno iskanje v snopu



Algoritem IDA*

- iterative-deepening A*
- deluje analogno kot **iterativno poglabljanje** (izvaja **preiskovanje v globino** z različnimi mejami), vendar za mejo ne uporabljamo globine, temveč vrednost funkcije $f(n)$
  - za mejo na začetku izberemo vrednost $f(n)$ začetnega vozlišča
  - na vsaki iteraciji razvijemo vsa vozlišča z $f(n) \leq$ mejni vrednosti
  - za naslednjo iteracijo izberemo mejo, ki je najmanjši $f(n)$ še nerazvitih vozlišč

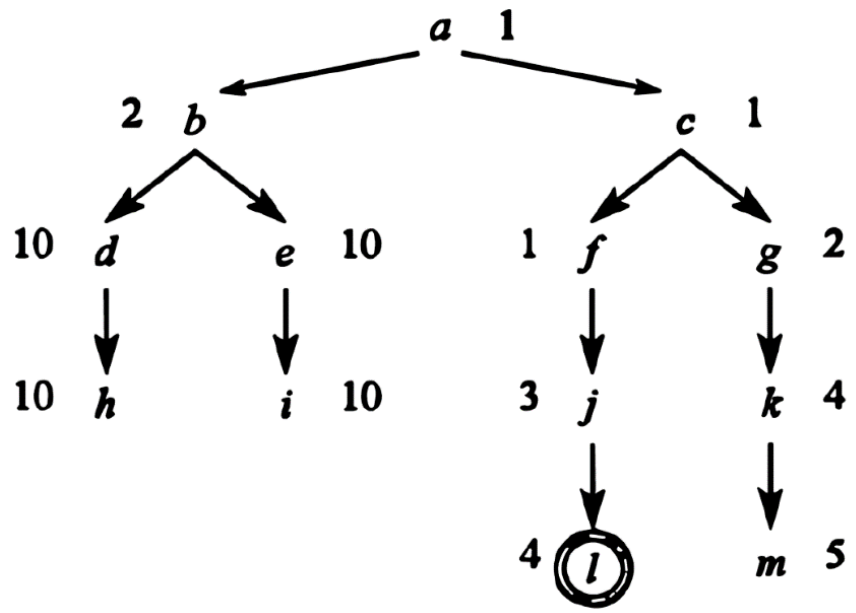
```
procedure ida_star(root)
  bound := h(root)
  path := [root]
  loop
    t := search(path, 0, bound)
    if t = FOUND then return (path, bound)
    if t = ∞ then return NOT_FOUND
    bound := t
  end loop
end procedure
```

```
function search(path, g, bound)
  node := path.last
  f := g + h(node)
  if f > bound then return f
  if is_goal(node) then return FOUND
  min := ∞
  for succ in successors(node) do
    if succ not in path then
      path.push(succ)
      t := search(path, g + cost(node, succ), bound)
      if t = FOUND then return FOUND
      if t < min then min := t
      path.pop()
    end if
  end for
  return min
end function
```

Psevdokoda: https://en.wikipedia.org/wiki/Iterative_deepening_A*

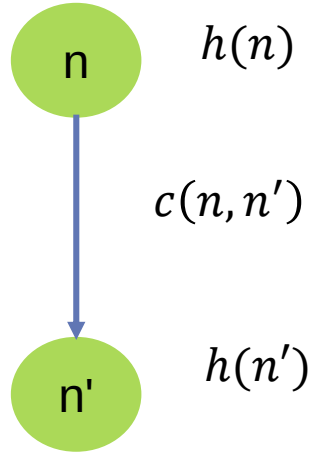
Algoritem IDA*

- primer:
  - podane so vrednost **f(n)** ($= g(n) + h(n)$) vozlišč
  - simuliraj preiskovanje z IDA*
- generirana vozlišča
  - 1. iteracija, meja=1: a/1, b/2, c/1, f/1, j/3, g/2
  - 2. iteracija, meja=2: a/1, b/2, d/10, e/10, c/1, f/1, j/3, g/2, k/4
  - 3. iteracija, meja=3: a/1, b/2, d/10, e/10, c/1, f/1, j/3, l/4, g/2, k/4
  - 4. iteracija, meja=4: a/1, b/2, d/10, e/10, c/1, f/1, j/3, **l/4**



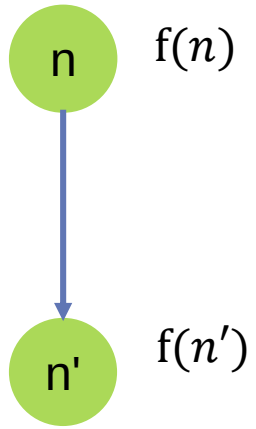
Učinkovitost algoritma IDA*

- redundanca: ponovno generiranje velikega števila vozlišč
- neučinkovit, če prevladujejo vozlišča z zelo raznolikimi vrednostmi funkcije $f(n)$
- vendar: v spominu hrani samo trenutno pot (podobno kot pri iskanju v globino) in ne vseh vozlišč kot A*
- želimo si, da je IDA* optimalen glede na: ceno najdene rešitve, porabljen prostor in porabljen čas:
  - to je možno, če IDA* razvije najmanjše potrebno število vozlišč, čemur rečemo, da jih razvija v *prioritetnem vrstnem redu*
  - IDA* razvija vozlišča v prioritetnem vrstnem redu, če je hevristična ocena $h(n)$ **monotona** ali **konsistentna** (monotone/consistent).
    To je res, kadar za vsaki povezani vozlišči n in n' velja (trikotniška neenakost):
    $h(n) \leq c(n, n') + h(n')$
    in $h(g) = 0$ za vsako končno vozlišče g



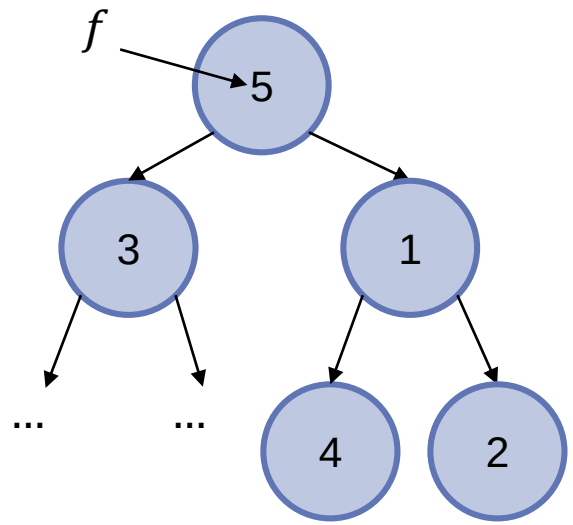
  - če je hevristika monotona/konsistentna, je tudi dopustna (!)

Učinkovitost algoritma IDA*



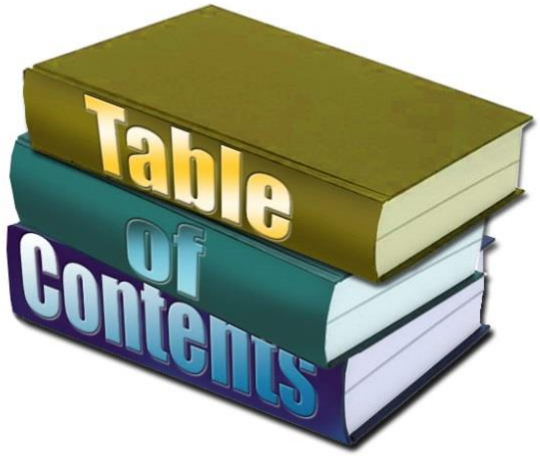
- poenostavitev: za zagotovitev razvijanja vozlišč v prioritetnem vrstnem redu ni nujno potrebno, da velja monotonost za hevristiko h, temveč **že zadošča, da je monotona cenilna funkcija f $(= g + h)$**
- cenilna funkcija f je monotona, če za vsak par povezanih vozlišč n in n' velja $f(n) <= f(n')$

- primer nemonotone funkcije f:
  - meja za f je enaka 5, vendar pa vozlišče z $f = 3$ razvijemo pred vozlišči z $f = 1$ in $f = 2$

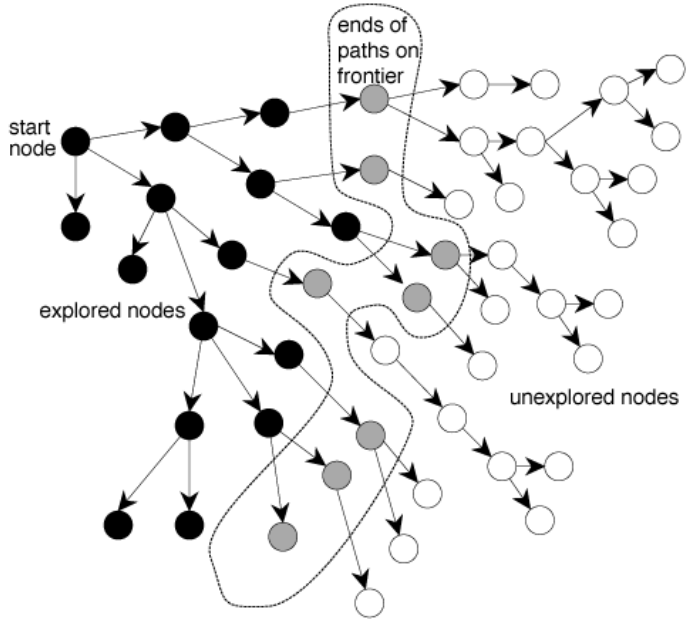


- premislek – izziv (objavi odgovor na forumu)!
  - Ali dopustna hevristična funkcija h zagotavlja, da je cenilna funkcija f monotona?
  - Ali za monotone cenilne funkcije f velja, da zadoščajo pogoju iz izreka o dopustnosti hevristik h?

Pregled

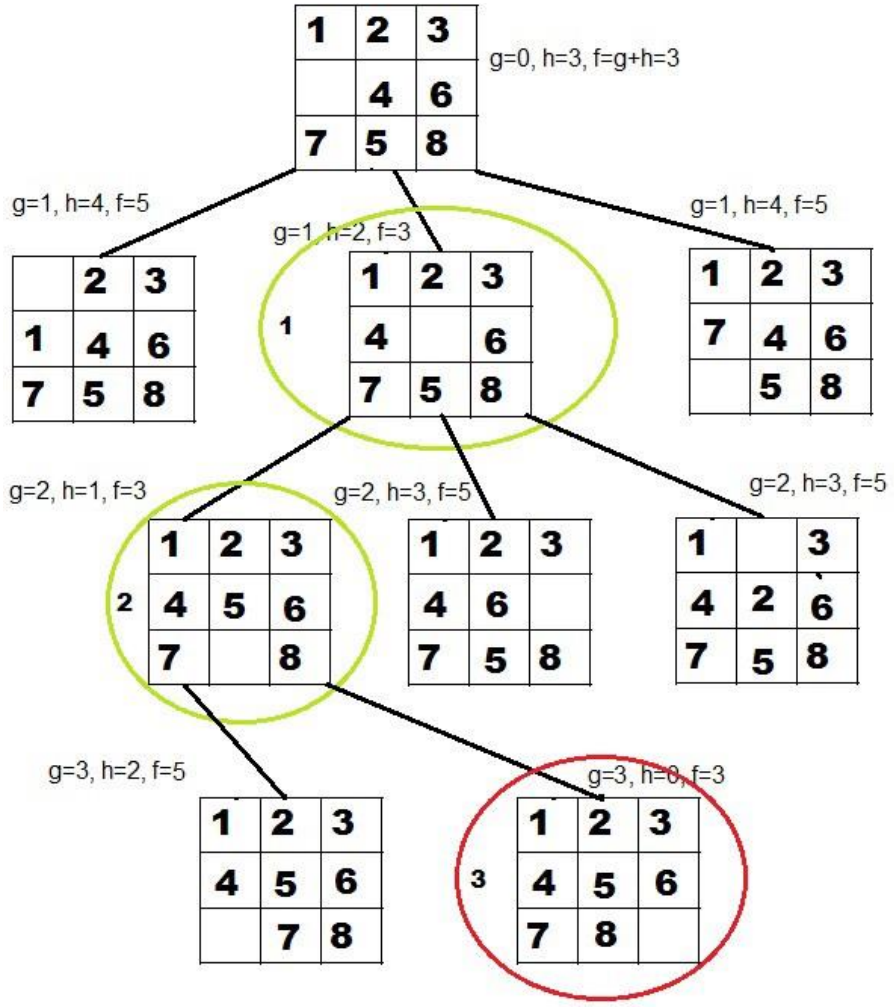


- **preiskovanje prostora stanj**
  - **informirani preiskovalni algoritmi**
    - hevristično preiskovanje (primer)
    - požrešno preiskovanje
    - A*
    - IDA*
    - kakovost hevrističnih funkcij
  - **lokalni preiskovalni algoritmi in optimizacijski problemi**
    - plezanje na hrib
    - simulirano ohlajanje
    - lokalno iskanje v snopu

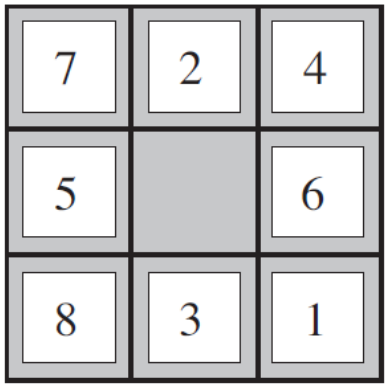


Kakovost hevrističnih funkcij

- primer: igra 8 ploščic
- povprečna dolžina rešitve je 22 korakov
- povprečni faktor vejanja je 3
  (2 potezi možni v vogalu, 3 ob robu, 4 v sredini)
- izčrpno preiskovanje bi torej pregledalo prostor 3^{22} stanj (za 3x3); na srečo je dosegljivih le približno 9!/2 = 181,440 stanj
- hevristična funkcija lahko učinkovito zmanjša prostorsko in časovno ceno iskanja
- pri iskanju ustrezne hevristike si pomagamo s poznavanjem problema



Kakovost hevrističnih funkcij

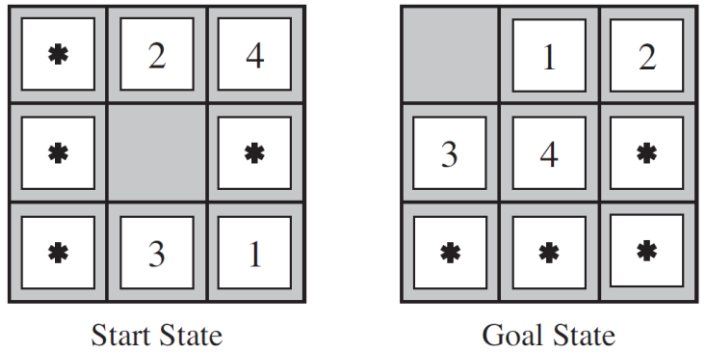


- primeri hevrističnih funkcij:
  - h_1 – število ploščic, ki niso na pravem mestu (za primer na desni: $h_1 = 8$)
  - h_2 – vsota manhattanskih razdalj ploščic do pravega mesta (za primer na desni: $h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 2 = 18$)
- kakovost h lahko ocenimo:
  - s številom generiranih vozlišč
  - z efektivnim faktorjem vejanja (koliko vozlišč N je algoritem generiral, da je na globini d našel rešitev)

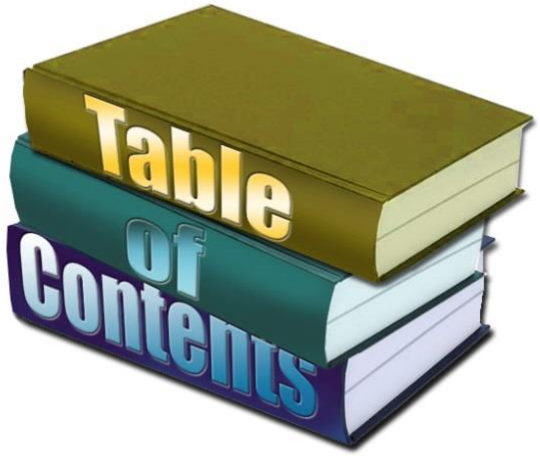
	število generiranih vozlišč			*efektivni faktor vejanja*		
Globina	**IDS**	**A*(h_1)**	**A*(h_2)**	**IDS**	**A*(h_1)**	**A*(h_2)**
2	10	6	3	2,45	1,79	1,79
4	112	13	12	2,87	1,48	1,45
6	680	20	18	2,73	1,34	1,30
8	6384	39	25	2,80	1,33	1,24
10	47127	93	39	2,79	1,38	1,22
12	3644035	227	73	2,78	1,42	1,24
14	?	539	113	?	1,44	1,23
16	?	1301	211	?	1,45	1,25
18	?	3056	363	?	1,46	1,26
20	?	7276	676	?	1,47	1,27
22	?	18094	1219	?	1,48	1,28
24	?	39135	1641	?	1,48	1,26

Kako do idej za hevristike?

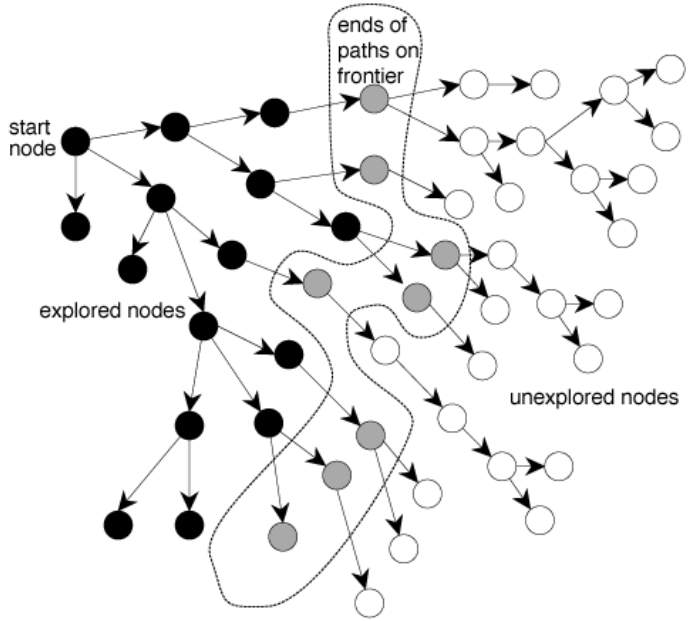
- želimo imeti dopustne hevristike:
  - s čim višjimi vrednostmi
  - s sprejemljivo ceno (časom) izračuna
- v prejšnjem primeru je h_2 boljša od h_1 (ker $h_2(n) \geq h_1(n)$ za vsak n, pravimo, da h_2 **dominira** h_1)
- pridobivanje hevristik:
  - iz poenostavljenega (relaksiranega) problema:
    - "ploščico lahko prestavimo na poljubno (tudi neprazno) polje"
    - "ploščico lahko prestavimo na poljubno (tudi nesosednje) prazno polje"
    - "ploščico lahko prestavimo na sosednje (tudi neprazno) polje"
  - z vzorci podproblemov (osredotočimo se npr. samo na iskanje rešitve za del problema)
  - z izkušnjami in uteževanjem kriterijev (npr. oddaljenost od cilja, število sosednjih ploščic, ki ne mejijo na ciljno mesto ipd.)



Pregled

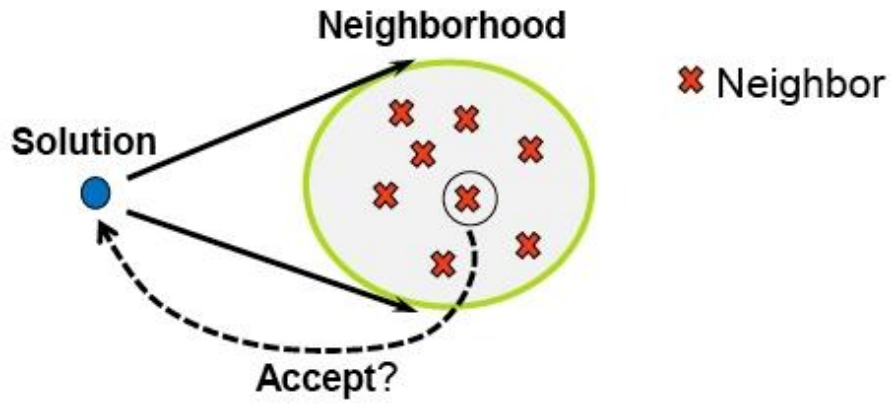


- **preiskovanje prostora stanj**
  - **informirani preiskovalni algoritmi**
    - hevristično preiskovanje (primer)
    - požrešno preiskovanje
    - A*
    - IDA*
    - kakovost hevrističnih funkcij
  - **lokalni preiskovalni algoritmi in optimizacijski problemi**
    - plezanje na hrib
    - simulirano ohlajanje
    - lokalno iskanje v snopu



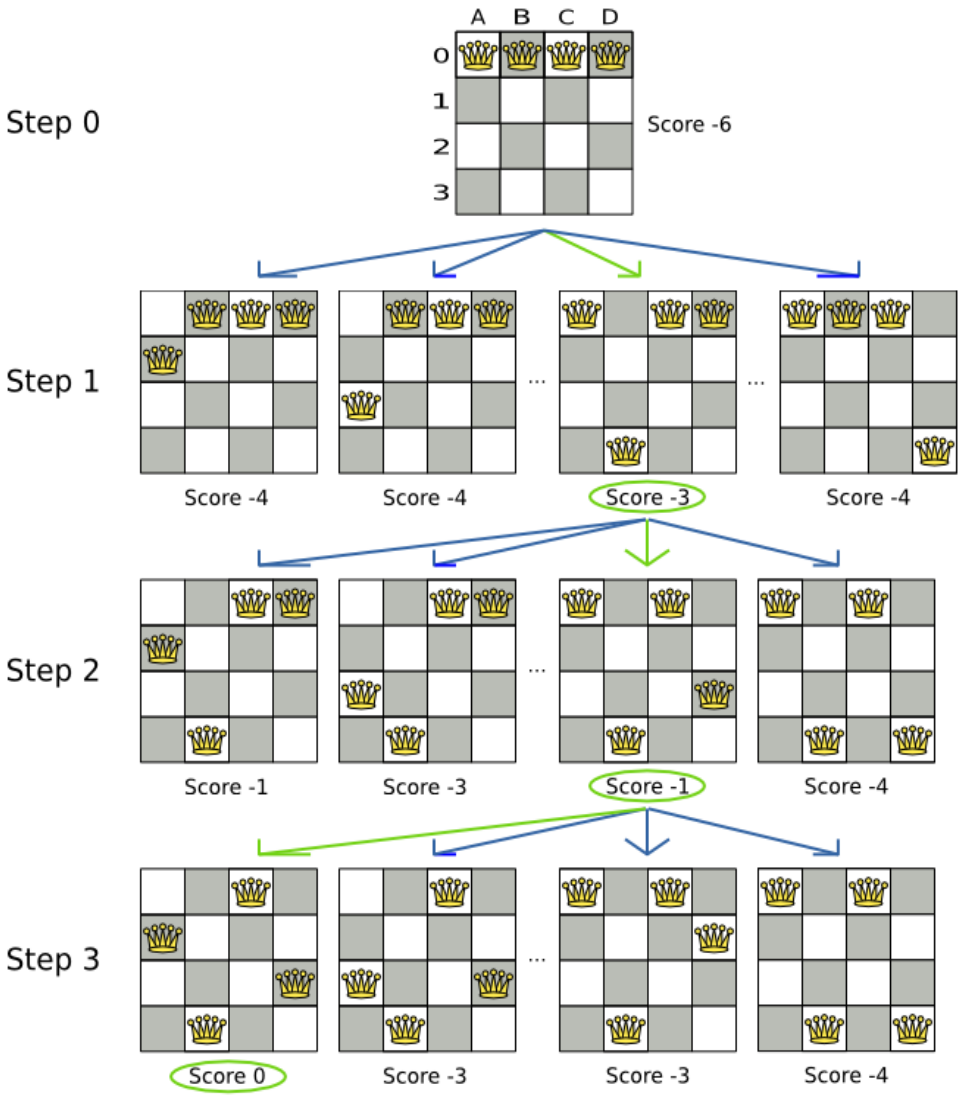
Lokalni preiskovalni algoritmi

- 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 **ne ohranjaj poti**
  - ponavljaj do **ustavitvenega pogoja**
- koristni v primerih:
  - če nas zanima samo kakovost rešitve (stanja) in ne 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



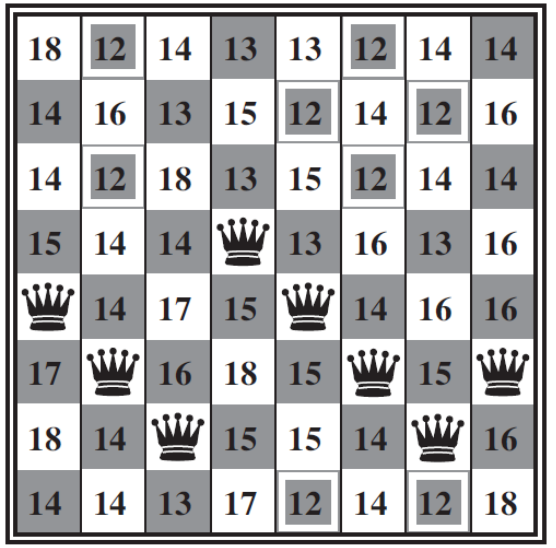
Lokalni preiskovalni algoritmi

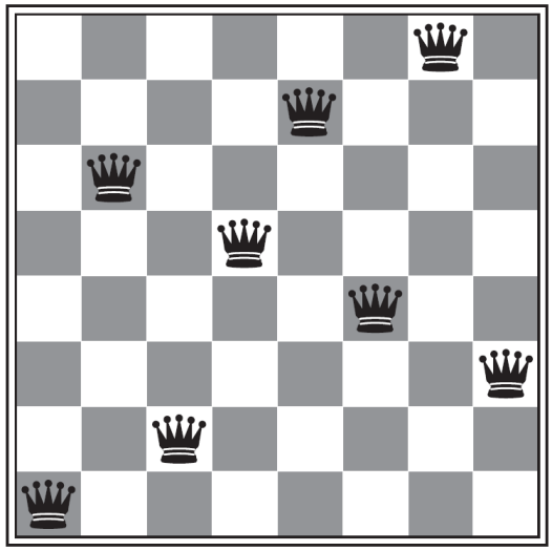
- primer: 4 kraljice na šahovnici
- kriterijska funkcija: maksimiziramo – (minus) število kraljic, ki se medsebojno napadajo
- če želimo ohraniti zaporedje korakov, ki vodijo do rešitve, je najdena **iskalna pot** pomembna, sicer pa ne; najdena iskalna pot za desni problem:
  - B0 na B3
  - D0 na D2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



Plezanje na hrib

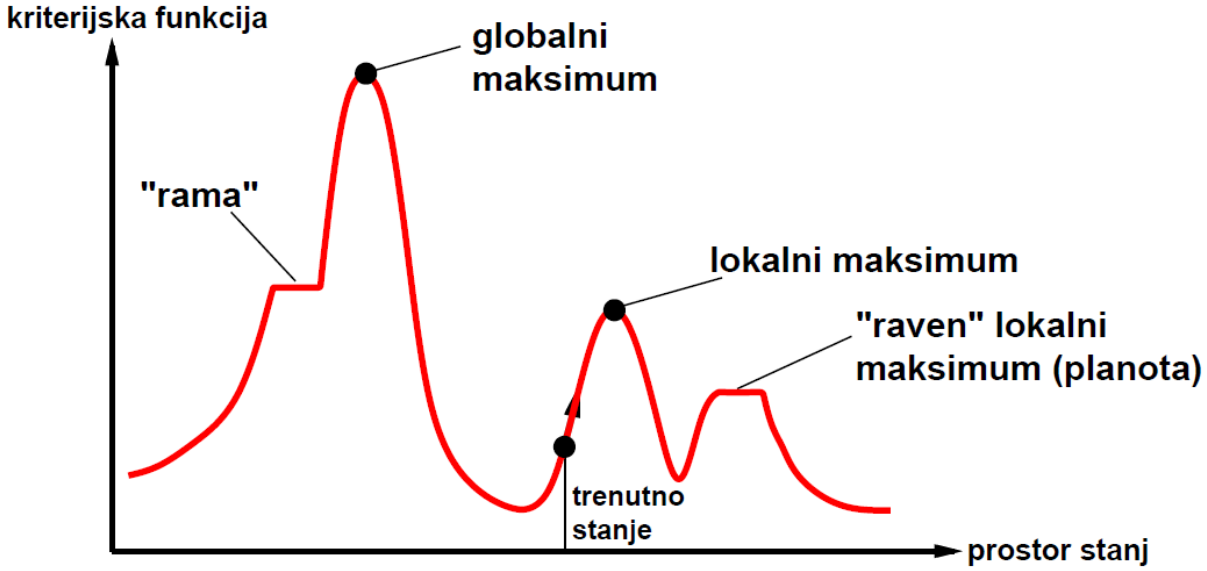
- angl. *hill-climbing search* (ali: *greedy local search*)
- premikaj se po prostoru stanj v smeri najboljše izboljšave kriterijske funkcije
- primer problema 8 kraljic:
  - kot "soseda" trenutnega stanja definiramo stanje, kjer 1 kraljico premaknemo na drugo polje znotraj istega stolpca
  - skupaj: 8×7=56 sosednih stanj
  - kriterijska funkcija: število kraljic, ki se napadajo (na zgornji sliki je prikazana vrednost kriterijske funkcije, če kraljico iz vsakega stolpca premaknemo na izbrano mesto znotraj stolpca), izvajamo minimizacijo
  - 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)

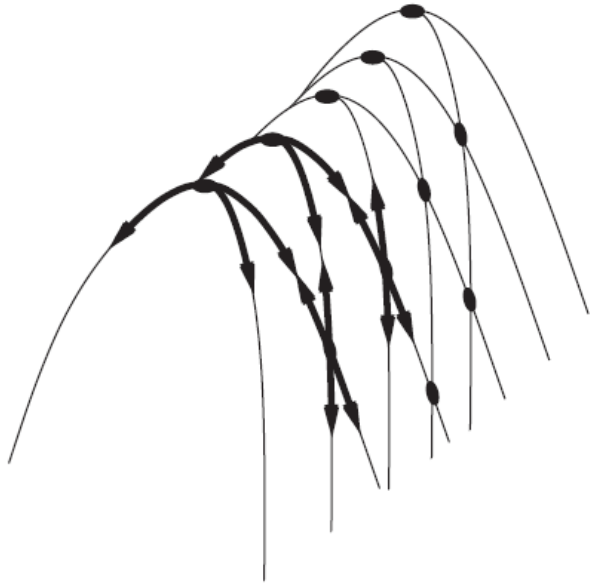




Lokalni preiskovalni algoritmi

- 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

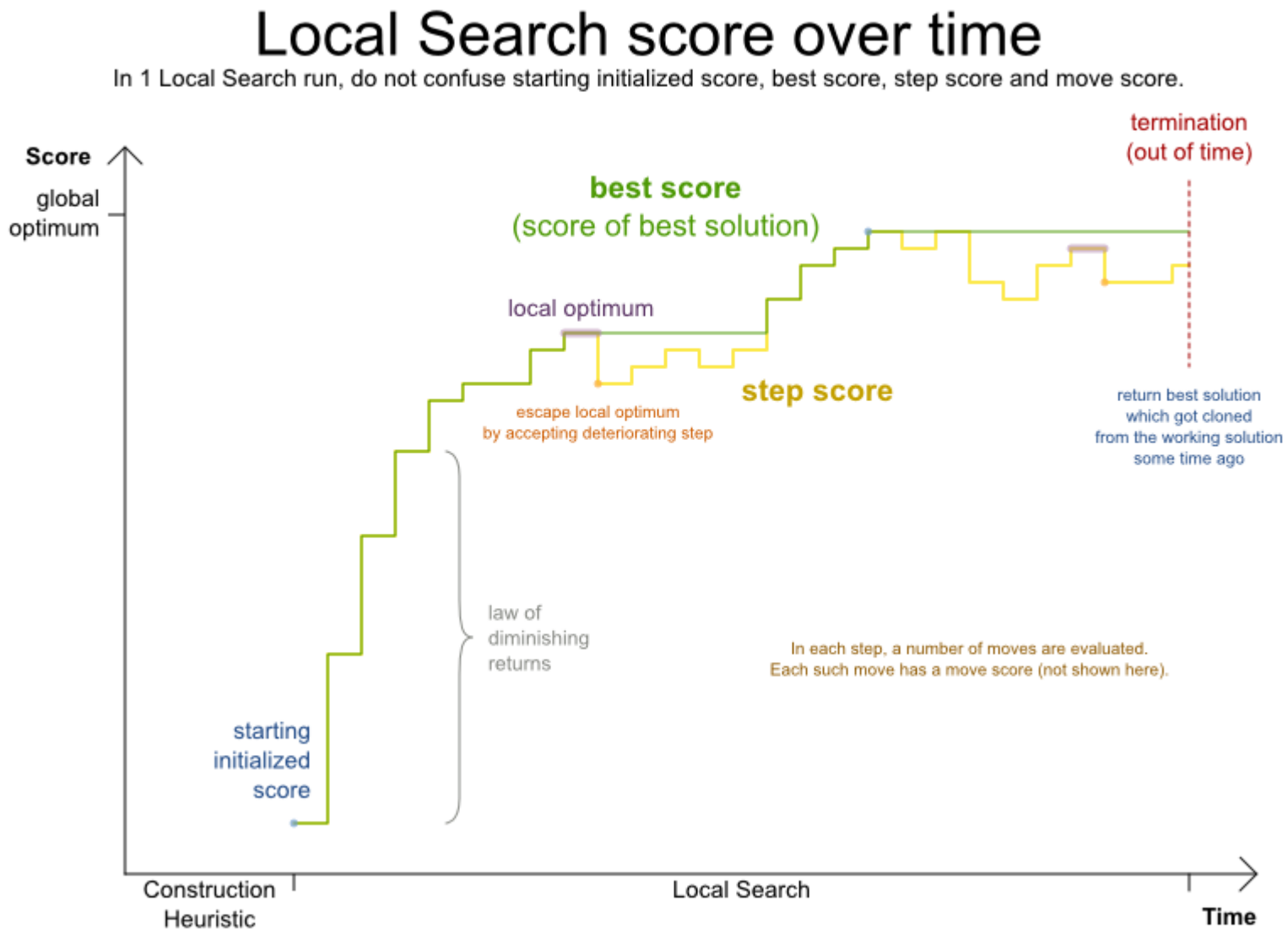
Reševanje iz lokalnih maksimumov

- **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, verjetnostno izberemo naslednje stanje (pri čemer upoštevamo, da imajo boljša stanja večjo verjetnost izbora)

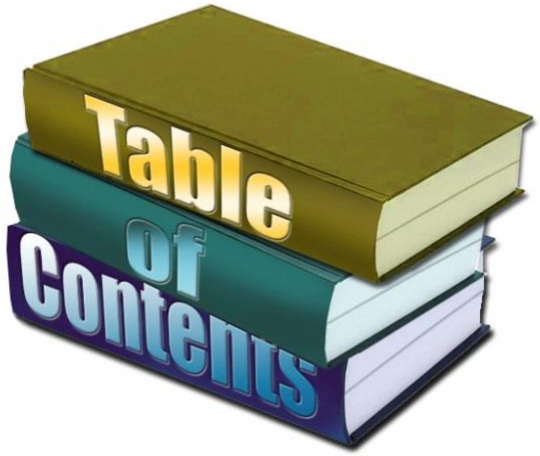
- **naključni ponovni zagon**: večkrat poženi plezanje na hrib iz naključnih začetnih stanj, dokler ne najdeš rešitve
  - če je verjetnost uspeha enega zagona p, je v povprečju potrebnih $1/p$ zagonov
  - za problem 8 kraljic:
    - $p = 0.14 \Rightarrow$ potrebnih je 7 zagonov (skupaj približno 22 korakov)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

Reševanje iz lokalnih maksimumov

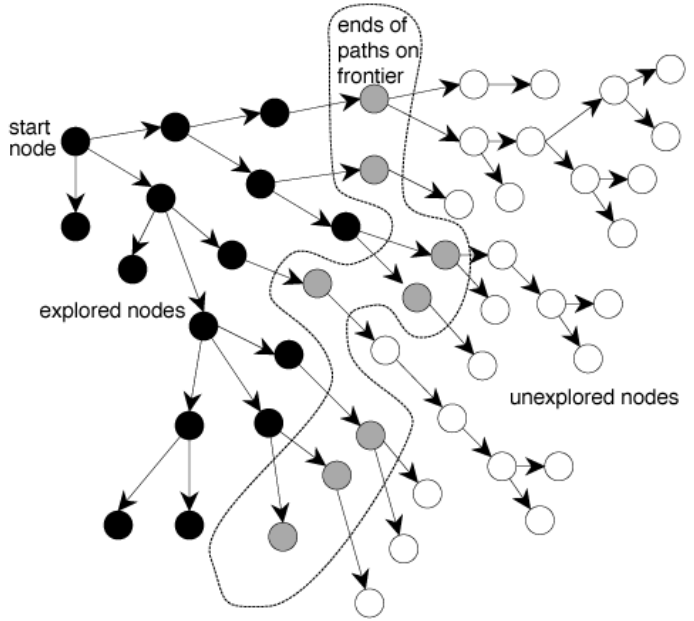


https://docs.optaplanner.org/6.5.0.CR1/optaplanner-docs/html/ch10.html#hillClimbing

Pregled



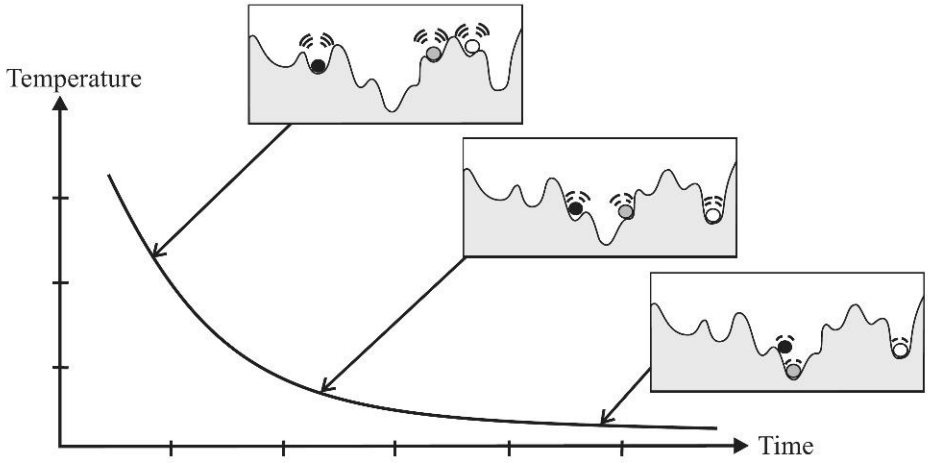
- preiskovanje prostora stanj
  - informirani preiskovalni algoritmi
    - hevristično preiskovanje (primer)
    - požrešno preiskovanje
    - A*
    - IDA*
    - kakovost hevrističnih funkcij
  - **lokalni preiskovalni algoritmi in optimizacijski problemi**
    - plezanje na hrib
    - simulirano ohlajanje
    - lokalno iskanje v snopu



Simulirano ohlajanje

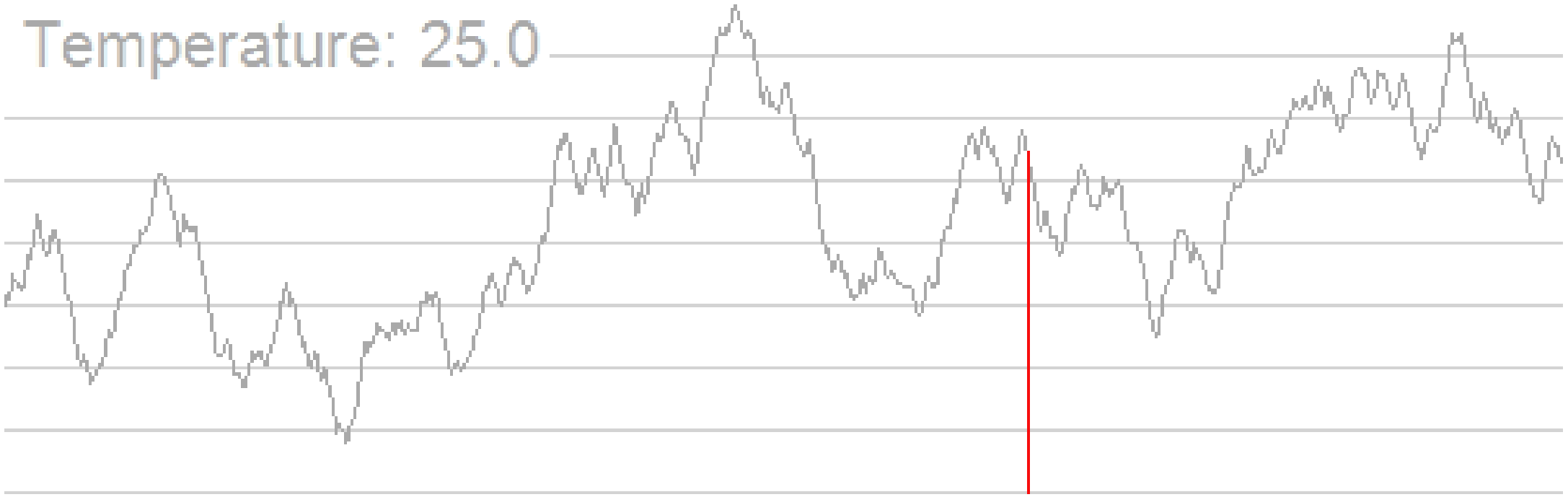
- angl. *simulated annealing*
- optimizacijski algoritem, ki izvira iz fizikalnih lastnosti v metalurgiji (ko je jeklo tekoče, so molekule v njem bolj gibljive; ko se ohlaja, se strjuje in molekule se umirjajo)
- analogija:
  - generiramo **naključne sosede** trenutnega stanja
  - če najdemo **boljše 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

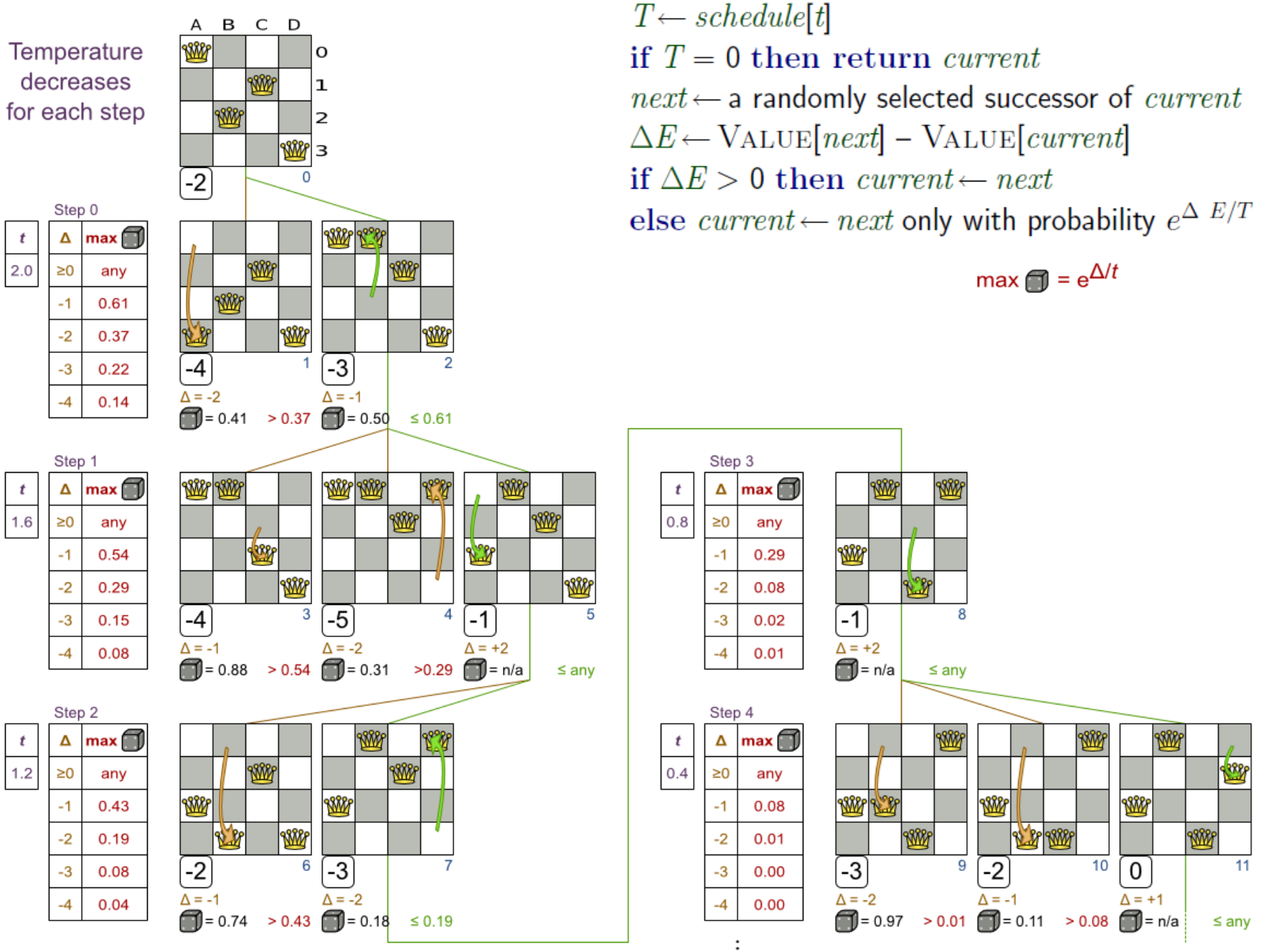
- iskanje globalnega maksimuma



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

Simulirano ohlajanje

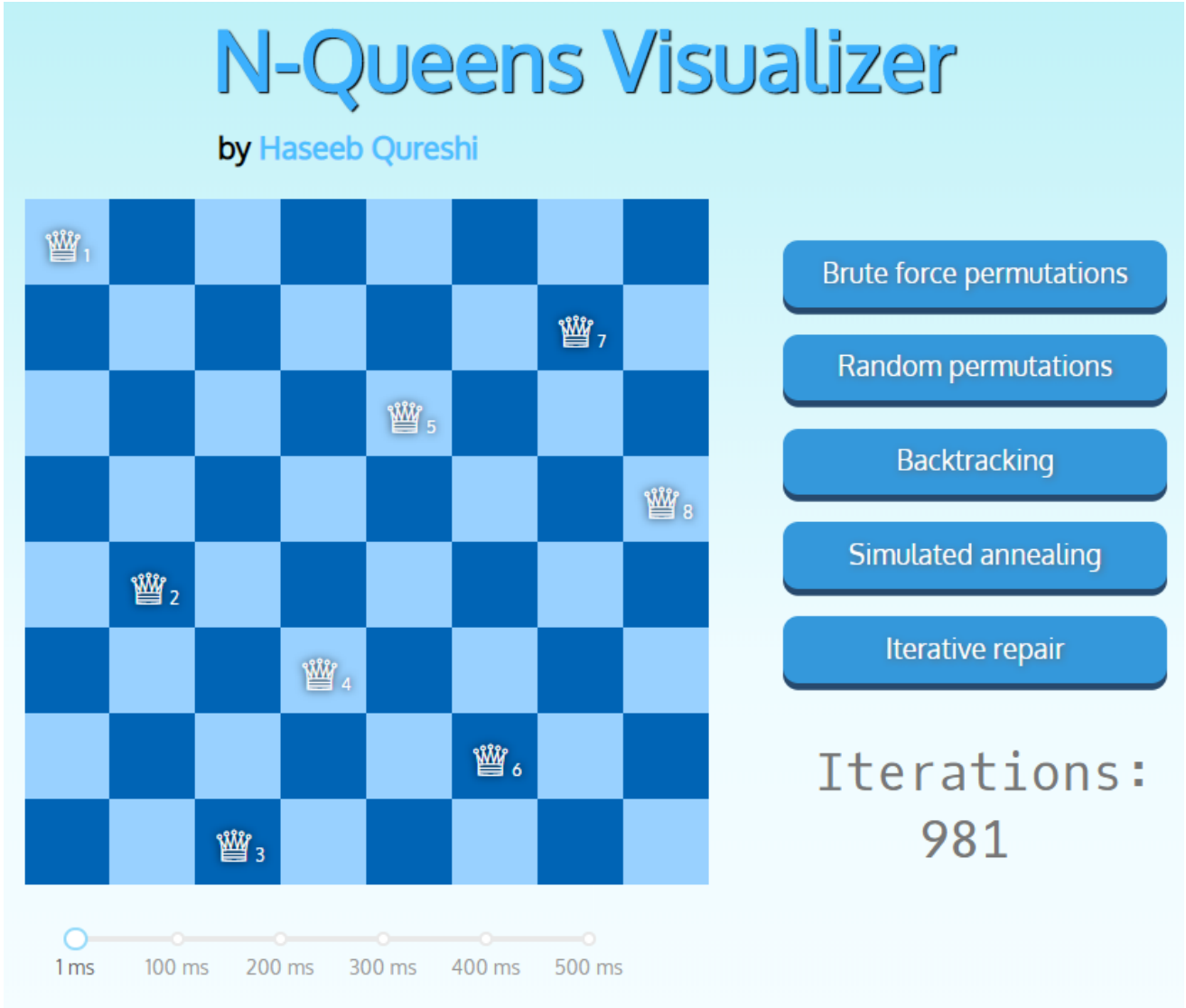
- za vsako slabše vozlišče naključno odločimo, ali ga izberemo kot naslednika
- če je vozlišče boljše, ga vedno izberemo za naslednika
- sčasoma (s padom temperature) simulirano ohlajanje preide v navadno plezanje na hrib



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

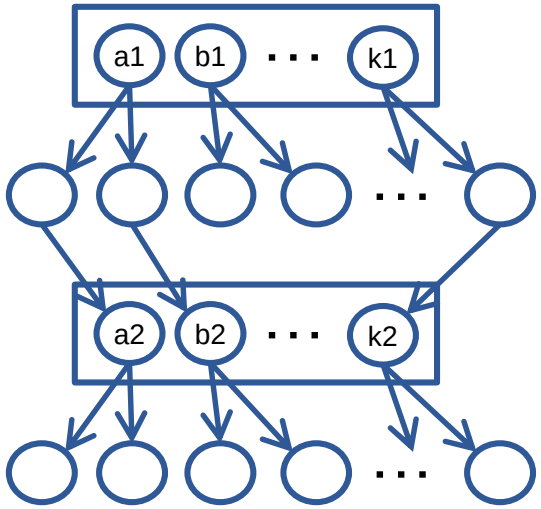
Demo

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



Lokalno iskanje v snopu

- algoritem:
  - v spominu hrani *k* aktualnih stanj namesto enega
  - izberi *k* optimalnih stanj od sosedov 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

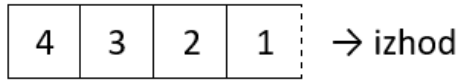
ponavljaj…

Primer izpitne naloge

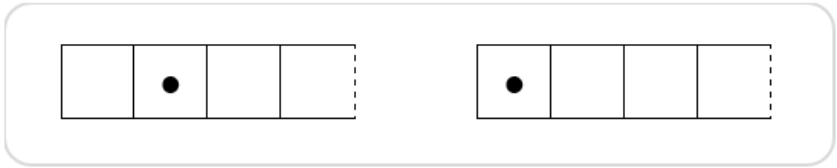
- 3. izpit, 7. 9. 2018

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 od 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