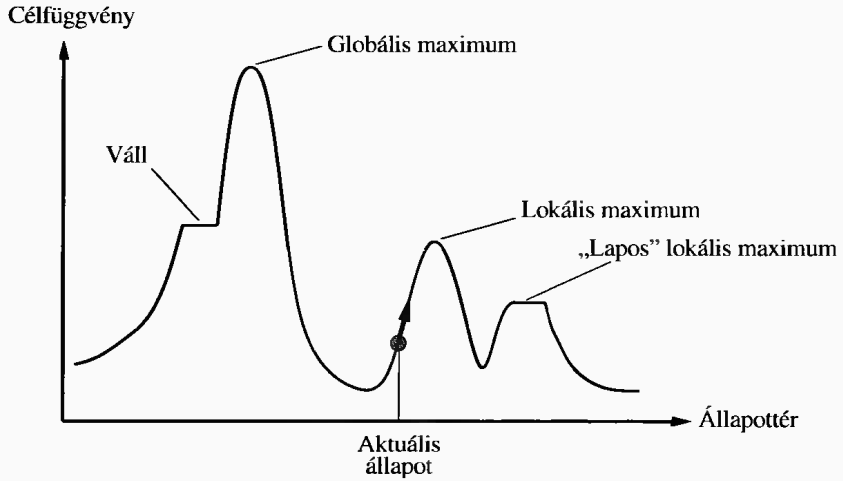
Lokális keresés

Lokális keresés

- egyes keresési problémák esetén csak a megoldás számít, az odavezető út nem (pl térképszínezés, n királynő)
- a lokális keresési algoritmusok
  1. csak egy [teljes] aktuális állapotot vesznek figyelembe
  2. az aktuális állapot valamelyik ígéretes szomszédjára lépnek tovább
  3. általában konstans mennyiségű memóriát használnak
  4. nagy / folytonos keresési térben is elfogadható megoldást adnak



*

Hegymászó keresés

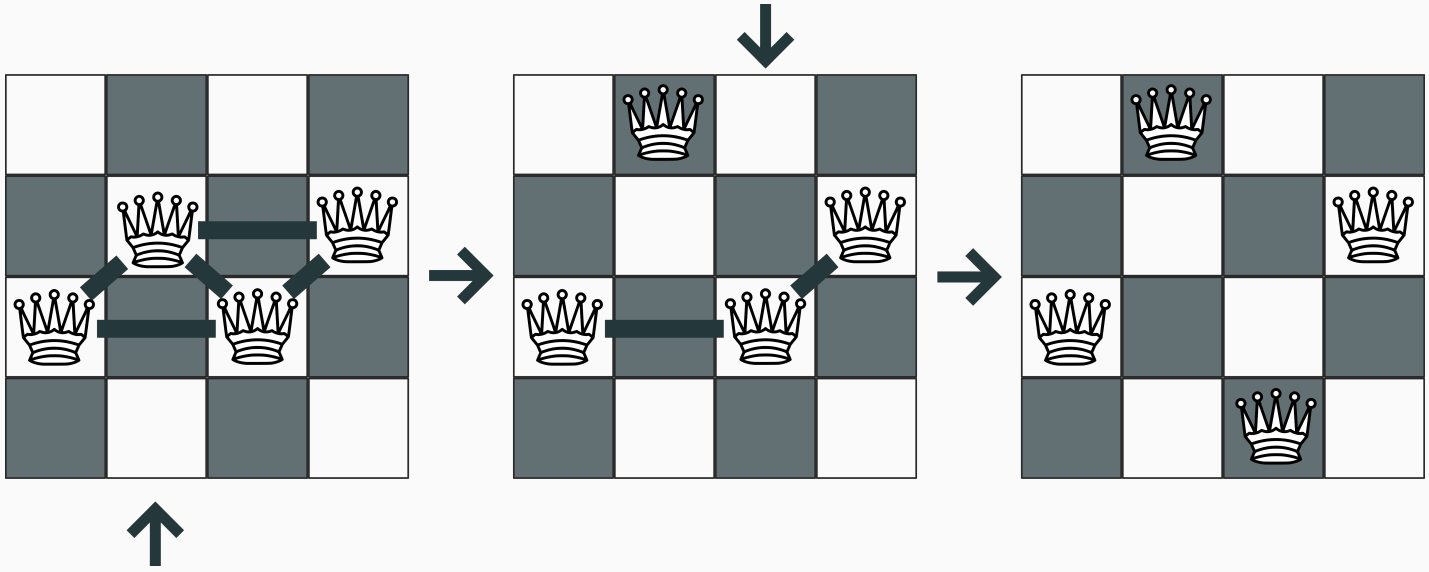
```
state hillclimb(state current, evaluator f) {
  while(true) {
    state succ = highest_f_neighbor_of(current, f);
    if f(succ) <= f(current) return current;
    current = succ;
  }
}
```

- *mintha a Mount Everest csúcsát szeretnénk megtalálni sűrű ködben és emlékezetkihagyásban szenvedve**
- a gyakorlatban igen jól teljesít
- megakad a lokális maximumokban (minimumokban)
- nem teljes

*Norvig & Russel, Mesterséges intelligencia modern megközelítésben, 2003

n-királynő probléma hegymászó kereséssel

- teljes állapot leírás (complete state formulation) – vs részleges hozzárendelés (backtrack)
- minimizálandó függvény: az egymást támadó királynő-párok száma
- szomszédos állapot: egy királynőt a saját oszlopán belül egy másik pozícióra helyezünk



Oldallépések

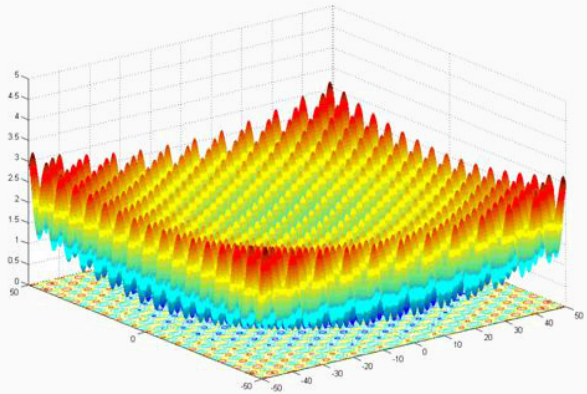
- **oldallépés**: akkor is elfogadunk egy szomszédos állapotot, ha az ugyanolyan jó, mint az aktuális
- a fennsíkokon és a vállakon való mozgást teszi lehetővé

```
1 state hillclimb(state current, evaluator f) {
2   while(true) {
3     state succ = highest_f_neighbor_of(current, f);
4     if f(succ) < f(current) return current;
5     current = succ;
6   }
7 }
```

- az egymás után alkalmazott oldallépések számát korlátozni kell

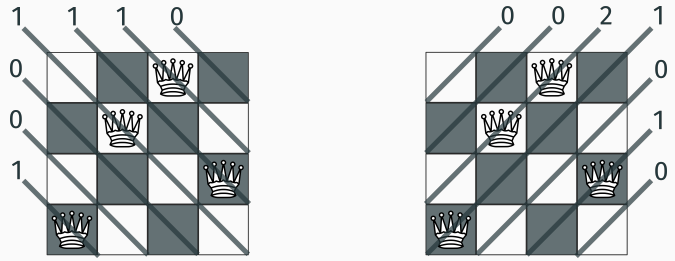
Változatok a hegymászó algoritmusra

- sztochasztikus hegymászó algoritmus (stochastic hill climbing)
  - a rákövetkezők állapotokból véletlenszerűen választ
  - a választás valószínűsége a lefele mutató irány meredekségével változhat
  - lassabban konvergál de kevésbé ragad bele a lokális optimumokba
- véletlen újraindítású hegymászás (random restart hill climbing)
- a hegymászó keresés sikere nagyban függ az állapottér felszínétől



n-királynő probléma, Gu & Susic, 1990*

- az $\{1..n\}$ egy permutációjával inicializálja a királynőket
- csak az átlós ütközésekre figyel: *mdiag*, *sdiag*
- heurisztika: ha két királynő, q_i és q_j egyike ütközik és a cseréjük csökkentené az ütközések számát, akkor cseréljük fel q_i-t q_j-vel
- ha csere történt, frissítsük diagonális tömböket



- q: [3 1 0 2]
- mdiag: [1 0 0 1 1 1 0], ütközések száma: 0
- sdiag: [0 0 2 1 0 1 0], ütközések száma: 1

*A Polynomial Time Algorithm for the N-Queens Problem

n-királynő probléma, Gu & Susic, 1990

```
1  function nqueen-sosic(queen: array [1..n] of integer)
2  repeat
3    queen = random_permutation(1,n)
4    repeat
5      nswaps = 0
6      for i = 1..n
7        for j = i+1..n
8          if is_attacked(queen[i]) or is_attacked(queen[j]) then
9            if swap(queen[i], queen[j]) reduces collisions then
10             perform_swap(queen[i], queen[j])
11             nswaps = nswaps+1
12           end
13         end
14       end
15     end
16   until nswaps == 0
17 until no collisions
```

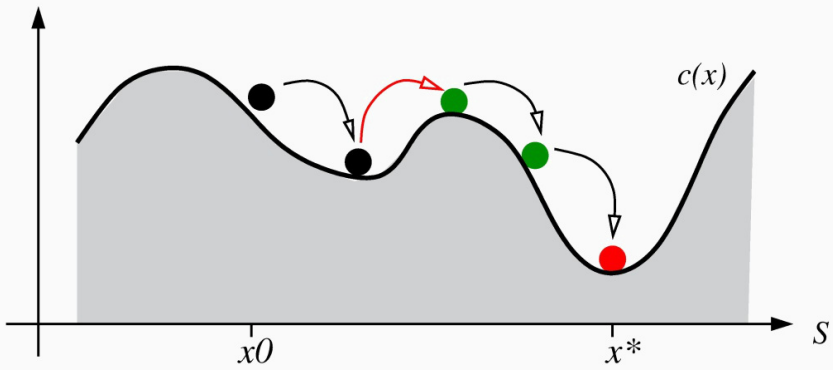
- n ≈18 000 királynőre 1s alatt talál megoldást (2.8GHz, i7)

n-királynő probléma, Gu & Susic, 1994*

- a kezdeti permutációt inkrementálisan állítja elő, úgy, hogy a benne levő ütközések minimálisak legyenek
- n≈3 000 000 királynőre 1s alatt talál megoldást (2.8GHz, i7)

*Efficient Local Search with Conflict Minimization

Szimulált lehűtés



- a hegymászást a (teljes) random walk-kal kombináljuk:
  - egy véletlen lépést teszünk
  - ha ez jobb állapotot eredményez, elfogadjuk
  - ha nem, akkor csak egy bizonyos valószínűséggel fogadjuk el, ez exponenciálisan csökken a lépés “rosszaságával” és a “hőmérséklet” (T) csökkenésével

Szimulált lehűtés algoritmus

```
1  state simulated_annealing(state act) {
2    for (int i = 0; i < MAX_ITER; ++i) {   //
3      double T = temperature(i);           //T - temperature
4      if (T <= eps) return act;            //temperature(·) - cooling
5      state next = random_neighbor(act);   //                 shedule
6      double DeltaE = f(next) - f(act);    //f(·) - function to
7      if (DeltaE < 0)                      //       minimize
8        act = next;
9      else
10       if (probability(exp(-DeltaE/T)))
11         act = next;
12     }
13 }
```

- exponenciális hűtés: $T(i) = T_0 \alpha^i$, $\alpha \in (0,1)$

Utazóügynök szimulált lehűtéssel

- állapot: az $\{1,2,\ldots,n\}$ halmaz egy σ permutációja
- minimizálandó függvény: $\sum_{i=1}^{n-1} d(\sigma(i),\sigma(i+1))$
- szomszédos állapot pl
  - egy útszakasz megfordítása
  - egy útszakasz kivágása, és egy másik pozícióra való beszúrása