UMELÁ INTELIGENCIA

STU FIIT

Ing. Lukáš Kohútka, PhD.

Problémy kde nehľadáme cestu

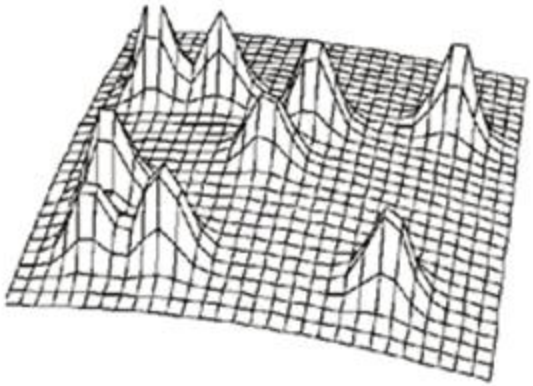
- Na príklade problému 8 dám sme si ukázali, že sú problémy, pri ktorých nehľadáme cestu riešenia, ale hľadáme cieľový stav, ktorý nepoznáme
  - poznáme iba spôsob, ako cieľový stav rozpoznať (cieľ je daný implicitne)
  - na hľadanie riešenia tohto druhu problémov možno použiť aj stratégie, opísané v predchádzajúcej časti
    - tie sa však zameriavajú na nájdenie najlepšej cesty, teraz avšak cesta nezohráva žiadnu úlohu
  - tým, že uvažujeme jednoduchší druh problémov, máme príležitosť sformulovať metódu hľadania riešenia, ktorá by bola jednoduchšia
- Asi najjednoduchšia metóda je cyklicky postupovať tak, že sa vyberie (vygeneruje) možné riešenie a testuje, či je riešením problému

Algoritmy cyklického vylepšovania

Algoritmus cyklického vylepšovania vychádza z ľubovoľného začiatočného stavu opisujúceho úplnú konfiguráciu a postupne mení stav tak, aby sa konfigurácia vylepšovala z hľadiska približovania sa k cieľovému stavu.

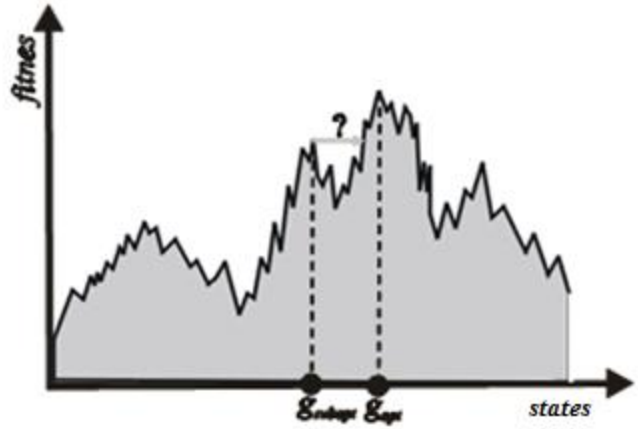
```
function GENEROVANIE-A-TESTOVANIE(problém, GENERUJ-STAV)
returns stav riešenia
    static: možné-riešenie, stav (ak riešením je stav)
            alebo posledný stav na ceste (ak riešením je cesta)

    loop do
        možné-riešenie ← GENERUJ-STAV(STAVY[problém])
        if CIEĽOVÝ-TEST[problém] aplikovaný na možné-riešenie je úspešný
                then return možné-riešenie
    end
```

Hľadanie optima fitnes funkciou



A

B



C

A. Znázornenie povrchu fitnes, ktorý vyjadruje závislosť výsledku fitnes funkcie od zvolených parametrov (stavu)
B. Kontúrový graf, priradený povrchu fitnes, oblak bodov znázorňuje vyskúšané kombinácie parametrov (stavy)
C. Znázornenie globálneho riešenia g_{opt} a na vedľajšom vrchole suboptimálneho riešenia g_{subop}

Generovanie stavov

- Systematické
  - exhaustívne (vyčerpávajúce, úplné) prehľadanie priestoru problému
  - lokálne zmeny
- Náhodné
  - nie je záruka, že sa riešenie nájde, aj ak riešenie existuje (algoritmus Britského múzea – hypotéza o opiciach a písacích strojoch)
  - slepý algoritmus

Algoritmus Britského múzea

- Hypotéza o opiciach a písacích strojoch
- Stačí posadiť dostatočný (nekonečný) počet opíc za dostatočný (nekonečný) počet písacích strojov a ich ťukaním skôr či neskôr vzniknú Shakespearove zobrané spisy
- Ale (v skutočnosti, aspoň podľa pokusu výskumníkov Plymouthskej univerzity v Anglicku, 2003)
- 6 opíc druhu makak dostalo jeden počítač a 4 týždne času. výsledok?
  - jeden chytil kameň a mlátil do klávesnice
  - ďalšie vykonali nad klávesnicou rôzne kombinácie malej a veľkej potreby
  - po čase sa dostali k tomu, že začali písať písmeno S
  - celkovo napísali 5 strán, okrem písmena S sa im do výsledného textu vkradlo aj zopár písmen A, J, L a M
- Projekt stál 2000 £
- Záver: opice nemožno redukovať na náhodné stroje. počítač ich nudí
- Napriek tomu, ak by bolo nie 6, ale 10813 opíc, nie 4 týždne, ale 5 rokov a každá mala svoj počítač, tak by vznikol sonet č. 3. samozrejme, od Shakespeara

Slepý algoritmus

Slepý algoritmus je základný stochastický algoritmus, ktorý opakovane generuje náhodne riešenie z oblasti D a zapamätá si ho len vtedy, ak bolo získané lepšie riešenie ako to, ktoré už bolo zaznamenané v predchádzajúcej histórii algoritmu.

```
procedure Blind_Algoritmus(input: t_max,k,n; output:
α_fin,f_fin);
begin f_fin:=∞; t:=0;
      while t<t_max do
      begin t:=t+1;
            α:=randomly generated binary vector
               of the length kn;
            if f(Γ(α))<f_fin then
            begin α_fin:=α; f_fin:=f(Γ(α)) end;
      end;
end;
```

Slepý algoritmus

Tento jednoduchý stochastický optimalizačný algoritmus poskytuje korektné globálne minimum t_{max} asymptoticky rastie do nekonečna

$$\lim_{t_{\max} \to \infty} P(t_{\max} / \alpha_{fin} = \alpha_{opt}) = \mathbf{1}$$

Slepý algoritmus neobsahuje žiadnu stratégiu konštrukcie riešení na základe predchádzajúcej histórie algoritmu. Každé riešenie je zostrojené úplne nezávisle (t.j. plne náhodne) od predchádzajúcich riešení. Zaznamenáva sa to riešenie, ktoré v priebehu aktivácie procedúry poskytuje zatiaľ najnižšiu funkčnú hodnotu.

Lokálne hľadanie

- Metóda hľadania s veľmi nízkymi požiadavkami na pamäť
- Hľadanie negeneruje strom hľadania, vždy sa pracuje len so zápisom súčasného stavu!
- Dá sa použiť len na problémy, kde riešením nie je cesta (napr. 8 dám) ale stav – iba ak by sa cesta nejako kódovala do stavu
- Podobnosti s metódami optimalizácie

Lokálne hľadanie

- V mnohých optimalizačných problémoch nie je dôležitá cesta do cieľa – riešením je cieľový stav
- Stavový priestor = množina „úplných" konfigurácií
- Treba nájsť konfiguráciu, ktorá spĺňa ohraničenia
- V takých prípadoch možno použiť lokálne hľadanie
- Udržiava sa “súčasný” stav, je snaha vylepšovať ho

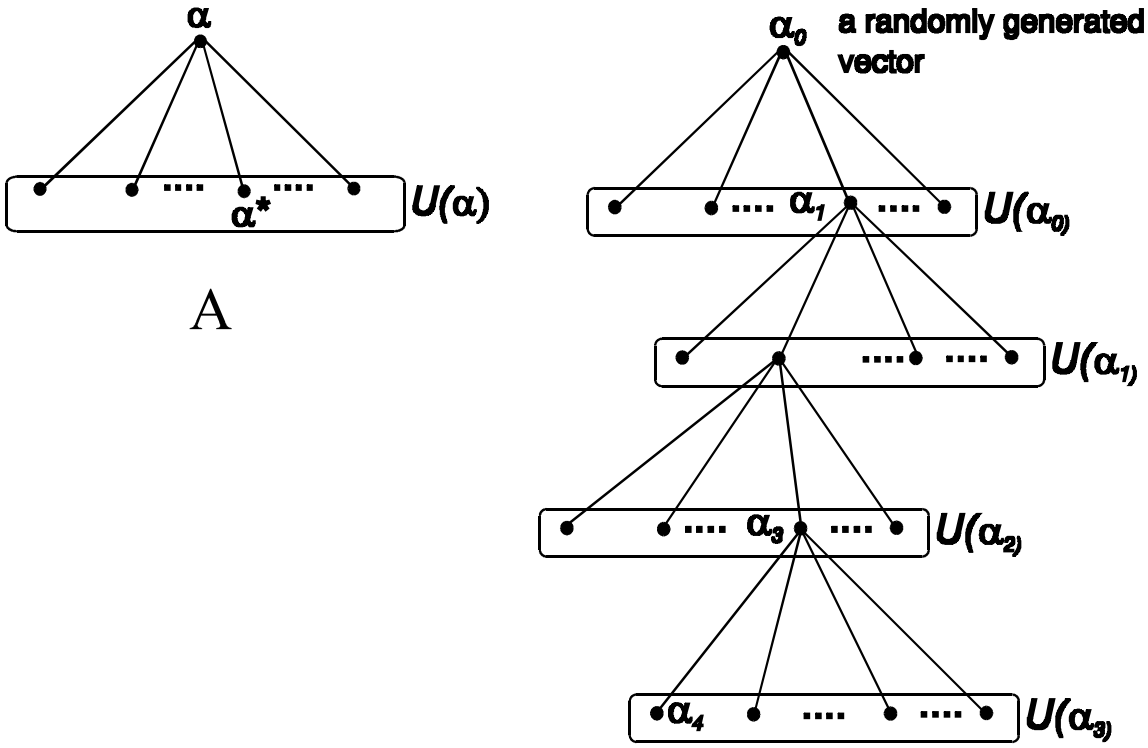
Stratégia lokálneho vylepšovania

Podstatou stratégie lokálneho vylepšovania je všeobecne používaná heuristika: pri hľadaní riešenia postupovať v každom kroku v smere, ktorý spôsobí lokálne zlepšenie z daného stavu.
Algoritmus neudržiava strom hľadania (nevybraté uzly sa nepamätajú, ale okamžite sa zabúdajú). Uzol obsahuje iba opis stavu a jeho ohodnotenie.

```
function LOKÁLNE-VYLEPŠOVANIE(problém) returns stav riešenia
    static: súčasný, uzol

    súčasný ← VYTVOR-UZOL(ZAČIATOČNÝ-STAV[problém])
    loop do
        if nejaký nasledovník uzla súčasný má lepšie ohodnotenie
            then súčasný ← lepšie ohodnotený nasledovník uzla súčasný
            else return STAV(súčasný)
    end
```

Stratégia lokálneho vylepšovania

Získané riešenie α^* použijeme ako "stred" v ďalšom iteračnom kroku

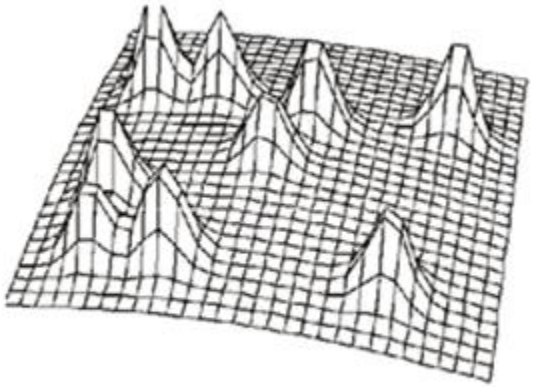


Stratégia lokálneho vylepšovania

- Cyklus hľadania, ktoré sa nepretržite pohybuje v smere zvyšujúcej sa hodnoty
  - skončí keď sa dosiahne vrchol
  - stratégia známa tiež ako lačné lokálne hľadanie
  - stratégia známa tiež ako horolezecký algoritmus (hillclimbing)
- Výstup na Mount Everest v úplnej hmle
- Hodnota ohodnocovacej funkcie (fitnes funkcia)
  - hodnota cieľovej funkcie
  - hodnota heuristickej funkcie
- Nepozerá sa dopredu pred bezprostredných susedov súčasného stavu
- Volí náhodne z množiny nasledovníkov, ak viac je hodnotených lepšie

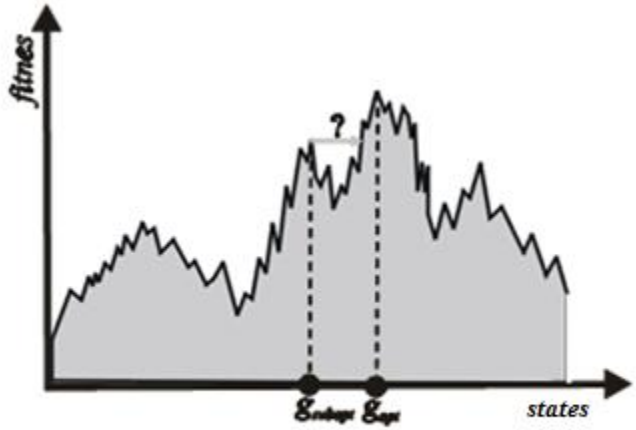
Stratégia lokálneho vylepšovania

- Ak existujú lokálne extrémy (optima), hľadanie nemusí viesť k optimálnemu riešeniu
- Jednoduchý spôsob nájdenia riešenia (a často efektívny)
- Viac pokusov hľadania
  - náhodné začiatky



A

B



C

Stratégia lokálneho vylepšovania – príklad

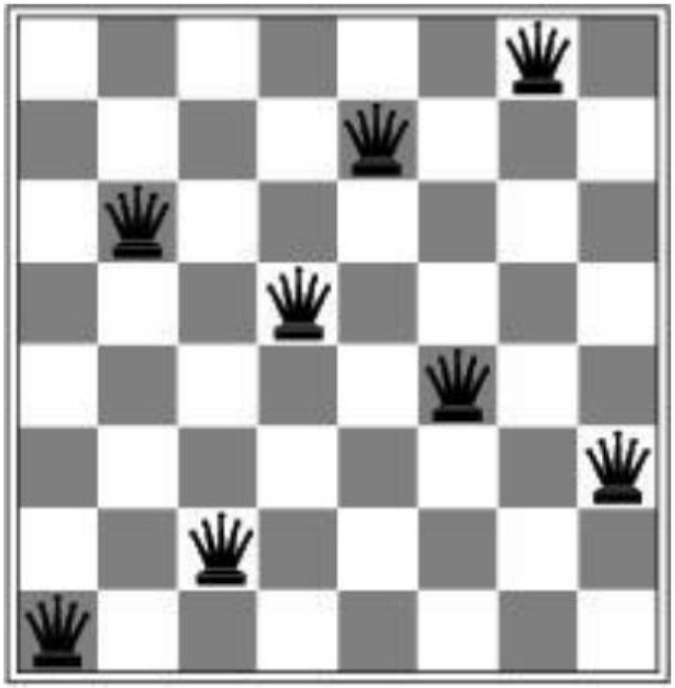
- Problém 8-dám, opis stavu zahŕňa úplnú konfiguráciu
  - všetkých 8 dám na doske v nejakej konfigurácii
- Funkcia nasledovníka:
  - presuň dámu na iné políčko v tom istom stĺpci.
- Príklad heuristickej funkcie h(u):
  - počet dvojíc dám, ktoré sa napádajú
  - (čiže toto chceme minimalizovať)

Stratégia lokálneho vylepšovania – príklad

Ohodnotenie súčasného stavu: h=17

Znázornené sú h-hodnoty každého možného nasledovníka v každom stĺpci

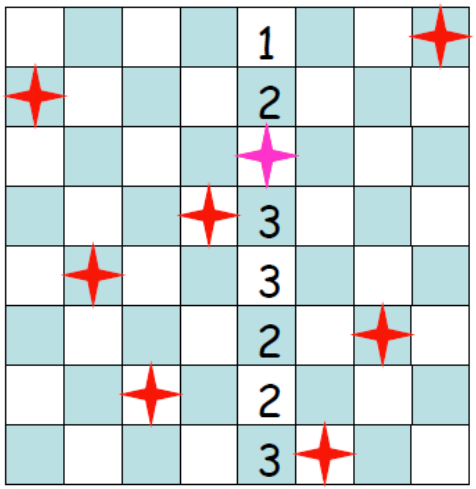
Lokálne minimum pre 8-dám

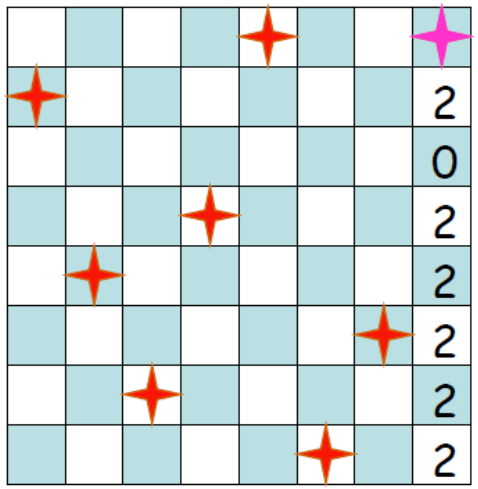


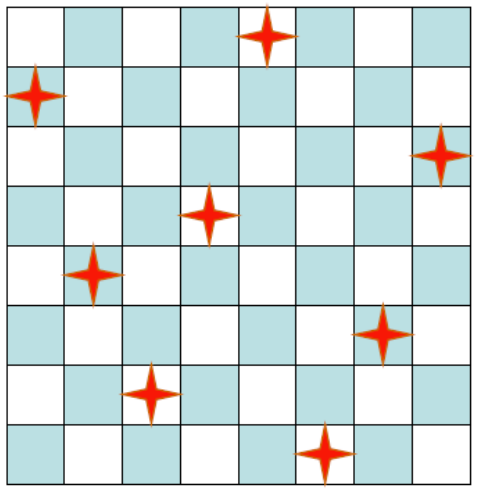
Lokálne minimum v stavovom priestore problému 8-dám (h=1)
mimochodom: prečo je tento stav lokálne minimum?

8-dám, trochu iná heuristika

1) zvoľ začiatočný stav S náhodne tak, že v každom stĺpci je práve 1 dáma
2) opakuj k razy:
   a) If GOAL?(S) then return S
   b) zvoľ náhodne dámu Q , ktorá je napadnutá
   c) presuň Q v jej stĺpci na políčko, kde ju bude napádať čo najmenej dám → nový stav S
3) Return neúspech



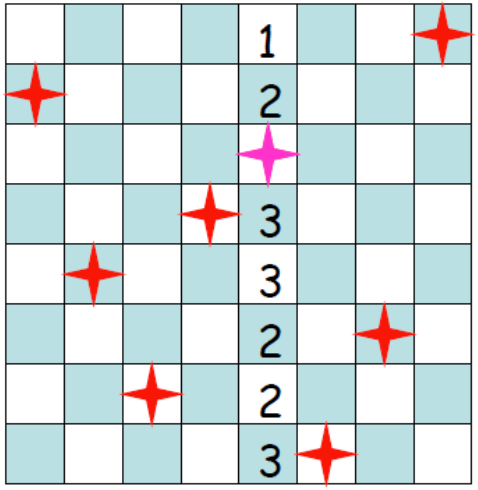


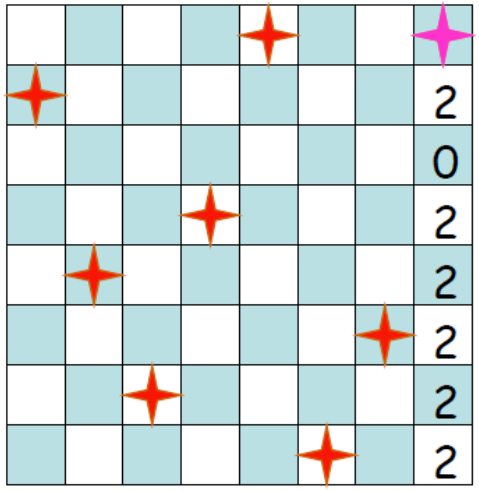


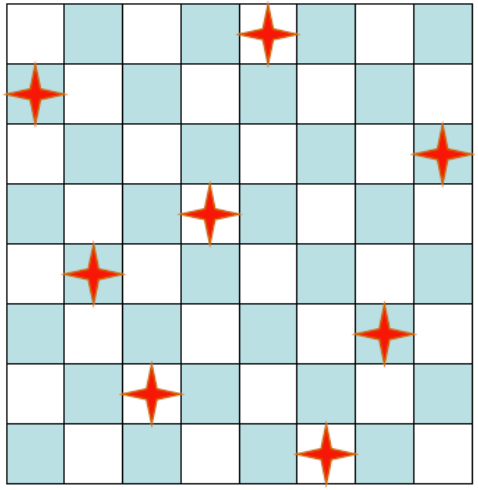
8-dám, trochu iná heuristika

prečo to funguje??

1) jestvuje veľa cieľových stavov, ktoré sú dobre rozdelené v stavovom priestore
2) ak sa nenájde riešenie po niekoľkých krokoch, treba radšej skončiť a začať odznova. hľadanie by bolo kvôli vysokému faktoru vetvenia neefektívne
3) čas hľadania je skoro nezávislý od počtu dám







Ako funguje stratégia lokálneho vylepšovania na problém 8-dám

- Začiatočné stavy sa generujú náhodne…
- 14% prípadov vyrieši problém
- 86% prípadov zapadne v lokálnom minime (problém zacyklenia)
- Avšak…
  - V prípade úspechu potrebuje len 4 kroky na nájdenie cieľového stavu
  - Len 3 kroky v priemere na zapadnutie v lokálnom minime
  - (v stavovom priestore s ~17 miliónmi stavov)

Možné riešenie… bočné úkroky

- Ak nie sú možné kroky nahor (nadol), treba povoliť bočné úkroky v nádeji, že hľadanie sa vyhne lokálnemu extrému
  - treba stanoviť hranicu na možný počet bočných úkrokov, aby nemohlo dôjsť k nekonečnému cyklu
- Pre 8-dám
  - nech je povolených (napr.) 100 bočných úkrokov
  - toto zvýši podiel úspešne vyriešených inštancií problémov z 14% na 94%
  - avšak….
    - 21 krokov do každého úspešného riešenia
    - 64 krokov do každého neúspechu

Variácie stratégie lokálneho vylepšovania

- Stochastická stratégia lokálneho vylepšovania (stochastic hill-climbing)
  - náhodný výber spomedzi krokov smerujúcich nahor
  - pravdepodobnosť výberu ďalšieho kroku môže byť daná strmosťou pohybu v smere príslušného kroku
- Stratégia lokálneho vylepšovania prvý berie (first-choice hill-climbing)
  - stochastická stratégia lokálneho vylepšovania generovaním nasledovníkov náhodne dovtedy, kým sa nájde lepší
  - užitočné, ak je veľmi veľa nasledovníkov
- Stratégia lokálneho vylepšovania s náhodným reštartom
  - pokúša sa vyhnúť uviaznutiu v lokálnom maxime
- Stratégia lokálnej optimalizácie
  - výber najlepšieho spomedzi krokov smerujúcich nahor

Stratégia lokálneho vylepšovania s náhodným reštartom

- Rôzne obmeny
- Pre každý opakovaný začiatok hľadania (reštart):
  - hľadá sa, kým neskončí
  - hľadá sa vopred stanovenú dobu
- Opakovanie hľadania:
  - vopred stanovený počet opakovaní (reštartov):
  - hľadá sa „donekonečna“

- Ako odhadnúť počet opakovaní?
- Ako odhadnúť počet krokov do nájdenia riešenia?

Stratégia lokálnej optimalizácie

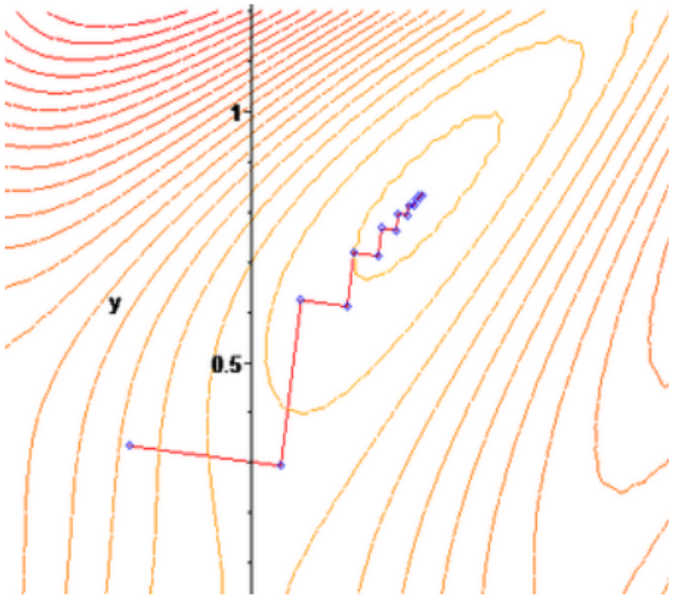
Známa aj ako **gradientové hľadanie**

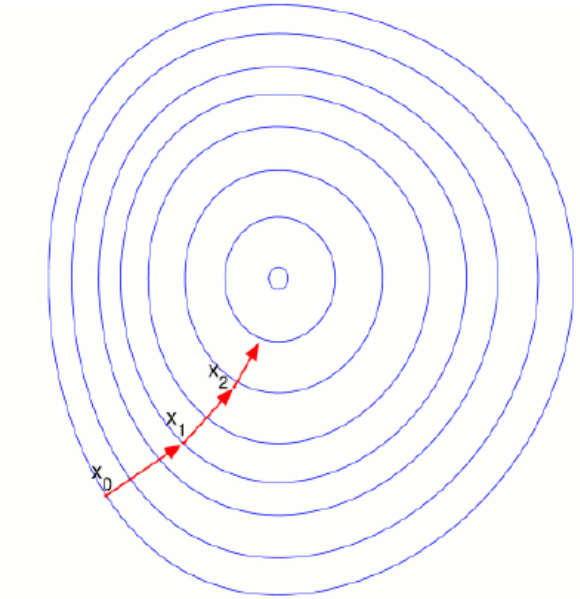
Ako ďalší uzol sa nevyberá hociktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.

```
function LOKÁLNA-OPTIMALIZÁCIA(problém) returns stav riešenia
    static: súčasný, uzol
            ďalší, uzol

    súčasný ← VYTVOR-UZOL(ZAČIATOČNÝ-STAV[problém])
    loop do
        ďalší ← najlepšie ohodnotený nasledovník uzla súčasný
        if HODNOTA[ďalší] < HODNOTA[súčasný]
                then return STAV(súčasný)
        súčasný ← ďalší
    end
```

Stratégia lokálnej optimalizácie

Ako ďalší uzol sa nevyberá hociktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.





Stratégia lokálneho vylepšovania v lúči

- Local beam search
- Udržiava si nie 1 ale k súčasných stavov
  - na začiatku: vyberie sa náhodne ***k*** stavov
  - ďalší krok: určia sa všetky nasledovníky všetkých k stavov
  - ak je hociktorý z nich riešením – return
  - inak vyberie sa *k* najlepších nasledovníkov, ďalší krok

Stratégia lokálneho vylepšovania v lúči

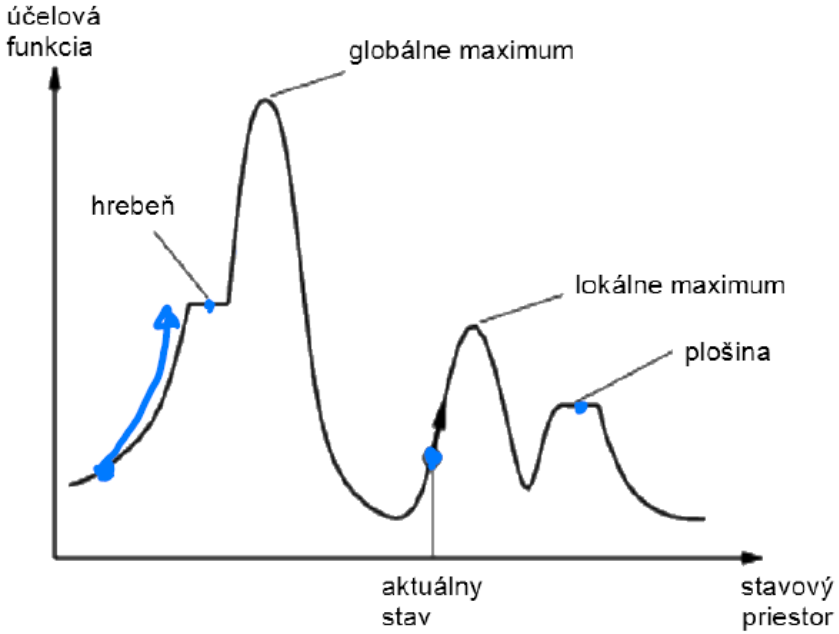
- Zdá sa, že je to len *k* paralelných hľadaní stratégiou lokálneho vylepšovania
- Nie, lebo informácie o hľadaní sú spoločné pre všetkých k vlákien
- Ak jeden stav generuje viacero dobrých nasledovníkov, môžu sa dostať (aj všetky) do ďalšieho kola
- Stavy, ktoré generujú zlé nasledovníky, sa odstránia

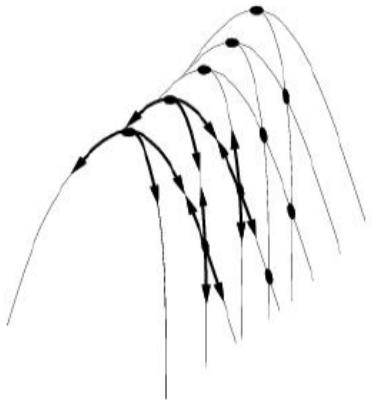
Stratégia lokálneho vylepšovania v lúči

- Spôsob výberu nasledovníkov je silná aj slabá stránka stratégie
- Silná:
  - neproduktívne stavy (smery hľadania) sa rýchlo zanechajú
  - stavy sľubujúce najväčší pokrok sa uprednostňujú
- Slabá:
  - nedostatočná rôznorodosť (hľadá sa v malom výseku stavového priestoru)
  - stavy aj tak môžu skonvergovať do spoločného miesta
  - stále náchylné na lokálne extrémy

Problém zacyklenia v lokálnom extréme

Stratégie lokálneho vylepšovania skrývajú v sebe aspoň tri úskalia:



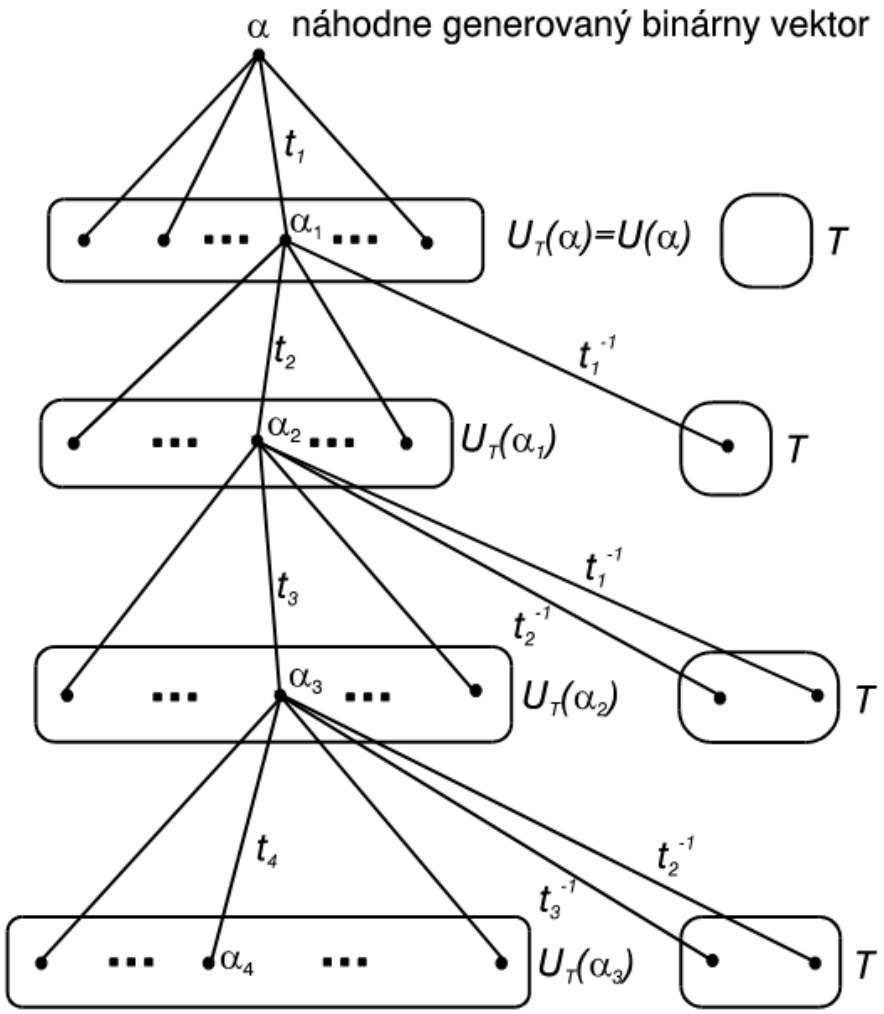


- hrebeň = postupnosť stavov ohodnotených ako lokálne maximá. Lačné lokálne hľadanie len ťažko naviguje po hrebeni
- plošina = oblasť stavového priestoru, v ktorej sú hodnoty vyhodnocovacej funkcie ploché (rovnaké).

Metóda zakázaného hľadania

- Známa ako “tabu search”
- Navrhnutá koncom 80-tých rokov Gloverom ako určité zovšeobecnenie horolezeckého algoritmu pre riešenie zložitých optimalizačných úloh
- Glover navrhol jednoduchú heuristiku ako odstrániť problém zacyklenia v lokálnom extréme
- Do horolezeckého algoritmu je zavedená tzv. krátkodobá pamäť, ktorá si pre určitý krátky interval predchádzajúcej histórie algoritmu pamätá stavy alebo inverzné transformácie k týmto stavom, ktoré poskytovali lokálne optimálne riešenia
- Tieto inverzné transformácie (resp. stavy) sú zakázané (tabu) pri tvorbe nového okolia pre dané aktuálne riešenie

Metóda zakázaného hľadania



Metóda zakázaného hľadania


```
1 sBest ← s0
2 bestCandidate ← s0
3 tabuList ← []
4 tabuList.push(s0)
5 while (not stoppingCondition())
6     sNeighborhood ← getNeighbors(bestCandidate)
7     bestCandidate ← sNeighborhood[0]
8     for (sCandidate in sNeighborhood)
9         if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10            bestCandidate ← sCandidate
11        end
12    end
13    if (fitness(bestCandidate) > fitness(sBest))
14        sBest ← bestCandidate
15    end
16    tabuList.push(bestCandidate)
17    if (tabuList.size > maxTabuSize)
18        tabuList.removeFirst()
19    end
20 end
21 return sBest
```

Zakázaný zoznam (tabu list)

- Numerické skúsenosti s algoritmom zakázaného hľadania ukazujú, že veľkosť zakázaného zoznamu je veľmi dôležitým parametrom pre prehľadávanie stavového priestoru s možnosťou vymaniť sa lokálnych extrémov
- Existujú aj varianty tabu search s viacerými pamäťami s rôznou veľkosťou
  - krátkodobá pamäť
  - strednodobá pamäť
  - dlhodobá pamäť

ĎAKUJEM ZA POZORNOSŤ

STU FIIT

lukas.kohutka@stuba.sk