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
```


Generovanie stavov

- **systematické**
  - exhaustívne (vyčerpávajúce, úplné) prehľadanie priestoru problému.
- **náhodné**
  - Nie je záruka, že sa riešenie nájde, aj ak riešenie existuje (algoritmus Britského múzea)
    hypotéza o opiciach a písacích strojoch

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. opakovane.
  - 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 10^{813} opíc, nie 4 týždne, ale 5 rokov a každá mala svoj počítač, tak by vznikol sonet č. 3. samozrejme, od Shakespeara.

lokálne hľadanie

- metóda hľadania s 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

- "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
    - Výstup na Mt Everest v úplnej hmle
- Hodnota (ohodnocovacej funkcie) je buď
  - Hodnota cieľovej funkcie
  - Hodnota heuristickej funkcie
- Nepozerá sa dopredu pred bezprostredných susedov súčasného stavu.
- Volí náhodne z množiny nasledovníkov, ak viac je hodnotených lepšie

7

Stratégia lokálneho vylepšovania a lokálne extrémy

- ak jestvujú lokálne extrémy, hľadanie nie je optimálne
- jednoduchý spôsob nájdenia riešenia (a často efektívny)
  - viac pokusov hľadania – náhodné začiatky

8

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)

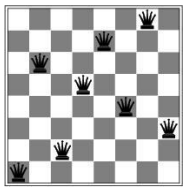
9

Stratégia lokálneho vylepšovania - príklad

Ohodnotenie súčasného stavu: h=17

Znázornené sú h-hodnoty každého možného nasledovníka v každom stĺpci

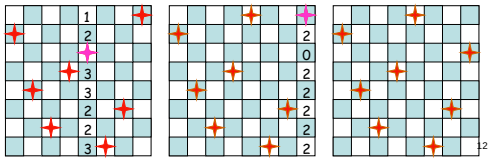
Lokálne minimum pre 8-dám



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



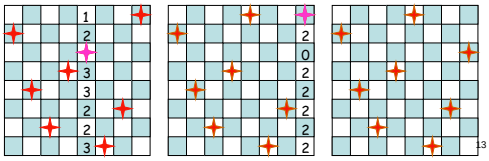
12

8-dám, trochu iná heuristika

opakuj s rozsy
1)
2)

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…
- v 14% prípadov vyrieši problém
- v 86% prípadov zapadne v lokálnom minime
- 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 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

Stratégia lokálneho vylepšovania - variácie

- 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 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 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 (local beam search)

- zdá sa, že je to 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 (local beam search)

- 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)
  - náprava: vybrať k nasledovníkov náhodne, s predispozíciou pre lepšie stavy, (tj s pravdepodobnosťou danou ohodnotením nasledovníka)

stratégia lokálnej optimalizácie

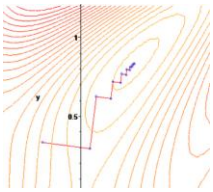
Ako ďalší uzol sa nevyberá hociktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.

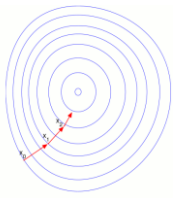
```
function LOKÁLNA-OPTIMALIZÁCIA(problém) returns stav riešenia
   static: súčasný, uzol
           ďalší, uzol

   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

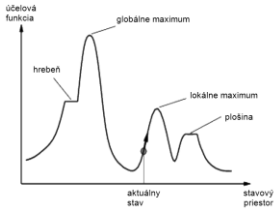
Ako ďalší uzol sa nevyberá hociktorý lepší než súčasný, ale najsľubnejší z jeho nasledovníkov.





stratégie lokálneho vylepšovania / optimalizácie

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).

Simulované žíhanie

- Upustenie od podmienky, že pri hľadaní sa nemôže ani o krok zostúpiť.
- Po uviaznutí v lokálnom maxime sa vykoná niekoľko krokov dolu kopcom, aby sme unikli z okolia lokálneho maxima, namiesto toho aby hľadanie pokračovalo z nejakého náhodne zvoleného miesta.
- Vnútorný cyklus simulovaného žíhania sa podobá lokálnej optimalizácii. Nevyberá sa však najlepší krok, ale náhodný krok.
- Ak sa pritrafí, že vybratý krok naozaj zlepšuje situáciu, tak sa aj vykoná. Inak vykonanie kroku určuje nejaká pravdepodobnosť, ktorá je menšia než 1.

Simulované žíhanie

```
function SIMULOVANÉ-ŽÍHANIE(problém, ROZVRH) returns stav riešenia
    inputs: problém
            rozvrh, zobrazenie času to "teploty"
    static: súčasný, uzol
            ďalší, uzol
            T, "teplota", ktorá riadi pravdepodobnosť krokov nadol

    súčasný ← VYTVOR-UZOL(ZAČIATOČNÝ-STAV[problém])
    for k ← 1 to ∞ do
        T ← ROZVRH[k]
        if T = 0 then return STAV(súčasný)
        ďalší ← náhodne vybratý nasledovník uzla súčasný
        ΔE ← HODNOTA[ďalší] – HODNOTA[súčasný]
        if ΔE > 0 then súčasný ← ďalší
                  else súčasný ← ďalší iba s pravdepodobnosťou e^(ΔE/T)
    end
```


Analógia so žíhaním

- funkcia HODNOTA zodpovedá celkovej energii atómov v materiáli
- *T* zodpovedá teplote.
- ROZVRH určuje, ako sa znižuje teplota.
- Jednotlivé kroky v stavovom priestore zodpovedajú náhodným fluktuáciám spôsobeným teplotným šumom.
- Ak sa teplota znižuje dostatočne pomaly, materiál prijme konfiguráciu s najnižšou energiou (dokonalé usporiadanie).
- V kontexte hľadania riešenia to zodpovedá tvrdeniu, že ak ROZVRH definuje znižovanie *T* dostatočne pomaly, algoritmus nájde globálne optimum.

Simulované žíhanie

- určenie počtu „krokov“ – rozvrh
- v každom kroku:
  - náhodne zaveď zmenu do súčasného stavu
  - nový stav prijmi vždy, ak zlepšuje hodnotu
  - (inak) náhodne prijmi aj stav, ktorý zhoršuje hodnotu
- pomaly znižuj pravdepodobnosť prijatia stavu, zhoršujúceho hodnotu

Simulované žíhanie

- znáhodňujúci algoritmus
- umožňuje rýchlo prehľadať veľké časti stavového priestoru
- prijatím zhoršujúcich zmien sa otvára možnosť vyhnutia sa lokálnym extrémom
- prijatie zhoršujúcej zmeny môže byť zlé
  - pravdepodobnosť ich prijatia sa postupne zmenšuje