Programování
s omezujícími podmínkami

Roman Barták
Katedra teoretické informatiky a matematické logiky

roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Prohledávání

- podmínky jsou užívány pasivně jako test
  - přiřazuji hodnoty proměnným a ...
  - ... zkouším, co to udělá
- **systematické prohledávání**
  - systematicky prochází prostor všech ohodnocení
  - GT, BT, BJ, BM, DB, IB, LDS
- **nesystematické prohledávání**
  - prochází prostor všech ohodnocení, části lze přeskočit
  - Credit Search, Bounded Backtrack
- **lokální prohledávání**
  - prochází prostor všech ohodnocení v krátkých krocích
  - HC, MC, RW, Tabu, GSAT, Genet, simulované žíhání

Systematické prohledávání

- systematicky prochází prostor všech ohodnocení
- systematicky = nic nevynechává

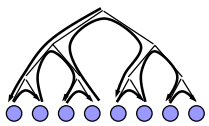
Vlastnosti:

+ **úplné** (pokud řešení existuje, najde ho)
- **efektivita** (může to trvat pěkně dlouho)

Klasifikace:

- **procházení úplných ohodnocení**

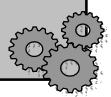
  - **generuj a testuj**
  - spíše u lokálního prohledávání (není systematické)
- **rozšiřování částečného ohodnocení**
  - stromové prohledávání



Generuj a testuj (GT)

- Asi nejobecnější metoda řešení problémů
  1) vygeneruj kandidáta na řešení
  2) ověř, zda se skutečně jedná o řešení
- Použití na řešení CSP
  1) vyber hodnoty pro všechny proměnné
  2) ověř, zda platí podmínky
- Postupně **prochází úplná, ale nekonzistentní ohodnocení**, dokud nenajde (úplné) konzistentní ohodnocení.

```
procedure GT(X:proměnné, C:podmínky)
    V ← vyber první úplné ohodnocení proměnných X
    while V nesplňuje všechny podmínky C do
        V ← systematicky vyber další ohodnocení proměnných X po V
    end while
    return V
end GT
```



Problémy metody GT

- Generuje **zbytečně mnoho ohodnocení**, o kterých je jasné, že nebudou řešením.

Příklad:

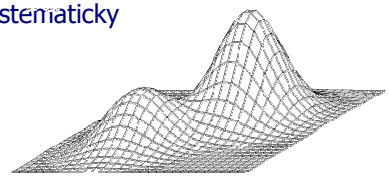
X::{1,2}, Y::{1,2}, Z::{1,2} X = Y, X ≠ Z, Y > Z

X	1	1	1	1	2	2	2
Y	1	1	2	2	1	1	2
Z	1	2	1	2	1	2	1

- **Jak zlepšit GT?**
  - **chytrý generátor**
    - další generované ohodnocení se „poučilo" z chyb předchozích ohodnocení
    - základ technik lokálního prohledávání
  - **spojené generování a testování**
    - podmínka je testována, jakmile to jde (znám hodnoty proměnných)
    - backtracking

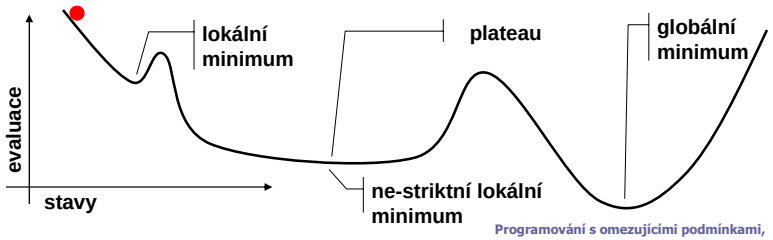
Lokální prohledávání

- **Metoda generuj a testuj**: prochází úplná nekonzistentní ohodnocení dokud nenajde konzistentní ohodnocení.
- Problém GT - generátor nepoužívá výsledku testu
- Následující ohodnocení můžeme generovat na základě výsledku testu.
  - budeme dělat jen **lokální změny** ohodnocení
  - další ohodnocení je „lepší" než předchozí
    - lepší = splňuje více podmínek
  - ohodnocení nejsou generována systematicky
    - ztrácíme úplnost algoritmu
    - výměnou za rychlost



Terminologie

- **stav** - ohodnocení všech proměnných
- **evaluace** - hodnota objektivní funkce (počet nesplněných podmínek)
- **okolí** - množina stavů lokálně se lišících od daného stavu (typicky se dva stavy liší v hodnotě jedné proměnné)
- **lokální optimum** - stav, v jehož okolí není „lepší" stav a zároveň není optimem
- **striktní lokální optimum** - stav, který není optimem a v jehož okolí jsou pouze „horší" stavy
- **ne-striktní lokální optimum** - lokální optimum, které není striktní
- **plateau** - množina stavů se stejnou evaluací
- **globální optimum** - stav, dosahující nejlepší evaluace



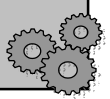
Největší stoupání (HC)

Asi nejznámější metoda lokálního prohledávání (hill climbing)

- začíná v náhodně vybraném stavu
- hledá vždy nejlepší stav v okolí
  - **okolí** = hodnota libovolné proměnné je změněna
  - velikost okolí = $\Sigma_{i=1..n}(D_i-1)$ (= n*(d-1))
- „útěk" ze striktního lokálního minima pomocí restartu

Algoritmus Hill Climbing

```
procedure hill-climbing(Max_Moves)
    restart: s ← random assignment of variables;
    for j:=1 to Max_Moves do            % omezený počet kroků
        if eval(s)=0 then return s
        if s is a strict local minimum then
            go to restart
        else
            s ← neighbourhood with the smallest evaluation value
        end if
    end for
    go to restart
end hill-climbing
```



Minimalizace konfliktů (MC)

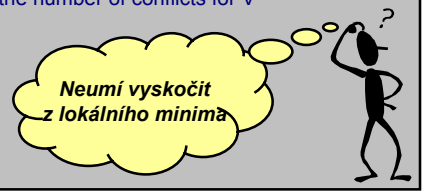
Pozorování:

- okolí u hill-climbing je poměrně velké (n*(d-1))
- pouze změna konfliktní proměnné může přinést zlepšení

- **Metoda minimalizace konfliktů (min-conflicts)**
  - vybere náhodně konfliktní proměnnou a zkusí ji změnit k lepšímu
    - **okolí** = hodnota zvolené proměnné *i* je změněna
    - velikost okolí = (D_i-1) (= (d-1))

Algoritmus Min-Conflicts

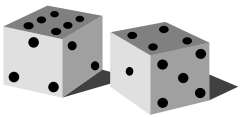
```
procedure MC(Max_Moves)
    s ← random assignment of variables
    nb_moves ← 0
    while eval(s)>0 & nb_moves<Max_Moves do
        choose randomly a variable V in conflict
        choose a value v' that minimises the number of conflicts for V
        if v' ≠ current value of V then
            assign v' to V
            nb_moves ← nb_moves+1
        end if
    end while
    return s
end MC
```



Náhodná procházka (RW)

Jak se dostat z lokálního optima bez restartu (tj. lokálním krokem)?

- Přidání „šumu" do algoritmu!



Náhodná procházka (random walk)

- **stav z okolí je volen náhodně** (resp. hodnota proměnné je volena náhodně)
- tato metoda samostatně těžko povede k řešení
- potřebujeme nějaké zacílení
  - náhodná procházka je kombinována s heuristikou určující další tah pomocí pravděpodobnostního rozložení:
    - p - pravděpodobnost náhodného kroku
    - (1-p) - pravděpodobnost použití směrové heuristiky

MCRW

- MC vede algoritmus k cíli (splnění všech podmínek) a RW umožňuje vyskočení z lokálního minima.

Algoritmus Min-Conflicts-Random-Walk

```
procedure MCRW(Max_Moves,p)
    s ← random assignment of variables
    nb_moves ← 0
    while eval(s)>0 & nb_moves<Max_Moves do
        choose randomly a variable V in conflict
        if probability p verified then
            choose randomly a value v' for V
        else
            choose a value v' that minimises the number of conflicts for V
        end if
        if v' ≠ current value of V then
            assign v' to V
            nb_moves ← nb_moves+1
        end if
    end while
    return s
end MCRW
```

0.02 ≤ p ≤ 0.1

SDRW

- V kombinaci s RW můžeme použít také HC heuristiku.
- Potom není potřeba restart.

Algoritmus Steepest-Descent-Random-Walk

```
procedure SDRW(Max_Moves,p)
    s ← random assignment of variables
    nb_moves ← 0
    while eval(s)>0 & nb_moves<Max_Moves do
        if probability p verified then
            choose randomly a variable V in conflict
            choose randomly a value v' for V
        else
            choose a move <V,v'> with the best performance
        end if
        if v' ≠ current value of V then
            assign v' to V
            nb_moves ← nb_moves+1
        end if
    end while
    return s
end SDRW
```

Tabu seznam

Pozorování: Setrvání v lokálním minimu je speciálním případem cyklu.

- **Jak se obecně zbavit cyklů?**
  - Stačí si **pamatovat předchozí stavy** a zakázat opakování stavu.
    - příliš paměťově náročné (mnoho stavů)
  - Můžeme si pamatovat pouze několik posledních stavů.
    - zabrání „krátkým" cyklům
- **Tabu seznam** = seznam **zakázaných stavů**
  - stav lze popsat význačným atributem (nemusí být uschován celý)
    - ⟨proměnná, hodnota⟩ - zachycuje změnu stavu (původní hodnotu)
  - tabu seznam má fixní délku k (**tabu tenure**)
    - „staré" stavy ze seznamu vypadávají s přicházejícími novými stavy
  - stav popsaný prvkem tabu seznamu je zakázaný (je tabu)
- **Aspirační kritérium** = odtabuizování stavu
  - do stavu lze přejít, i když je jeho abstrakce v tabu seznamu
  - například: krok vede k celkově lepšímu stavu

Prohledávání s tabu seznamem

- Tabu seznam **zabraňuje krátkému cyklení.**
- Povoleny jsou pouze tahy mimo tabu seznam resp. tahy splňující aspirační kritérium.

Algoritmus Tabu Search

```
procedure tabu-search(Max_Moves)
    s ← random assignment of variables
    num_moves ← 0
    initialise randomly the tabu list
    while eval(s)>0 & num_moves<Max_Moves do
        choose a move <V,v'> with the best performance among the non-tabu
            moves and the moves satisfying the aspiration criteria
        introduce <V,v> in the tabu list, where v is the current value of V
        remove the oldest move from the tabu list
        assign v' to V
        num_moves ← num_moves+1
    end while
    return s
end tabu-search
```