Programování s omezujícími podmínkami

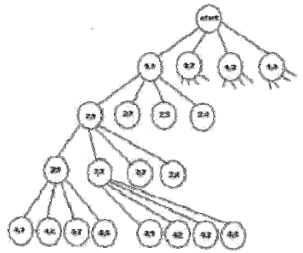
4

Roman Barták
Katedra teoretické informatiky a matematické logiky

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

Backtracking

- **prochází částečná konzistentní ohodnocení dokud nenajde úplné (konzistentní) ohodnocení**
  - oproti metodě „generuj a testuj" je fáze testování splnění podmínek realizována v průběhu generování
  - pokud můžeme test provést nad částečně vygenerovaným kandidátem řešením, je BT vždy lepší než GT, protože nemusí procházet všechny kandidáty
- Základní princip prohledávání s navracení při řešení CSP:
  1. postupně ohodnocuj proměnné
  2. po každém ohodnocení otestuj podmínky, jejichž všechny proměnné již mají hodnotu



Problémy backtrackingu

- **thrashing** — Look Back
  - neumí využít informace o důvodu (zdroji) konfliktu

  Příklad: A,B,C,D,E:: 1..10, A>E
  - vyzkouší všechny možnosti pro B,C,D, než odhalí, že A≠1

  Řešení: **backjumping** (skok na původce chyby)

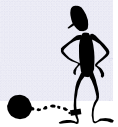
- **redundantní práce**
  - nepamatuje si již odhalené konflikty mezi přiřazením proměnných

  Příklad: A,B,C,D,E:: 1..10, B+8<D, C=5*E
  - při hledání hodnot pro C a E, stále zkouší přiřadit do D hodnoty 1,..,9

  Řešení: **backmarking** (pamatuje si dobrá/chybná přiřazení)
- **pozdní odhalení chyby** — Look Ahead
  - nesplnění podmínky odhalí teprve, když ohodnotí její proměnné

  Příklad: A,B,C,D,E:: 1..10, A=3*E
  - odhalí že A>2, až při ohodnocování proměnné E

  Řešení: **forward checking** (kontrola podmínek dopředu)



Backjumping

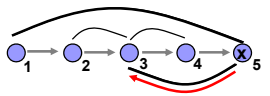
Backjumping je metoda pro **odstranění thrashingu**

Jak?

1) identifikuj důvod neúspěchu (nelze přiřadit hodnotu proměnné)
2) skoč až na konfliktní proměnnou

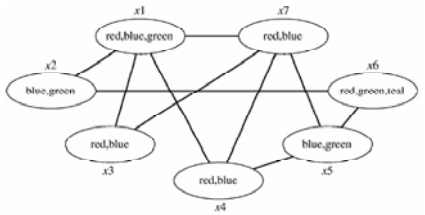
- Stejný dopředný běh jako u backtrackingu, pouze zpět skáčeme dále, tj. odstraníme prohledávání nerelevantních kombinací hodnot!
- **Jak zjistíme kam až skočit? Co je důvodem neúspěchu?**
  - vyber podmínky, které obsahují ohodnocovanou proměnnou a jsou testovány na splnění
  - z těchto podmínek zjisti nejbližší (v pořadí prohledávání) proměnnou, kterou obsahují

Grafem řízený backjumping (graph-directed backjuping)



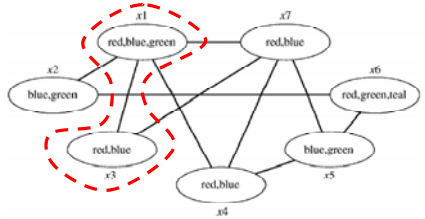
Backjumping
řízený grafem

- Uvažujme problém **barvení grafu**, kde barvy přiřazujeme v pořadí x1, x2, …, x7.



- Kam skočíme, pokud neobarvíme x4?
  - **x1**
- Kam skočíme, pokud neobarvíme x5?
  - **x4**
  - A co když už nelze obarvit ani x4?
  - **x1**

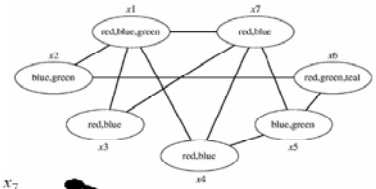
- Vypadá to, že když nelze obarvit vrchol, **můžeme skočit na jeho nejbližšího předka**, ale …



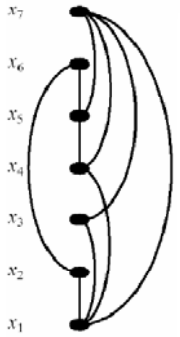
- Kam skočíme, pokud neobarvíme x7?
  - **x5**
  - A co když už nelze obarvit ani x5?
  - **x4**
  - A co když už nelze obarvit ani x4?
  - **x3**

Backjumping
řízený grafem

- Při skoku zpět stačí **uvolnit některý slepý uzel** (slepý uzel = uzel, kde nelze vybrat hodnotu proměnné).



- z listu x skočíme na nejbližšího předchůdce x v grafu podmínek
  (x7 → x5, x4, x3, x1)
- z vnitřního vrcholu x skočíme na nejbližšího předchůdce ale všech slepých uzlů, přes které jsme do x „doskákali
  (x7 → x5, x4, x3, x1 → x4, x3, x1 → x3, x1)



- označme **anc(x) předchůdce proměnné x** v grafu podmínek uspořádaném dle pořadí při ohodnocování (lze nastavit před prohledáváním dle struktury grafu)
  - anc(x7) = {x5, x4, x3, x1}
- nechť jsme se do uzlu x vrátili z uzlů y1,…,yk a nechť pro proměnnou x nemáme další hodnotu kompatibilní s předchůdci
- potom **skočíme na nejbližší proměnnou z anc(x) ∪ anc(y1) ∪ … ∪ anc(yk) – {x,y1,…,yk}**

BJ řízený grafem
rekurzivně

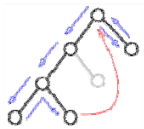
```
procedure GraphBJ(X:variables, V:assignment, C:constraints)
    if X = {} then return V
    x ← select a not-yet assigned variable from X
    conflict ← anc(x)
    for each value h from the domain of x do
       if constraints C are consistent with V ∪ {x/h} then
          R ← GraphBJ(X – {x}, V ∪ {x/h}, C)
          if R = fail(JumpSet) then                 % skok zpět (backjump)
             if x∉JumpSet then return R             % na proměnnou před x
             conflict ← conflict ∪ JumpSet – {x}    % na x
          else return R                             % řešení nalezeno
    end for
    return fail(conflict)
end GraphBJ
```

volá se GraphBJ(X, {}, C)

BJ řízený grafem
iterativně

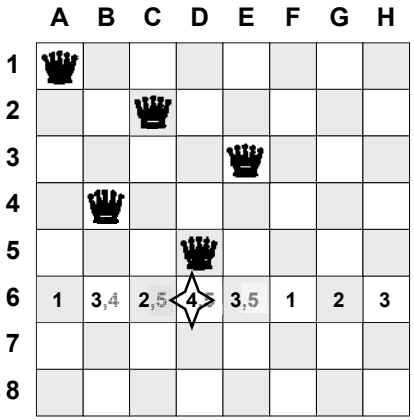
```
procedure GraphBJ(X:variables, C:constraints)
    i ← 1, D'i ← Di , li ← anc(xi)
    while 1 ≤ i ≤ n do
       instantiate_and_check(i, C)
       if xi = null then
          iprev ← i, i ← latest index in li, li ← li ∪ liprev – {xi}
       else
          i ← i + 1, D'i ← Di , li ← anc(xi)
       end if
    end while
    if i = 0 then return fail
    return {x1,…, xn}
end GraphBJ
```

```
procedure instantiate_and_check(i, C:constraints)
    while D'i is not empty do
       select and delete some element b from D'i
       xi ← b
       if constraints C consistent with {x1,…, xi} then return
    end while
    xi ← null
end instantiate_and_check
```



Backjumping
zjemnění skoků

Problém N-dam



Dámy přiřazujeme po řádcích, tj. pro každou dámu hledáme sloupec.

Dámu v řádku 6 nelze přiřadit!

1. Do políčka zapisujeme čísla konfliktních dam.

2. V každém políčku vybereme nejvzdálenější dámu.

3. Mezi políčky vybereme nejbližší dámu a tam skočíme.

Poznámka:

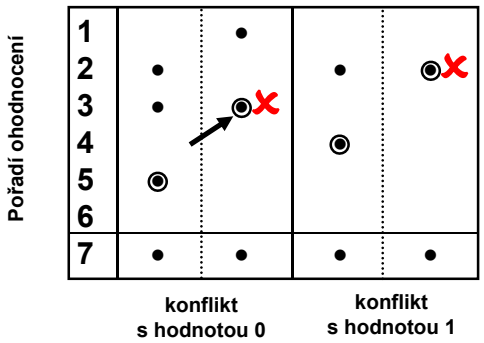
grafem řízený backjumping se zde chová stejně jako chronologický backtracking (graf podmínek je úplný)!

Konfliktní proměnné

Jak obecně najdeme konfliktní proměnnou?

Situace:

- ohodnocujeme proměnnou č. 7 (možné hodnoty 0 a 1)
- symbol • vyznačuje jaké proměnné jsou zahrnuty v konfliktní (nesplněné) podmínce



Sedmé proměnné nelze přiřadit žádnou z hodnot 0,1!

1. U každé nesplněné podmínky najdeme nejbližší proměnnou (o).

2. Z nesplněných podmínek pro hodnotu vybereme vzdálenější z těchto proměnných (x).

3. Z proměnných získaných pro jednotlivé hodnoty zvolíme tu nejbližší a na ní skočíme.

Test konzistence
Gaschnigův backjumping

kromě testování konzistence podmínek také počítáme konfliktní úroveň

```
procedure consistent(Labelled, Constraints, Level)
    J ← Level                  % úroveň, kam skákat
    NoConflict ← true          % identifikace konfliktu
    for each C in Constraints do
      if all variables from C are Labelled then
          if C is not satisfied by Labelled then
              NoConflict ← false
              J ← min {J, max{L | X ∈ vars(C) & X/V/L in Labelled & L<Level}}
          end if
      end if
    end for
    if NoConflict then return true
                  else return fail(J)
end consistent
```

- **V je hodnota proměnné X**
- **L je pořadové číslo (úroveň) proměnné X při ohodnocování**

Gaschnigův backjumping

```
procedure GBJ(Unlabelled, Labelled, Constraints, PreviousLevel)
    if Unlabelled = {} then return Labelled
    pick first X from Unlabelled
    Level ← PreviousLevel+1
    Jump ← 0
    for each value V from D_X do
        C ← consistent({X/V/Level} ∪ Labelled, Constraints, Level)
        if C = fail(J) then
            Jump ← max {Jump, J}
        else
            Jump ← PreviousLevel
            R ← GBJ(Unlabelled-{X},{X/V/Level} ∪ Labelled,Constraints, Level)
            if R ≠ fail(Level) then return R          % úspěch nebo skok dál
        end if
    end for
    return fail(Jump)          % skok ke konfliktní proměnné
end GBJ
```

volá se GBJ(Variables,{},Constraints,0)

Gaschnigův backjumping
iterativně

```
procedure GBJ(X:variables, C:constraints)
    i ← 1, D`i ← Di , jumpi ← 0
    while 1 ≤ i ≤ n do
        xi ← select_value(i, C)
        if xi = null then
            i ← jumpi
        else
            i ← i + 1
            D`i ← Di
            jumpi ← 0
        end if
    end while
    if i = 0 then return fail
    return {x1,..., xn}
end GBJ
```

```
procedure select_value(i, C:constraints)
    while D`i is not empty do
        select and delete some element b from D`i
        consistent ← true
        k ← 1
        while k<i and consistent do
            if k>jumpi then jumpi ← k
            if xi=b consistent with {x1,..., xk} in C then
                    k ← k + 1
            else    consistent ← false
        end while
        if consistent then return b
    end while
    return null
end instantiate_and_check
```


Backjumping
krátké shrnutí

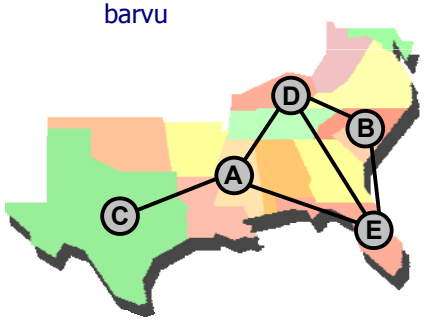
- **Grafem řízený backjumping**
  - řídí se pouze strukturou sítě podmínek (nebere v úvahu splnění/nesplnění konkrétní podmínky)
  - umí realizovat několik zpětných skoků
- **Gaschnigův backjumping**
  - bere v úvahu, které podmínky skutečně způsobily konflikt
  - skočí pouze jeden skok (pokud se do proměnné podaří přiřadit hodnotu, jde se zpět pouze o jednu úroveň jako v chronologickém backtrackingu)
- **Konfliktem řízený backjumping (CBJ)**
  - můžeme spojit výhody obou metod (upřesnění kam skočit přes nesplněné podmínky a několik skoků)
  - při návratu budeme vracet nalezenou konfliktní množinu, která si využije při dalším skoku (pokud nenajdeme novou hodnotu proměnné, na kterou jsme skočili)
    - přenášíme důvod konfliktu k již ohodnoceným proměnným

Problémy backjumpingu

Při skoku zpět zapomíná (zahazuje) již udělanou práci!

Příklad:

- obarvěte graf třemi barvami tak, že sousední vrcholy mají různou barvu

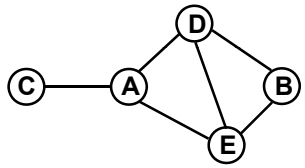


uzel	barva		
A	1		1
B	2		1
C	*1* 2		*1* 2
D	*1 2* 3	skok zpět	*1* 2
E	*1 2 3*		*1 2* 3

Při druhém průchodu (ohodnocení) C děláme zbytečnou práci, stačilo nechat původní hodnotu 2, změnou B se nic neporušilo.

Dynamický backtracking
příklad

Stejný graf (A,B,C,D,E), stejné barvy (1,2,3) ale jiný postup.



Backjumping
+ pamatování si důvodu konfliktu
+ přenos důvodu konfliktu
+ změna pořadí proměnných

= **DYNAMICKÝ BACKTRACKING**

uzel	1	2	3
A	•		
B		•	
C	A	•	
D	A	B	•
E	A	B	D

skok zpět
+ přenos důvodu chyby

uzel	1	2	3
A	•		
B		•	
C	A	•	
D	A	B	AB
E	A	B	

skok zpět
+ přenos důvodu chyby
+ změna pořadí B, C

uzel	1	2	3
A	•		
C	A	•	
B	•	A	
D	A	•	
E	A	D	•

• vybraná barva
AB důvod konfliktu

Vrchol C (respektive celý graf, který na něm případně „visel“) není potřeba přebarvovat.

Dynamický backtracking

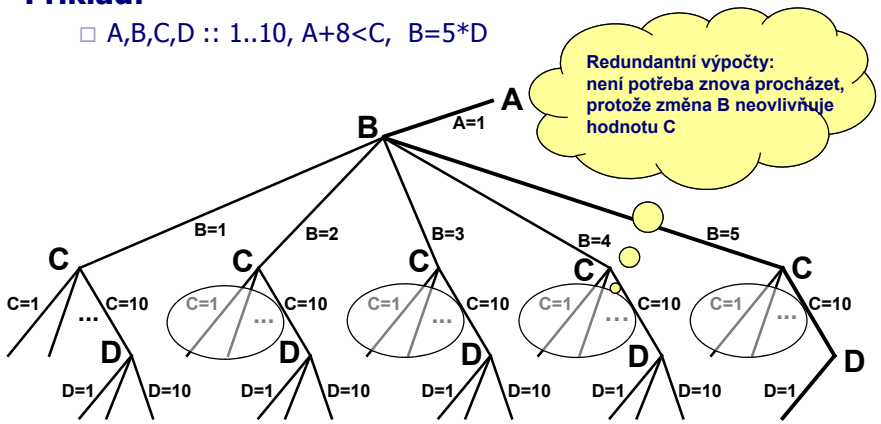
```
procedure DB(Variables, Constraints)
    Labelled ← {};   Unlabelled ← Variables
    while Unlabelled ≠ {} do
        select X in Unlabelled
        Values_X ← D_X - {values inconsistent with Labelled using Constraints}
        if Values_X = {} then
            let E be an explanation of the conflict (set of conflicting variables)
            if E = {} then failure
            else
                let Y be the most recent variable in E
                unassign Y (from Labelled) with eliminating explanation E-{Y}
                remove all the explanations involving Y
            end if
        else
            select V in Values_X
            Unlabelled ← Unlabelled - {X}
            Labelled ← Labelled ∪ {X/V}
        end if
    end while
    return Labelled
end DB
```


Redundance backtrackingu

- **Co je to redundantní práce?**
  - opakování výpočtu, jehož výsledek už máme k dispozici

Příklad:

- A,B,C,D :: 1..10, A+8<C, B=5*D

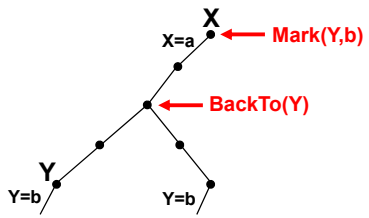


Základy backmarkingu

Základní princip (pracujeme s binárním CSP):

- **Mark(X,V)** u každé hodnoty V z domény proměnné X si pamatujeme nejvzdálenější konflikt (nejvzdálenější proměnnou)
- **BackTo(X)** u každé proměnné X si pamatujeme místo nejvzdálenějšího návratu (od chvíle posledního ohodnocení X)

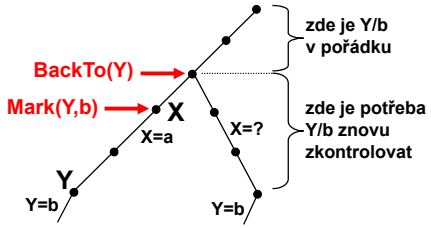
Situace 1
(Mark<BackTo)



Y/b je nekonzistentní s X/a (konzistentní se vším nad X)

Y/b je s X/a stále nekonzistentní, nemusíme kontrolovat

Situace 2
(Mark≥BackTo)



Y/b je nekonzistentní s X/a (ale je konzistentní se vším předtím)

Backmarking - příklad

Problém N-dam

	A	B	C	D	E	F	G	H	
1	♛								1
2	1	1	♛						1
3	1	2	1	2	♛				1
4	1	♛							1
5	1	4	2	♛	1	2	3	♛	1
6	1	3	2	4	3	1	2	3	5
7									1
8									1

1. Dámy přiřazujeme po řádcích, tj. pro každou dámu hledáme sloupec.

2. Vedle šachovnice píšeme úrovně návratu (BackTo). Na začátku všude 1.

3. Do políčka zapisujeme čísla nejvzdálenějších konfliktních dam (Mark). Na začátku všude 1.

4. Dámu v řádku 6 nelze přiřadit!

5. Vracíme se na 5, opravíme BackTo.

6. Když znova přijdeme na 6, všechny pozice jsou stále špatné (Mark<BackTo).

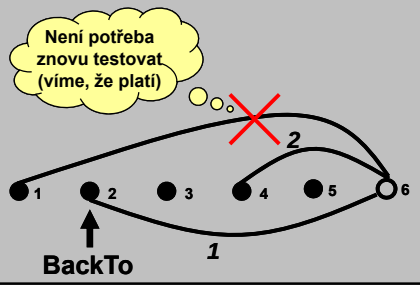
Poznámka:
backmarking lze kombinovat s backjumpingem (a máme to zadarmo)

Test konzistence

backmarking

- testování konzistence jen u podmínek, kde došlo ke změně, plus počítání nejvzdálenější konfliktní úrovně

```
procedure consistent(X/V, Labelled, Constraints, Level)
    for each Y/VY/LY in Labelled such that LY≥BackTo(X) do
      % bereme pouze proměnné Y, které se mohly změnit
      % jdeme v rostoucím pořadí podle LY (od nejstarší)
       if X/V is not compatible with Y/VY using Constraints then
            Mark(X,V) ← LY
            return fail
       end if
    end for
    Mark(X,V) ← Level-1
    return true
end consistent
```



Backmarking

algoritmus

```
procedure BM(Unlabelled, Labelled, Constraints, Level)
  if Unlabelled = {} then return Labelled
  pick first X from Unlabelled          % pořadí proměnných je pevné
  for each value V from D_X do
    if Mark(X,V) ≥ BackTo(X) then       % hodnota se musí znova kontrolovat
      if consistent(X/V, Labelled, Constraints, Level) then
        R ← BM(Unlabelled-{X}, Labelled ∪ {X/V/Level}, Constraints, Level+1)
        if R ≠ fail then return R       % řešení nalezeno
      end if
    end if
  end for
  BackTo(X) ← Level-1                   % budeme se vracet k předch. proměnné
  for each Y in Unlabelled do           % řekni všem o návratu
    BackTo(Y) ← min {Level-1, BackTo(Y)}
  end for
  return fail                           % návrat k předchozí proměnné
end BM
```