Programování 3 omezujícími podmínkami

Roman Barták, KTIML

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

Co bylo minule

Formální definice CSP
- proměnné, konečné domény a podmínky

Binarizace podmínek
- duální kódování
- kódování se skrytou proměnou

Základní algoritmy systematického prohledávání
- generuj a testuj
- backtracking

  problémy backtrackingu
  - thrashing → *backjumping*
  - redundantní práce → *backmarking*
  - pozdní odhalení chyby → forward checking

Backjumping

Odstranění thrashingu

Jak?

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

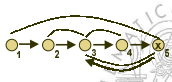
Stejný běh jako u backtrackingu, pouze zpět skáčeme dále, tj. odstraníme prohledávání nerelevantních kombinací hodnot!

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

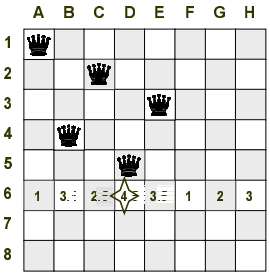
Grafem řízený (graph-directed) backjumping



vylepšení: vybírej proměnnou pouze u nesplněných podmínek

Backjumping - příklad

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 (úplný graf)!

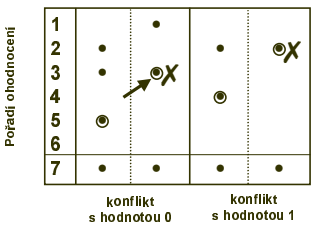
Identifikace konfliktní proměnné

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 ni skočíme.

Test konzistence u backjumpingu

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 in 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
```

Algoritmus backjumpingu

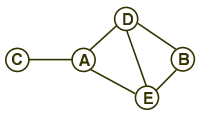
```
procedure BJ(Unlabelled, Labelled, Constraints, PreviousLevel)
    if Unlabelled = {} then return Labelled
    pick first X from Unlabelled
    Level ← PreviousLevel+1
    Jump ← 0
    for each value V from D_X do
        C ← consistent({X/V/Level} ∪ Labelled, Constraints, Level)
        if C = fail(J)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

volání BJ(Variables,{},Constraints,0)
```


Problémy backjumpingu

Při skoku zpět zapomíná (zahazuje) již udělanou práci!

Příklad:

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



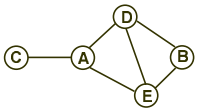
uzel	barva	
A	1	1
B	2	1
C	1 2	1 2
D	1 2 3	1 2
E	1 2 3	1 2 3

skok zpět

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

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	B	•

• vybraná barva
AB důvod konfliktu

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

Algoritmus dynamického backtrackingu

```
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
```

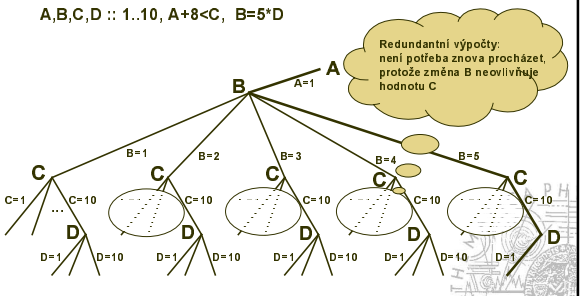

Redundance backtrackingu

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

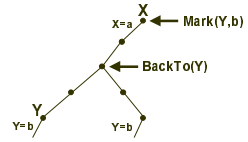


Základy backmarkingu

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)

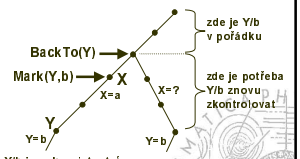
Situace 1
(Mark<BackTo)



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

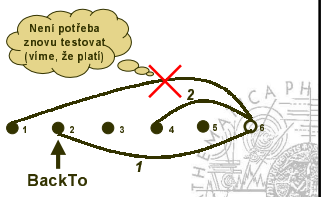
1. Dámy přiřazujeme po řádcích, tj. pro každou dámu hledáme sloupec.
2. Vedle šachovnice píšeme úrovně návratu (BackTo). Na začátku všude 1.
3. Do políčka zapisujeme čísla nejvzdálenějších konfliktních dam (MarkTo). Na začátku všude 1.
4. Dámu v řádku 6 nelze přiřadit!
5. Vracíme se na 5, opravíme BackTo.
6. Když znova přijdeme na 6, všechny pozice jsou stále špatné (MarkTo<BackTo).

Poznámka:
backmarking lze kombinovat s backjumpingem (zdarma)

Test konzistence u backmarkingu

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
```



Algoritmus backmarkingu

```
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
```