Programování s omezujícími podmínkami

7

Roman Barták
Katedra teoretické informatiky a matematické logiky

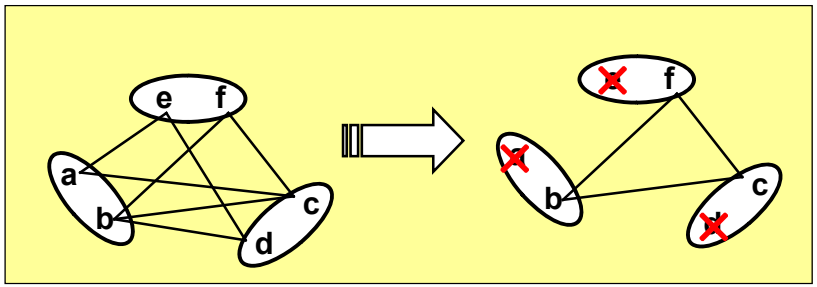
roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Na půli cesty od AC k PC

Jak oslabit PC, aby algoritmus:
- neměl paměťové nároky PC,
- neměnil graf podmínek,
- byl silnější než AC?

- **Testujeme PC jen v případě, když je šance, že to povede k vyřazení hodnoty z domény proměnné!**

Příklad:

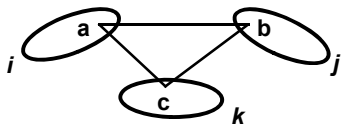


Omezená konzistence po cestě

- PC hrany se testuje pouze tehdy, pokud vyřazení dvojice může vést k vyřazení některého z prvků z domény příslušné proměnné.
- Jak to poznáme?
  - Jedná se o jedinou vzájemnou podporu.

Definice:

Vrchol i je **omezeně konzistentní po cestě** (**restricted path consistent**) právě když:
- každá hrana vedoucí z i je hranově konzistentní (AC)
- pro každé $a \in D_i$ platí:
  je-li b jediná podpora a ve vrcholu j, potom v každém vrcholu k (spojeném s i a j) existuje hodnota c tak, že (a,c) a (b,c) jsou kompatibilní s příslušnými podmínkami (PC).



Algoritmus RPC

inicializace

Založeno na AC-4: počítání podpor + seznam cest pro PC

```
procedure INITIALIZE(G)
    Q_AC ← {} , Q_PC ← {} , S ← {}                    % vyprázdnění datových struktur
    for each (i,j)∈arcs(G) do
        for each a∈D_i do
            total ← 0
            for each b∈D_j do
                if (a,b) is consistent according to the constraint C_i,j then
                        total ← total + 1,  S_j,b ← S_j,b ∪ {<i,a>}
            end for
            counter[(i,j),a] ← total
            if counter[(i,j),a] = 0 then
                Q_AC ← Q_AC ∪ {<i,a>},  delete a from D_i
            else if counter[(i,j),a] = 1 then
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        Q_PC ← Q_PC ∪ {(<i,a>,j,k)}
            end if
        end for
    end for
    return (Q_AC, Q_PC)
end INITIALIZE
```

Algoritmus RPC

kontrola AC

```
procedure PRUNE(Q_AC, Q_PC)
    while Q_AC non empty do
        select and delete any pair <j,b> from Q_AC
        for each <i,a> from S_j,b do
            counter[(i,j),a] ← counter[(i,j),a] - 1
            if counter[(i,j),a] = 0 & "a" is still in D_i then
                    delete "a" from D_i
                    Q_AC ← Q_AC ∪ {<i,a>}
            else if counter[(i,j),a] = 1 then
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        Q_PC ← Q_PC ∪ {(<i,a>,j,k)}
                else
                    for each k such that (i,k)∈arcs(G) & (k,j)∈arcs(G) do
                        if counter[(i,k),a] = 1 then
                            Q_PC ← Q_PC ∪ {(<i,a>,k,j)}
            end if
        end for
    end while
    return Q_PC
end PRUNE
```


Algoritmus RPC

Nejprve uděláme AC a potom testuje vybrané PC, případně se vracíme k AC.

```
procedure RPC(G)
    (Q_AC, Q_PC) ← INITIALIZE(G)
    Q_PC ← PRUNE(Q_AC, Q_PC)              % první běh AC
    while Q_PC non empty do
        select and delete any triple (<i,a>,j,k) from Q_PC
        if a∈D_i then
            {<j,b>} ← {<j,x> ∈ S_ia | x∈D_j }   % jediná podpora pro a
            if {<k,c> ∈ S_ia ∩ S_jb | c ∈ D_k } = ∅ then
                counter[(i,j),a] ← 0
                delete "a" from D_i
                Q_PC ← PRUNE({<i,a>}, Q_PC)   % opakujeme AC
            end if
        end if
    end while
end RPC
```

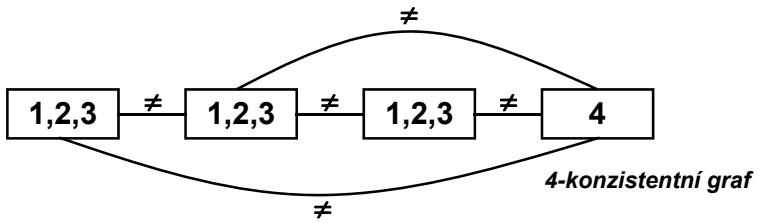

k-konzistence

Mají AC a PC něco společného?

- AC: rozšiřujeme jednu hodnotu do druhé proměnné
- PC: rozšiřujeme dvojici hodnot do třetí proměnné
- ... můžeme pokračovat

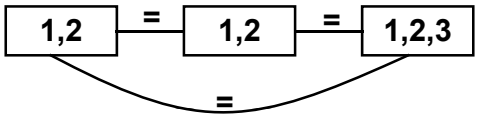
Definice:

CSP je k-konzistentní, právě když libovolné konzistentní ohodnocení (k-1) různých proměnných můžeme rozšířit do libovolné k-té proměnné.



4-konzistentní graf

Silná k-konzistence



3-konzistentní graf

není 2-konzistentní graf!

Definice:

CSP je silně k-konzistentní, právě když je j-konzistentní pro každé $j \le k$.

Vlastnosti:

Zřejmě: silná k-konzistence ⇒ k-konzistence

Dokonce: silná k-konzistence ⇒ j-konzistence $\forall j \le k$

Obecně neplatí: k-konzistence ⇒ silná k-konzistence

Pojmenování:

NC = silná 1-konzistence = 1-konzistence

AC = (silná) 2-konzistence

PC = (silná) 3-konzistence

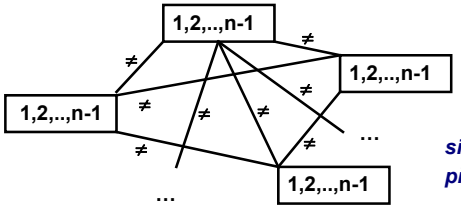
- někdy se říká **silná konzistence po cestě**, pokud NC+AC+PC

Jak velké k potřebujeme?

Máme-li graf s n vrcholy, jak silnou konzistenci potřebujeme abychom přímo našli řešení?

Pro graf s n vrcholy potřebujeme silnou n-konzistenci!

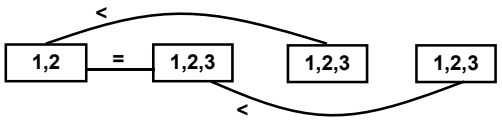
- n-konzistence nestačí - viz předchozí příklad
- silná k-konzistence pro k<n také nestačí



graf s n vrcholy
domény 1..(n-1)

silně k-konzistentní pro každé k<n
přesto nemá řešení

A co tento graf?

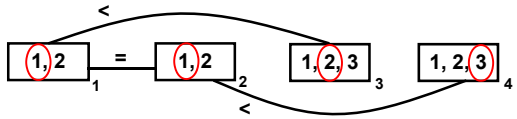


Stačí nám pouze (D)AC!
Protože se jedná o strom.

Řešení bez navracení

Definice:

CSP problém vyřešíme bez navracení, pokud při nějakém uspořádání proměnných můžeme pro každou proměnnou vždy najít hodnotu kompatibilní s již ohodnocenými proměnnými.



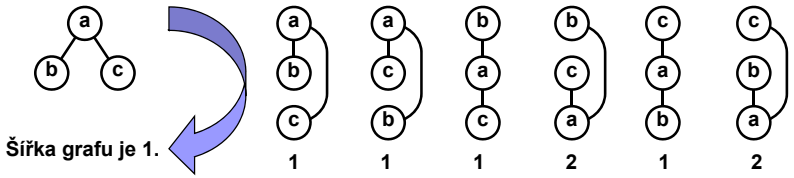
Jak zjistit úroveň konzistence potřebnou pro daný graf?

Pozorování:

- proměnná musí být kompatibilní s již ohodnocenými proměnnými tj. s tolika proměnnými, kolik má „zpětných" hran
- pro k zpětných hran potřebujeme (k+1)-konzistenci
- je-li m maximum počtu zpětných hran pro všechny vrcholy, stačí nám silná (m+1)-konzistence
- při různém uspořádání vrcholů je počet zpětných hran různý
- samozřejmě hledáme uspořádání s nejmenším m

Šířka grafu

- **Uspořádaný graf** je graf s lineárním uspořádáním vrcholů.
- **Šířka vrcholu** v uspořádaném grafu je počet hran vedoucích z tohoto vrcholu do předchozích vrcholů.
- **Šířka uspořádaného grafu** je maximum z šířek jeho vrcholů.
- **Šířka grafu** je minimum z šířek všech jeho uspořádaných grafů.




```
procedure MinWidthOrdering((V,E))
    Q ← {}
    while V not empty do
        N ← select and delete node with the smallest #edges from (V,E)
        enqueue N to Q
    return Q
end MinWidthOrdering
```

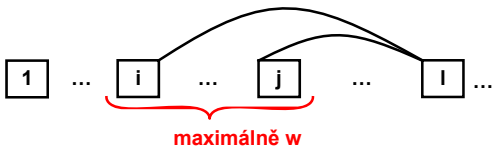

Šířka grafu a stupeň konzistence

Tvrzení:

Pokud je graf podmínek silně k-konzistentní a k>w, kde w je šířka grafu podmínek, potom existuje uspořádání proměnných pro vyřešení CSP bez navracení.

Důkaz:

- graf má šířku w, tj. existuje uspořádaný graf s touto šířkou
- speciálně, počet zpětných hran pro každou proměnnou je max. w
- proměnné ohodnocujeme v pořadí uspořádání grafu
- nyní, pokud ohodnocujeme proměnnou:
  - musíme najít hodnotu kompatibilní se všemi již ohodnocenými proměnnými, které jsou s proměnnou spojené podmínkou (hranou)
  - nechť takových proměnných je m, potom $m \leq w$
  - graf je (m+1)-konzistentní, tedy taková hodnota musí existovat

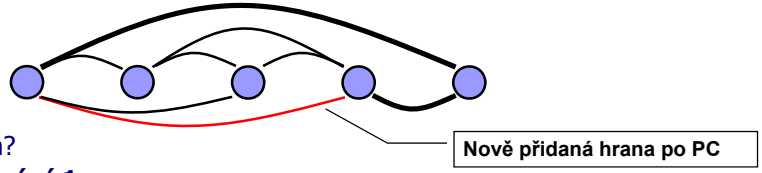


Obecná směrová konzistence

AC (silná 2-konzistence) stačí na stromové grafy (šířka 1).

Jak je to s PC a vyššími typy konzistence?

- PC mění strukturu grafu - přidává nové hrany!
- Tedy, pokud vezmeme graf šířky 2 a provedeme PC, můžeme zvětšit šířku grafu!



Co s tím?

Pozorování 1:

- na stromy nám stačí DAC (děláme ve směru ke kořenu)

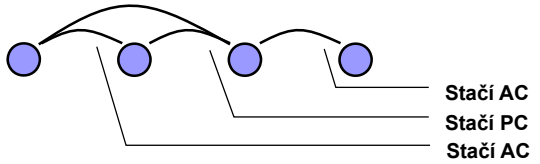
Definice:

CSP je směrově k-konzistentní při nějakém uspořádání proměnných, právě když libovolné konzistentní ohodnocení (k-1) různých proměnných můžeme rozšířit do libovolné k-té proměnné, která je v uspořádání za touto (k-1)-ticí.

Adaptivní konzistence

Pozorování 2:

- v celém grafu nepotřebujeme všude stejnou konzistenci

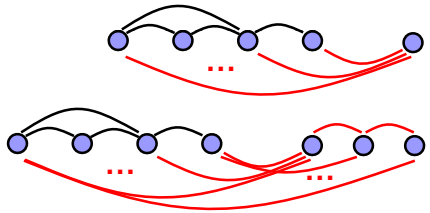


- **Adaptivní konzistence**
  - zajišťuje směrovou i-konzistenci, kde i se mění podle šířky zpracovávaného vrcholu
  - vrcholy jsou zpracovávány ve směru proti uspořádání grafu
  - nové hrany přibývají pouze v dosud nezpracované části
  - výslednou topologii (šířku) grafu lze zjistit před spuštěním algoritmu

(i,j)-konzistence

Při k-konzistenci rozšiřujeme (k-1) proměnných o další proměnnou, tj. vyřazujeme (k-1)-tice, které nelze rozšířit na další proměnnou.

Můžeme ještě zobecnit!



Definice:

CSP je **(i,j)-konzistentní**, právě když libovolné konzistentní ohodnocení i různých proměnných můžeme rozšířit do libovolné množiny j nebo méně než j dalších proměnných.

CSP je **silně (i,j)-konzistentní**, právě když je (k,j)-konzistentní pro každé $k \leq i$.

- k-konzistence = (k-1,1)-konzistence
- AC = (1,1)-konzistence
- PC = (2,1)-konzistence

Inverzní konzistence

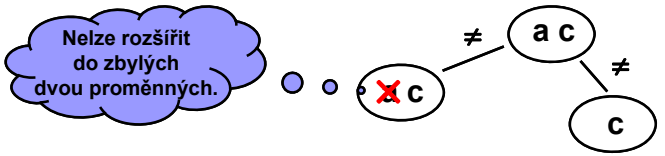
Je-li v (i,j)-konzistenci i větší než 1, **musíme pracovat s i-ticemi**, což znamená velké paměťové nároky (viz PC).

Co to zkusit naopak, tj. **i necháme 1 a zvětšujeme j**?

První náznak jsme již měli:
RPC je (1,1)-konzistence a občas (1,2)-konzistence

Definice:

- (1,k)-konzistenci nazýváme **inverzní konzistencí**.
  Pro danou hodnotu hledáme podporu v dalších k proměnných. Pokud taková podpora chybí, vyřadíme hodnotu z domény.
- **inverzní hranová konzistence** = hranová konzistence
- **inverzní konzistence po cestě (PIC)** = (1,2)-konzistence



(Neighbourhood Inverse Consistency) NIC

Pozorování:

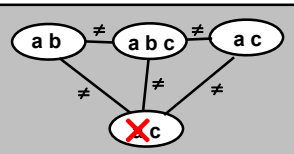
Zjišťování inverzní konzistence má smysl, pokud je alespoň jedna testovací proměnná svázaná s danou proměnnou.

Můžeme zjišťovat konzistenci právě jen v okolí každé proměnné.

Definice:

CSP je inverzně konzistentní pro okolí (NIC), právě když pro libovolnou hodnotu h libovolné proměnné X existuje řešení problému vzniklého z okolí X, které [řešení] je konzistentní s h.

```
procedure NIC((V,E))
  Q ← V
  while Q not empty do
    V ← select and delete a variable from Q
    deleted ← false
    for each H in D_V do
      if no solution for Neighbourhood(X) compatible with H then
        remove H from D_V
        deleted ← true
        if D_V empty then return fail
    if deleted then Q ← Q ∪ Neighbourhood(X)
  return true
end NIC
```



Bodová konzistence

Můžeme libovolnou lokální konzistenční techniku dále posílit?

ANO! Zkusíme, zda pro každou hodnotu je zbylý problém konzistentní.

Definice:

CSP je bodově A-konzistentní (singleton A-consistency), kde A je libovolná konzistenční technika, právě když pro libovolnou hodnotu h libovolné proměnné X je problém omezený na X=h A-konzistentní.

Vlastnosti:

- **+ podobně jako NIC a RPC odstraňuje jen hodnoty z domény proměnných**
- **+ snadná implementace**
- **- může být časově náročnější (používat opatrně)**

1) bodová A-konzistence ≥ A-konzistence

2) A-konzistence ≥ B-konzistence ⇒
bodová A-konzistence ≥ bodová B-konzistence

3) bodová (i,j)-konzistence > (i,j+1)-konzistence (SAC>PIC)

4) silná (i+1,j)-konzistence > bodová (i,j)-konzistence (PC>SAC)

Přehled konzistenčních technik

- NC = 1-konzistence
- AC = 2-konzistence = (1,1)-konzistence
- PC = 3-konzistence = (2,1)-konzistence
- PIC = (1,2)-konzistence