Programování
s omezujícími podmínkami

8

Roman Barták
Katedra teoretické informatiky a matematické logiky

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

Ne-binární podmínky

Dosud jsme se zabývali především **binárními podmínkami**.

Mohli jsme si to dovolit, protože **každé CSP lze převést na binární CSP**!

Dělá se to tak ale v praxi?

- v aplikacích se často objevují právě nebinární podmínky
  např. $a+b+c \leq 5$
- pro takové podmínky můžeme dělat řadu lokálních inferencí (tj. odvození v rámci podmínky)
  např., pokud víme, že $a,b \geq 2$, můžeme odvodit $c \leq 1$
- v rámci jedné podmínky můžeme omezit doménu zvolené proměnné tak, aby její hodnoty vyhovovaly dané podmínce
  ↳ **zobecněná hranová konzistence**

Zobecněná hranová konzistence

- **Hodnota** x proměnné V je **zobecněně hranově konzistentní** (generalized arc consistent) vzhledem k podmínce P, právě když existují hodnoty v aktuálních doménách ostatních proměnných v P takové, že dohromady s x splňují P.

  Příklad: A+B≤C, A in {1,2,3}, B in {1,2,3}, C in {1,2,3}
  Hodnota 1 pro C není GAC (nemá podporu), 2 a 3 jsou GAC.

- **Proměnná** V je **zobecněně hranově konzistentní** (generalized arc consistent) vzhledem k podmínce P, právě když všechny hodnoty z aktuální domény D_V jsou GAC vzhledem k P.

  Příklad: A+B≤C, A in {1,2,3}, B in {1,2,3}, C in {2,3}
  C je GAC

- **Podmínka** je **zobecněně hranově konzistentní**, právě když jsou všechny její proměnné GAC.

  Příklad: pro A in {1,2}, B in {1,2}, C in {2,3} je A+B≤C GAC

- **Problém** je **zobecněně hranově konzistentní**, právě když jsou všechny jeho podmínky GAC.

Jak na GAC?

Upravíme AC-3 pro nebinární podmínky.

- Podmínku můžeme vidět jako sadu propagačních metod, z nichž každá udělá jednu proměnnou GAC:
  $A + B = C$: $A + B \rightarrow C$, $C - A \rightarrow B$, $C - B \rightarrow A$
- Provedením všech metod bude daná podmínka GAC.
- Opakujeme revize metod, dokud se mění domény.

```
procedure GAC-3(G)
    Q ← {Xs →Y | Xs →Y is a method for some constraint in G}
    while Q non empty do
        select and delete (As→B) from Q
        if REVISE(As→B) then
            if D_B=∅ then stop with fail
            Q ← Q ∪ {Xs →Y | Xs →Y is a method s.t. B∈Xs}
        end if
    end while
end GAC-3
```

Konzistence okrajů

Někdy je GAC příliš drahé (výpočtově náročné), například pokud jsou domény proměnných velké.

Poznámka:

Směrové GAC nepomůže, je-li úplná aplikace i jediné metody výpočtově náročná.

V takových případech se používá **slabší verze GAC**: GAC vlastnost je požadována pouze po okrajových hodnotách každé domény.

Definice:

Proměnná V je **okrajově konzistentní** (bounds consistent) vzhledem k podmínce P, právě když okraje domény D_V jsou GAC vzhledem k P.

Poznámky:

- předpokládáme uspořádání hodnot v doménách
- doména proměnné pak může být reprezentována jako interval, tj. pomocí dvou okrajů
- v praxi často využívaná technika, protože je hodně rychlá (ILOG Solver)

Globální podmínky

- A co když chceme dosáhnout GAC **rychleji než obecný GAC algoritmus**?
  - např. filtraci podmínky A < B lze udělat rychleji než zkoumáním kompatibilních dvojic (konzistence okrajů dá hranovou konzistenci).
- Je možné popsat **filtrační algoritmus pro podmínku**, kde může být **různá arita**?
  - např. podmínka all_different
- Využijeme **sémantiku podmínky** pro efektivní filtrační algoritmy, které pracují s libovolným počtem proměnných.

☞ globální podmínky ☜

Motivace: Sudoku

- Logická hádanka**, jejímž cílem je doplnit chybějící čísla 1-9 do tabulky 9×9 tak, aby se čísla neopakovala v žádném řádku, sloupci ani v malých čtvercích 3×3.**

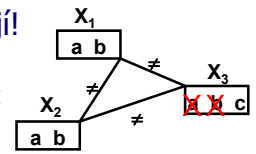
9	6	3	1	7	4	2	5	8
1	7	8	3	2	5	6	4	9
2	5	4	6	8	9	7	3	1
8	2	1	4	3	7	5	9	6
4	9	6	8	5	2	3	1	7
7	3	5	9	6	1	8	2	4
5	8	9	7	1	3	4	6	2
3	1	7	2	4	6	9	8	5
6	4	2	5	9	8	1	7	3

Jak takový problém formálně namodelujeme?

- proměnné **popisují políčka**
- podmínka **nerovnosti svazuje každou dvojici proměnných v řádku, sloupci nebo čtverci**

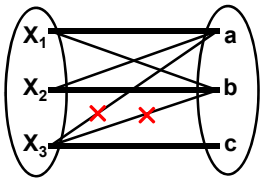
Takové podmínky ale „málo" propagují!

- příklad napravo je AC, ale
- přesto by ale šly nějaké hodnoty odstranit



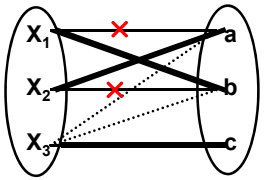
(Régin, AAAI 1994) Podmínka `all-different`

- modeluje množinu navzájem různých proměnných
- **`all_different`**$(\{X_1,\ldots, X_k\}) = \{(d_1,\ldots, d_k) \mid \forall i\ \ d_i \in D_i\ \ \&\ \forall i \neq j\ \ d_i \neq d_j\}$
- filtrace založena na hledání **párování v bipartitních grafech** (vrcholy = proměnné+hodnoty, hrany = domény)



Inicializace:

1) najdi maximální párování
2) odstraň všechny hrany, které neleží v žádném max. párování



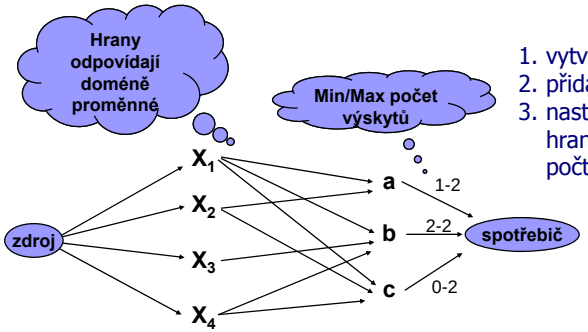
Inkrementální propagace ($X_1 \neq a$):

1) odstraň neplatné hrany
2) najdi nové maximální párování
3) odstraň všechny hrany, které neleží v žádném max. párování

Podmínka gcc

(Régin, AAAI 1996)

- Zobecnění podmínky all-different (**global cardinality** cons.)
  - pro každou hodnotu ze sjednocení domén je určen minimální a maximální počet výskytů
- Efektivní propagaci lze založit na **hledání toků v sítích**.



1. vytvoříme graf podmínky
2. přidáme zdroj a spotřebič
3. nastavíme dolní mez a kapacitu hran (u hran z „hodnot" podle počtu výskytů, ostatní 0-1)

Maximální tok odpovídá konzistentnímu ohodnocení proměnných! Najdeme hrany, které mají tok 0 v každém maximálním toku, a podle hran odfiltrujeme příslušné hodnoty z domén.

Motivace: bořič symetrií

Přítomnost **symetrických řešení** zhoršuje efektivitu řešení podmínek (prohledávají se symetrické prostory bez řešení).

Klasický příklad je problém **rozvrhování sportovních turnajů.**

- hraje n týmů
- hraje se systémem každý s každým, tj. (n-1) kol
- každý tým hraje v každém kole jako domácí nebo host

Jak takový problém formálně namodelujeme?

- Kolo i může být popsáno posloupností K_i **kódů zápasů**.
  - $K_{i,j}$ je kód j-tého zápasu kola i
- Zápasy v rámci kola je možné prohodit – **symetrie zápasů**.
  - symetrii zápasů odstraníme podmínkou $K_{i,j} < K_{i,j+1}$
- Kola pro různé týdny je možné prohodit – **symetrie týdnů**.
  - symetrii týdnů odstraníme podmínkou $K_i <_{lex} K_{i+1}$.

Podmínka lex

(Frisch et al., CP 2002)

- modeluje **lexikografické uspořádání dvou vektorů**
- **lex**$(\{X_1,\dots, X_n\}, \{Y_1,\dots, Y_n\}) \equiv (X_1 \le Y_1) \wedge (X_1 = Y_1 \Rightarrow X_2 \le Y_2) \wedge \dots$
  $\dots \wedge (X_1 = Y_1 \wedge \dots \wedge X_{n-1} = Y_{n-1} \Rightarrow X_n < Y_n)$
- **globální filtrace** používá dva ukazatele:
  α: proměnné před α jsou instanciované a mají stejnou hodnotu
  β: vektory začínající β jsou uspořádané „obráceně"
  floor$(\{X_\beta,\dots, X_n\}) >_{lex}$ ceiling$(\{Y_\beta,\dots, Y_n\})$

$X = \langle \{2\}, \{1,3,4\}, \{1,2,3,4,5\}, \{1,2\}, \{3,4,5\} \rangle$ nejprve inicializujeme ukazatele
$Y = \langle \{0,1,2\}, \{1\}, \{0,1,2,3,4\}, \{0,1\}, \{0,1,2\} \rangle$
$\alpha\uparrow$ $\uparrow\beta$

$X = \langle \{2\}, \{1,3,4\}, \{1,2,3,4,5\}, \{1,2\}, \{3,4,5\} \rangle$ upravíme Y_1, aby alespoň $X_1 = Y_1$
$Y = \langle \{2\}, \{1\}, \{0,1,2,3,4\}, \{0,1\}, \{0,1,2\} \rangle$ a posuneme ukazatel α
$\alpha\uparrow$ $\uparrow\beta$

$X = \langle \{2\}, \{1\}, \{1,2,3,4,5\}, \{1,2\}, \{3,4,5\} \rangle$ upravíme Y_2, aby alespoň $X_2 = Y_2$
$Y = \langle \{2\}, \{1\}, \{0,1,2,3,4\}, \{0,1\}, \{0,1,2\} \rangle$ a opět posuneme ukazatel α
$\alpha\uparrow$ $\uparrow\beta$

$X = \langle \{2\}, \{1\}, \{1,2,3\}, \{1,2\}, \{3,4,5\} \rangle$ protože $\alpha = \beta - 1$
$Y = \langle \{2\}, \{1\}, \{2,3,4\}, \{0,1\}, \{0,1,2\} \rangle$ vynutíme podmínku $X_\alpha < Y_\alpha$
$\alpha\uparrow$ $\uparrow\beta$

Motivace: rostering

Rostering

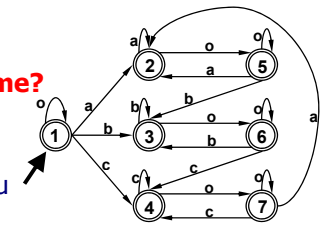
- **rozvrhování směn**, např. v nemocnicích či v dopravě
- speciální omezení na posloupnost směn přiřazenou téže osobě (daná např. zákoníkem práce, odbory ,…)

Příklad:

- směny: a, b, c, o (o je volno)
- podmínky:
  - stejná směna se může opakovat každý den
  - mezi a, b, mezi b, c a mezi c,a musí být alespoň jedno o
  - a-o*-c, b-o*-a, ani c-o*-b není povoleno (o* je libovolná sekvence o)
- První den může mít daná osoba libovolnou směnu, druhý den směny b, o, třetí den směny a, c, o, čtvrtý den směny a, b, o a pátý den směnu a.

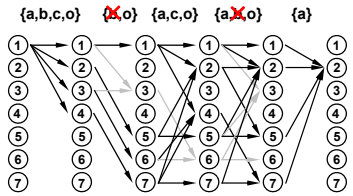
Jak takový problém formálně namodelujeme?

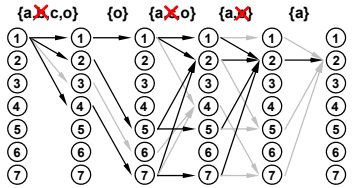
- proměnná určující směnu daný den
- a co podmínky?
  - například pomocí konečného automatu



(Pesant, CP 2004) Podmínka `regular`

- modeluje sekvenci znaků **přijímaných konečným automatem**
- **`regular`**(A, {X_1,..., X_k}) = {(d_1,..., d_k) | ∀i $d_i \in D_i \wedge d_1 \ldots d_k \in L(A)$}
- filtrace založena na „rozkladu" **výpočtu konečného automatu** pomocí vrstveného orientovaného grafu (vrstvy=stavy, hrany=přechody)



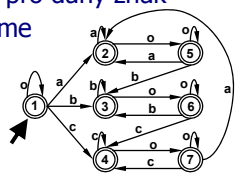


Inicializace

1. přidáme hrany vedoucí z počátku a odpovídající symbolům v doménách
2. při zpětném chodu odstraníme hrany, které nemají pokračování
3. odstraníme znaky bez hrany

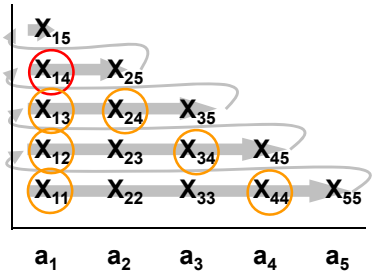
Inkrementální propagace ($X_4 \neq o$):

1. odstraníme hrany pro daný znak
2. změnu propagujeme oběma směry
3. odstraníme znaky bez hrany



(Quimper & Walsh, Sellmann, CP 2006) Podmínka `grammar`

- A co modelovat sekvenci znaků generovaných **bezkontextovou gramatikou**?
- **`grammar`**(G, {X_1,..., X_k}) = {(d_1,..., d_k) | ∀i $d_i \in D_i \wedge d_1 \ldots d_k \in L(G)$}
- filtraci založíme na **algoritmu CYK** pro Chomského NF ($A \rightarrow BC \mid x$)
  myšlenka: $X_{i,j} = \{A \mid A \Rightarrow^* a_i a_{i+1} \ldots a_j\}$
  začneme: $X_{i,i} = \{A \mid (A \rightarrow a_i) \in P\}$
  pokračujeme: $X_{i,j} = \{A \mid \exists k: i \le k < j \; B \in X_{i,k} \wedge C \in X_{k+1,j} \wedge (A \rightarrow BC) \in P\}$
  pokud $S \in X_{1,n}$, potom $a_1 a_2 \ldots a_n$ patří do jazyka



S → AB | BC
A → BA | a
B → CC | b
C → AB | a

{S,A,C}				
-	{S,A,C}			
-	{B}	{B}		
{S,A}	{B}	{S,C}	{S,A}	
{B}	{A,C}	{A,C}	{B}	{A,C}
b	a	a	b	a

(Walsh et al., 2006) Podmínka `slide`

- Vraťme se ještě k podmínce **`regular`**, která se chová jako když speciální **podmínka klouže po posloupnosti proměnných**.
- Tento princip můžeme zobecnit!
- **`slide`**$_j$(C, {X_1,..., X_n}) ≡ ∀i C(X_{ij+1},..., X_{ij+k})
  - C je k-ární podmínka
  - konstanta j určuje, o kolik políček skáčeme / kloužeme

Příklady:

- **`regular`**(A, {X_1,..., X_n}) ≡ **`slide`**$_2$(C, {Q_0,X_1,Q_1, ..., X_n, Q_n})
  C(P,X,Q) reprezentuje přechod δ(P,X) = Q, $Q_0 = \{q_0\}$, Q_n = F
- **`lex`**({X_1,..., X_n}, {Y_1,..., Y_n}) ≡ **`slide`**$_3$(C, {B_0,X_1,Y_1,B_1, ..., X_n, Y_n, B_n})
  C(B,X,Y,C) ≡ B=C=1 or (B=C=0 and X=Y) or (B=0, C=1 and X<Y)
  B_0 = 0, B_n = 1 (striktní lex), B_n in {0,1} (ne-striktní lex)
- **`stretch`**({X_1,..., X_n}, s, l , t) ≡ **`slide`**$_2$(C, {X_1,S_1, ..., X_n, S_n})
  $C(X_i, S_i, X_{i+1}, S_{i+1}) \equiv X_i = X_{i+1}, S_{i+1} = 1+S_i, S_{i+1} \le l(X_i)$,
  or $X_i \neq X_{i+1}, S_i \ge s(X_i), S_{i+1} = 1, (X_i, X_{i+1}) \in t$
  $S_1 = 1$

...

Motivace: rozvrhování

Rozvrhovací problém je o alokaci aktivit na zdroje a čas.

Uvažujme alokaci v čase na unární zdroj.

unární zdroj - v dané chvíli může zpracovávat jedinou aktivitu

každá aktivita má pevnou dobu trvání a přiřazené **časové okno**, ve kterém musí být zpracována

Jak takový problém formálně namodelujeme?

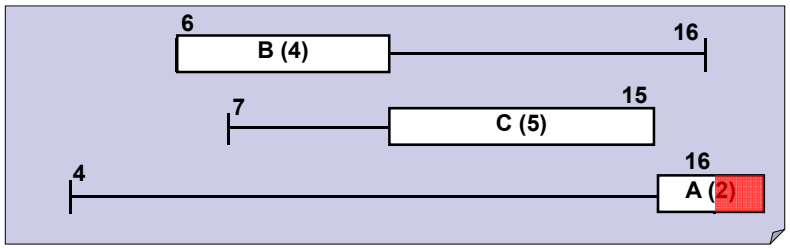
- **proměnné** popisují startovní čas aktivit **start(A)**
- **podmínky** zajišťují nepřekrývání se aktivit v čase
  start(A) + p(A) ≤ start(B) ∨ start(B) + p(B) ≤ start(A)
  (odtud také název **disjunktivní zdroj**)
- případně další relace typu **precedence** A « B
  start(A) + p(A) ≤ start(B)

Disjunkce ale tradičně „málo" propagují!

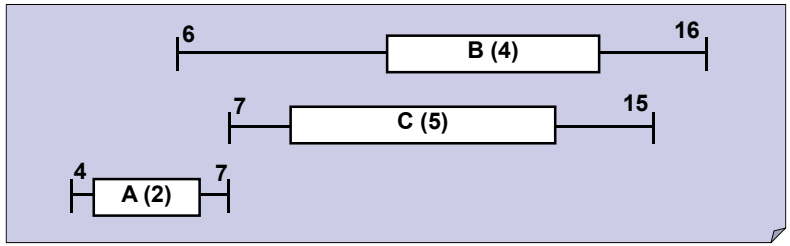
(Baptiste & Le Pape, 1996)

Podmínka edge_finding

Co když disjunktivní podmínka (A<<B ∨ B<<A) nezabere?



min(start(Ω)) + p(Ω) + p(A) > max(end(Ω ∪ {A})) ⇒ A<<Ω



(Baptiste & Le Pape, 1996)

Podmínka edge_finding

filtrační pravidla

Filtrační pravidla:

- **p(Ω ∪ {A}) > lct(Ω ∪ {A}) - est(Ω) ⇒ A«Ω**
- **p(Ω ∪ {A}) > lct(Ω) - est(Ω ∪ {A}) ⇒ Ω«A**
- **A«Ω ⇒ end(A) ≤ min{ lct(Ω') - p(Ω') | Ω'⊆Ω }**
- **Ω«A ⇒ start(A) ≥ max{ est(Ω') + p(Ω') | Ω'⊆Ω }**

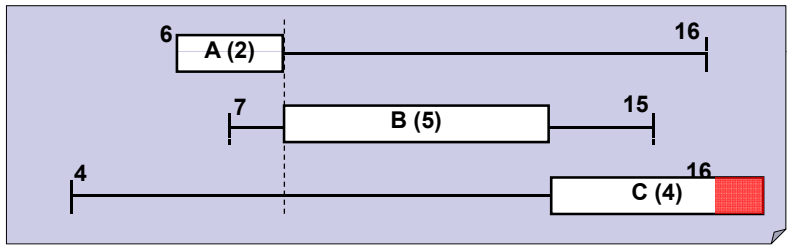
V praxi:

- musíme uvažovat n.2^n dvojic (A,Ω) (to je moc!)
- místo všech Ω uvažujme pouze **intervaly úloh** [X,Y]
  {C | est(X) ≤ est(C) ∧ lct(C) ≤ lct(Y)}
  - ↳ časová složitost O(n^3), často užívaný inkrementální algoritmus
- existují také algoritmy se složitostí O(n^2) a O(n.log n)

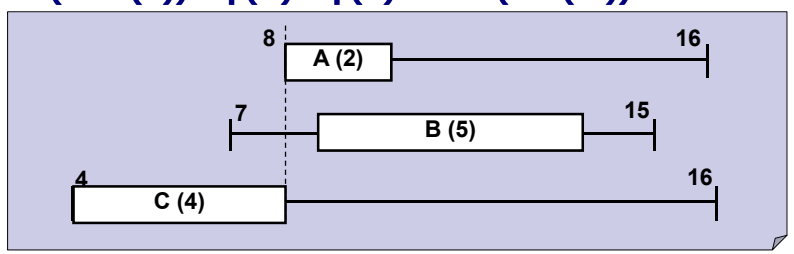
(Torres & Lopez, 2000)

Podmínka not_first

Co se stane, pokud A bude zpracována první?



min(start(A)) + p(Ω) + p(A) > max(end(Ω)) ⇒ ¬ A<<Ω



(Torres & Lopez, 2000)

Podmínka not_first

filtrační pravidla

Pravidla pro not_first:

p(Ω∪{A}) > lct(Ω) - est(A) ⇒ ¬ A«Ω
¬ A«Ω ⇒ start(A) ≥ min{ ect(B) | B∈Ω }

Symetrická pravidla pro not_last:

p(Ω∪{A}) > lct(A) - est(Ω) ⇒ ¬ Ω«A
¬ Ω«A ⇒ end(A) ≤ max{ lst(B) | B∈Ω }

V praxi:

- lze implementovat s časovou složitostí O(n^2) a prostorovou složitostí O(n)