Programování s omezujícími podmínkami

11

Roman Barták
Katedra teoretické informatiky a matematické logiky

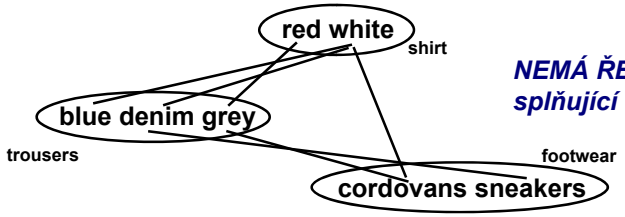
roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Motivace - problém šatníku

Vyberte oblečení tak, aby navzájem ladilo.

Proměnné:
- shirt: {red, white}
- footwear: {cordovans, sneakers}
- trousers: {blue, denim, grey}

Podmínky:
- shirt x trousers: red-grey, white-blue, white-denim
- footwear x trousers: sneakers-denim, cordovans-grey
- shirt x footwear: white-cordovans



NEMÁ ŘEŠENÍ splňující všechny podmínky

Problémy, kde nelze najednou splnit všechny podmínky, se nazývají **příliš omezené** (pře-omezené, over-constrained).

První řešení problému šatníku

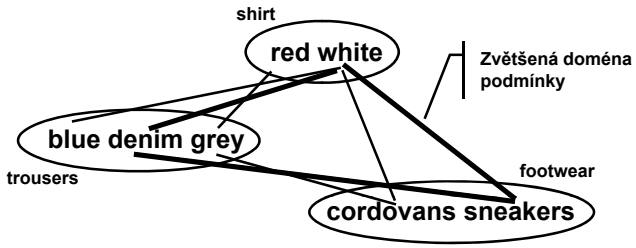
Problém sice nemá řešení, ale mi nějaké potřebujeme.

1) dokoupíme oblečení
   zvětšíme doménu proměnné
2) slevíme z vkusu
   zvětšíme doménu podmínky
3) boty a neladící košile nevadí
   vyřadíme podmínku
4) nevezmeme si boty
   vyřadíme proměnnou

Doména je určena unární podmínkou

Doména podmínky je úplná

Vyřadíme podmínky s proměnnou



Částečné splňování podmínek

- Definujme nejprve prostor problémů jako částečně uspořádanou množinu CSP problémů (PS,≤), kde $P_1 \leq P_2$ právě když množina řešení P_2 je podmnožinou množiny řešení P_1.
- Prostor problémů získáme oslabováním původního problému.
- **Problém částečného splňování podmínek** (Partial Constraint Satisfaction Problem, PCSP) je definován jako ⟨P,(PS,≤),M,(N,S)⟩
  - P je původní problém
  - (PS,≤) je prostor problémů obsahující P
  - M je metrika na tomto prostoru definující vzdálenost problémů
    - M(P,P') může být například počet různých řešení P a P'
    - nebo počet různých n-tic v podmínkách
  - N je maximální možná vzdálenost
  - S je postačující vzdálenost (S<N)
- Řešením PCSP je problém P' (a jeho řešení), takový že P'∈PS a M(P,P')<N. Postačujícím řešením je řešení, kde M(P,P') ≤ S. Optimálním řešením je řešení s minimální vzdáleností od P.

Částečné splňování podmínek
v praxi

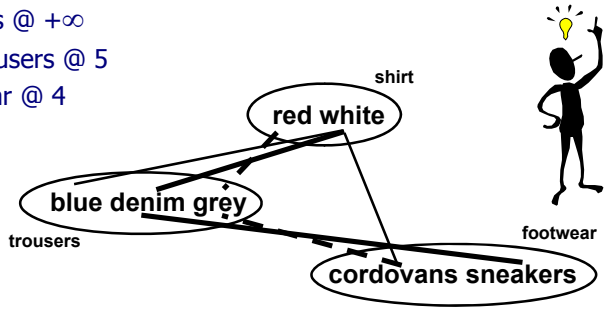
- Při hledání řešení PCSP negenerujeme další problémy, ale **pracujeme s původním problémem**:
  - používá se evaluační funkce g, která každému (i částečnému) ohodnocení proměnných přiřazuje numerickou hodnotu
  - hledáme ohodnocení minimalizující/maximalizující funkci g
- **PCSP je zobecněním CSOP**:
  - g(x) = f(x), je-li ohodnocení x řešením CSP
  - g(x) = ∞, jinak
- **PCSP se používá pro řešení:**
  - **příliš omezených problémů**
  - **příliš složitých problémů**, jejichž úplné řešení je časově náročné
  - **s daným množstvím prostředků** (například času)
  - **v reálném čase** (anytime algoritmy)
- Pro řešení PSCP je potřeba upravit propagační algoritmy nebo lze použít lokální prohledávání.

Druhé řešení problému šatníku

Každé podmínce můžeme přiřadit **váhu** popisující důležitost splnění podmínky.

Cílem je **minimalizovat součet vah** nesplněných podmínek.

- shirt x trousers @ +∞
- footwear x trousers @ 5
- shirt x footwear @ 4



Vzniklý problém se nazývá **Vážené (Weighted) CSP**.
Použitý princip může být zobecněn na tzv. **Oceněné (Valued) CSP**.

Oceněné splňování podmínek
(Valued CSP)

- Základní myšlenka:
  - každé podmínce je přiřazeno jisté **ocenění**.
  - ocenění nesplněných podmínek jsou agregována
  - za řešení je vybráno ohodnocení dávající **nejmenší agregované ocenění**
- **Struktura ocenění** je (E,⊗,>,⊥,T), kde:
  - E je množina ocenění úplně uspořádaná pomocí > s minimálním prvkem ⊥ a maximálním prvkem T
  - ⊗ je komutativní a asociativní binární operace na E s jednotkovým prvkem ⊥ (⊥⊗a=a) a absorpčním prvkem T (T⊗a=T), která zachovává monotonii (a ≥ b ⇒ a⊗c ≥ b⊗c)
- Podmínky C jsou mapovány na ocenění E pomocí funkce φ: C→E.
- **Řešením problému** je takové ohodnocení proměnných, které má nejmenší agregované ocenění v(A), kde:

$$v(A) = \bigotimes_{\substack{c \in C \\ \text{A violates c}}} \varphi(c)$$

Valued CSP

- **Struktura ocenění** je (E,⊗,>,⊥,T), kde:
  - E je množina ocenění úplně uspořádaná pomocí > s minimálním prvkem ⊥ a maximálním prvkem T
  - ⊗ je komutativní a asociativní binární operace na E s jednotkovým prvkem ⊥ (⊥⊗a = a) a absorpčním prvkem T (T⊗a = T), která zachovává monotonii (a ≥ b ⇒ a⊗c ≥ b⊗c)
- Podmínky C jsou mapovány na ocenění E pomocí funkce φ: C→E.
- **Řešením problému** je takové ohodnocení proměnných, které má nejmenší agregované ocenění v(A), kde:

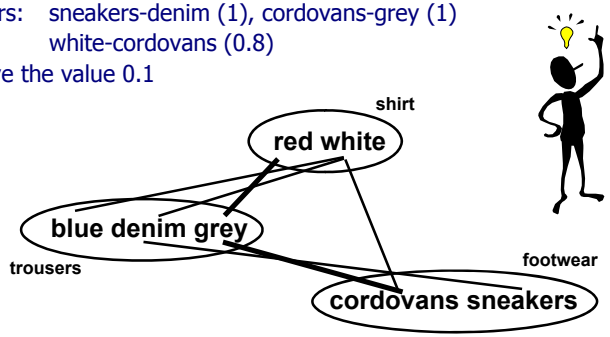
$$v(A) = \bigotimes_{\substack{c \in C \\ \text{A violates c}}} \varphi(c)$$

Framework	E	⊗	>	⊥	T
Classical CSP	{true,false}	∧	>	true	false
Weighted CSP	$\mathbb{N} \cup \{+\infty\}$	+	>	0	+∞
Probabilistic CSP	⟨0,1⟩	×	<	1	0
Possibilistic CSP	⟨0,1⟩	max	>	0	1
Lexicographic CSP	$\mathbb{N}^{\langle 0,1\rangle} \cup \{T\}$	∪	$>_{lex}$	∅	T

Třetí řešení problému šatníku

- Každé dvojici (n-tici) hodnot v podmínce můžeme přiřadit **preferenci** popisující, jak dobře dané hodnoty podmínku splňují.
- Cílem je **maximalizovat součin preferencí** získaných projekcí daného ohodnocení na všechny podmínky.

shirt x trousers: red-grey (1), white-blue (1), white-denim (0.9)
footwear x trousers: sneakers-denim (1), cordovans-grey (1)
shirt x footwear: white-cordovans (0.8)
all other pairs have the value 0.1



Vzniklý problém se nazývá **Pravděpodobnostní (Probabilistic) CSP** zobecněním získáme **CSP nad polo-okruhy** (**semiring-based CSP**).

Podmínky nad polo-okruhy

- Základní myšlenka:
  - každé n-tici hodnot proměnných v podmínce je přiřazena **preference** vyjadřující, jak dobře daná n-tice splňuje podmínku
  - agregujeme preference pro projekci ohodnocení na proměnné všech podmínek
  - za řešení je vybráno takové ohodnocení dávající **největší agregovanou preferenci**
- **Struktura C-polo-okruhu** je $(A,+,\times,0,1)$, kde
  - A je množina preferencí,
  - $+$ je komutativní, asociativní a idempotentní (a+a=a) binární operace na A s jednotkovým prvkem 0 (0+a=a) a absorpčním prvkem 1 (1+a=1)
    tato operace zároveň definuje uspořádání $a \le b \Leftrightarrow a+b=b$.
  - $\times$ je komutativní, asociativní binární operace na A s jednotkovým prvkem 1 ($1\times a=a$) a absorpčním prvkem 0 ($0\times a=0$), která je distributivní nad +.
- **Řešením** je takové ohodnocení proměnných V, které dává největší agregovanou preferenci p(V):

$$p(V) = \underset{c \in C}{\times} \delta_c\left(V \downarrow vars(c)\right)$$

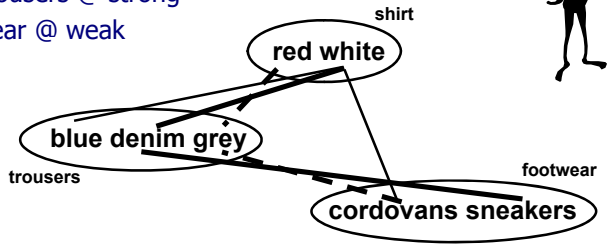
Semiring-based CSP

- **Struktura C-polo-okruhu** je $(A,+,\times,0,1)$, kde
  - A je množina preferencí,
  - + je komutativní, asociativní a idempotentní (a+a=a) binární operace na A s jednotkovým prvkem 0 (0+a=a) a absorpčním prvkem 1 (1+a=1)
    tato operace zároveň definuje uspořádání $a \le b \Leftrightarrow a+b=b$.
  - $\times$ je komutativní, asociativní binární operace na A s jednotkovým prvkem 1 ($1\times a=a$) a absorpčním prvkem 0 ($0\times a=0$), která je distributivní nad +.
- N-tice v podmínce C jsou mapovány na polo-okruh funkcí δ_C: $V \to A$.
- **Řešením** je takové ohodnocení proměnných V, které dává největší agregovanou preferenci p(V): $p(V) = \underset{c \in C}{\times} \delta_c\left(V \downarrow vars(c)\right)$

Framework	A	+	×	1	0
Classical CSP	{false,true}	∨	∧	true	false
Weighted CSP	$N \cup \{+\infty\}$	min	+	0	$+\infty$
Probabilistic CSP	⟨0,1⟩	max	×	1	0
Possibilistic CSP	⟨0,1⟩	min	max	0	1
Fuzzy CSP	⟨0,1⟩	max	min	1	0
Lexicographic CSP	$N^{(0,1)} \cup \{T\}$	$\max_{lex}$	∪	∅	T

Čtvrté řešení problému šatníku

- Podmínkám můžeme přiřadit **preferenci** určující, jaké podmínky mají být přednostně splněny.
- Preference určují striktní přednost
  silnější podmínka je vždy preferována na úkor slabších
  - shirt x trousers @ required
  - footwear x trousers @ strong
  - shirt x footwear @ weak



Podmínky s preferencemi tvoří hierarchii, hovoříme proto o **hierarchii omezujících podmínek**.

Hierarchie podmínek

- Každé podmínce přiřadíme **preferenci** (množina preferencí je lineárně uspořádaná)
  - význačná je preference required, podmínka s touto preferencí musí být splněna (nutná podmínka, **hard constraint**)
  - ostatní podmínky jsou preferenční a nemusí být splněny (**soft constraints**)
- **Hierarchie podmínek** H je konečná (multi) množina podmínek.
  - H_0 je množina nutných podmínek (s odstraněnou preferencí)
  - H_1 je množina nejvíce preferovaných podmínek
  - ...
- **Řešením hierarchie** je ohodnocení proměnných, které splňuje nutné podmínky a nejlépe splňuje preferenční podmínky.
  - $S_{H,0} = \{\sigma \mid \forall c \in H_0 , c\sigma \text{ platí}\}$
  - $S_H = \{\sigma \mid \sigma \in S_{H,0} \wedge \forall \omega \in S_{H,0} \neg \text{better}(\omega,\sigma,H) \}$

Komparátory

- **Porovnání ohodnocení proměnných vzhledem k dané hierarchii.**
  - anti-reflexivní, transitivní relace, která **respektuje hierarchii**
  - pokud nějaké ohodnocení splňuje všechny podmínky až do úrovně k, potom totéž splňují všechna lepší ohodnocení
- **Chybová funkce** $e(c,\sigma)$ - jak dobře ohodnocení splňuje podmínku
  - predikátová chybová funkce (splňuje/nesplňuje)
  - metrická chybová funkce - vzdálenost od řešení, $e(X \geq 5,\{X/3\}) = 2$
- **Lokální komparátory**
  - porovnávají chybu zvlášť u každé podmínky

  $\text{locally\_better}(\omega,\sigma,H) \equiv \exists k>0$
  $\forall i<k \ \forall c \in H_i \ e(c,\omega)=e(c,\sigma) \wedge \forall c \in H_k \ e(c,\omega) \leq e(c,\sigma) \wedge \exists c \in H_k \ e(c,\omega)<e(c,\sigma)$
- **Globální komparátory**
  - agregují chybu pro celou úroveň pomocí funkce g

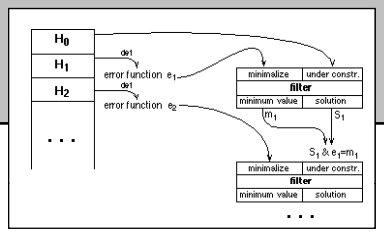
  $\text{globally\_better}(\omega,\sigma,H) \equiv \exists k>0 \ \forall i<k \ \ g(H_i,\omega)=g(H_i,\sigma) \wedge g(H_k,\omega)<g(H_k,\sigma)$
  - používají se vážené součty, součty čtverců, nejhorší případ ...

DeltaStar

- Řešení podmínek po úrovních postupným zjemňováním množiny řešení.
- Máme řešič podmínek s funkcí filter: P(řešení) × podmínky → P(řešení)
  - z dosud nalezené množiny řešení vybere řešení splňující co nejlépe podmínky (realizuje tak komparátor)
  - řešení může být předáváno v implicitním tvaru

Algoritmus DeltaStar

```
procedure DeltaStar(H: constraint hierarchy)
    i ← 1
    Solution ← solution of required constraints from H
    while not unique Solution and i<number of levels do
        Solution ← filter(Solution,Hi)
        i++
    endwhile
    return Solution
end DeltaStar
```



- filtr může být realizován simplexem
- podmínky dalších úrovní jsou zachyceny v chybové funkci

Lokální propagace

Hledání řešení postupným splňováním podmínek.

- podmínka je popsána **metodami** pro výpočet hodnot jejích proměnných

  A+B = C A ← C-B, B ← C-A, C ← A+B,
- každé proměnné přiřadíme metodu některé podmínky, která určí její hodnotu
- tato hodnota se použije jako vstup do další podmínky

výhody:

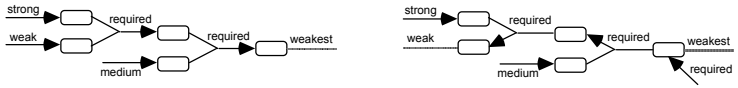
- přes síť podmínek můžeme propagovat změnu hodnoty
- metody lze dopředu zkompilovat

omezení:

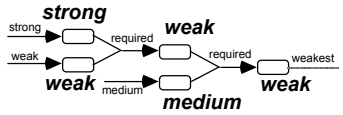
- funguje pouze pro funkcionální podmínky (rovnosti)
- v grafu metod nesmí být cyklus
- najde pouze jedno řešení
- funguje pouze s variantami lokálního predikátového komparátoru

Základy DeltaBlue

- Pracuje s **metodami** majícími **jediný výstup**.
  - nejprve vybere pro každou podmínku metodu (plánování)
  - potom přes zvolenou síť propaguje hodnoty (exekuce)
- **Inkrementální** plánování - po přidání podmínky upraví síť metod



- Abychom věděli, kterým směrem síť metod upravovat, používá algoritmus tzv. průchozí preference (**walkabout strength**).
- **Průchozí preference** proměnné je slabší z preference podmínky, jejíž metoda určuje hodnotu proměnné, a z průchozích preferencí proměnných, které jsou výstupem zbylých metod této podmínky.

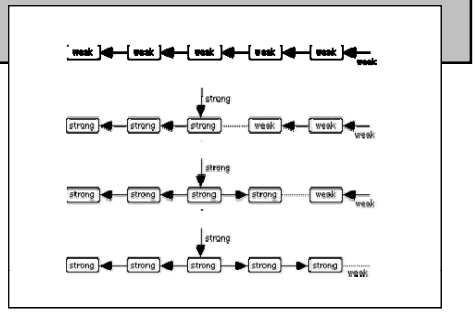


DeltaBlue

Algoritmus DeltaBlue

```
procedure AddConstraint(c: constraint)
    select the potential output variable V of c with the weakest walkabout
    strength
    if walkabout strength of V is weaker than strength of c then
        c'← the constraint currently determing the variable V
        make c' unsatisfied
        select the method determing V in c
        recompute walkabout strengths of downstream variables
        AddConstraint(c')
    endif
end AddConstraint
```

- přidáním podmínky se síť metod lokálně upraví
- přepočtou se průchozí preference
- vyřazená podmínka se zkusí znova přidat



Indigo

- **DeltaBlue** umí pracovat pouze s funkcionálními podmínkami, kde metody mají jediný výstup.
- **SkyBlue** zobecňuje DeltaBlue o metody s více výstupy.
- Oba algoritmy vyžadují existenci acyklické sítě metod a neumí nefunkcionální podmínky, tj. třeba A<B.
- Algoritmus **Indigo** byl navržen pro acyklické hierarchie s nefunkcionálními podmínkami a s lokálním metrickým komparátorem.
  - **propaguje okraje domén** (dělá problém okrajově konzistentní)
    - vždy spustí všechny metody pro podmínku
  - postupně přidává podmínky od nejsilnější po nejslabší
    - přidáním podmínky se omezí okraje některých domén
    - změna okrajů domén se propaguje přes další podmínky

Indigo v příkladě

c1: required a>=10
c2: required b>=20
c3: required a+b=c
c4: required c+25=d
c5: strong d<=100
c6: medium a=50
c7: weak a=5
c8: weak b=5
c9: weak c=100
c10: weak d=200

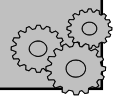
action	a	b	c	d	note
	(-inf,inf)	(-inf,inf)	(-inf,inf)	(-inf,inf)	initial bounds
add c1	[10,inf)	(-inf,inf)	(-inf,inf)	(-inf,inf)	
add c2	[10,inf)	[20,inf)	(-inf,inf)	(-inf,inf)	
add c3	[10,inf)	[20,inf)	[30,inf)	(-inf,inf)	
add c4	[10,inf)	[20,inf)	[30,inf)	[55,inf)	
add c5	[10,inf)	[20,inf)	[30,inf)	[55,100]	
	[10,inf)	[20,inf)	[30,75]	[55,100]	propagate bounds using c4
	[10,55]	[20,65]	[30,75]	[55,100]	propagate bounds using c3
add c6	[50,50]	[20,65]	[30,75]	[55,100]	
	[50,50]	[20,25]	[70,75]	[55,100]	propagate bounds using c3
	[50,50]	[20,25]	[70,75]	[95,100]	propagate bounds using c4
add c7	[50,50]	[20,25]	[70,75]	[95,100]	c7 is unsatisfied
add c8	[50,50]	[20,20]	[70,75]	[95,100]	c8 is unsatisfied but its error is minimized
	[50,50]	[20,20]	[70,70]	[95,100]	propagate bounds using c3
	[50,50]	[20,20]	[70,70]	[95,95]	propagate bounds using c4
add c9	[50,50]	[20,20]	[70,70]	[95,95]	c9 is unsatisfied
add c10	[50,50]	[20,20]	[70,70]	[95,95]	c10 is unsatisfied

Projekční algoritmus

Řešení lineárních rovností a nerovností Gaussovou a Fourierovou eliminací.

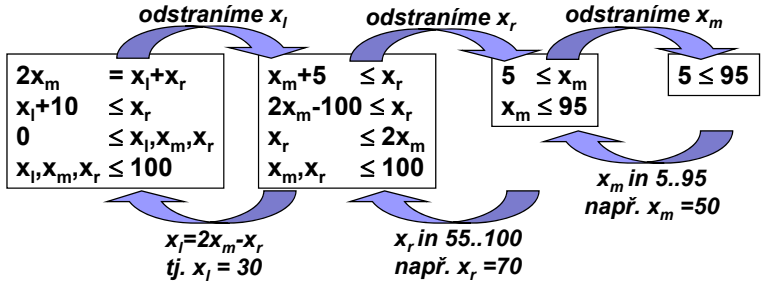
- C(0,x) - podmínky neobsahující x
- C(=,x) - rovnosti obsahující x
- C(+,x) - nerovnosti, které lze převést na tvar x≤e
- C(-,x) - nerovnosti, které lze převést na tvar e≤x

```
procedure project(C: set of constraints, x: variable)
    if ∃ c ∈ C(=,x) where c is x=e then
        D ← C-{c} with every occurence of x replaced by e
    else
        D ← C(0,x)
        for each c in C(+,x) where c is x ≤ e+ do
            for each c in C(-,x) where c is e- ≤ x do
                D ← D ∪ {e- ≤ e+}
            endfor
        endfor
    endif
    return D
end project
```



Projekční algoritmus v praxi

- Projekcemi postupně **odstraníme všechny proměnné** a potom zpětně dopočteme hodnoty proměnných.



A co hierarchické podmínky?

- pro lokální metrický komparátor
  - podmínky e?b @ pref převedeme na e?v_e @ required, v_e=b @ pref (v_e je nová proměnná, ? je relace =,≤,≥)
  - proměnné v nutných podmínkách eliminujeme od nejslabších
  - bereme hodnotu nejbližší konstantě b z nejsilnější podmínky