Programování s omezujícími podmínkami

11

Roman Barták, KTIML

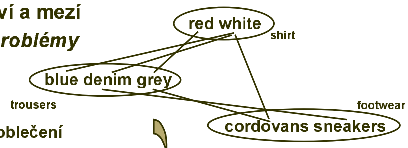
bartak@ktiml.mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

Co bylo minule

Lokální prohledávání
- Tabu Search, GSAT, Localizer

Optimalizační techniky
- metoda větví a mezí

Příliš omezené problémy



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

Částečné splňování podmínek

Definujme nejprve **prostor problémů** jako částečně uspořádanou množinu CSP problémů (PS,≤), kde $P_1 \le P_2$ 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 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)

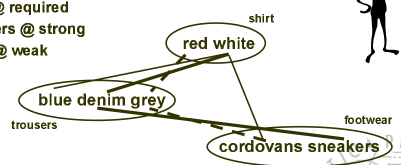
Pro řešení PSCP je potřeba upravit propagační algoritmy nebo lze použít lokální prohledávání.

Druhé řešení problému šatníku

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 omezujících 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.

$S_{H,0} = \{\sigma \mid \forall c \in H_0 ,\ c\sigma \text{ platí}\}$

$S_H = \{\sigma \mid \sigma \in S_{H,0} \ \&\ \forall \omega \in S_{H,0}\ \neg\ \text{better}(\omega,\sigma,H)\}$

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, $e(X>5,\{X/3\}) = 2$

Lokální komparátory

porovnávají chybu zvlášť u každé podmínky

$\text{locally\_better}(\omega,\sigma,H) \equiv \exists k>0$
$\forall i<k\ \forall c\in H_i\ e(c,\omega)=e(c,\sigma)\ \&\ \forall c\in H_k\ e(c,\omega) \le e(c,\sigma)\ \&\ \exists c\in H_k\ e(c,\omega)<e(c,\sigma)$

Globální komparátory

agregují chybu pro celou úroveň pomocí funkce g

$\text{globally\_better}(\omega,\sigma,H) \equiv \exists k>0\ \forall i<k\ \ g(H_i,\omega)=g(H_i,\sigma)\ \&\ g(H_k,\omega)<g(H_k,\sigma)$

používají se vážené součty, součty čtverců, nejhorší případ ...

Existence řešení hierarchie podmínek

Když se nám podaří splnit nutné podmínky (množina $S_{H,0}$ je neprázdná), existuje vždy řešení hierarchie (množina S_H je neprázdná)?

NE!

Příklad:

required X>0, strong X=0, kde $X\in R$ (reálná čísla), používáme metrický komparátor

$S_{H,0} = \{\{ X/d \} \mid d\in R\ \&\ d>0\}$ … neprázdná množina

$S_H = \emptyset$ protože *better({X/(d/2)},{X/d})*

Co tam vadí?
- $S_{H,0}$ je nekonečná
- doména R je nekonečná
- používáme metrický komparátor

Věty o existenci řešení

Tvrzení 1: Je-li $S_{H,0}$ neprázdná a konečná, je S_H neprázdná.

Důkaz: sporem - předpokládejme, že S_H je prázdná

vezměme $\sigma_1\in S_{H,0}$ (množina je neprázdná)

protože $\sigma_1\notin S_H$, existuje $\sigma_2\in S_{H,0}$ tak, že better(σ_2,σ_1)

takto sestrojíme nekonečný řetězec σ_1, σ_2 ,…

ohodnocení σ_i a σ_j ($i\neq j$) jsou navzájem různá (transitivita a anti-reflexivita)

množina $S_{H,0}$ obsahuje všechna taková σ_i - SPOR (množina je konečná)

Tvrzení 2: Je-li $S_{H,0}$ neprázdná a používám predikátový komparátor, je S_H neprázdná.

Důkaz: sporem - předpokládejme, že S_H je prázdná

podobně jako v 1 najdeme nekonečný řetězec σ_1, σ_2 ,…

σ_i a σ_j ($i\neq j$) splňují různé množiny podmínek (predikátový komparátor)

tedy máme nekonečně mnoho podmnožin dané hierarchie - SPOR

Tvrzení 3: Je-li $S_{H,0}$ neprázdná a doména proměnných je konečná, je S_H neprázdná.

Důkaz: Konečná doména znamená konečně mnoho ohodnocení proměnných. Tedy $S_{H,0}$ je konečná a dle 1, je S_H neprázdná.

Monotonie a inkrementalita

Klasické CSP je monotónní, tj. přidáním podmínek se nezvětší množina řešení.

Můžeme používat inkrementální algoritmy, které po přidání podmínky pouze upraví předchozí řešení.

Platí totéž i o hierarchiích?

NE!

Definice: Komparátor je monotónní, pokud pro libovolné hierarchie H a J platí $S_{H\cup J} \subseteq S_H$.

Tvrzení: Pokud pracujeme s netriviální doménou (alespoň dvouprvková), potom žádný komparátor, který respektuje hierarchii, není monotónní.

Důkaz:

vezměme H={weak X=a} a J={strong X=b}

S_H = {X/a} zatímco $S_{H\cup J}$ = {X/b}

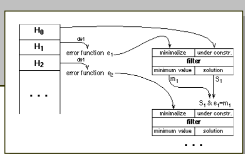
DeltaStar

Řešení podmínek po úrovních postupným zjemňováním množiny řešení.

Máme řešič podmínek s funkcí filter: řešení $\times$ podmínky $\rightarrow$ řešení
- z dosud nalezeného řešení vybere řešení splňující co nejlépe podmínky (realizuje tak komparátor)
- řešení může být předáváno v explicitním 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,H_i)
        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

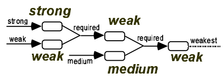
Pracuje s metodami majícími **jediný výstup**.

- nejprve vybere pro každou podmínku metodu (plánování)
- potom přes zvolenou sítě 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).

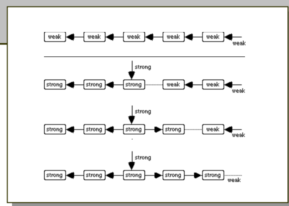
Průchozí preference proměnné je slabší z preference podmínky, jejíž metoda určuje hodnotu proměnné, a z průchozích preferencí proměnných, které jsou výstupem zbylých metod této podmínky.



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

Projekční algoritmus

Řešení lineárních rovností a nerovností Gaussovou a Fourierovou eliminací.

C(0,x) - podmínky neobsahující x

C(=,x) - rovnosti obsahující x

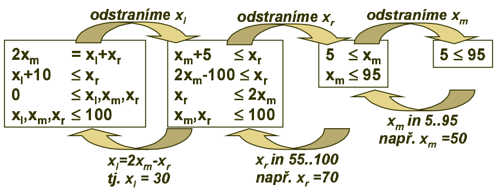
C(+,x) - nerovnosti, které lze převést na tvar $x \le e$

C(-,x) - nerovnosti, které lze převést na tvar $e \le x$

```
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)
        foreach c in C(+,x) where c is x ≤ e+ do
            foreach 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