Programování 2
omezujícími podmínkami

Roman Barták, KTIML

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

Problém splňování podmínek

CSP (Constraint Satisfaction Problem)
- konečná množina proměnných
- každé proměnné je přiřazena konečná množina hodnot = doména
- konečná množina podmínek omezujících hodnoty, které mohou proměnné současně nabývat

Částečné/úplné ohodnocení proměnných
- některé/každá proměnná má přiřazenu hodnotu

Řešení CSP
- úplné ohodnocení proměnných, které splňuje všechny podmínky

Úkolem je najít:
- jedno nebo všechna řešení
- optimální řešení (vzhledem k objektivní funkci) COSP

Reprezentace CSP

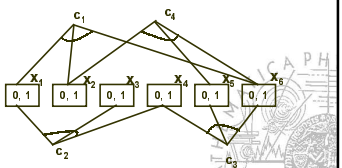
Reprezentace podmínek:
- intenzionální (matematická/logická formule)
- extenzionálně (výčet k-tic kompatibilních hodnot, 0-1 matice)

Reprezentace CSP problému pomocí (hyper)grafu
- vrcholy = proměnné
- (hyper)hrany = podmínky

Příklad:

proměnné $x_1,\ldots,x_6$ s doménou {0,1}

c_1: $x_1+x_2+x_6=1$
c_2: $x_1-x_3+x_4=1$
c_3: $x_4+x_5-x_6>0$
c_4: $x_2+x_5-x_6=0$



Binární podmínky

Svět není binární ...
ale lze na binární tvar transformovat!

Binární CSP
CSP + všechny podmínky jsou binární

Poznámka: unární podmínky jsou nezajímavé, lze je kódovat v doméně proměnné

Ekvivalence CSP
Dva CSP problémy jsou ekvivalentní, pokud mají stejnou množinu řešení.

Rozšířená ekvivalence
Řešení problémů lze mezi sebou vzájemně „syntakticky“ převést.

Lze každý CSP problém transformovat na (rozšířeně) ekvivalentní binární CSP?

Duální kódování

Prohoďme proměnné a podmínky.

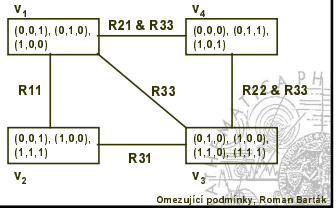
k- ární podmínku c převedeme na duální proměnnou v_c s doménou obsahující konzistentní k-tice

pro každou dvojici podmínek c a c' sdílejících proměnné zavedeme binární podmínku mezi v_c a $v_{c'}$ omezující duální proměnné na k-tice, ve kterých mají sdílené proměnné stejnou hodnotu

Příklad:

proměnné $x_1,\ldots,x_6$ s doménou {0,1}

c_1: $x_1+x_2+x_6=1$
c_2: $x_1-x_3+x_4=1$
c_3: $x_4+x_5-x_6>0$
c_4: $x_2+x_5-x_6=0$



Kódování se skrytou proměnnou

Pro (nebinární) podmínky zaveďme nové proměnné.

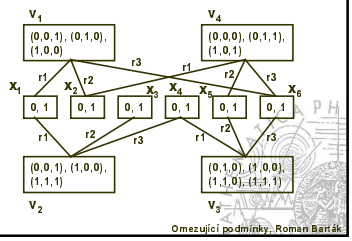
k- ární podmínku c převedeme na duální proměnnou v_c s doménou obsahující konzistentní k-tice

pro každou proměnnou x v podmínce c zavedeme podmínku mezi x a v_c omezující k-tice duální proměnné na kompatibilní s x

Příklad:

proměnné $x_1,\ldots,x_6$ s doménou {0,1}

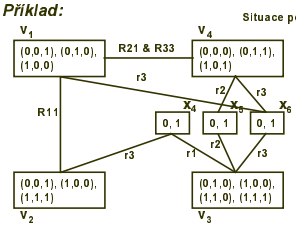
c_1: $x_1+x_2+x_6=1$
c_2: $x_1-x_3+x_4=1$
c_3: $x_4+x_5-x_6>0$
c_4: $x_2+x_5-x_6=0$



Převody mezi kódováními

Kódování se skrytou proměnnou lze převést na duální kódování
- cesty délky 2 mezi každou dvojicí duálních proměnných nahraď binární podmínkou, která je spojením relací na smazaných cestách (Př. r1 a r1 přejde na R11)
  pozor na hrany sdílené více cestami
- pokud se původní proměnná stane izolovanou nebo je spojena jen s jednou duální proměnnou, vyřaď ji

Příklad:



V každém kroku transformace máme ekvivalentní binární CSP
⇒ „hybridní" kódování

Transformaci lze provést i v obráceném směru.

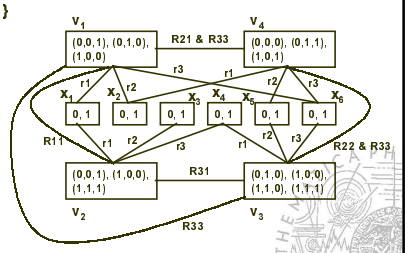
Zdvojené kódování

Kódování se skrytou proměnnou obohatíme o podmínky duálního kódování.

Příklad:

proměnné $x_1,\ldots,x_6$ s doménou {0,1}

c_1: $x_1+x_2+x_6=1$
c_2: $x_1-x_3+x_4=1$
c_3: $x_4+x_5-x_6>0$
c_4: $x_2+x_5-x_6=0$



Poznámky k binarizaci

Proč binarizujeme?
- unifikovaný tvar CSP problému
- řada řešicích algoritmů navržena pro binární CSP
- tradice

Jaký způsob binarizace je lepší?
- těžko říct
- duální kódování:
  lepší propagace vs. extenzionální reprezentace podmínky
- kódování se skrytou proměnnou:
  zachovává původní proměnné vs. oslabení propagace

Binární vs. nebinární podmínky
- složitější propagační algoritmy pro nebinární podmínky
- využití sémantiky nebinárních podmínek pro lepší propagaci

Splňování podmínek prohledáváním

podmínky jsou užívány pasivně jako test
přiřazuji hodnoty proměnným a ...
... zkouším, co to udělá

systematické prohledávání
systematicky prochází prostor všech ohodnocení
GT, BT, BJ, BM, DB, IB, LDS

nesystematické prohledávání
prochází prostor všech ohodnocení, části lze přeskočit
Credit Search, Bounded Backtrack

lokální prohledávání
prochází prostor všech ohodnocení v krátkých krocích
HC, MC, RW, Tabu, GSAT, Genet, simulované žíhání

Systematické prohledávání

systematicky prochází prostor všech ohodnocení
systematicky = nic nevynechává

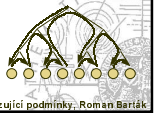
Vlastnosti:
+ úplné (pokud řešení existuje, najde ho)
- může to trvat pěkně dlouho

Klasifikace:

procházení úplných ohodnocení

generuj a testuj
spíše u lokálního prohledávání (není systematické)

rozšiřování částečného ohodnocení



stromové prohledávání

Generuj a testuj (GT)

Asi nejobecnější metoda řešení problémů
1) vygeneruj kandidáta na řešení
2) ověř, zda se skutečně jedná o řešení

Použití na řešení CSP
1) vyber hodnoty pro všechny proměnné
2) ověř, zda platí podmínky

Postupně prochází úplná, ale nekonzistentní ohodnocení, dokud nenajde (úplné) konzistentní ohodnocení.

Algoritmus GT(X:proměnné, C:podmínky)
V ← vyber první úplné ohodnocení proměnných X
while V nesplňuje všechny podmínky C do
  V ← systematicky vyber další ohodnocení proměnných X po V
end while
return V

Nedostatky a vylepšení metody GT

Generuje zbytečně mnoho ohodnocení, o kterých *je jasné*, že nebudou řešením.

Příklad:

X::{1,2}, Y::{1,2}, Z::{1,2} X = Y, X ≠ Z, Y > Z

X	1	1	1	1	2	2	2
Y	1	1	2	2	1	1	2
Z	1	2	1	2	1	2	1

Jak zlepšit GT?

Chytrý generátor

další generované ohodnocení se „poučilo“ z chyb předchozích ohodnocení

základ technik lokálního prohledávání

Spojené generování a testování

podmínka je testována, jakmile to jde (znám hodnoty proměnných)

backtracking

Backtracking

Asi nejběžnější algoritmus systematického prohledávání

prohledávání do hloubky s navracením

Použití na řešení CSP

1) postupně ohodnocuj proměnné
2) po každém ohodnocení otestuj podmínky, jejichž všechny proměnné již mají hodnotu

Postupně prochází částečná konzistentní ohodnocení, dokud nenajde úplné (konzistentní) ohodnocení.

Otevřené otázky:

v jakém pořadí se mají proměnné ohodnocovat?

- proměnné s menší doménou dříve
- proměnné účastnící se více podmínek dříve
- klíčové proměnné dříve

v jakém pořadí se mají zkoušet hodnoty?

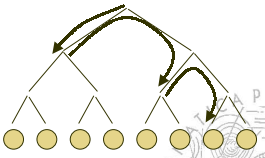
- problémově závislé

Algoritmus chronologického backtrackingu

rekurzivní definice

Algoritmus BT(X:proměnné, V: ohodnocení, C:podmínky)

```
if X={} then return V
x ← vyber neohodnocenou proměnnou z X
for each hodnota h proměnné x do
    if podmínky C splněny s V+x/h then
        R ← BT(X-x, V+x/h, C)
        if R≠fail then return R
end for
return fail
```



volá se BT(X, {}, C)

Backtracking je vždy lepší než generuj a testuj!

Problémy backtrackingu

thrashing

neumí využít informace o původu konfliktu

Příklad: A,B,C,D,E:: 1..10, A>E

vyzkouší všechny možnosti pro B,C,D, než odhalí, že A≠1

Řešení: backjumping (skok na původce chyby)

redundantní práce

nepamatuje si jednou odhalené konflikty

Příklad: A,B,C,D,E:: 1..10, B+8<D, C=5*E

při hledání hodnot pro C a E, stále zkouší přiřadit do D hodnoty 1,..,9

Řešení: backmarking, backchecking (pamatuje si dobrá/chybná přiřazení)

pozdní odhalení chyby

nesplnění podmínky odhalí teprve, když ohodnotí její proměnné

Příklad: A,B,C,D,E::1..10, A=3*E

odhalí že A>2, až při ohodnocování proměnné E

Řešení: forward checking (kontrola podmínek dopředu)