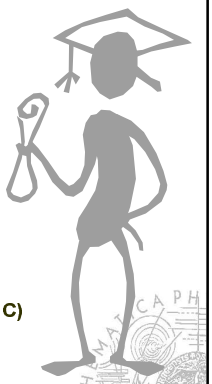
Programování omezujícími podmínkami

10

Roman Barták, KTIML

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

Co bylo minule



Řízení prohledávacího algoritmu
- uspořádání proměnných
- uspořádání hodnot

Redukce problému
- metoda eliminace cyklů (CC)
- rozšířená verze MAC (MACE)

Lokální prohledávání
- metoda největšího stoupání (HC)
- metoda minimalizace konfliktů (MC)
- vyskočení z lokálního minima
  - restartem algoritmu
  - náhodná procházka (MCRW, SDRW)

Tabu seznam

Pozorování: Setrvání v lokálním minimu je speciálním případem cyklu.

Jak se obecně zbavit cyklů?

Stačí si pamatovat předchozí stavy a zakázat opakování stavu.
- příliš paměťově náročné (mnoho stavů)

Můžeme si pamatovat pouze několik posledních stavů.
- zabrání „krátkým“ cyklům

Tabu seznam = seznam zakázaných stavů

stav lze popsat význačným atributem (nemusí být uschován celý)

⟨proměnná, hodnota⟩ - zachycuje změnu stavu (původní hodnotu)

tabu seznam má fixní délku *k* (tabu tenure)

„staré“ stavy ze seznamu vypadávají s přicházejícími novými stavy

stav popsaný prvkem tabu seznamu je zakázaný (je tabu)

Aspirační kritérium = odtabuizování stavu

do stavu lze přejít, i když je ještě v tabu seznamu

například: krok vede k celkově lepšímu stavu

Prohledávání s tabu seznamem

Tabu seznam zabraňuje krátkému cyklení.

Povoleny jsou pouze tahy mimo tabu seznam resp. tahy splňující aspirační kritérium.

Algoritmus Tabu Search

```
procedure tabu-search(Max_Iter)
    s ← random valuation of variables
    nb_iter ← 0
    initialise randomly the tabu list
    while eval(s)>0 & nb_iter<Max_Iter do
        choose a move <V,v'> with the best performance among the non-tabu
            moves and the moves satisfying the aspiration criteria
        introduce <V,v> in the tabu list, where v is the current value of V
        remove the oldest move from the tabu list
        assign v' to V
        nb_iter ← nb_iter+1
    end while
    return s
end tabu-search
```


SAT a lokální prohledávání

Řadu problémů lze formulovat v podobě SAT

= splnitelnost logické formule v konjunktivní normální formě

CNF = konjunkce klauzulí

klauzule = disjunkce literálů (podmínka)

literál = atom nebo negace atomu

Příklad:

$(A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee \neg A)$

SAT je NP-úplný problém, nelze tedy očekávat rychlý úplný algoritmus.

Lokálním prohledáváním lze řešit poměrně velké formule.

Poznámka:

splnitelnost formule v disjunktivně normální formě lze řešit v lineárním čase

Algoritmus GSAT

GSAT řeší SAT problém postupným „překlápěním“ proměnných.

Cílem je maximalizovat (vážený) počet splněných klauzulí.

Algoritmus GSAT

```
procedure GSAT(A,Max_Tries,Max_Flips)
    A: is a CNF formula
    for i:=1 to Max_Tries do
        S ← random instantiation of variables
        for j:=1 to Max_Iter do
            if A satisfiable by S then return S
            V ← the variable whose flip yield the most important raise
                in the number of satisfied clauses
            S ← S with V flipped
        end for
    end for
    return the best instantiation found
end GSAT
```

GSAT a heuristiky

GSAT lze kombinovat s různými heuristikami, které zvyšují jeho efektivitu (zvláště při řešení strukturovaných problémů):

Random-Walk

stejně jako u MCRW

Vážení klauzulí

vychází z pozorování, že některé klauzule zůstávají po řadu iterací nesplněné, klauzule tedy nemají v problému stejnou důležitost

splnění „těžké“ klauzule lze preferovat přidáním váhy

váhu může systém sám odvodit

- na začátku mají všechny klauzule stejnou váhu
- po každém pokusu je zvětšena váha u nesplněných klauzulí

Průměrování řešení

standardně každý pokus začíná z náhodného ohodnocení, tj. „zapomene“ se již získané řešení

lze zachovat společné části předchozích řešení

restartovací stav se vypočte ze dvou posledních výsledků bitovým srovnáním (stejné bity jsou zachovány, ostatní nastaveny náhodně)

Localizer

Prezentované algoritmy lokálního prohledávání mají společnou kostru, kterou lze doplnit procedurami realizujícími konkrétní algoritmus.

Algoritmus Local Search

```
procedure local-search(Max_Moves,Max_Iter)
    s ← random valuation of variables
    for i:=1 to Max_Tries while Gcondition do
        for j:=1 to Max_Iter while Lcondition do
            if eval(s)=0 then
                return s
            end if
            select n in neighbourhood(s)
            if acceptable(n) then
                s ← n
            end if
        end for
        s ← restartState(s)
    end for
    return s
end local-search
```

Optimalizace pomocí omezujících podmínek

Dosud jsme hledali libovolné řešení splňující všechny podmínky.

Často je potřeba najít optimální řešení, kde kvalita je definována objektivní funkcí.

Definice: Problém optimálního splňování podmínek (Constraint Satisfaction Optimisation Problem - CSOP) se skládá z CSP problému P a objektivní funkce *f* mapující řešení P na čísla.

Řešením CSOP je řešení problému P minimalizující / maximalizující hodnotu funkce *f*.

Pro řešení CSOP obecně potřebujeme procházet všechna řešení, tj. používají se metody schopné poskytnout všechna řešení CSP.

Metoda větví a mezí

Metoda větví a mezí (branch and bound) je nejběžnější optimalizační technika založená na ořezávání větví, ve kterých se nenachází řešení.

Používá heuristickou funkci *h* odhadující objektivní funkci.

korektní heuristika pro minimalizaci splňuje $h(x) \leq f(x)$

[korektní heuristika pro maximalizaci splňuje $f(x) \leq h(x)$]

lepší je heuristika bližší k objektivní funkci

Při prohledávání stromu řešení je podstrom uříznut, pokud:

- v podstromu neexistuje žádné řešení
- v podstromu není optimální řešení

  mez ≤ h(x), kde *mez* je maximální hodnota přípustného řešení

Jak získáme mez?

například hodnota dosud nejlepšího řešení

BB a splňování podmínek

S objektivní funkcí pracujeme jako s podmínkou

„optimalizujeme“ hodnotu proměnné *v*, kde v = f(x)

- první řešení získáme bez omezení proměnné *v*
- po dalších řešeních požadujeme, aby byla lepší (*v*<Bound)
- opakujeme, dokud nalézáme lepší řešení

Algoritmus Branch & Bound

```
procedure BB-Min(Variables, V, Constraints)
    Bound ← sup
    NewSolution ← fail
    repeat
        Solution ← NewSolution
        NewSolution ← Solve(Variables,Constraints ∪ {V<Bound})
        Bound ← value of V in NewSolution (if any)
    until NewSolution = fail
    return Solution
end BB-Min
```


Poznámky k metodě větví a mezí

Heuristika *h* je ukryta v propagaci podmínky v = f(x)

Efektivita řešení je závislá na:

- dobré heuristice (propagace přes objektivní funkci)
- včasném nalezení dobrého řešení

  můžeme pomoci přesnější počáteční mezí

Nalezení optimálního řešení může být rychlé

dlouho trvá důkaz optimálnosti (musí se projít zbylá řešení)

Většinou nepotřebujeme optimální řešení, stačí dobré řešení

- BB můžeme ukončit po dosažení stanovené meze

Zrychlení BB získáme zpřesňováním dolní a horní meze půlením

```
repeat
    TempBound ← (UBound+LBound) / 2
    NewSolution ← Solve(Variables,Constraints ∪ {V≤TempBound})
    if NewSolution=fail then
        LBound ← TempBound+1
    else
        UBound ← TempBound
until LBound = UBound
```

Trochu motivace - problém šatníku

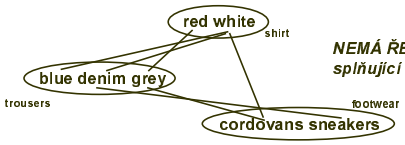
Vyberte oblečení tak, aby navzájem ladilo.

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 přílišomezené (pře-omezené, over-constrained).

První řešení problému šatníku

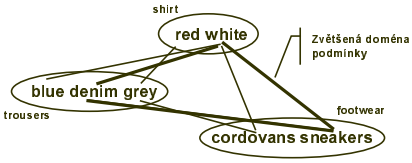
Problém sice nemá řešení, ale mi nějaké potřebujeme.

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 $(PS,\leq)$, 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 $\langle P,(PS,\leq),M,(N,S)\rangle$

- P je původní problém
- $(PS,\leq)$ 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 $P' \in PS$ a $M(P,P') < N$. Postačujícím řešením je řešení, kde $M(P,P') \leq S$. 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.

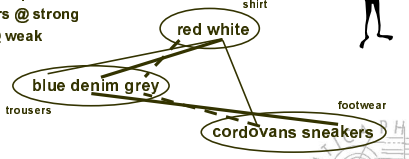
Druhé řešení problému šatníku

Podmínkám můžeme přiřadit preferenci určující, jaké podmínky mají být 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.

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

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