Plánování a rozvrhování

6

Roman Barták, KTIML

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

Na úvod

Dosud prezentované plánovací systémy používaly **ad-hoc algoritmy**, tj. speciální plánovací algoritmy.

Můžeme plánovací problém **kódovat v jiném formalismu** a využít jeho řešící techniky?

Ano! Např. použitím **SAT a CSP**.

Proč?

Zlepšení obecných řešících algoritmů pro SAT a CSP povede okamžitě ke zlepšení efektivity plánovače.

Problém:

SAT i CSP používají **statické kódování**, my ale dopředu **neznáme velikost plánu**!

Řešení:

Budeme hledat **plány do dané maximální délky** a v případě neúspěchu se délka zvětší.

Plánování
jako SAT

Základní myšlenka

1. **Plánovací problém kódujeme jako výrokovou formuli.**
   - Jak popsat stavy?
   - Jak popsat stavové přechody?
2. Prostředky SATu **zjistíme splnitelnost této formule.**
   - Davis-Putnam, GSAT, ...
3. Z pravdivostních hodnot proměnných zpětně **dekódujeme řešení plánovacího problému**.

Kódování stavů

- **Stavy** v klasické reprezentaci jsou **množiny** (konjunkce) **instanciovaných atomů**.
  Příklad: at(r1,l1) ∧ ¬loaded(r1)
- Stačí **atomy** kódovat jako **výrokovou proměnnou**.
  Příklad: předchozímu stavu odpovídá model
  {at(r1,l1)←true, loaded(r1)←false}
  - **Nemůže pak ale být robot být na dvou místech najednou?**
    Např. máme-li výrokovou proměnnou pro at(r1,l2), dostáváme dva modely předchozího stavu:
    - {at(r1,l1)←true, loaded(r1)←false, at(r1,l2)←true}
    - {at(r1,l1)←true, loaded(r1)←false, at(r1,l2)←false}

    z nichž pouze druhý model odpovídá naší představě.
  - **Ve stavu musíme kódovat i „zřejmé" fakty!**
    tj. at(r1,l1) ∧ ¬at(r1,l2) ∧ ¬loaded(r1)

  Poznámka:
  Výroková formule může bez problémů reprezentovat množiny stavů:
  ((at(r1,l1) ∧ ¬at(r1,l2)) ∨ (¬at(r1,l1) ∧ at(r1,l2))) ∧ ¬loaded(r1)

Kódování přechodů

- Formule kódující stav nepokrývá dynamiku systému!
- **Přechod** použitím akce move(r1,l1,l2) lze popsat dvěma formulemi:
  - at(r1,l1) ∧ ¬at(r1,l2) % počáteční stav
  - ¬at(r1,l1) ∧ at(r1,l2) % stav po přechodu
- **V jiných stavech tedy platí jiné fakty!**
- Přidáme do atomů i označení stavu (tzv. **fluents**) a potom můžeme **přechod kódovat jedinou formulí**:
  - at(r1,l1,s1) ∧ ¬at(r1,l2,s1) ∧ ¬at(r1,l1,s2) ∧ at(r1,l2,s2)
- Do formule přidáme také novou výrokovou proměnnou označující akci, která přechod způsobila:
  - move(r1,l1,l2,s1) ∧
    at(r1,l1,s1) ∧ ¬at(r1,l2,s1) ∧ ¬at(r1,l1,s2) ∧ at(r1,l2,s2)
- Jak ale víme, že na stav s1 se použije právě move?

Plánovací problém

Budeme hledat řešení plánovacího problému s danou **maximální délkou plánu** (n).

Pro každý krok i plánu máme

- **proměnné** (**fluents**) L popisující možný stav, např. at(r1,l1,i)
- **proměnné** popisující možné **akce** A, např. move(r1,l1,l2,i)

Plánovací problém kódujeme jako konjunkci následujících formulí

- Formule popisující **počáteční stav** (úplný popis):
  $\wedge\{p_0 \mid p \in s_0\} \wedge \wedge\{\neg p_0 \mid p \in L - s_0\}$
- Formule popisující **cíl** (částečný popis):
  $\wedge\{p \mid p_n \in g^+\} \wedge \wedge\{\neg p_n \mid p \in g^-\}$
- Pro každou **akci** máme formule popisující, jaké změny tato akce přinese, pokud se použije v i-tém kroku plánu:
  - $a_i \Rightarrow \wedge\{p_i \mid p \in \text{precond}(a)\} \wedge \wedge\{e_{i+1} \mid e \in \text{effects}(a)\}$
- **Axiom úplného vyloučení**:
  - Pro každou dvojici různých akcí a, b máme formule říkající, že se tyto akce nemohou použít najednou:
    $\neg a_i \vee \neg b_i$

Je to vše?

Problém rámce

- Musíme zaručit, že proměnné, které nejsou změněny použitou akcí, zůstanou stejné.

- **Axiomy rámce**:
  formule popisující co se mezi stavy nemění
  existují různé způsoby, jak je zapsat
  - **klasické axiomy rámce**
    - u každé akce a popíšeme vlastnosti p, které se nemění ($p \notin \text{effects}(a)$)
      $a_i \Rightarrow (p_i \Leftrightarrow p_{i+1})$
    - ještě je potřeba zaručit, že v každém kroku je jedna akce vybrána
      $V\{a_i \mid a \in A\}$
  - **axiomy rámce s vysvětlením**
    - pravdivostní hodnota fluent proměnné p se mění pouze, pokud to udělá nějaká akce
      $(\neg p_i \wedge p_{i+1} \Rightarrow V\{a_i \mid p \in \text{effects}^+(a)\})$;; někdo přidal p
      $\wedge\ (p_i \wedge \neg p_{i+1} \Rightarrow V\{a_i \mid p \in \text{effects}^-(a)\})$;; někdo ubral p
    - potřebují **axiom úplného vyloučení**, ten ale vede k úplně uspořádaným plánům, což může omezovat
    - stačí **axiom konfliktního vyloučení**, tj. stačí aby v jednom kroku nebyly dohromady závislé akce

Jiné kódování akcí

- Zkusme zmenšit počet proměnných pro reprezentaci akcí (méně proměnných = menší prostor k prohledání)
- Situace: 3 roboti a 3 místa
  - Pro akci move(r,loc1,loc2) potřebujeme v každém kroku $3 \times 3 \times 3 =$ **27 proměnných**.
  - Proměnnou pro move(r1,loc1,loc2) nahradíme konjunkcí 3 proměnných:

    $\text{move1(r1)} \wedge \text{move2(loc1)} \wedge \text{move3(loc2)}$
  - Protože akce move teď sdílejí stejné proměnné (např. move2(loc1) se použije pro move(r1,loc1,loc2) i pro move(r2,loc1,loc3)) dostáváme v každém kroku 3+3+3= **9 proměnných.**
- **Proměnné můžeme sdílet i mezi akcemi** různého typu, např. move a fly.

Závěrečné poznámky

- **Celkový rámec pro plánování jako SAT**

  for n = 0, 1, 2, …,
  kóduj (P,n) jako SAT Φ
  if Φ is satisfiable then
  z modelu pro Φ extrahuj plán a vrať ho
  - **Extrakce plánu**
    - v každém kroku i máme právě jednu proměnnou a_i s pravdivostní hodnotou true, ta určuje příslušnou akci do plánu
- **Jak dobře to funguje?**
  - **samo o sobě není moc praktické** (paměťové a časové nároky)
  - vhodné v **kombinaci** s dalšími technikami, např. **plánovacím grafem**
    - systém **BlackBox** kóduje plánovací graf jako SAT, tj. máme proměnné pouze pro akce a atomy z grafu
    - splnitelnost získané formule ověřuje existenci plánu

Příklad

- **Plánovací doména:**
  - jeden robot: r1
  - dvě propojené lokace: l1, l2
  - jeden operátor: move
- **Kódujeme plánovací problém** (P,n) pro plán délky n = 1
  - počáteční stav: {at(r1,l1)}
    at(r1,l1,0) $\wedge$ $\neg$**at(r1,l2,0)**
  - cíl: {at(r1,l2)}
    at(r1,l2,1)
  - operátor: move(r,l,l') = (precond: at(r,l), effects: at(r,l'), $\neg$at(r,l))
    move(r1,l1,l2,0) $\Rightarrow$ **at(r1,l1,0)** $\wedge$ **at(r1,l2,1)** $\wedge$ $\neg$**at(r1,l1,1)**
    move(r1,l2,l1,0) $\Rightarrow$ **at(r1,l2,0)** $\wedge$ **at(r1,l1,1)** $\wedge$ $\neg$**at(r1,l2,1)**
  - axiom úplného vyloučení:
    $\neg$**move(r1,l1,l2,0)** $\vee$ $\neg$**move(r1,l2,l1,0)**
  - axiomy rámce s vysvětlením:
    $\neg$**at(r1,l1,0)** $\wedge$ **at(r1,l1,1)** $\Rightarrow$ **move(r1,l2,l1,0)**
    $\neg$**at(r1,l2,0)** $\wedge$ **at(r1,l2,1)** $\Rightarrow$ **move(r1,l1,l2,0)**
    at(r1,l1,0) $\wedge$ $\neg$**at(r1,l1,1)** $\Rightarrow$ **move(r1,l1,l2,0)**
    at(r1,l2,0) $\wedge$ $\neg$**at(r1,l2,1)** $\Rightarrow$ **move(r1,l2,l1,0)**

Plánování
a splňování podmínek

Základní použití

- **Plánování jako CSP**
  - plánovací problém kódujeme a řešíme jako CSP
    - SAT je vlastně speciálním případem CSP
    - my ale použijeme jinou reprezentaci plánovacího problému, **reprezentaci se stavovou proměnnou**
- **CSP v plánování**
  - CSP techniky se používají přímo v plánovacích algoritmech (např. Graphplan)
  - jedná se o **běžnější** způsob využití CSP technik v plánování

Stavové proměnné

- **Stavové proměnné** popisují vlastnost nějakého objektu v závislosti na stavu formou funkce.
  - rloc: robots $\times$ S $\rightarrow$ locations
  - rload: robots $\times$ S $\rightarrow$ containers $\cup$ {nil}
  - cpos: containers $\times$ S $\rightarrow$ locations $\cup$ robots
- **Neměnné vztahy** se (pořád) reprezentují relacemi.
  - adjacent(loc1,loc2)
  - robots(r1) ;; popisuje typy konstant
- **Operátory** popisují změny stavových proměnných.
  - **move(r,l,m)**
    ;; robot r at location l moves to an adjacent position m
    precond: rloc(r)=l, adjacent(l,m)
    effects: rloc(r)$\leftarrow$m
  - **load(c,r,l)**
    ;; robot r loads container c at location l
    precond: rloc(r)=l, cpos(c)=l, rload(r)=nil
    effects: rload(r)$\leftarrow$c, cpos(c)$\leftarrow$r
  - **unload(c,r,l)**
    ;; robot r unloads container c at location l
    precond: rloc(r)=l, rload(r)=c
    effects: rload(r)$\leftarrow$nil, cpos(c)$\leftarrow$l

Plánování jako CSP
proměnné

- Opět modelujeme omezený plánovací problém, tj. hledáme **plán do dané délky k**.
- Pro každou instanciovanou **stavovou proměnnou** x_i budeme mít CSP proměnnou $x_i(j)$ $0 \leq j \leq k$ s doménou odpovídající oboru hodnot.
  - **Příklad**:
    - rloc(r1,j)$\in${l1,l2,l3}, $0 \leq j \leq 4$
    - rload(r1,j)$\in${c1,c2,c3,nil}, $0 \leq j \leq 4$
    - cpos(c,j)$\in${l1,l2,l3,r1}, $0 \leq j \leq 4$, c$\in${c1,c2,c3}
- Pro každé j, $0 \leq j \leq k-1$, máme **akční proměnnou** act(j) s doménou obsahující možné akce (včetně no-op).
  - **Příklad**:
    - act(j) $\in${move(r1,l1,l2),move(r1,l1,l3),…, no-op}, $0 \leq j \leq 3$

Plánování jako CSP
kódování stavů

- **Počáteční a koncový stav** kódujeme formou **unárních omezujících podmínek**:
  - má-li stavová proměnná x_i ve stavu s_0 hodnotu v_i, potom přidáme podmínku
    - $x_i(0) = v_i$

    Příklad:
    rloc(r1,0)=l1, rload(r1,0)=nil, cpos(c1,0)=l1, cpos(c2,0)=l2, cpos(c3,0)=l2
  - má-li stavová proměnná x_i v cíli g hodnotu v_i, potom přidáme podmínku
    - $x_i(k) = v_i$

    Příklad:
    cpos(c1,4)=l2, cpos(c2,4)=l1

Plánování jako CSP

kódování akcí

- Akce kódujeme formou podmínek svazujících akční proměnnou act(j) s „okolními" stavovými proměnnými $x_i(j)$ a $x_i(j+1)$.
- **Předpoklad akce** a ve tvaru $\mathbf{(x_i=v_i)}$ se přeloží do podmínek
  $act(j)=a \Rightarrow x_i(j)=v_i,\ 0 \le j \le k-1.$
- **Předpoklad akce** a ve tvaru $\mathbf{(x_i \in D_i)}$ se přeloží do podmínek
  $act(j)=a \Rightarrow x_i(j) \in D_i,\ 0 \le j \le k-1.$
- Přiřazení $\mathbf{x_i \leftarrow v_i}$ z **efektů akce** a se přeloží do podmínek
  $act(j)=a \Rightarrow x_i(j+1)=v_i,\ 0 \le j \le k-1.$

Příklad:

- act(j)=move(r1,l1,l2) ⇒ rloc(r1,j)=l1 ;; předpoklad
- act(j)=move(r1,l1,l2) ⇒ rloc(r1,j+1)=l2 ;; efekt

Poznámka: další předpoklad adjacent(l1,l2) se vzal v úvahu ihned při tvorbě akce z operátoru (dosazení konstant)

Plánování jako CSP

axiomy rámce

- Podobně jako u SAT kódování je potřeba zajistit, že stavové proměnné, které nejsou změněny vybranou akcí, si podrží svojí hodnotu – **axiomy rámce**.
- Použijeme **ternární podmínky** svazující akční proměnnou act(j) s „okolními" stavovými proměnnými $x_i(j)$ a $x_i(j+1)$.
- Konkrétně, není-li stavová proměnná x_i v efektu akce a (x_i je vzhledem k a **invariantní**), potom přidáme podmínky:
  $act(j)=a \Rightarrow x_i(j)=x_i(j+1),\ 0 \le j \le k-1.$

Příklad:

- act(j)=move(r1,l1,l2) ⇒ rload(r1,j)=rload(r1,j+1)
- act(j)=move(r1,l1,l2) ⇒ cpos(c,j)=cpos(c,j+1), c∈{c1,c2,c3}

CSP v plánování

- **CSP techniky lze použít jako součást existujících plánovacích algoritmů.**
- plánování v prostoru plánů
  - testování konzistence částečného plánu pomocí propagace omezujících podmínek (rychlé, ale neúplné)
  - úplný test konzistence může být proveden když je nalezen kompletní plán
- **plánování s plánovacím grafem**
  - plánovací graf je statická struktura a může být zakódovaná jako CSP
  - CSP se používá ve fázi extrakce plánu z plánovacího grafu

Do & Kambhampati (2000)

Graphplan jako CSP

- Použijeme proměnné pro atomy a tyto proměnné budou určovat, jaká akce přidala daný atom.

CSP model:

- **proměnné**
  - výrokové uzly ze stavových vrstev $P_{j,m}$ (atom p_j ve vrstvě m)
  - číslovány jsou pouze stavové vrstvy
- **doména proměnných**
  - aktivity, které mají daný atom jako pozitivní efekt
  - $\perp$ pro neaktivní atom
- **omezující podmínky**
  - spojují pozitivní efekty s předpoklady akce
  - relace vzájemné výlučnosti (mutex)

Graphplan jako CSP

podmínky

$P_{4,m}=a \Rightarrow P_{1,m-1}\neq\bot \wedge P_{2,m-1}\neq\bot \wedge P_{3,m-1}\neq\bot$

- akce *a* má předpoklady p_1, p_2, p_3 a pozitivní efekt p_4
- podmínka je přidána pro každý pozitivní efekt akce *a*

$P_{i,m}=\bot \vee P_{j,m}=\bot$

- mutex mezi atomy p_i a p_j

$P_{i,m}\neq a \vee P_{j,m}\neq b$

- akce *a*, *b* jsou označeny jako mutex, p_i je pozitivní efekt *a* a p_j je pozitivní efekt *b*

$P_{i,k}\neq\bot$

- p_i je cílový atom a k je index poslední vrstvy

žádné paralelní akce

- maximálně jedna akce různá od no-op je přiřazena proměnným ve stejné vrstvě

žádné prázdné vrstvy

- minimálně jedna akce různá od no-op je přiřazena proměnným v každé vrstvě

Graphplan jako BCSP

- Použijeme Boolean proměnné pro výroky i akce a hodnota true bude určovat přítomnost výroku/akce.

Boolean CSP (SAT) model:

- **proměnné**
  - Boolean proměnné pro akční uzly $A_{j,m}$ a pro výrokové uzly $P_{j,n}$
  - vrstvy jsou číslovány jako v Graphplanu, tj. stavové vrstvy od nuly a akční vrstvy od jedné
- **doména proměnných**
  - hodnota **true** znamená, že akce/výrok je aktivní
- **omezující podmínky**
  - spojují akce s předpoklady a pozitivními efekty
  - relace vzájemné výlučnosti (mutex)

Graphplan jako BCSP

podmínky

- **podmínky pro předpoklady akce**
  - $\mathbf{A_{i,m}} \Rightarrow \mathbf{P_{j,m-1}}$
  - p_j je předpokladem akce a_i
- **podmínky pro následující stav**
  - $\mathbf{P_{i,m}} \Leftrightarrow \quad (\vee_{p_i \in effects^+(a_j)} \mathbf{A_{j,m}}) \vee \ (\mathbf{P_{i,m-1}} \wedge (\wedge_{p_i \in effects^-(a_j)} \neg \mathbf{A_{j,m}}))$
  - p_i je aktivní, pokud je přidáno nějakou akcí nebo pokud je aktivní v předchozí vrstvě a žádná akce ho nesmazala
  - no-op se zde nepoužívají!
  - Pozor! Podmínky umožňují, aby byl atom jednou akcí přidán a zároveň jinou akcí smazán, takže podmínky pro mutex jsou potřeba!
- **podmínky pro mutex**
  - $\neg \mathbf{A_{i,m}} \vee \neg \mathbf{A_{j,m}}$ mutex akcí a_i a a_j ve vrstvě m
  - $\neg \mathbf{P_{i,n}} \vee \neg \mathbf{P_{j,n}}$ mutex atomů p_i a p_j ve vrstvě n
- **počáteční stav a cíle**
  - $\mathbf{P_{i,k}}$**=true** p_i je cílový atom a k je index poslední stavové vrstvy
  - $\mathbf{P_{i,0}}$**=true** p_i je atom v počátečním stavu
- **další podmínky**
  - žádné paralelní akce – maximálně jedna akce ve vrstvě je aktivní
  - žádné prázdné vrstvy – minimálně jedna akce ve vrstvě je aktivní

CSP vs. SAT

- **Podobný základní princip**
- **CSP kódování je kompaktnější**
  - jednodušší reprezentace stavu s méně proměnnými (díky použití stavové proměnné)
  - nejsou potřeba axiomy vyloučení akcí (díky použití proměnné pro akci)

Poznámka:

- CSP a SAT jsou NP-úplné zatímco plánování je PSPACE- nebo NEXPTIME-úplné. Jak to?
- Převod plánovacího problému na SAT resp. CSP přináší **exponenciální nárůst velikosti problému**.
  - SAT: exponenciální počet Boolean proměnných
  - CSP: lineární počet proměnných, ale exponenciální nárůst je ve velikosti CSP díky doménám