PLÁNOVÁNÍ V UMĚLÉ INTELIGENCI I

MICHAL PĚCHOUČEK

Úvod

V mnoha knihách o umělé inteligenci je definována *ui* jako inteligentní interakce mezi:

AGENTEM ↔ ÚLOHOU ↔ PROSTŘEDÍM

Plánování je ve své podstatě uvažování o hypotetické interakci mezi agentem a prostředím. Motivací plánovacího procesu je tedy uvažovat o možných akcíh, které vyvolají změny v prostředí za cílem dosažení cílového stavu (úloha).

Plánování není rozvrhování → Při plánování je třeba uvažovat že jednotlivé komponenty plánu spolu mohou souviset a interagovat. Při rozvrhování toto není problém a jediné co je třeba řešit je optimálně přiřadit v řase jednotlivé zdroje daným úlohám. Plánování se spíš hodí řešit znalostním inženýrstvím, zatímco rozvrhování softocomputingem (*symblic functionalism* je vlastně teorií pro *knowledge engineering*)

Representace stavů a cílů

Popis domény plánování se skládá z jazyka L pro popis domény problému a z množiny oprátorů O pro popis schopností agenta.

Jazyk L je definován jako omezený jazyk skládájící se z predikátů, constant, proměnných a negací. *Termem* zde rozumíme konstanty a proměnné v L. *Atom* je výraz

$$p(t_1, \ldots, t_n)$$

kde p je predikáty arity n a t_i jsou termy. *Literál* je atom nebo jeho negace. Každý stav je tedy popsán jako množina representující konjunkci literálů. Cíl je vlastně instance stavu.

Příklad:

```
have(white-wine) ∨ have(red-wine)
```

```
at(X) ∧ sells(X,milk)
```

značí být v nějakém obchodě, kde prodávají mléko.

Representace Operátorů

Operátor je definován jako

$$\alpha \equiv \langle \textit{akce}, \textit{podmínky}, \textit{efekty}, \textit{cena} \rangle$$

O operátoru řekneme, že je *částeční instanciován* když jeho meta-popis obsahuje proměnné (jinak – *operátorové schema*). Operátor je *plně instanciován* když jsou všechny jeho proměnné nahrazeny knostantami.

O operátoru α řekneme, že je *aplikovatelný* na množině plně instancovaných literálů popisující daný stav světa – S_n, když všechny *podmínky* α platí v množině S_n. Aplikace operátoru α má z následek přechod staveu S_n do stavu S_{n+1}. Nazveme *Del* množinu literálů odstraněných aplikací α. Pak

$$S_{n+1} \rightarrow (S_n - Del) \cup Eff(\alpha)$$

Representace Plánovacího Problému

Plánovací problém je tedy definován jako:

$$pp \equiv \langle S_{init}, S_{goal}, O \rangle$$

kde *init* je množina literálu počátečního stavu, *goal* je množina literálu cílového stavu a O je množina plánovacích operátorů.

Algorimtus, který řeší plánovací problem nazvěmě plánovačem. V podstatě rozlišujeme *progresní* a *regresní* plánovače, které prohledávají stavový prostor situací. Budeme-li porhledávat prostor plánů je plánovací problem definován jako

$$pp \equiv \langle \Pi_{částečný}, \Pi_{úplný}, O \rangle, \text{ kde}$$

plánovací operátory nad světem plánů jsou přidání kroku, vazba mezi proměnnými, přidání uspořádání.

Representace Plánu

Plně uspořádaný plán

Sekvence plně instanciovaných operátorů $\underline{\Pi}$ je řešením plánovacího problému kde každý operátor je chápán jako *krok* plánování. Prvn í krok je aplikovatelný na počáteční stav, *i*-tý krok na produkt aplikování kroku *i*-1. Každý literál cílového stavu platí pro následníka posledního kroku. Plán $\underline{\Pi}$ je *úplně uspořádaným plánem* a je považován za *korektní*.

Částečně uspořádaný plán

Plán Π nazveme *částečně uspořádaným plánem* není-li pořadí všech operátorů plně specifikováno, t.j. existují-li alespoň dva operátory které nejsou navzájem uspořádány. Částečné uspořádání R na množině U je nereflexivní a transitivní binární relace. Platí:

- $\forall\ a, a \in U: (a, a) \notin R$
- $\forall\ a, b, c \in U: ((a, b) \in R \wedge (b, c) \in R) \rightarrow (a, c) \in R$

Používáme značení $s_1 \prec s_2$ pro každou dvojici $(s_1,s_2) \in R$. Úplně uspořádaný plán lze odvodit z $\underline{\Pi}$ *linearizací* plánu, t.j. přidáním dalších relací uspořádání. *Plně instanciovaný plán* je takový plán, kde jsou všechny proměnné vázány na konstaty.

Příklad: Pop $\{\{a \prec b\}\ \{a \prec c\}\ \{c \prec d\}\ \{b \prec e\}\ \{e \prec f\}\ \{d \prec f\}\} = \{\{a \prec b \prec e \prec f\}\ \{a \prec c \prec d \prec f\}\}$ odpovídá množině šesti Top .

Struktura plánu

Plán je tedy definován jako struktura obashující:

- množinu kroků, kde krok je aplikace jednoho z operátorů plánovacího problému $S(\Pi)$
- uspořádání kroků $R(\Pi)$
- množina specifikující vazby mezi proměnnými jednotlivých kroků $B(\Pi)$
- množina kauzálních linek $C(\Pi)$

Kauzální Linka

Každý částečně uspořádaný plán může obsahovat *kauzální linky* – relace mezi kroky ve smyslu splnitelnosti podmínek. Kauzální linka $\langle s_1 \rightarrow_p s_2 \rangle$ říká, že krok s_1 nastaví podmínkový literál pro s_2, t.j.

- p je podmínkou pro s_2
- p je efektem pro s_1
- $s_1 \prec s_2$ platí v částečně uspořádaném plánu

Vlastnosti Korektního Plánu

Chápejme *korektní plan*, jako plán který lze vykonat a zaručí nám že se dostaneme z počátečního do cílového stavu jako řešení plánovacího problému. Řešením plánovacího problému je *úplný* a *konzistetní* plán. Plán je úplný v případě, že každá podmínka každého operátoru je splněna aplikací jiného operátoru a není zrušena jiným operátorem:

$$\forall\, S\ \forall\, c, c \in \mathrm{pre}(S) : \exists\, S' : c \in \mathrm{eff}(S') \wedge \neg \exists\, S'' : \neg c \in \mathrm{eff}(S'')$$

Konzistetní plan je takový plan, kde uspořádání mezi operátory ani vazební omezení nepřináší kontradikci.

$$\forall S_1 \forall S_2 : S_1 \prec S_2 \Rightarrow \neg\, S_2 \prec S_1$$

Úplně stejně nelze bvázat jednu proměnnou na dvě konstaty během jednoho kroku $v = A$ a $v = B$. Vzheledm k transitivnosti relace $\prec$ platí že $S_1 \prec S_2 \wedge S_2 \prec S_3 \wedge S_3 \prec S_1$ je rovněž nekonzistentní.

Příklad:

POPIS SVĚTA:

at(*place*), *have*(*item*), *sells*(*where*,*what*)

start: $\{at(home) \wedge sells(HWS, drills) \wedge sells(SM, milk) \wedge sells(SM, banana)\}$

finish: $\{have(drill) \wedge have(banana) \wedge have(milk)\}$, EFF:`nil`

OPERÁTORY:

$\langle$ACTION: **go(there),** PREC: $\{at(here)\}$, EFF: $\{\neg at(here) \wedge at(there)\}\rangle$

$\langle$ACTION: **buy(this),** PREC: $\{at(store) \wedge sells(this, store)\}$, EFF: $\{have(this)\}\rangle$

Ohrožení kauzání linky

Kauzální linka $\langle s_i \rightarrow_p s_j \rangle$ je *ohrožena* krokem s_k pokud tento krok znemožňuje aplikaci kauzální linky. Krok s_k je pak chápan jako *ohrožení* linky $\langle s_i \rightarrow_p s_j \rangle$. Ohrožení může být buď pozitivní nebo negativní. Negativní ohrožení kuazální linky $\langle s_i \rightarrow_p s_j \rangle$ pomocí kroku s_k v rámci plánu Π platí-li:

(i) vazby $s_i \rightarrow s_k$ a $s_k \rightarrow s_j$ jsou konzistentní s plánem Π a

(ii) existuje efekt $q \in Eff(s_k)$ tak že $\neg\, q \in Pre(s_j)$

Pozitivní ohrožení kuazální linky $\langle s_i \rightarrow_p s_j \rangle$ pomocí kroku s_k v rámci pláni Π platí-li:

(i) vazby $s_i \rightarrow s_k$ a $s_k \rightarrow s_j$ jsou kinzistentní s plánem Π a

(ii) existuje efekt $q \in Eff(s_k)$ tak že $q \in Pre(s_j)$

Ohrožení se ošetří pomocí *demotion* t.j. přidáním pomocného omezení $s_k \rightarrow s_i$, nebo *promotion* přidáním pomocného omezení $s_j \rightarrow s_k$, nebo *separation* t.j. oddělenou vazbou proměnných parametrů.

Algoritmy Plánování

POPLAN – Partial-Order, Backward Chaining Algorithm

```
(i)    OpenList ← ⟨init → goal⟩
(ii)   Π ← chepest(OpenList), OpenList ← OpenList – Π
(iii)  if correct(Π) return (Π)
(iv)   if not(OpenList ← OpenList + expand(Π,O)) return(failed)
(v)    if exists-threats(Π)  Π ← resolve-threat(Π)
(vi)   get-back (ii)
```

`Correct` – Plán Π považujeme za korektní je-li ho možno dekomponovat na množinu plně uspořádaných plánů. Jedná se vlastně o takový plán Π, který nebosahuje žádnou otevřenou podmínku p kroku s, t.j. podmínku pro kterou neexistuje žádná kauzální linka $\langle s_1 \rightarrow_p s_2 \rangle$.

`Expand` – snaží se zprava rozbalit vazby → částečného uspořádání, tak že identifikují otevřené podmínky p kroku co nejblíže k cíli tím, že nachází operátory, které mají p jako efekt.

```
(i)    find in Π step S with unachieved precondition c
(ii)   if not(S ← find in Π step S´, c ∈ Eff(S´))
       a. Links(Π) ← Links(Π) + S´→c S
       b. Ordering(Π) ← Ordering(Π) + S´≺ S
       c. if not present
            i. Steps(Π) ← Steps(Π)  + S´
           ii. Ordering(Π) ← Ordering(Π) + Start≺ S´≺ Goal
(iii)  return(Π)
```

`Resolve-threads` – snaží se ošetřit detekovaná *ohrožení* kauzálních linek pomocí *promotion*, *demotion*, *separation*.

```
For each St threatening S1→c S2 ∈ Π
     do either:
     (a)   Ordering(Π) ← Ordering(Π) + St≺ S1
     (b)   Ordering(Π) ← Ordering(Π) + S2≺ St
     If not consistent(Π) than fail
```

TOPLAN – Total Ordered, Forward-Chaining Algorithm

```
(i)    OpenList ← ⟨init⟩
(ii)   Π ← chepest(OpenList), OpenList ← OpenList – Π
(iii)  if goal ∈ Π → return(Π)
(iv)   OpenList ← OpenList + expand(Π)
(v)    get-back (ii)
```

Expand – Najde všechny možné následníky z pro daný plán, t.j. přidá do Π za každý krok s_i, který nemá následníka všechny operátory α, takové, že $Eff(s_i) \equiv Pre(\alpha)$, s ohledem na vazbu proměnných