PLÁNOVÁNÍ V UMĚLÉ INTELIGENCI II

MICHAL PĚCHOUČEK

1 Opakování

(plánovací problém, úplně uspořádaný plán, částečně uspořádaný plán, lineární, nelineární plánování, transformace na řešení problémů)

2 Hierarchické Plánování

Pro řešení komplexních úloh bude potřeba vytvořit dlouhé plány, složené z primitivních operací. Proto i stavový prostor takovéto úlohy bude uhromný a bude náročné ho efektivně prohledat. Proto byla vytvořena teorie použití makroperátorů (meta-representací), které byly schopny popsat podle svých PODMÍNEK a EFEKTŮ specifické kusy plánovacího procesu. Proto je potřeba dělit operátory na

- primitivní – nepopisují další operace, nedekomponují se dále
- ne-primitivní – představují meta-representaci dalších shluků operátorů

Ke každému neprimitvnímu operátoru je třeba vytvořit metodu DECOMOSE, která redukuje operátor na problém plánování na nižší úrovni.

```
decompose(α):
      find-decomposition-relation(α,init,goal,O)
      Π ← plan(init,goal,O)
      return(Π)
```

POPLAN plánovací algoritmus poté plánuje stejným způsobem jako u nehierarchického plánování. Rozlišujem mezi rekurentním *plánováním do hloubky* a *plánováním do délky*. Zatímco plánování do hloubky pokaždé při specifikaci správného makroperátoru spustí algoritmus plánování na nižší úrovni, při plánování do délky algoritmus vytvoří nejdřív celý plán na abstraktní úrovni a pak postupuje do hloubky.

PROHLEDÁVÁNÍ DO HLOUBLKY:

```
(i)    if correct(Π) return (Π)
(ii)   if exists-threats(Π)  Π ← resolve-threat(Π)
(iii)  if not( Π ← Π + expand-sub-goals(Π,O)) return(failed)
(iv)   get-back (i)

expand-sub-goals(Π,O):
      either return(Π + choose-primitive-operators(Π,O))
      or     Π´← choose-operators-decompostion(Π,O)
             return(Π + plan(Π´))
```

PROHLEDÁVÁNÍ DO DÉLKY:

```
(i)    if correct(Π) return (Π)
(ii)   if correct(first(Π)) plan(rest(Π))
(iii)  if exists-threats(Π)  Π ← resolve-threat(Π)
(iv)   if not( Π ← Π + expand-sub-goals(Π,O)) return(failed)
(v)    get-back (i)

expand-sub-goals(Π,O):
either return(Π + choose-primitive-operators(Π,O))
      or     Π´← choose-operators-decompostion(Π,O)
                  return(Π + Π´)
```

2.1 Plánování Důkazů

Jedná se o aplikaci hierarchického plánování do problematiky dokazování matematických vět. Vlastně celý svět operátorů a meta-operátorů popisuje inferenční znalost matematika když postupuje při vytváření důkazu. Svět je tedy rozdělěn na *metody* a *taktiky*, které řídí dokazovací proces. Existují tři fáze algoritmu:

- plánování
- ohodnocení
- exekuce

(viz obrázek, příklad)

2.2 Plánování Rozhodování

Plánování rozhodování je obecná metodologie representace znalostí pro libovolný rozhodovací proces. Znalosti jsou zde vlastně representovány pomocí více-úrovňových *rozhodovacích grafů*. Motivace simulace rozhodovacího procesu je tedy prohledávání rozhodovacího grafu za účelem nalezení cesty. Která representuje výsledné rozhodnotí.

Existují dva typy rozhodovacího grafu – *abstraktní* a *specifický*.

(viz příklady) – zdůraznit distribuovanost, použití, strojové učení

3 Problém Rámce

Jedním ze základních filosofických problémů plánovaní v umělé inteligenci je problém známý rámce – *frame problem* a jemu příslušný *ramfication problem*. Problém rámce tkví v tom jak vhodně a efektivně representovat znalost v situačním kalkulu o tom že se nějaká vlastnost objektů po aplikování jisté akce nemění. Pro toto používáme rámcové predikáty. Těch ale může být strašně moc a nikdo neví kolik jich bude v budoucnu potřeba. To je něco co umělá inteligence ještě nezvládla vyřešit. (*representační problém*)

Ramifikační problém naopak řeší potřebu representovat jak spolu naopak souvsí vykonané akce.

4 Přehled Plánovacích Systémy

- QA3 THEOREM PROVER - 1969 Green, založen na situačním kalkulu, QA (question askingng – systém se snaží otestovat jsou-li předpoklady operátoru pravdivé. Nejsou-li pak vrátí altrnativy jak je nastavit ve smyslu dalších operaátorů, vazby řazení, …), vrátí sekvenci akcí

- KOWALSKI – 1979 (prolog), založen na jazyku víčřádové predikátové logiky, vytváří partial ordered plány
- STRIPS – 1972 (Fikes, Hart, Nilson) nejznámčjší plánovač, pracuje na myšlkenkách ↑, založeno na GPS, means end analysis (zmenšuje počet nesedících atributů), používal *trojúhelníkovou tabulku* pro representaci základníb *teleologické struktury* plánu. (příklad)
- HACKER – 1973 (Susman) první plánovač který se učí ze svých chyb, mčl jsitou galerii kritik, INTERPLAN - pokračování (Tate) HACKER
- WALDINGER – 1977, problém se systémy ↑, je v tom že stráví čas na špatném plány a když selžou staví plán znovu. Zde je požitá methodologie *regrese cílů*, kde se plán staví tak aby dosáhl nčjakého cíle a byl dále modifikován aby dosáhl dalšího, etc.
- WARPLAN – 1974 stejné jako ↑, ale používá *regrese akcí*. (najde akci která splňuje cíl a snaží se najít místo v plánu kam by se dala co nejmírumilovnčji vložit.
- NOAH – 1977 nelineární plánovač který jako první zavedl pojem procedurálních sítí pro representaci částečnč uspořádaných plánů. Použil hierarchickou dekomposici. *Least-Commitment*
- NONLIN – Tate ↑
- MOLGEN – Stefik based on constrain satisfaction, constrain propagation
- PRS – procedural reasoning system, používá automat konečných stavů a byl použit pro RT.
- O-PLAN – (Tate) open planning architecture, je založen na filosofii tabule a využívá mnohé další ↑ methodologie, MAS plánování.