Report

Popis formalizace

Zadaný problém jsem formalizoval pomocí 4 hlavních predikátů, kterými popisuji aktuální stav stavového prostoru. Sleduji stav robota, stav jeho robotické ruky, aktuální obsazenost stolu, kde se manipuluje s deskami, a nekonec stav jednotlivých desek. V doméně se jedná konkrétně o tyto predikáty:

```
(robot-s ?s - robot-state)
(robot-hand ?s - tool)
(wood-state ?w - wood ?l - location ?c - color ?s - shape)
(table-s ?ts - table-state)
```

Poté jsem zavedl pomocné predikáty pro možnost upgradovat robota a pro sledování počtu desek na stole.

```
(can-upgrade ?rs0 ?rsN - robot-state)
(table-count-incr ?ts ?tsN - table-state)
(table-count-decr ?ts ?tsN - table-state)
```

Akce definované v doméně jsou tyto: vzít a položit nástroj, položit desku na stůl a zase ji sejmout. Dále vyřezat dřevo, nabarvit dřevo a nakonec upgradovat robota. Akce jsem se snažil volit tak, abych nemusel použít negativní `precondition`, protože se ukázalo, že proto je potřeba další `requirenments` a to `negative-preconditions`.
Funkce, které se v doméně vyskytují jsou dvojího druhu, bezparametrická celková cena všech akcí `(total-cost)`, kterou optimalizujeme. A dále funkce pro ceny pro jednotlivé úkony robota, které závisí na jeho aktuálním stavu.
Abych se vyhnul desetinným číslům v průběhu optimalizace, vynásobil jsem všechny počáteční hodnoty hodnoty konstantou 2. Všechny akce jsem poté zlevňoval ve stejném poměru. Vy výsledcích tedy uvádím mnou posunuté (tj. dvojnásobné hodnoty) optimalizační funkce.
Pro správnou funkčnost solverů jsem uvedl do sekce `requirenments` tyto proměnné: `:strips :equality :typing :action-costs`.

Plánovače

Pro porovnání jsem vybral plánovače:

1. `lama2008`
2. `sgplan6`
3. `roamer`

Testovací prostředí

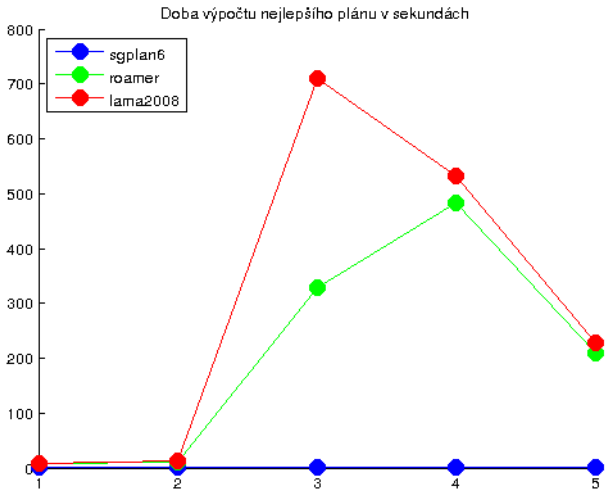
Plánovače jsem testoval v nativní prostředí, konkrétně na operačním systému Linux Xubuntu x86_64 s jádrem 3.11.0-18. Hardwarové parametry počítače jsou Core2Duo na frekvenci 2.8 Ghz a 8GB RAM.

Porovnání plánovacích časů

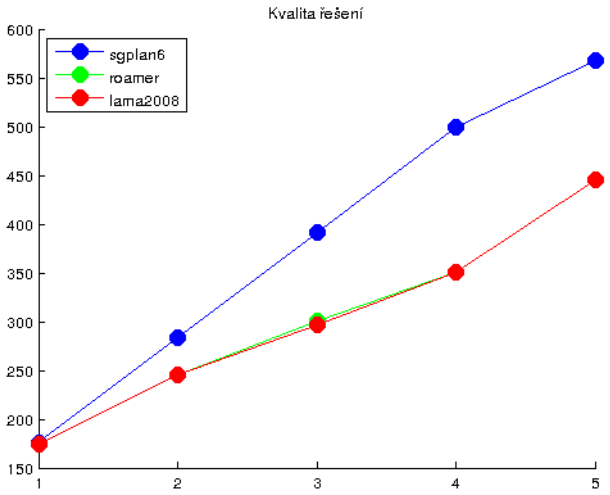
Porovnání výpočetních časů plánovačů je vidět na obrázku 1. Z grafu by bylo možné říct, že chování plánovačů je na první pohled nelogické. Nejedná se o chybu - výpočty byly ukončeny, protože plánovače spotřebovali veškerou paměť . Plánovač `sgplan6` naplánoval všechny úlohy ve zlomku sekundy pro libovolný z vytvořených problémů. Plánovače `lama2008` a `roamer` vždy narazili na paměťový limit testovacího počítače. Plánovač `lama2008` vždy o trochu dříve, protože byl kompilovaný pro 32b verzi systému, plánovač `roamer` vždy po vyčerpání zhruba 6GB paměti RAM.

Srovnání kvality řešení

Porovnání výpočetních časů plánovačů je vidět na obrázku 2. Z výsledků je vidět, že pro výslednou hodnotu je relevantní doba výpočtu. Plánovač `sgplan6` sice najde výsledek téměř okamžitě, ale poskytuje jednoznačně nejhorší řešení – u složitějších úloh zhruba 2x horší. Plánovače `roamer` a `lama2008` si vedou hodně podobně.



Obrázek 1: Srovnání časů mnou použitých plánovačů



Obrázek 2: Srovnání hodnoty nejlepších řešení použitých plánovačů

Závěr

Podařilo se mi formulovat úlohu tak, aby ji plánovače byly schopné vyřešit (téměř všechny, až na `yahsp2`, kvůli chybějící knihovně). Nicméně předpokládám, že úloha by šla formalizovat i lépe - nemyslím si, že na takto malých testovacích problémech, by plánovače `lama` a `roamer` neměli vyčerpat veškerou paměť a navíc by měli dojít k optimálnímu řešení, což se jim u některých problémů nepodařilo. V úloze jsem si vyzkoušel jak formalizovat problém do jazyka PDDL, tak že mu rozumí široká paleta plánovačů. Na základě jejich výsledků bych pak mohl řešit různé úlohy. Například kdyby mi solver `sgplan6` rychle vrátil výsledek, měl bych jistotu že nějaké řešení existuje a pak bych mohl zkoušet sofistikovanější plánovače pro zlepšení hodnoty optimalizované funkce. Za závěr mi přijde dobré vědět, že není nutné pro každý problém psát si vlastní plánovač nebo nějaký jednoduchý CSP solver, ale stačí problém formalizovat pomocí PDDL a použít již hotový software k vyřešení problému.