VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

POROVNÁNÍ KLASICKÝCH METOD PLÁNOVÁNÍ

COMPARISON OF CLASSICAL PLANNING METHODS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **REBEKA ČERNIANSKA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. FRANTIŠEK ZBOŘIL, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2021/2022

Zadání bakalářské práce

Studentka: **Černianska Rebeka**
Program: Informační technologie
Název: **Porovnání klasických metod plánování**
Comparison of Classical Planning Methods
Kategorie: Umělá inteligence

Zadání:

1. Seznamte se s klasickými metodami plánování, vedle systému STRIPS také se systémy HTN, GraphPlan, případně SatPLAN.
2. Zvolte příklady a úlohy pro plánování tak, aby vystihovaly různé problémy plánování, například konfliktní cíle, částečně uspořádané úlohy a podobně.
3. Implementujte, nebo zvolte aktuální implementace plánovacích metod a proveďte jejich srovnání na zvolených příkladech.
4. Na základě získaných výsledků diskutujte, pro jaké typy jsou nebo nejsou jednotlivé metody vhodné.

Literatura:

- A. Blum and M. Furst (1997).Fast planning through planning graph analysis. Artificial intelligence. 90:281-300
- H. A. Kautz and B. Selman (1992). Planning as satisfiability. In Proceedings of the Tenth European Conference on Artificial Intelligence

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Zbořil František, doc. Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 3. listopadu 2021

Abstrakt

Táto bakalárska práca sa zaoberá problematikou automatizovaného plánovania a existujúcich metód, ktoré riešia jeho problémy. Ide o analýzu a porovnanie správania metód STRIPS, Graphplan a HTN, ktoré implementujú rôzne prístupy plánovania. Ich fungovanie je testované na typických plánovacích úlohách, ktoré sú zamerané na ich rôzne problémy. Cieľom bolo nájsť vhodné využitie jednotlivých metód, ako aj ich výhody a nevýhody v závislosti od zadanej úlohy. Na testovanie boli zvolené existujúce implementácie skúmaných metód, na ktorých je ukázané ich správanie.

Abstract

This bacherlor's thesis deals with the topic of automated planning and the methods which deal with its' problems. The analysis and comparison of the methods STRIPS, Graphplan and HTN takes place, which implement different approaches to planning. Their performance is tested on typical planning problems, which focus on various planning challenges. The goal was to find suitable planning problems for each of the methods, as well as their strong and weak points regarding the given problem. For testing purposes, existing implementations of the methods were used, which showcase their behaviour.

Kľúčové slová

Automatizované plánovanie, Plánovacie metódy, Plánovač, STRIPS, Graphplan, HTN, Hierarchické plánovanie, Plánovací graf, Plánovacia úloha, Plánovacia doména, PDDL, Klasické plánovanie

Keywords

Automated planning, Planning methods, Planner, STRIPS, Graphplan, HTN, Hierarchical planning, Planning graph, Planning problem, Planning domain, PDDL, Classical planning

Citácia

ČERNIANSKA, Rebeka. *Porovnání klasických metod plánování.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. František Zbořil, Ph.D.

Porovnání klasických metod plánování

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracovala samostatne pod vedením pána docenta Ing. Františka Zbořila Ph.D. Uviedla som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpala.

.....
Rebeka Černianska
9. mája 2022

Poďakovanie

V prvom rade ďakujem pánovi docentovi Ing. Františkovi Zbořilovi Ph.D. za jeho čas a cenné rady pri tvorbe práce. Ďalej ďakujem rodine a blízkym, obzvlášť priateľovi Ondrejovi, za ich psychickú podporu a neustále povzbudzovanie pri štúdiu. Bez nich by táto práca nebola hotová.

Obsah

Kapitola 1

Úvod

Jeden z cieľov informačných technológií v ľudských životoch je uľahčovať riešenie problémov, s ktorými sa ľudia opakovane stretávajú. Jedná sa o problémy, ktoré je možné vyriešiť automatizovane, bez ľudského pričinenia. Jedna z oblastí informatiky zaoberajúca sa práve automatizovaným hľadaním riešenia je umelá inteligencia.

Jej snahou je riešiť problémy prístupom podobným, ako by to robil ľudský mozog. Existuje veľa rôznych prístupov a spôsobov v rámci odboru umelej inteligencie, z ktorých jedným je plánovanie.

Cieľom tejto práce je skúmať oblasť plánovania, typické prístupy, ktoré sa v tejto oblasti využívajú a niekoľko metód, ktoré riešia jeho úlohy. V rámci plánovania existuje rad metód, ktoré pomocou rôznych prístupov riešia úlohy a snažia sa o čo najlepšie a najefektívnejšie riešenie.

Úlohou plánovania je nájsť postupnosť krokov, pomocou ktorých je možné prísť k riešeniu problému, ktorý nastal alebo bol zadaný.

V tejto bakalárskej práci sa budú skúmať a porovnávať vybrané metódy riešiace plánovacie problémy. Cieľom je demonštrovať rôzne typy plánovacích úloh a spôsoby akými sú riešené pomocou daných metód. Pomocou testovania na viacerých druhoch plánovacích problémov sa bude analyzovať vhodnosť jednotlivých metód pri zvolených typoch problémov.

Kapitola 2

Plánovanie v umelej inteligencii

Plánovanie je proces, pri ktorom sa pred vykonaním nejakej aktivity preskúma, akým spôsobom je najlepšie postupovať. Je to snaha nájsť najlepší možný spôsob vykonania procesu pomocou dostupných prostriedkov bez aktívneho skúšania rôznych prístupov. Plánovanie predchádza samotnému vykonaniu aktivity a má za úlohu nájsť čo najlepšie riešenie, ktoré bude možné aplikovať. [10]

Takisto, ako pri ľudskom plánovaní, snahou pri automatizovanom plánovaní je nájsť čo najlepšie riešenie nejakého problému predtým, ako sa samotný problém začne riešiť. Základným princípom plánovania v umelej inteligencii je nájsť cestu, ktorá zabezpečí prechod z jedného stavu systému do druhého. Jeho úlohou je teda nájsť plán, ktorý zaručene vykoná prechod medzi aktuálnym a cieľovým stavom. Tento prechod je riešením ľubovoľne zadaného problému, ktorého riešenie je žiaduce nájsť. Pôvodný stav je situácia, v ktorej sa aktuálne nachádza prostredie, v rámci ktorého je potrebné vyriešiť nejaký problém. Cieľovým stavom sa rozumie stav, do ktorého je požadované prostredie dostať a sú v ňom platné zadané ciele. Zmena medzi týmito dvoma postaveniami je preto problém, ktorý je nutné vyriešiť a na to bude slúžiť práve nájdenie plánu pre jeho vyriešenie. Plán bude pozostávať z postupnosti krokov, ktoré po vykonaní spôsobia zmenu aktuálneho stavu na cieľový stav.

2.1 Pojmy potrebné pre definíciu plánovania

Pre proces plánovania je potrebné definovať niekoľko pojmov, ktoré sa v tejto problematike využívajú. Plánovanie je vždy vykonávané v rámci jedného vymedzeného priestoru, v ktorom je požadované vyriešiť nejaký problém. Bude prebiehať v určenom priestore alebo systéme, v rámci ktorého existujú objekty a akcie. Problém, pre ktorý sa bude hľadať riešenie, sa bude vzťahovať na objekty a stavy v rámci vymedzeného priestoru a bude využívať akcie tu existujúce. Tento systém môže byť otvorený alebo uzatvorený, ale je nutné, aby bol vopred stanovený. Objekty v tomto priestore môžu mať medzi sebou rôzne vzťahy. Tieto vzťahy sú dané stavom systému, ktorý môže nastať.

Stav reprezentuje vzťah objektov v priestore voči sebe. Stav systému je popísaný pomocou predikátov, ktoré majú názov, objekty a jasne definujú vzťah týchto objektov voči sebe. To, aký stav môže v priestore nastať je definované jednotlivými predikátmi a ich názvami, ktoré sú vopred určené a takisto objektmi, ktoré sú v priestore. Množina stavov, ktoré je možné dosiahnuť, môže byť buď konečná, alebo nekonečná. To je závislé na tom, či je definovaný priestor konečný alebo nekonečný. Tieto pojmy boli definované na základe zdroja [24].

Ďalej sa jedná o pojem plán, ktorý je definovaný podľa [18], ktorý v tomto kontexte predstavuje jasnú postupnosť krokov, ktoré sú riešením zadaného problému. Kroky pozostávajú z vykonaných akcií, ktoré spôsobujú zmeny stavu. Plán umožňuje prechod medzi pôvodným a cieľovým stavom. Pozostáva z krokov, ktoré majú určené poradie postupnosti vykonania. Samotný krok v pláne pozostáva z akcie, ktorá je v rámci neho vykonaná. Pokiaľ po nájdení plánu budú vykonané kroky, ktoré stanovuje, bude zadaný problém možné vyriešiť. Konečným cieľom plánovania je buď nájsť funkčný plán, ktorý umožní riešenie zadaného problému, alebo vyhlásiť, že taký plán neexistuje. Zároveň je cieľom vytvoriť čo najkratší a najrýchlejší plán.

Pre zmenu stavu systému je možné využívať plánovacie operátory, ktorých popis a definícia je založená na [9]. Definujú akým spôsobom a za akých podmienok je možné vykonať akciu v priestore, čím dôjde k zmene jeho stavu. Akciou rozumieme konkrétne využitie plánovacieho operátoru, kde budú všetky premenné nahradené za konkrétne literály, teda priamo zapísané hodnoty namiesto premenných. Pomocou plánovacieho operátoru je možné zmeniť stav v priestore a je to kľúčový prvok, ktorý umožňuje plánovanie. Je definovaný menom, objektmi, podmienkami a následkami. Hovorí o tom, čo sa udeje s objektmi a ako bude ovplyvnený vzťah medzi nimi, po splnení vstupných podmienok. Umožňuje teda prechod medzi rôznymi stavmi a pri hľadaní plánu sa využíva na rôzne manipulácie s objektmi pre dosiahnutie cieľového stavu.

2.2 Špecifikácia klasického plánovania

Táto podkapitola je parafrázovaná z [10]. Zložitosť procesu plánovania je závislá od niekoľkých faktorov. Podľa toho, aké vlastnosti majú jednotlivé prvky potrebné pre plánovanie, je možné rozdeliť plánovanie na rôzne druhy. Druh plánovania, ktorý je potrebný definovať pre túto prácu je klasické plánovanie, ktorého úlohy budú testovať skúmané metódy. Podstata hľadania postupnosti krokov, ktorá umožní prechod medzi aktuálnym a požadovaným stavom, je stále rovnaká ako pri všeobecnom pojme plánovania. Rozdiely nastávajú v spôsobe, ako je definovaný plánovací systém a takisto akcie v rámci systému. Obmedzenia charakteristické pre klasické plánovanie ovplyvňujú nielen spôsob hľadania riešenia problému, ale takisto aj ako je problém zadávaný.

Prvú vlastnosť, ktorú je potrebné špecifikovať, je determinizmus plánovacích operátorov. Ich dôsledok na aktuálny stav je jasne daný a je pri použití ľubovoľných objektov rovnaký. Vždy je možné presne určiť pre ľubovoľnú akciu, s ľubovoľnými objektmi, aké budú jej následky. Táto vlastnosť sa využíva pri hľadaní rôznych možností postupovania v pláne, s prihliadaním na možný výsledný stav po zmene systému.

Dĺžka priebehu akcie je okamžitá, teda nie je meraný jej čas a nijak nevplýva na priebeh plánu a plánovania.

Poradie akcii vo finálnom pláne môže byť čiastočne alebo úplne usporiadané, čo znamená, že poradie akcií v nájdenom pláne musí byť splnené aspoň v jednotlivých úsekoch plánu.

Systém, v ktorom sa odohráva klasické plánovanie, je konečný. Je presne ohraničený objektmi, ktoré sa v ňom môžu vyskytovať, stavmi, ktoré môžu nastať a operátormi, ktoré môžu byť použité.

Systém je statický, čo znamená, že neexistujú udalosti, ktoré by sa mohli v systéme spontánne udiať. Jediné zmeny systému, ktoré môžu nastať, sú tie po aplikovaní akcie. Stav systému je možné plne sledovať, všetko čo sa v ňom nachádza a deje je možné monitorovať.

Plánovač sa zaoberá obmedzenými cieľmi, ktoré sú zadané ako cieľový stav, alebo skupina cieľových stavov. Úlohou je dosiahnuť cieľový stav pomocou zvolených krokov plánovača.

Plánovanie prebieha offline, teda plánovač sa zaoberá stavom systému, ktorý mu bol zadaný ako pôvodný, alebo aktuálny a nezaoberá sa zmenami, ktoré nastali počas jeho práce. Predpokladá, že stav systému sa nijako nezmenil, teda nesleduje zmeny, ktoré sa udiali počas plánovania nad daným systémom.

2.3 Všeobecný priebeh plánovania

Činnosť hľadania plánu vykonáva plánovač, ktorý je zodpovedný za nájdenie plánu. Tomu musia byť poskytnuté informácie o tom, aký je aktuálny stav prostredia, v ktorom sa nachádza, čo je jeho cieľový stav a taktiež aké existujú prostriedky na prechod medzi týmito stavmi [18]. Plánovač je implementovaný určitou zvolenou plánovacou metódou. Pomocou existujúcej metódy, ktorej úlohou je riešiť plánovacie problémy, nájde riešenie pre zadaný problém. Existuje veľké množstvo metód, ktoré sú určené na plánovanie a každá z nich má svoje špecifické vlastnosti. Ich cieľ a účel je ale spoločný a ide im o vyriešenie zadanej úlohy čo najlepším spôsobom.

Práca plánovania začína zadaním úlohy, kedy je pomocou zvolenej reprezentácie potrebné zadefinovať priestor, v ktorom sa bude plánovanie vykonávať. Samotná úloha je potom riešená v zadanom priestore, pričom je potrebné poskytnúť jeho počiatočný stav. Zadaním je potom cieľový stav, ktorý je potrebné dosiahnuť. [18]

Po zadaní úlohy môže začať jej riešenie, ktoré bude vykonávané pomocou zvolenej plánovacej metódy. Tento proces je individuálny pre každý prístup, pričom základ tohto procesu je snaha zmeniť stav systému čo najefektívnejšie pomocou krokov v ňom. Možné zmeny v systéme sú vopred definované a pri procese plánovania je hľadaná postupnosť zmien systému, ktorá vyrieši zadaný problém. Plánovanie končí, keď sa podarí nájsť riešenie zadanej úlohy pomocou povolených prostriedkov.

Kapitola 3

Zadávanie plánovacích problémov

Pre samotné plánovanie je potrebné definovať problém alebo úlohu, ktorú bude plánovač riešiť [9]. Existuje niekoľko zaužívaných spôsobov, ktoré sa na definovanie úlohy využívajú. Tri druhy prístupov k zapisovaniu plánovacieho problému sú pomocou množín, klasického zápisu a nakoniec pomocou stavového popisu. V tejto práci sa bude využívať zápis pomocou klasického prístupu a to najmä z dôvodu jeho popularity. Jedná sa o typický prístup volený v prípade uzatvoreného stavovo prechodového priestoru, ako uvádza zdroj [10].

3.1 Reprezentácia problémov pomocou formalizmov

Klasická reprezentácia využíva zápis, ktorý čerpá z predikátovej logiky prvého rádu.
Pre túto reprezentáciu je potrebné definovať jazyk, ktorým sa vyjadruje. Jazyk predikátovej logiky má konečný počet predikátových symbolov, konštantných symbolov a neexistujú v ňom symboly funkcií. Preto pre tento jazyk platí, že každý jeho term je buď konštantný symbol alebo premenný symbol. Ďalším prvkom bude sled alfanumerických znakov, ktoré budú používané ako názvy predikátov konštantných symbolov, pričom tento rad bude dlhý aspoň dva znaky. Pre popisovanie premenných znakov budú takisto použité alfanumerické znaky, pričom premennú bude reprezentovať práve jeden znak. Odsek bol parafrázovaný z [10].

V príkladovej doméne `Simple Blocks` 6.4.1, ktorá je neskôr používaná na testovanie, je definovaný operátor polohy kocky. Pričom názov predikátu je `on` a názvy premenných symbolov sú `block` a `table`. Pre ten istý predikát sú podobne definované konštantné symboly, ktorých názvy sú `A` a `T`. Tieto sa budú vyskytovať v zápise pri aplikovaní operátora na konkrétny stav.

```
on(block, table) -> on(A, T)
```

Jednotlivé stavy sú popísané pomocou atómov, pri ktorých je sledovaná ich pravdivosť v danom kontexte. Atóm pozostáva z predikátu a n termov. Stav je skupina základných atómov, ktoré patria do definovaného jazyka. Z definície jazyka vyplýva, že počet dosiahnuteľných predikátov je konečný. Pre túto reprezentáciu sa používa predpoklad uzatvoreného sveta, ktorý reprezentuje predpoklad, že predikáty, ktoré nie sú explicitne uvedené v stave systému, sú nepravdivé [9]. Pomocou predikátov sa popisuje celkový stav priestoru, v ktorom sa pracuje a umožňujú prácu v ňom. Sú základným prvkom potrebným pre prehľadnú reprezentáciu plánovacích problémov. Cieľom plánovania je sledovať jednotlivé predikáty a počas plánovania ich meniť tak, aby systém našiel riešenie.

3.2 Domény

Na to, aby bolo možné zadať konkrétny problém na vyriešenie, je potrebné definovať doménu, v ktorej sa systém bude pohybovať. Doména špecifikuje prostredie, v ktorom sa rieši úloha. Sú v nej definované objekty, ktoré sa môžu v systéme vyskytovať. Takisto sú v doméne popísané plánovacie operátory, ktoré sú v danej doméne k dispozícii. V doméne je potrebné definovať predikáty, ktoré sa budú v systéme vyskytovať a využívať. [3]

Napríklad pre doménu `Simple Blocks` je možné definovať operátor nasledovne.

```
objects: b, t1, t2
action: move(b, t2)
preconditions: on(b, t1)
postconditions: on(b, t2)
```

Pričom `move` je operátor v tejto doméne a jeho parametre `b, t1` a `t2` predstavujú objekty, s ktorými pracuje. Operátor predstavuje presun bloku `b` z jeho pozície `t1` na pozíciu `t2`. Takisto tu je zobrazená reprezentácia predikátu `on(block, table)`, ktorá definuje pozíciu objektov voči sebe.

3.3 Úlohy

Samotná úloha, ktorá sa bude riešiť, je definovaná oddelene od domény vzhľadom na to, že v rámci jednej domény môže byť riešený ľubovoľný počet úloh. Pri definovaní úlohy je potrebné plánovaču predať informácie o aktuálnom stave systému a o cieľových stavoch, ku ktorému sa má plánovač dopracovať. Tu je špecifikované nielen všeobecne aké objekty sa môžu v systéme nachádzať, ale konkrétne reprezentácie týchto objektov, s pomenovaním a s určitým vzťahom voči systému, ktorý je definovaný pomocou predikátov v systéme. Na predchádzajúcom príklade je možné tieto údaje definovať nasledujúcim spôsobom:

```
initial state: on(b, t1)
goal state: on(b, t2)
```

3.4 Operátory v doméne

Na to, aby bolo možné nejaký stav zmeniť, je potrebné definovať operátory, ktoré môžu byť na stav aplikované. Plánovací operátor je trojica prvkov: meno, vstupné podmienky a výstupné podmienky. [10]

Meno, alebo názov operátora, ho jedinečne identifikuje a využíva sa v systéme na jeho označovanie. Jeho vstupné a výstupné podmienky sú definované predikátmi a sú potrebné na overenie jeho použiteľnosti v danom stave systému a na opísanie vplyvu operátora na systém. Vstupné podmienky hovoria o predikátoch, ktoré musia byť platné, alebo naopak nesmú platiť predtým, ako je operátor aplikovaný na daný systém. Predikáty vstupných podmienok musia byť v súlade s aktuálnym stavom systému, inak nie je možné použiť operátor v danej chvíli. Výstupné podmienky predstavujú to, ako sa zmenia predikáty aktuálneho stavu systému po aplikovaní operátora. Zmeny sa týkajú toho, aké predikáty prestanú platiť a aké naopak nadobudnú platnosť po použití operátora. Príklad definície je možné vidieť v kapitole 3.2.

3.5 Jazyky využiteľné pre definíciu plánovacích úloh

Ako prvý veľký plánovací systém bol vyvinutý STRIPS [6] - STanford Research Institute Problem Solver - ktorý sa neskôr v tejto práci bude podrobne skúmať ako jedna z plánovacích metód. Pre zadávanie problémov je potrebné spomenúť, že STRIPS reprezentuje nielen metódu, ale neskôr bol týmto názvom pomenovaný aj spôsob zapisovania plánovacích úloh. Tento prístup bol jeden z prvých zavedených postupov pre zapisovanie úloh, ktorý bol postupom času vylepšený, až ho úplne nahradil najaktuálnejší spôsob zápisu v podobe jazyku PDDL [24]. Tieto reprezentácie sú veľmi podobné, pričom ich odlišuje menšie obmedzenie vyjadrovania v PDDL, ktoré umožňuje voľnejší zápis.

3.6 Reprezentácia plánovacích úloh pomocou PDDL

PDDL [7] - Planning Domain Definition Language - je jazyk vytvorený v snahe o štandardizáciu zápisov plánovacích úloh. Bol vytvorený s hlavným využitím pre medzinárodné plánovacie súťaže (International Planning Competition [13]), v ktorých rôzne plánovače súťažia v riešení úloh. Úlohou tohto jazyka bolo zjednotiť spôsob zadávania úloh na súťažiach, avšak je potrebné podotknúť, že je veľké množstvo plánovačov, ktoré tento jazyk nepodporujú v plnej miere, alebo vôbec. Aj keď ide o snahu o štandardizáciu, nejde o oficiálny štandard v rámci plánovania.

PDDL od svojho vzniku prešlo niekoľkými vylepšeniami, a preto existuje niekoľko verzií tohto jazyka. Každá verzia so sebou priniesla prvky, ktoré umožňujú rozsiahlejšie definovanie úlohy [7]. Pre potreby porovnania metód nie je potrebné využívať rozširujúce prvky jazyka a postačí jeho základná verzia. Aj v rámci základnej verzie sa vyskytujú prvky, ktoré niektoré plánovače nevyužívajú, alebo nevedia využiť pre svoje fungovanie.

3.6.1 Vyjadrovacie prvky v reprezentácii PDDL

Popis prvkov jazyka PDDL je vytvorený na základe zdrojov, ktoré tento jazyk využívali na ukázanie fungovania domén, ako napríklad [10], [23]. Pri definovaní plánovacej úlohy je pri použití jazyka PDDL potrebné predstaviť jeho základné prvky. Jedná sa o pôvodný stav systému, ďalej o cieľový stav, ktorý predstavuje úlohu, ktorá sa bude riešiť, plánovacie operátory, ktoré sa v doméne nachádzajú a je možné s nimi narábať. Ďalej ide o predikáty, ktoré sa v systéme vyskytujú a o objekty, ktoré budú pre danú úlohu relevantné. V tomto jazyku sa úloha typicky definuje v 2 osobitných súboroch, pričom ide o súbor, v ktorom je definovaná doména a o súbor, v ktorom je zadaná úloha. Preto je vždy úloha zadávaná touto dvojicou súborov, ktoré sú prepojené názvom domény.

3.6.2 Štrukturálne rozdelenie zadávaných informácií

V doménovom súbore sú definované operátory, ktoré sa budú v doméne používať, a takisto sú tu definované predikáty, ktoré sa môžu v doméne vyskytovať. Okrem toho, sa v tomto súbore nachádzajú úvodné formálne definície, ktoré popisujú názov domény, aké požiadavky na ňu existujú, typy objektov v doméne, avšak tieto prvky nie sú kľúčové pre fungovanie jazyka. Nie všetky plánovače prvok predikátov používajú, vzhľadom na to, že vyskytujúce sa predikáty je možné zistiť z plánovacích operátorov definovaných pre doménu. Pochopiteľne, tento súbor môže byť párovým súborom pre viacero úloh. V rámci jednej domény je možné riešiť viacero úloh a preto je možné pre každú doménu vytvoriť niekoľko súborov s úlohami.

Príklad domény `Simple Blocks` v jazyku PDDL:

```
(define (domain blocksworld)
  (:requirements :strips)
  (:predicates
    (on ?b ?t))
  (:action move
     :parameters (?b ?t1 ?t2)
     :precondition (and (on ?b ?t1) (not (on ?b ?t2)))
     :effect (and (on ?b ?t2) (not (on ?b ?t1)))))
```

Operátory nachádzajúce sa v súbore musia byť definované pomocou 4 prvkov. Ide v prvom rade o názov, ktorým ju bude systém identifikovať. V príklade je možné vidieť definovanie názvu za kľúčovým slovom `:action`, za ktorým sa vyskytuje samotné meno operátora.

Ďalej je potrebné definovať parametre tejto akcie. Spravidla ide o nenulový počet objektov, ktoré operátor berie ako vstup a s ktorými bude v rámci svojej činnosti pracovať. Ich definícia nasleduje za kľúčovým slovom `:parameters`, kde sú postupne vymenované. Každý parameter je postupne definovaný názvom za znakom `?`.

Následne je pre akciu definovaný sled podmienok, ktoré musia byť splnené pred jej použitím Takéto podmienky sú definované za kľúčovým slovom `:precondition`. Ide o konjunkciu predikátov, ktoré vyjadrujú v akých vzťahoch k systému a k sebe navzájom sa jednotlivé vstupné objekty musia nachádzať pred aplikovaním operátora. Pokiaľ niektorý z predikátov nespĺňa požadovanú pravdivostnú hodnotu, plánovač tento operátor nebude brať do úvahy ako použiteľný v danom stave systému.

Pokiaľ je možné operátor aplikovať, sú vykonané definované zmeny v systéme za kľúčovým slovom `:effect`. Tie sú súčasťou definície operátora a jedná sa o takzvaný efekt na systém alebo o výstupné podmienky operátora. Táto informácia je zložená z konjunkcie predikátov, pre ktorú platí, že predikáty s kladnou pravdivostnou hodnotou sú po prevedení akcie do systému pridané, pričom tie so zápornou sú zo systému odstránené, pokiaľ sa v ňom nachádzali.

Kľúčové slovo `not` značí, že daný predikát nesmie byť platný v danom stave.

Definícia úlohy v jazyku PDDL:

```
(define (problem move-blocks-between-tables)
    (:domain blocksworld)
  (:objects a b c table1 table2)
  (:init (on a table2) (on b table1)
       (on c table2) (on d table1) (on e table2))
  (:goal (and (on a table1) (on b table2)
   (on c table1) (on d table2) (on e table1))))
```

V súbore pre definovanie úlohy sú uvedené všetky informácie potrebné pre jej stanovenie. Jedná sa v prvom rade o konkrétne objekty s ich názvami, s ktorými sa bude v rámci danej domény pracovať, definovanými za kľúčovým slovom `:objects`. Iné objekty sa nemôžu vyskytovať v pôvodnom ani v cieľovom stave. Nie je možné pracovať v rámci zadanej úlohy s inými objektami, než tými, ktoré sú definované v tejto časti úlohy.

Následne je definovaný počiatočný stav systému, v ktorom sa nachádza na začiatku riešenia úlohy, ktorý je definovaný za kľúčovým slovom `:init`. Tento stav je popísaný pomocou konjunkcie predikátov, ktoré sú platné v systéme pred začiatkom hľadania riešenia. Keďže jeden z predpokladov, ktorý bol v kapitole 2.2 uvedený pre klasické plánovanie, očakáva uzatvorený svet, predikáty uvedené v tejto konjunkcii nesmú byť definované ako neplatné.

Tie, ktoré sa tu nevyskytujú, budú brané ako nepravdivé v danom stave. Preto v tejto časti nie je potrebné definovať záporné predikáty.

Posledná časť tohto súboru definuje cieľový stav systému za kľúčovým slovom `:goal`. Ide o stav, do ktorého sa má systém dostať a úlohou plánovača je nájsť spôsob ako to zabezpečiť. Pre cieľový stav platia rovnaké podmienky ako pre pôvodný stav - teda je definovaný rovnakou konjunkciou s rovnakými obmedzeniami a významom.

Kapitola 4

Metódy riešenia problémov klasického plánovania

Princíp plánovania bol spomenutý v kapitole o klasickom plánovaní 2. Snaha plánovača je nájsť postupnosť krokov, ktorá bude riešiť zadaný problém. Spôsob, akým je problém zadávaný bol popísaný v predchádzajúcej kapitole 3.6. Samotné hľadanie plánu môže prebiehať veľa rôznymi spôsobmi, v závislosti od zvolenej plánovacej metódy. Väčšina metód má spoločný základ fungovania, ktorý sa vylepšuje a optimalizuje pomocou nejakého prvku alebo prístupu. V tejto práci sa bude pracovať s tromi rôznymi metódami, ktoré sa líšia v spôsobe riešenia úlohy.

Existuje množstvo rôznych prístupov k riešeniu plánovacích problémov, od priamočiarych až po tie, ktoré problém transformujú a snažia sa o jeho abstraktné riešenie. Metódy zvolené na porovnanie sú zvolené z rôznych skupín podľa spôsobu, akým sa metódy stavajú k riešeniu. Porovnávať sa bude metóda STRIPS [6], ktorej prístup je najviac priamočiary zo zvolených, no zároveň staval základy ostatných metód svojou prelomovosťou. Ďalšie metódy sa snažia o efektívnejšie hľadanie riešenia, pričom ide o Graphplan [3], ktorého veľká časť riešenia sa zaoberá zefektívnením prehľadávacieho priestoru. Posledným prístupom je HTN [8], ktorého prístup je značne odlišný od predchádzajúcich, ako je aj jeho cieľ. V ostatných metódach bola snaha o nájdenie plánu, ktorý umožní prechod z aktuálneho, do cieľového stavu. HTN sa snaží o rozloženie zadanej úlohy na jednoduché kroky, kedy cieľový stav je abstraktná úloha a nie stav priestoru, ako uvádza zdroj [10]. Ide o plánovaciu metódu, ktorá získava popularitu a stojí za zmienku zaoberať sa ňou a porovnávať jej fungovanie narozdiel od ostatných metód.

4.1 Plánovanie v stavovom priestore - State-space planning

Podkapitola sa opiera o [10]. Medzi najjednoduchšie plánovacie metódy patria algoritmy prehľadávania stavového priestoru, ktorý je možné reprezentovať pomocou grafu. Uzlami grafu sú jednotlivé stavy a jeho hranami sú prechody medzi nimi. Tento graf je tvorený postupne, popri tom ako plánovač hľadá aplikovateľné operátory. Pokiaľ je niektorá z možností testovaná, uzol v grafe pribudne a v prípade že nie je vhodný, nebude z neho pokračovať ďalej. Naopak, pokiaľ je operátor aplikovaný, plánovač zvolí túto cestu a hľadá ďalší postup. Plánom sa potom stane cesta v prehľadávacom priestore. Tento spôsob prehľadávania implementuje niekoľko jednoduchých metód, z ktorých fungovania vychádza jedna zo skúmaných metód a to STRIPS. Je potrebné poznamenať, že vzniknutý graf je len pomocná

štruktúra, vďaka ktorej je možné zobraziť, akým spôsobom plánovač rieši úlohu a nie je hlavným cieľom metódy vytvoriť ho.

4.1.1 Typické prístupy - dopredné hľadanie a spätné hľadanie

Prvý intuitívny spôsob plánovania pomocou prehľadávania stavového priestoru, je hľadanie cesty od aktuálneho stavu k cieľovému. Ten bol historicky považovaný za nevýhodný a príliš nepraktický pre veľké úlohy, ktorými rozumieme úlohy s veľkým prehľadávacím priestorom. Jeden z dôvodov je problematika toho, ako zvoliť vhodnú akciu na vykonanie, najmä za predpokladu, že ich výber je veľký. Z toho vyplýva, že plánovače fungujúce týmto prístupom môžu častokrát skúmať riešenia, ktoré nie sú relevantné. Ďalším problémom je veľkosť stavového priestoru, ktorý je v prípade zdanlivo malých problémov rozvetvený do značných veľkostí.

Reprezentácia akcií v zadanej úlohe, ktorá bola opísaná v kapitole 3.6, umožňuje aj opačný prístup, kde sa od cieľového stavu plánovač snaží vrátiť k aktuálnemu stavu. Tento prístup sa nazýva spätné plánovanie a je možný len v prípade, ak je možné určiť čo predchádza zmenu stavu pomocou akcie a čo nasleduje po nej. Na to, aby tento prístup fungoval, je potrebná vhodná reprezentácia domény a úlohy. Pre plánovač je potrebné vedieť, aký stav musí predchádzať aplikovaniu operátora a aké zmeny budú v stave vykonané. Pokiaľ sú tieto informácie pre plánovač zadané, je schopný spätne prehľadávať priestor. Táto podkapitola stavia na publikácii [24].

4.2 STRIPS

Plánovacia metóda STRIPS, STanford Research Institute Problem Solver, je prvou z metód, ktoré budú v tejto práci skúmané. Táto metóda vznikla v roku 1971 na Stanford Research Institute, podľa ktorého bola aj pomenovaná. Vytvorili ju R.E. Fikes a N.J. Nilsson, pričom funguje nad modelom sveta reprezentovaným logikou prvého rádu. Modelom sveta môžeme rozumieť doménu, v ktorej sa plánovanie odohráva a zároveň stav sveta v zadanej úlohe. Ide o jednu z prvých plánovacích metód, ktorej základ je dodnes využívaný a vylepšovaný pre implementáciu rôznych plánovačov. [6]

STRIPS je jeden z prvých pokusov o optimalizáciu prístupu spätného plánovania. Táto optimalizácia je vytvorená zmenšením prehľadávaného stavového priestoru, vďaka čomu sa plánovač vyhne skúšaniu nefunkčných riešení. [10]

Zadávanie problému pre STRIPS prebieha spôsobom opísaným v kapitole 3.6. Ako už bolo spomenuté, spolu so systémom STRIPS, bol s rovnakým názvom zavedený aj spôsob popísania problému. Na jeho základe stavia reprezentácia PDDL [7], využitá pre zadanie problémov pre metódy v tejto práci. STRIPS pre svoje fungovanie využíva menej informácií, než poskytuje PDDL. Ako základ fungovania vyžaduje táto metóda počiatočný stav systému, cieľový stav a nakoniec operátory v tejto doméne, ako uvádza zdroj[17]. Popis operátorov pozostáva zo vstupných podmienok aktuálneho stavu a zoznamu stavov, ktoré sa z aktuálneho stavu vymažú a zoznamu stavov, ktoré do aktuálneho stavu pribudnú. V rámci PDDL reprezentácie sa tieto zoznamy spájajú do jedného, ktorý sa nazýva efekt na stav systému.

4.2.1 Priebeh metódy STRIPS

Fungovanie metódy sa dá principiálne popísať ako snaha transformovať aktuálny stav systému na cieľový, pomocou aplikovania plánovacích operátorov 3.4. Nájdené operátory vytvoria plán, pomocou ktorého bude možné vyriešiť zadaný problém. Plánovač pracuje spätným prehľadávaním, kedy postupným pridávaním operátorov mení stav systému, až sa prepracuje od cieľového stavu k pôvodnému stavu systému.

Popis fungovania metódy je nasledovný a opiera sa o publikáciu [6]:

1. Vytvorí sa prázdny zoznam, ktorý reprezentuje plán a vytvorí sa zoznam aktuálneho stavu.

2. Vytvorení sa zásobník, do ktorého sa vloží cieľový stav po jednotlivých predikátoch.

3. Z vrchu zásobníka je odobratý prvok a je vyhodnotený. Pokiaľ sa jedná o predikát a nachádza sa v aktuálnom stave, je zo zásobníka vyradený, pretože je splnený. Vo chvíli, keď je zásobník prázdny, je úloha vyriešená a prehľadávanie je ukončené.

4. Keď ide o konjunkciu predikátov, je potrebné ich rozloženie a sú vložené na vrch zásobníka namiesto pôvodného zápisu.

5. Pokiaľ sa zvolený predikát nenachádza v aktuálnom stave, vyberie sa operátor, ktorého efekt na systém obsahuje nesplnený predikát a nahradí v zásobníku pôvodný predikát. Operátor je vložený na vrch zásobníka a spolu s ním aj jeho vstupné podmienky, ktoré musia byť splnené.

6. V prípade, že je na vrchu zásobníka operátor, je odstránený, vloží sa do zoznamu plánu a v aktuálnom stave sa vykonajú zmeny podľa efektu operátora.

7. Pokiaľ nie je zásobník prázdny, postup sa opakuje od kroku 3.

Vo chvíli, kedy sa plánovač snaží vyriešiť nejaký podcieľ, stáva sa prioritným a k pôvodnému cieľu sa vráti až keď je podcieľ splnený. [10] Pokiaľ sú predikáty splnené, zvolený operátor je vložený do finálneho plánu a predikáty v aktuálnom stave sú upravené v závislosti od efektu na stav, ktorý bol pre daný operátor definovaný. Pre aktuálny stav to znamená, že predikáty, ktoré majú nadobudnúť platnosť sú pridané a naopak tie, ktoré prestanú platiť sú odstránené. Po tom, ako sú splnené predikáty podcieľa, vráti sa opäť k riešeniu pôvodného cieľa a postupuje opäť rovnakým spôsobom.

Keď plánovač vyhodnotí operátor ako aplikovateľný, záväzne ho prevezme ako časť plánu a nevracia sa späť v snahe skúšať iný operátor. Vďaka tejto vlastnosti je zredukovaný prehľadávaný priestor, ale takisto spôsobuje neúplnosť tejto metódy, keďže môže prísť o riešenie úlohy, aj napriek tomu že existuje. [10]

Je možné, že plánovač nevie nájsť operátor, pomocou ktorého by bolo možné dosiahnuť potrebný predikát. Vtedy sa hľadanie riešenia ukončí neúspešne a plánovač nemôže vrátiť žiadny funkčný plán. V tom prípade vyhlási, že nevie nájsť plán, ktorý by vedel dosiahnuť cieľový stav z aktuálneho. Pokiaľ tomu tak nie je a plánovač úspešne splní všetky predikáty z cieľového stavu pomocou postupnosti operátorov a tak odstráni rozdiel medzi aktuálnym a cieľovým stavom, našiel riešenie úlohy. To vráti prostredníctvom postupnosti aplikovaných akcií, ktoré počas svojho hľadania ukladal. [10]

4.3 Plánovanie pomocou plánovacieho grafu

Táto podkapitola bola písaná na podklade zdrojov [10] a [23]. Iným prístupom k plánovaniu je prehľadávanie plánovacieho priestoru. Motiváciou vytvorenia efektívnejšej plánovacej metódy bolo vetvenie prehľadávacieho priestoru, ktoré môže narásť do obrovských rozmerov.

Tento prístup opäť využíva prehľadávanie priestoru, ale jeho vylepšenie spočíva v snahe vytvoriť lepší a prepracovanejší graf, ktorý poskytuje lepší priestor pre rozhodovanie plánovacej metódy. Jeho tvorba je odlišná od toho, akým spôsobom bol tvorený graf pri plánovaní v stavovom priestore. Plánovanie začína od uzlu reprezentujúceho prázdny plán. Postupnou tvorbou grafu sa snaží nájsť uzol so zoznamom predikátov, ktoré sú zhodné s úlohou zadaným cieľom. V tomto prístupe plánovania graf nie je len vizualizačnou pomôckou, ale stáva sa dôležitou časťou celkového plánovacieho procesu. Jeho štruktúra a obsah sú prínosné prvky, ktoré zlepšia a zefektívnia proces plánovania. Jednotlivé plánovacie algoritmy využívajúce tento prístup môžu voliť rôzne prístupy alebo obmeny plánovacieho grafu pre ich efektívne fungovanie.

Výsledný plán, ktorý je nájdený, je odlišný od výsledku, ktorý vrátil STRIPS. V prípade plánovacieho priestoru nie je nájdený úplne usporiadaný, lineárny plán. Plánovací priestor definuje výsledný plán voľnejšou štruktúrou. Ide o čiastočne usporiadaný plán, ktorého niektoré časti majú presne určenú postupnosť a niektoré časti môžu byť usporiadané ľubovoľne.

Pri tvorbe grafu sa plánovač nezameriava na prísne obmedzenia špecifikované úlohou. Jeho snahou je uvoľniť niektoré z týchto pravidiel, ktoré ho zväzujú pri tvorbe grafu a vytvára priestor, ktorý bude o niečo rozsiahlejší. Toto uvoľnenie obmedzení znamená, že vytvorenie grafu je jednoduchšie, pretože nemusí spĺňať veľa podmienok. Z toho vyplýva, že bude väčší, ale vždy bude obsahovať riešenie prísnejšie špecifikovaného problému. Uvoľnením vybraných obmedzení úlohy je možné predísť zdĺhavému výberu vhodných operátorov vo chvíli, kedy to nie je nevyhnutne potrebné pre proces.

Ďalšou odlišnosťou tejto metódy je rozdelenie plánovacieho priestoru na podplány, tak ako už bolo spomenuté. Plánovač bude jednotlivé podplány brať ako samostatné problémy, ktoré je potrebné vyriešiť nezávisle od ostatných.

V tomto prístupe je celkový proces plánovania rozdelený na dve časti: v prvom kroku ide o zvolenie jednotlivých akcií, v nasledujúcom usporadúva akcie do takého poradia, aby splnili zadaný cieľ.

4.3.1 Plánovací graf

Prvok nevyhnutný pre prácu v plánovacom priestore, je samotný plánovací graf. Jeho štruktúra a spôsob jeho tvorby sú dôležité pre efektivitu plánovania v ňom a je silným nástrojom pre zlepšenie fungovania plánovacej metódy.

Vytvorenie grafu je prvým krokom metód pracujúcich v plánovacom priestore. Bude reprezentovať prehľadávací priestor pre uvoľnenú verziu zadanej úlohy [10]. Jeho základnou myšlienkou je skúmať, aké stavy sú dosiahnuteľné v jednotlivých fázach plánovania.

Proces tvorby tohto stromu je nasledovný: začne v počiatočnom uzle, ktorý reprezentuje nejaký stav a pokúša sa zistiť, ktoré stavy sú z aktuálnej polohy v strome dosiahnuteľné na základe nejakých podmienok. Prehľadávanie je vykonávané vždy do vopred určenej hĺbky, pričom počiatočná hĺbka začína na hodnote 1 a počíta sa od koreňového uzla. To znamená, že hľadá práve jeden nasledujúci stav, ktorý vie dosiahnuť zo zadaného bodu. Pri pre-

hlbovaní grafu dochádza k hľadaniu nasledovníkov stavov nájdených v predchádzajúcich hĺbkach. [23]

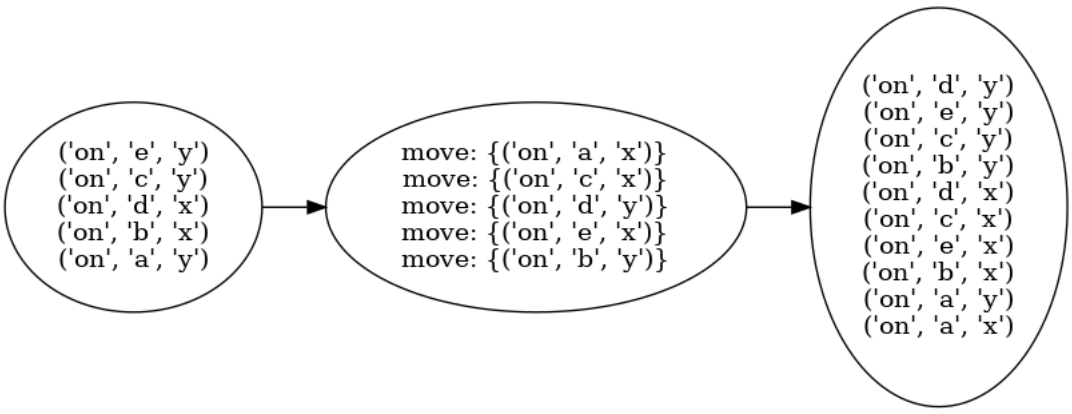
Vzhľadom na to, že pri tomto prístupe môže byť vytvorený prehľadávací priestor obrovských rozmerov, je potrebné zakomponovať vyššie uvedený uvoľnený problém 4.3. To je vykonané tým, že jednotlivé hĺbky - levely, úrovne - sú brané ako jeden stav pri prehlbovaní prehľadávania. Následne je možné si strom predstaviť tak, akoby na každej úrovni mal práve jeden uzol. Ten je zložený zo všetkých stavov nájdených v danej hĺbke. Zjednocovanie všetkých stavov jednotlivých úrovní pomáha značne redukovať veľkosť vznikajúceho prehľadávacieho grafu.

Stavy na úrovniach sú hľadané pomocou aplikovania všetkých možných akcií, ktoré spĺňajú aktuálne podmienky, ale nie je kontrolovaná ich vzájomná kompatibilita. Problém, ktorý môže nastať pri aplikovaní akcií bez kontroly ich vzájomných vplyvov na seba, je nezlučiteľná platnosť niektorých predikátov, z ktorých sa skladajú stavy na úrovniach. To znamená, že po vykonaní dvoch rôznych akcii sa môže jeden predikát stať zároveň platným a zároveň neplatným. [23]

Týmto prístupom teda vznikajú konflikty a nezrovnalosti v grafe, ktoré je následne potrebné odstrániť a predísť tak nesprávnym riešeniam. Bez dodatočnej úpravy tohto grafu budú vznikať neplatné a nesprávne plány.

Na vyriešenie týchto konfliktov je do grafu zavedený kontrolný prvok mutex - mutual exclusion, alebo vzájomné vylúčenie - ktorý bude riešiť vzniknuté konfliktné situácie. Mutex bude mať dva rôzne druhy - pôjde o mutex pre predikáty a pre akcie. Bude sa využívať na každej úrovni osobitne a bude ukazovať, ktoré z prvkov nachádzajúcich sa na danej úrovni nemôžu platiť súčasne, alebo nemôžu byť vykonané súčasne. Mutex je reprezentovaný zoznamom dvojíc, ktoré nemôžu byť súčasne platné. [3]

Úrovne v grafe majú takisto dva druhy, ktoré sa pravidelne striedajú, ako je možné vidieť na obrázku 4.1. Na prvej úrovni sa budú ukladať informácie o predikátoch, ktoré tu existujú a na nasledujúcej úrovni sa budú nachádzať informácie o akciách. Na počiatočnej úrovni grafu, kde je definovaný počiatočný stav systému, sa nachádzajú platné predikáty.



Obr. 4.1: Plánovací graf so striedajúcimi sa úrovňami predikátových a operátorových vrstiev, ktorý reprezentuje riešenie úlohy `Simple Blocks`.

4.3.2 Proces tvorby plánovacieho grafu

Popis procesu čerpá zo zdrojov [23] a [3]. Priebeh tvorby plánovacieho grafu pre ľubovoľný zadaný problém začína v koreňovom uzle, ktorý reprezentuje počiatočný stav systému. Na tejto úrovni bude pomocou jednotlivých predikátov reprezentovaný stav, ktorý je na nej platný. V nasledujúcom kroku sa bude tvoriť ďalšia úroveň, obsahujúca akcie aplikovateľné v danom kroku. Úroveň bude reprezentovaná pomocou akcií, ktoré na príslušnej úrovni majú splnené vstupné podmienky a je možné ich aplikovať. Aplikovateľné akcie sú filtrované pomocou vstupných podmienok a pomocou predikátových mutexov, pre ktoré musí platiť, že žiadna dvojica predikátov vstupných podmienok akcie sa nesmie nachádzať v mutexe. Budú sa tu nachádzať všetky aplikovateľné obmeny zvolených operátorov s objektmi z predikátov na predchádzajúcej vrstve.

Po vytvorení prvých dvoch úrovní je potrebné vytvoriť nasledujúce mutexy, ktoré upravia práve vzniknuté časti grafu. [3]

- Ako prvý, sa vytvorí mutex pre akcie. Vzájomne sa vylučujú akcie, ktoré navzájom rušia svoje výstupné podmienky. Teda nejaká dvojica predikátov z dvoch akcii má vzájomne opačnú platnosť po jej vykonaní. Akcie sa navzájom rušia aj pokiaľ majú konfliktné vstupné podmienky, ktoré nemôžu byť súčasne splnené. Tieto pravidlá nezaručujú nájdenie všetkých konfliktných akcií, ale sú schopné predísť ich väčšine.

- V ďalšom kroku je vytvorený mutex, ktorý bude platný pre predikáty na nasledujúcej úrovni. Dvojica predikátov sa navzájom zruší v prípade, že neexistuje akcia na predchádzajúcej úrovni, ktorá je schopná obidva predikáty vytvoriť a zároveň, ak všetky dvojice akcií na predchádzajúcej úrovni, ktoré tieto predikáty produkujú ako platné, sú navzájom vylúčené v mutexe.

Po vytvorení zoznamov mutexov sa proces opakuje a na ďalšej úrovni grafu sa opäť budú vyskytovať platné predikáty. Týmto procesom sa postupným prehlbovaním a opakovaním postupu vytvára prehľadávací graf. Graf sa prestane tvoriť v momente ako úroveň predikátov obsahuje rovnaké predikáty ako cieľový stav zadanej úlohy.

Posledný prvok, ktorý sa v plánovacom grafe vyskytuje, je operácia bez následkov, často označovaná `NoOp` operátor. Ide o akcie, ktoré sú vložené do grafu, aby v ňom nevznikali prázdne miesta. Tieto operátory neaplikujú žiadnu zmenu v systéme a neovplyvňujú stav predikátov. Ich úlohou v grafe je zaplniť miesta, kde nie je potrebné aplikovanie žiadneho operátora, pretože nejaká požiadavka na predikát bola splnená vo vyššej vrstve grafu, ktorý ale ďalej hľadá riešenie ostatných predikátov. Ide o prvok, ktorý pomáha efektivite algoritmu neskôr využívajúci tento graf.

4.4 Historické opodstatnenie metódy Graphplan

Graphplan je prvým plánovačom využívajúcim techniku plánovacieho grafu. Bol vyvinutý v roku 1995 a jej autormi sú Avrim Blum a Merrick Furst. Plánovače, ktoré v danej dobe boli vyvíjané a používané vychádzali z prístupu podobnému metóde STRIPS alebo s prístupom čiastočného usporiadania. Primárnou motiváciou hľadania metódy s iným prístupom bola linearita STRIPS, ktorá spôsobuje možné rušenie už dosiahnutých cieľov opísaných v kapitole 4.2.1. Tento prístup sa snaží o rozkladanie plánovacieho priestoru na menšie časti, ktoré sa pokúša vyriešiť samostatne, aby vytvoril celistvé riešenie. Využívaním plánovacích grafov sa podarilo metóde Graphplan značne zrýchliť proces hľadania plánu. Na základe tejto

metódy stavali aj ďalšie spôsoby plánovania, ktoré buď imitovali časti fungovania, alebo sa snažili zakomponovať plánovací graf. Graphplan bol podstatný pre vývoj ďalších plánovacích metód, menovite boli na jeho základe ďalej rozvíjané heuristiky dosiahnuteľnosti (reachability heuristics), ktoré boli využívané v ďalších plánovačoch. Od jeho základného fungovania, ktoré je predstavené v tejto práci, bolo vyvinutých niekoľko už aj vyššie spomenutých spôsobov vylepšenia tejto metódy. Z toho je možné usúdiť, že nielen metóda ako taká bola úspešným príspevkom do oblasti plánovania, ale jej myšlienky pomohli ďalšiemu rozvoju v tejto problematike. Odsek bol prevzatý a parafrázovaný z [3].

4.4.1 Priebeh metódy Graphplan

Druhou metódou skúmanou v tejto práci je Graphplan. Ide o metódu využívajúcu práve plánovací graf na svoje fungovanie, v ktorom následne hľadá validný plán.

Validný plán je v prípade plánovania v plánovacom grafe definovaný množinami akcií, ktoré majú byť vykonané, a čas, v ktorom majú byť vykonané. To znamená, že jednotlivé množiny akcií majú určené poradie, v ktorom sa majú vykonať [3]. Pre príklad, validný plán bude vytvorený z niekoľkých množín akcií, pričom ako prvé sú vykonané akcie prvej množiny, následne akcie druhej množiny, a tak ďalej. Jednotlivé akcie nemajú presne určenú postupnosť, ako tomu bolo pri prehľadávaní stavového priestoru, ale sú združené do skupín, ktoré majú pridelené poradie. Akcie v rámci množiny môžu byť vykonané v ľubovoľnom poradí, pokým nepríde k ich vzájomnému konfliktu, ktorý sa rieši pri tvorení grafu. Preto vo výsledku pôjde o plán s čiastočným usporiadaním akcií.

Priebeh metódy Graphplan je popísaný na základe publikácie [3] a začína sa vytvorením plánovacieho grafu. Postup je taký istý, ako bol popísaný v kapitole 4.3.2 s jedným krokom navyše. S postupným prehlbovaním grafu je potrebné hľadať samotný plán, čo znamená, že je potrebné kontrolovať, či graf dosiahol cieľový stav zadaný úlohou. Preto vždy po pridaní vrstvy akcií a nasledujúcej vrstvy predikátov plánovač skontroluje aktuálny stav predikátov a porovná ho s cieľovým stavom. Z toho vyplýva, že sa snaží nájsť výsledný plán v aktuálne najnovšej vrstve grafu, ktorá bola práve vytvorená. Vďaka tomuto prístupu je proces tvorby grafu zastavený v momente, keď je nájdené riešenie. Pokiaľ je nájdená zhoda medzi aktuálnym a cieľovým stavom, metóda prejde do fázy vrátenia platného plánu. V opačnom prípade povie, že plán na zadanú úlohu nie je možné nájsť v aktuálnej vrstve, alebo vo vyššej vrstve. Vtedy je potrebné prehĺbenie plánovacieho grafu, aby bolo možné ďalej hľadať riešenie. Tento proces je opakovaný pokiaľ nie je nájdený plán, alebo graf dosiahol maximálny povolený počet vrstiev. Toto opatrenie je potrebné pre zastavenie hľadania v prípade, že nie je schopný nájsť riešenie úlohy.

Algoritmický priebeh fungovania metódy [3]:

1. Plánovací graf je vytvorený s jednou úrovňou, do ktorej je vložený aktuálny stav vo forme jednotlivých predikátov. Takisto je vytvorený zoznam cieľov, ktorý obsahuje požadovaný cieľový stav.

2. Pokiaľ aktuálne najnovšia vrstva grafu spĺňa cieľový stav, začne prehľadávanie grafu za účelom nájdenia plánu v bode 4. V prípade, že cieľ splnený nie je, v danej hĺbke grafu nie je možné nájsť riešenie problému a musí dojsť k jeho expanzii.

3. Pokiaľ je potrebné, je vykonané prehĺbenie plánovacieho grafu, pričom sú vytvorené obe vrstvy, jedna s operátormi a jedna s predikátmi 4.3.1. Následne sú skontrolované a vytvorené mutex vzťahy 4.3.2 a fungovanie sa vracia k bodu 2.

4. Keď je nájdená zhoda na aktuálnej úrovni grafu a v cieľovom stave, dochádza k spätnému prehľadaniu grafu. Ten je prechádzaný od najnovšej vrstvy, pričom na každej predchádzajúcej vrstve operátorov hľadá tie, ktoré dosiahli pridanie cieľových predikátov.

Vo chvíli, ako metóda vyhodnotí, že splnila požadovaný cieľový stav, prechádza do fázy hľadania plánu, pomocou ktorého je možné prejsť z počiatočného stavu do cieľového. Graphplan hľadanie plánu vykonáva spätným prehľadávaním výsledného grafu. Začína od najnižšej vrstvy, kde bol dosiahnutý cieľový stav a prostredníctvom spätného hľadania skúma, ktoré akcie je potrebné aplikovať v pláne. Rekurzívne hľadá, ktoré akcie boli použité pre splnenie predikátov na aktuálnej vrstve a následne pomocou ich vstupných podmienok vytvorí podcieľ, pre ktorý hľadá nasledujúce akcie v ďalšej vrstve v poradí. Pokiaľ zistí, že už nevie spätne nájsť ďalšie aplikovateľné akcie, vráti sa a skúša hľadať pomocou inej aplikovateľnej akcie na danej vrstve. Tento proces sa opakuje, pokým nenarazí na počiatočnú vrstvu predikátov zadaných úlohou v koreňovom uzle.

4.5 Hierarchické plánovanie

Predchádzajúce metódy, STRIPS a Graphplan, využívali prístup a myšlienky klasického plánovania. Hierarchické plánovanie sa zoberá iným druhom plánovania a riešenia plánovacích úloh. Ako názov napovedá, dôležitým prvkom pre tento plánovací prístup bude hierarchia. Základné fungovanie tejto metódy využíva zadaný aktuálny stav systému, abstraktne zadanú úlohu, ktorú má plánovač splniť a nakoniec hierarchicky usporiadanú sieť operátorov - pomocou ktorých je možné meniť stav systému. Prístup hierarchického plánovania, z angličtiny Hierarchical Task Network (HTN) Planning, je prístup, ktorý je podobne starý ako predchádzajúce prístupy. Jeho počiatky je možné nájsť v roku 1975 v podobe plánovača Nets of Action Hierarchies (NOAH) [25], ktorý ako prvý predstavil myšlienky hierarchického plánovania. Tie stavajú na prístupe HTN a na tomto prístupe implementujú svoje fungovanie. Pojmom HTN sa rozumie prístup k plánovaniu, nie samotná metóda, ktorá by bola schopná riešiť konkrétne úlohy. Na definícii HTN plánovania stavajú plánovače, ktoré sa snažia o implementovanie jeho myšlienok. Aktuálne je jeden zo spôsobov HTN plánovania prístup plánovača SHOP. V angličtine sa tento plánovač nazýva Simple Hierachical Ordered Planner (SHOP) [20], vytvorený bol v 90-tych rokoch minulého storočia, pričom vznik dopomohol k rozšíreniu HTN plánovania. Ide o prístup hierarchického plánovania v jednoduchom algoritme, ktorý je schopný rýchlo riešiť aj komplexné úlohy, na úkor podrobne špecifikovanej domény. Pre túto prácu bol zvolený práve prístup SHOP, ktorý dobre reprezentuje schopnosti a obmedzenia HTN plánovania.

Hierarchický spôsob plánovania vychádza z ľudského prístupu k riešeniu úloh. Pokiaľ má niekto problém, ktorý musí vyriešiť, napríklad sa potrebuje dopraviť z domu na letisko, jeho prístup k riešeniu tohto problému bude pravdepodobne rozložiť ho na menšie konkrétne časti úlohy. Napríklad sa rozhodne, že na letisko sa dopraví pomocou taxi služby, preto musí nejaké vozidlo zavolať. Nastúpi do auta, ktoré ho následne odvezie k letisku a nakoniec zaplatí za dopravu. Fungovanie plánovača teda môže byť zjednodušene chápané tak, že sa snaží komplexnú úlohu rozložiť na jednoduchšie kroky, ktoré rozkladá dovtedy, kým sa dostane na krok, ktorý už môže samostatne vykonať. Myšlienka tejto metódy vychádza z predpokladu, že zložité plány majú často nejakú štruktúru. Túto štruktúru je možné vytvoriť pomocou hierarchie úloh, ktoré sú rozkladané na stále menej zložité časti na takzvané podplány. Ďalší využívaný predpoklad je, že podplány sú od seba veľmi často

nezávislé. Nie je tomu tak vo všetkých prípadoch, ale vo veľkom množstve plánovacích úloh to platí. Príklad takého nájdeného plánu je možné vidieť v niektorých testovaných úlohách, napríklad v časti 6.4.3. Príklad opísaný v tomto odstavci bol inšpirovaný z [10].

Predchádzajúce prístupy využívali na hľadanie problému algoritmické princípy, kedy skúšali aplikovať možné cesty, ktoré by mohli poskytnúť riešenie. Každý na to využil trochu odlišný postup, ale ich fungovanie sa do istej miery dá prirovnať k skúšaniu hrubou silou [9], i keď využívajú prvky ktoré umožňujú zúžiť výber potenciálnych riešení pred ich skúšaním. Ako bolo už vyššie načrtnuté, iným prístupom môže byť riešenie podobné tomu ľudskému, kedy sa využijú znalosti o doméne na riešenie problému.

HTN pre svoje fungovanie potrebuje podrobnejšie definovanie domény, v ktorej bude pracovať, ako predchádzajúce metódy, ako uvádza zdroj [8]. Aj bez ďalšieho definovania jeho fungovania je možné vidieť, že na to, aby bolo možné určiť akým spôsobom sa môžu dekomponovať väčšie, abstraktné úlohy (ako napríklad dopravenie sa na letisko), je potrebné vedieť toho o doméne viac a podrobnejšie, ako pri predchádzajúcom definovaní operátorov a predikátov. Nestačí popísať, aké operátory je možné využiť a za akých podmienok. V HTN musia byť navyše zakomponované do celého obrazu hierarchie plánov, akým spôsobom do nej zapadajú a na akých iných krokoch sú závislé. Každý klasický problém plánovania je možné transformovať do potrebného zápisu a riešiť ho týmto prístupom. [10]

Je možné vidieť, že doména cestovania [10], ktorá bola vyššie opísaná je ideálnym príkladom využitia hierarchického plánovania. Zložené operátory predstavujú rôzne možnosti cestovania, pričom jednoduché operátory potom predstavujú veľmi konkrétne úlohy, ktoré je možné jednotlivo vykonať. Takýto typ úlohy je typickým príkladom tohto spôsobu plánovania, a je možné vidieť, že definícia takejto domény možná je aj pre predchádzajúce metódy. Avšak takáto úloha nie je pre ne príliš vhodná, keďže v tomto prípade ide zjavne o hľadanie najjednoduchšej postupnosti krokov a nie o zmenu stavu systému. Úlohy typické pre hierarchické plánovanie je možné definovať aj pre metódy s iným prístupom, čím sa zaoberá publikácia [16]. V tomto prípade sa však nejedná o vhodné úlohy, čo sa spôsobu ich riešenia týka.

4.6 HTN plánovanie

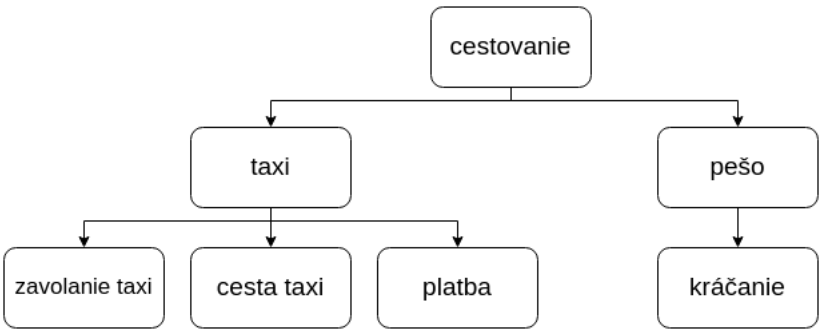
Základ štruktúry, pomocou ktorej boli definované stavy a operátory doteraz, platia aj vo fungovaní HTN plánovania. Popisovanie stavov a vzťahov v doméne pomocou predikátov ostáva rovnaké. Predikáty pomáhajú definovať aktuálny a cieľový stav systému. Rozšírením pôvodnej definície bude pridanie nových druhov operátorov, ktoré umožňujú vytváranie hierarchickej štruktúry [16]. Informáciu, ktorú plánovač dostáva navyše, je možné chápať ako zložený operátor, taktiež niekedy nazývaný metóda, ktorý definuje akým spôsobom je možné abstraktnú úlohu rozložiť na podúlohy. Operátory využívané v tejto metóde rozdeľujeme na dva druhy podľa publikácie[10]. Ten prvý je primitívny operátor, ktorý je rovnaký a definovaný zhodne ako operátory v oboch predchádzajúcich metódach. Definuje, za akých okolností môže dojsť k nejakej konkrétnej zmene stavu systému, pomocou vstupných podmienok, výstupných podmienok a parametrov, ktoré využíva. Nižšie je možné vidieť jeho príklad, ktorý reprezentuje operátory z obrázku 4.2. Tento typ operátora nie je ďalej rozložený na jednoduchší, ale aplikuje sa, a tak ako v predchádzajúcich úlohách vytvorí zmeny systému.

```
taxi_ride(x)
task: taxi_journey(x)
```

```
preconditions: taxi_called(x)
subtasks: none
```

Druhý typ je operátor zložený alebo neprimitívny, ktorý popisuje abstraktnú úlohu. Nejedná sa o konkrétnu zmenu systému, ktorá môže byť v plánovači aplikovaná a vykoná priamu transformáciu stavu. Namiesto toho ide o definovanie abstraktnej úlohy reprezentujúcej skupinu podúloh, ktoré popisujú jej význam a činnosť. Ide o zoznam niekoľkých primitívnych operátorov, ktoré spolu môžu byť definované ako väčšia úloha, ktorá by mala byť splnená 4.2. Operátor má takisto svoje vstupné podmienky, ktoré musia byť splnené na to, aby mohol byť aplikovaný. Jeho výstupnými podmienkami môžu byť chápané primitívne operátory, z ktorých je zložený [9]. Na príklade v časti `subtasks:` je možné vidieť na aké podúlohy operátor `taxi_journey(x)` môže byť rozložený. Medzi nimi sa vyskytuje aj vyššie definovaný jednoduchý operátor `taxi_journey(x)`.

```
transfer2(x)
task: travel_by_taxi(x)
preconditions: none(x)
subtasks: call_taxi(x), taxi_journey(x), pay(x)
```



Obr. 4.2: Príklad rozkladu operátorov na príklade domény presunu

Zložený operátor môže mať viacero spôsobov, ako môže byť rozložený na primitívne operátory [16]. To znamená, že je tu takisto možnosť skúšať rôzne spôsoby, ako rozložiť abstraktnú úlohu, podľa toho, čo bude v danom prípade najviac vyhovujúce. Napríklad pri operátore, ktorý definuje transport, môže byť jeho rozloženie buď chôdza, alebo cestovanie taxi službou. Je možné, aby sa zložený operátor rozložil do ďalšieho zloženého operátora. V tomto konkrétnom príklade sa transport pomocou taxi môže ďalej rozložiť na zavolanie taxi, cestovanie a platbu. Táto dekompozícia zložených operátorov na primitívne tvorí prvok, podľa ktorého je pomenovaná HTN metóda, a to sieť úloh. Tento prvok zostavuje hierarchiu, ktorou je riadené usporiadanie operátorov v doméne.

```
move1(x)
task: move_one_block(block, source, dest)
preconditions: none
subtasks: get_block(block), put_block(block)
```

Iný príklad jednoduchých a zložených operátorov je využitý aj ako testovacia úloha 6.4.6. Je tu možné vidieť zložený operátor presunu kocky. Na jej presun je potrebné využiť

ďalšie dva operátory a to `get_block()` a `put_block()`. To znamená, že kocku je potrebné zobrať z jej aktuálnej polohy a niekam ju presunúť. Tieto operátory sú ďalej rozložiteľné.

```
put1(x)
task: put_block(block, source, dest)
preconditions: is_table(dest)
subtasks: putdown(block, dest)

put2(x)
task: put_block(block, source, dest)
preconditions: is_block(dest)
subtasks: stack(block, dest)
```

Príklad zloženého operátora `put_block()` ukazuje, že je možné jeho rozloženie dvoma rôznymi spôsobmi, v závislosti od aktuálnej polohy kocky, ktorá je kontrolovaná v podmienkach `predconditions:`. Plánovač sa teda pri tomto operátore musí rozhodnúť, ktorá z možností je v danej chvíli vhodná.

```
put_down(x)
task: putdown(block, dest)
preconditions: holding(block)
subtasks: none
effects: on(block, dest)
```

Jednoduchý operátor `putdown()` vykonáva priamu zmenu systému a nie je rozložený na podúlohy. Ide teda o klasický operátor, aký sa vyskytoval aj v prechádzajúcich úlohách a je možné ho definovať aj pre predchádzajúce metódy.

Aj pri tomto prístupe existujú nejaké obmedzenia, ktoré pomáhajú usmerňovať plánovací proces. Okrem vstupných podmienok primitívnych a zložených operátorov je potrebné hľadieť na to, akým spôsobom budú usporiadané. Je potrebné kontrolovať, v akom poradí môžu byť vykonané jednotlivé operátory. Podobne ako pri metóde Graphplan, je potrebné kontrolovať vzájomné vylúčenie operátorov. Teda či vykonanie jedného operátora nespôsobí znemožnenie aplikovanie iného operátora.

Plánovanie začína zadanou úlohou, ktorú má plánovač vyriešiť. Je potrebné, aby bol definovaný aktuálny stav systému pomocou predikátov a objektov vyskytujúcich sa v ňom. Tieto budú ďalej slúžiť na kontrolu vstupných podmienok operátorov. Zadané môžu byť aj viaceré úlohy, ktoré majú byť splnené, pričom úloha môže byť zadaná priamo nejakým zloženým operátorom, ktorý v danej doméne existuje. Riešenie úlohy prebieha tým, že sa plánovač snaží o postupné rozkladanie zadanej úlohy, až pokým nebude nájdené riešenie pomocou primitívnych operátorov, alebo dekompozícia nebude možná kvôli zadaným obmedzeniam. Pokiaľ nejaký zložený operátor nebol schopný dosiahnuť riešenie úlohy, plánovač sa vráti späť do bodu aplikovania tohto operátora a pokúša sa nájsť iný operátor, ktorý v danom bode môže byť aplikovaný. [16]

4.6.1 Priebeh metódy HTN

Priebeh plánovania je popísaný na základe [10]. Plánovač začne od zadanej úlohy, kde do plánu vloží operátory, ktorými bola definovaná zadaná úloha. Pokiaľ plán obsahuje iba primitívne úlohy - operátory - našiel riešenie pre zadaný problém. V opačnom prípade zvolí

jednu zo zložených úloh v pláne a rozloží túto úlohu podľa jednej z definícii na podúlohy. Skontroluje podmienky a obmedzenia stanovené podúlohami. Pokiaľ rozklad nie je možný, vráti sa do predchádzajúcej definície úlohy. Následne sa proces vráti do plánu a znova hľadá zloženú úlohu, ktorú by mohol nahradiť.

Podobne, ako iné plánovacie metódy, aj pre HTN existuje množstvo rozšírení, ktoré sa pokúšajú o vylepšenie jeho fungovania. Ide napríklad o predchádzanie hrozbám konfliktov, pridávanie podmienok, heuristík a podobne. Pre porovnanie plánovacích metód je zámerom porovnať základnú formu metódy, tak ako to bolo v prípade predchádzajúcich metód.

V prípade plánovača SHOP ide o konkrétnu implementáciu HTN plánovača, ktorá bola predstavená na Marylandskej univerzite [20]. Zavedenie tohto prístupu hierarchického plánovania viedlo k vytvoreniu ďalších plánovacích metód založených na tomto prístupe. Zároveň od tejto metódy boli odvodené vylepšené prístupy, ktoré boli schopné zúčastňovať sa plánovacích súťaží a konkurovať v nich napríklad SHOP2 [1] a SHOP3 [21]. Pre túto prácu bol zvolený základný prístup tejto metódy, teda SHOP. Ide o jednoduchú formu hierarchického plánovania, ktorá funguje spôsobom dopredného vyhľadávania, teda plánuje v tom istom poradí, ako bude vykonávaný finálny plán.

Algoritmický priebeh fungovania metódy [10]:

1. Zoznam, do ktorého je vložená zadaná úloha je vytvorený.

2. Zo zoznamu je zvolený jeden z operátorov reprezentujúcich zadanú úlohu. Pokiaľ ide o zložený operátor, sú nájdené aplikovateľné rozloženia tohto operátora. Pokiaľ žiadne neexistujú, úlohu nie je možné ďalej zjednodušiť a teda nie je možné nájsť možný plán riešenia. V opačnom prípade je zvolené jedno z rozložení, ktoré je vložené do zoznamu úlohy namiesto pôvodného zloženého operátora.

3. Pokiaľ sa jedná o jednoduchý operátor, sú nájdené spôsoby jeho aplikovania, ak žiadne neexistujú, nie je možné týmto spôsobom riešiť úlohu. V opačnom prípade je do zoznamu zaradená nájdená akcia namiesto jednoduchého operátora.

4. Keď sa v zozname nachádzajú iba akcie, hľadanie je ukončené s plánom nájdeným v tomto zozname.

Kapitola 5

Implementácia klasických metód plánovania

Metódy, ktoré boli zvolené na porovnávanie sú často používané a zakomponované do rôznych projektov, preto boli zvolené už existujúce implementácie, ktoré boli testované. Porovnávané budú tri predstavené metódy, ktoré implementujú rôzne prístupy k plánovaniu. V tejto kapitole budú popísané jednotlivé implementácie metód, zadávanie vstupov a zaujímavé časti implementácie, ktoré je vhodné spomenúť.

5.1 Závislosť plánovačov na doméne

Pre implementáciu plánovača je potrebné v prvom rade zvoliť plánovaciu metódu, ktorú bude využívať. Druhou veľmi dôležitou časťou implementácie je spôsob, akým bude metóda implementovaná. Plánovač môže byť implementovaný troma rôznymi spôsobmi, ktoré sú uvedené v [10]. Tieto spôsoby sa od seba líšia v tom, akým spôsobom je program prispôsobený pre spracovanie vstupnej domény. Množstvo plánovačov je implementovaných pre fungovanie v rámci jednej konkrétnej domény. Sú prispôsobené priamo pre plánovanie v rámci nej a nie je možné ich priamo aplikovať v inej doméne. Tieto implementácie sa nazývajú doménovo závislé a ich výhoda spočíva v možnosti ich optimalizácie pre fungovanie s konkrétnymi operátormi a objektmi. Pochopiteľne, výhoda tohto prístupu spočíva v ideálnej implementácii s danou problematikou s čím súvisí aj zrejmá nevýhoda, pretože plánovač vtedy nie je možné bez úpravy použiť pre fungovanie v inej doméne. Väčšina plánovačov používaných v reálnom svete je implementovaná doménovo závislým spôsobom. Dôvodom je príliš vysoká náročnosť vytvorenia doménovo nezávislého plánovača, ktorý by mohol pracovať vo všetkých možných doménach s vysokou efektivitou. Zároveň je vtedy možné plánovač dobre optimalizovať pre konkrétne úlohy, ktoré bude riešiť, a tak zvýšiť efektivitu použitej metódy.

Druhý spôsob implementácie sú doménovo nastaviteľné plánovače, ktoré pracujú bez ohľadu na vstup, ale na ich fungovanie je neskôr potrebné doplniť informácie o konkrétnej doméne, v ktorej budú pracovať. Informácie navyše hovoria o spôsobe riešenia úloh v príslušnej doméne.

Plánovače v tejto práci boli všetky implementované tretím spôsobom - doménovo nezávislým - vzhľadom na to, že budú riešiť úlohy v niekoľkých rôznych doménach. V tomto prípade je vhodnejší práve tento prístup. Doménovo nezávislé plánovače sú implementované tak, aby boli schopné prijať vstup v ľubovoľnej doméne. Ich fungovanie nie je závislé

od konkrétnych akcií a objektov, ale všeobecne aplikuje kroky zvolenej plánovacej metódy alebo algoritmu na riešenie všeobecného problému.

5.2 Implementácia v jazyku Python

Programovací jazyk zvolený na implementáciu je Python. Vzhľadom na to, že sa jedná o objektovo orientovaný programovací jazyk, jeho vlastnosti ho robia vhodným na implementáciu týchto metód. Takisto sú výhodou využité knižnice umožňujúce vizualizáciu a zobrazovanie výsledkov pre jednoduchšie porovnanie fungovania metód, konkrétne ide o knižnice `plotly` a `pydot`.

5.3 Plánovač STRIPS

Implementácia plánovača STRIPS bola prevzatá od W. Tansey [26], pričom ide o celistvý program zabezpečujúci celý proces plánovania. Od spracovania vstupu, jeho prevedenia do potrebných dátových štruktúr, až po samotné plánovanie a vrátenie výstupu. Táto konkrétna implementácia ako vstup berie textový súbor vo formáte STRIPS. Vzhľadom na to, že súčasná reprezentácia v plánovaní v jazyku PDDL vychádza z tohto zápisu, bolo možné program upraviť tak, aby prijímal vstup práve v tomto jazyku. Takisto to bolo vhodné s ohľadom na ďalšej metódy, ktorá bude porovnávaná, pre ktorú bude vstup zhodný. Ich porovnanie bolo uľahčené testovaním na rovnakých vstupoch, pričom rovnaký formát vstupu zjednodušil vytváranie testovacích domén.

Príklad vstupu domény a úlohy vo formáte STRIPS:

```
Initial state: At(Table1), BlockAt(A, Table1), BlockAt(B, Table1),
BlockAt(C, Table1)
Goal state: On(A, B), On(B, C), OnTop(A)
Actions:
        Go(From, To)
        Preconditions: At(From), !At(To)
        Postconditions: At(To), !At(From)
```

Príklad podobného vstupu domény vo formáte PDDL:

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:predicates  (on ?b ?t))
  (:action move
     :parameters (?b ?t ?k)
     :precondition (and (on ?b ?t) (not (on ?b ?k)))
     :effect (and (on ?b ?k) (not (on ?b ?t)))))
```

Príklad podobného vstupu úlohy v príslušnej doméne vo formáte PDDL:

```
(define (problem move-blocks)
    (:domain blocksworld)
  (:objects a b c table1)
  (:init (on a table1) (on b table1)
       (on c table1))
```

```
(:goal (and (on a b)
 (on b c) (on c table1))))
```

Vstupom programu sú dva súbory vo formáte PDDL, pričom jeden zo súborov obsahuje informácie týkajúce sa domény, zatiaľ čo druhý súbor nesie informácie o zadávanej úlohe. Nachádzajú sa v nich informácie už spomínané v kapitole 3.6.2. Jeho výstupom je nájdený plán, reprezentovaný postupnosťou akcií, ktoré majú byť aplikované na jeho vykonanie. V prípade, že plán nebol nájdený, plánovač túto skutočnosť oznámi na výstupe. Nižšie je zobrazený príklad výstupu plánovača na testovacej úlohe.

```
move(b, c) -> move(a, b)
```

Celý plánovač je implementovaný v jednom zdrojovom súbore, kde je pomocou tried a funkcií popísaný doménový priestor a takisto postup plánovania. Kľúčovými časťami programu sú funkcie spracovania vstupu `create_world()` a samotný proces plánovania `linear_solver()`. Funkcia `create_world()` spracuje vstupné súbory a zadefinuje potrebné informácie z nich do tried využívaných plánovačom. Následne sa pomocou funkcie `linear_solver()` hľadá riešenie zadanej úlohy pomocou plánovacej metódy STRIPS.

Okrem spomínaných funkcií sú v programe implementované pomocné funkcie. Tieto napomáhajú vo fungovaní plánovača, pričom ide o funkcionality ako napríklad kontrola vstupných podmienok akcií, hľadanie aplikovateľných akcií, splnenie cieľu alebo kontrola zhodnosti predikátov. Ide o funkcie, ktoré umožňujú hladký proces hľadania finálneho plánu pre riešenie úlohy. Samotná funkcia plánovača implementuje algoritmus popísaný v kapitole 4.2.1.

Pri spracovaní vstupu sú jednotlivé časti domény zadefinované v programe pomocou tried, ktoré umožňujú jednoduchú prácu s týmito dátami. Konkrétne ide napríklad o triedy `World` a `Action`. Trieda `World` nesie reprezentáciu domény v jednotlivých atribútoch, aké objekty sa v nej nachádzajú, aký je aktuálny stav, cieľový stav a implementuje niekoľko pomocných funkcií využívaných pri plánovacom procese.

Trieda `Action` definuje akciu a všetky informácie potrebné na jej reprezentáciu a následné využitie ako napríklad parametre, vstupné podmienky a efekt na stav. Pri spracovávaní vstupu sú vytvorené objekty `Action` pre jednotlivé operátory domény, kde sa zadefinujú ich vlastnosti. Tieto objekty sa potom priradia do objektu `World`, do ktorého sú takisto vložené aj aktuálny a cieľový stav. Reprezentované sú pomocou triedy `Condition`, v ktorej sú definované jednotlivé predikáty reprezentujúce stav. Táto trieda obsahuje názov predikátu, objekty z domény, ktoré sa môžu vyskytnúť v zápise domény a nakoniec pravdivostnú hodnotu predikátu.

Na reprezentáciu operátorov a predikátov z domény sú využívané všeobecné zápisy, kde nefigurujú konkrétne objekty, s ktorými sa pracuje. Na tento účel slúžia triedy s predponou `Grounded`, ktoré obsahujú konkrétne názvy objektov zo zadanej úlohy v operátoroch a predikátoch. Ide o triedy `GroundedCondition` pre zápis predikátov a `GroundedAction` pre zápis akcií. Je potrebná reprezentácia v oboch formách, vzhľadom na to, že stav domény a akcie na vykonanie musia obsahovať konkrétne objekty, pričom pri procese plánovania je potrebné vedieť všeobecný tvar predikátu alebo operátora. Takisto to je potrebné pre metódu, ktorá musí vedieť, ako vyzerá zápis operátora alebo predikátu a mohol z neho odvodiť konkrétnu obmenu s objektmi.

V prípade implementácie programu STRIPS bolo potrebné upraviť časť programu zodpovednú za spracovávanie vstupu. Vzhľadom na to, že reprezentácie STRIPS a PDDL sú podobné v zapisovaní jednotlivých predikátov, bolo najmä potrebné pozmeniť to, kde boli

jednotlivé dáta v súbore uložené. Táto zmena oproti pôvodnej implementácii nastala vo funkcii `create_world()`, ktorá bola takmer celá nahradená, pričom jej fungovanie je inšpirované pôvodným prístupom. Vzhľadom na to, že vstup programu je spoločný aj pre metódu Graphplan, niektoré časti vstupu program ignoruje. Zameriava sa najmä na prečítanie dát potrebných pre jeho fungovanie. Dáta sú zapísané rovno do príslušných tried, ktoré boli už vyššie popísané.

Samotná funkcionalita metódy sa sústredí do funkcie `linear_solver()`, ktorá ďalej deleguje priebeh plánovania. Dôležitým prvkom tejto metódy je rekurzívne riešenie v prípade, keď je potrebné vyriešiť neskôr vytvorené podmienky, ktoré musia byť dodatočne splnené. Pomocné funkcie v programe sú určené na hľadanie aplikovateľných operátorov, na kontrolu vstupných podmienok a sledovanie splnenia predikátov.

Pre podrobnejšie sledovanie priebehu metódy je možné nastaviť sledovanie jej fungovania. Pokiaľ je program spustený bez tejto možnosti, je možné sledovať, aký aktuálny plán metóda našla. V rozšírenej možnosti je možné sledovať akým spôsobom sa pokúša o aplikovanie jednotlivých operátorov a akým spôsobom vyberá podciele, ktoré sa snaží splniť. Táto funkcionalita je veľmi vhodná pri sledovaní a analyzovaní správania metódy STRIPS a umožňuje podrobne vidieť, aký je jej priebeh pri rôznych úlohách a čo jej spôsobuje problémy.

5.4 Plánovač Graphplan

Druhou metódou, ktorej implementácia bola prevzatá od D. Nirwana zo zdroja [22], je plánovanie Graphplan prístupom. Táto implementácia je najrozsiahlejšia zo všetkých zvolených a vyplýva to z potreby vytvorenia pomocného plánovacieho grafu pred samotným procesom hľadania plánu. V ostatných metódach je prehľadávací priestor tvorený postupne a teda cesta, ktorá nebude súčasťou finálneho plánu, je uzatvorená veľmi rýchlo a nie je ďalej prehľadávaná. V prípade Graphplan sú vytvorené všetky možné aplikovateľné cesty ako sa uvádza v zdroji [3] a vzhľadom na to je priestor prirodzene väčší pri tvorbe. Je potrebné podotknúť, že spôsob akým je tvorený minimalizuje tvorbu nevhodných ciest a zlučovanie aplikovateľných operátorov na každej vrstve do jedného uzlu taktiež umožňuje zmenšený prehľadávací priestor.

V náväznosti na to je potrebné vytvoriť obmedzenia a usmernenia v grafe, ktoré pomôžu hľadaniu plánu tak, aby nedochádzalo k vzájomne konfliktným akciám, alebo nevznikali navzájom nezlučiteľné stavy, ktoré sú opísané v [3]. Tento proces je odlišný od prístupu STRIPS. V jeho prípade bola najprv nájdená akcia, ktorá by vedela dosiahnuť požadovaný stav a následne boli skontrolované podmienky tejto akcie a či je vôbec možné ju aplikovať. V prípade Graphplanu sú navrhnuté všetky akcie, ktoré spĺňajú podmienky ako možné riešenia a až pri spätnom prechádzaní grafu sa hľadí na to, či dosiahli požadovaný stav a vtedy sú zvolené do finálneho riešenia.

Implementácia tejto metódy je rozsiahlejšia a štrukturovanejšia, čo vyplýva z jej samotného fungovania. Metóda GraphPlan je v implementovaná ako knižnica, ktorú je možné nainštalovať a integrovať do ľubovoľného programu, pričom je spúšťaná s ľubovoľnými vstupnými súbormi. Na výstupe programu plánovač oznámi, či vedel nájsť plán vhodný pre vyriešenie úlohy alebo nie a príslušný plán vráti. Jeho štruktúra je čiastočne usporiadaná a teda vráti plán po vrstvách usporiadaných v poradí, v akom musia nasledovať, v rámci ktorých sú akcie ľubovoľne usporiadané. Plán je vždy definovaný v triede po jednotlivých vrstvách. Príslušná štruktúra má názov `Plan`, pričom pre celkový plán na všetkých vrstvách je definovaná trieda `LayeredPlan`, pomocou ktorej sú usporiadané plány na jednotlivých vrstvách.

Plán na jednej vrstve pozostáva z operátorov definovaných triedou `Operator`, ktorá je definovaná v module `pddlpy`. Špeciálny prázdny operátor 4.3.2 definuje samostatná trieda `NoOpAction`.

Samotný program pozostáva z 3 kľúčových častí, pričom prvou je kód zodpovedný za spracovanie vstupných súborov poskytujúcich informácie o doméne a úlohe. Modul `pddlpy`, je zodpovedný za spracovanie PDDL vstupu do príslušných dátových štruktúr, pričom hlavná časť dát je ukladaná do triedy `PlanningProblem`. Pomocou nej sú definované operátory, počiatočný stav a cieľový stav v doméne. `Pddlpy` roztriedi dáta z doménového a úlohového súboru do tried na operátory, predikáty, vstupný a výstupný stav.

Druhou časťou je modul `planning_graph`, ktorý definuje triedy definujúce prvky grafu a práce s ním. Najdôležitejšie triedy v programe sú `Graph` modelujúca plánovací graf, ktorý bude vytvorený pre riešenú doménu, trieda `PlanningGraph` zastrešujúca prácu s triedami `Graph` a `PlanningProblem`. V rámci tejto triedy je implementovaná tvorba tohto grafu. Ide o jeho expandovanie pri každej novej potrebnej vrstve, ktorá vzniká pri hľadaní cieľových predikátov a ich dosiahnutia. Trieda implementuje ako pridávanie aplikovateľných akcií, tak hľadanie mutex predikátov a operátorov. Metóda prebieha tak, ako bola popísaná v kapitole 4.4, pričom je implementovaná v triede `GraphPlanner`. Tvorí poslednú kľúčovú časť implementácie metódy a je definovaná v module `planning_graph_planner`. Trieda `GraphPlanner` využíva doteraz popísané štruktúry na svoje fungovanie, pričom najmä triedu `PlanningGraph`, ktorej tvorbu riadi a následne prehľadáva pre finálny plán.

Príklad výstupu programu na testovacej úlohe:

```
move {'?b1': 'b', '?dest': 'c'}
move {'?b1': 'a', '?dest': 'b'}
```

Súčasťou tejto implementácie je aj grafická vizualizácia pomocou knižnice `pydot`. Po nájdení plánu je možné vygenerovať vzniknutý graf, ktorý bol tvorený počas behu programu. Ten umožňuje zobraziť, akým spôsobom bol expandovaný graf a cez aké akcie a predikáty plánovač prechádzal.

5.5 Plánovač HTN

Poslednou metódou je plánovanie pomocou HTN, ktorého implementácia bola prevzatá od autora Dana Nau z [19]. Táto metóda, ako už bolo skôr ukázané, sa zaoberá iným prístupom k plánovaniu a to hierarchickým. V tomto druhu plánovania existuje opäť množstvo spôsobov, ako k nemu pristupovať, pričom HTN je jedným z nich. Najjednoduchším HTN plánovačom je prístup nazývaný SHOP (Simple Hierarchic Ordered Planner) [20], ktorý jednoduchým spôsobom implementuje podstatu metódy HTN, čím je rozkladanie abstraktnej úlohy na jednoduchšie a menšie podúlohy. Na tomto prístupe stavajú ďalšie plánovače, akým je napríklad SHOP2 [1], ktorý súťažil v plánovacích súťažiach a bol schopný dostať aj ocenenia. Tieto konkrétne plánovače sú spomínané kvôli tomu, že ich spoluautor je autorom zvolenej implementácie, ktorá bude porovnávaná.

Názov tohto plánovača je Pyhop a ide o jednoduchú implementáciu HTN plánovania, ktorej priebeh je podobný tej v SHOP plánovači [20]. Je to doménovo nezávislý plánovač, ktorý však ako vstup neprijíma PDDL súbory. Vzhľadom na spôsob, akým funguje HTN plánovanie, je potrebná podrobnejšia definícia domény, než akú poskytuje formát PDDL. Existujú rozšírenia tohto jazyka [11], ktoré umožňujú definovanie domény pre hierarchické plánovače, avšak táto implementácia nie je prispôsobená na úpravu tak, aby bolo možné problémy takto zadať. Bola vyvinutá snaha o zjednotenie vstupu pre všetky testované

metódy, avšak pri tejto konkrétnej nebolo možné zhodné zadanie vstupu. Aj napriek tomu bola táto implementácia zvolená, vzhľadom na jej autora a jeho prínos v implementácii podobných plánovačov, ako aj v literatúre, keďže ide o spoluautora niekoľkých zdrojov [10], [9] citovaných v tejto práci. Ďalším dôvodom je fakt, že spracovanie vstupu nemá vplyv na výkon a fungovanie samotnej plánovacej metódy, ktorou sa zaoberá táto práca. Takisto je potrebné poznamenať, že vytvorenie domény pre HTN vyžaduje jej rovnaké pochopenie bez ohľadu na to, či je definovaná vo formálnom jazyku alebo nie. V každom prípade je potrebné vedieť, akým spôsobom je možné zlúčiť niektoré operátory do skupín a vytvoriť z nich zložené operátory. Takisto je dôležité v akom poradí, za akých podmienok a na čo je potrebné dávať pozor pri ich rozkladaní. Pre definovanie domény pre HTN plánovač je preto potrebné jej veľmi dobré pochopenie. V opačnom prípade nie je plánovač schopný nájsť dobré riešenia alebo fungovať efektívne.

V Pyhop implementácii sú operátory definované ako funkcie, a stavy ako triedy. Program neberie vstupné súbory, ale je potrebné ich importovať do testovacieho skriptu, z ktorého bude riadené zadanie úlohy. Na definovanie domény je potrebné zadať operátory, tak ako v ostatných metódach, ale takisto aj zložené operátory. Tieto informácie sú definované pomocou funkcií, kde sú popísané vstupné parametre, vstupné podmienky a takisto akým spôsobom bude stav zmenený po aplikovaní operátora. Úlohu je možné zadať vytvorením objektov `State` a `Goal`, do ktorých sú ako do atribútov zadané predikáty domény a v akom stave sa nachádzajú. Ide o priamočiary a prehľadný zápis, ktorý uľahčuje prácu s týmto plánovačom.

Príklad definície vyššie spomínaného operátora move v implementácii Pyhop:

```
def move(state, b, table1, table2):
    if state.pos[b] == table1 and table1 != table2:
        state.pos[b] = table2
        return state
    else:
        return False
```

Samotná funkcionalita metódy je implementovaná v module `pyhop.py`, ktorý je možné využiť ďalej ľubovoľne. Tento modul uloží a zjednotí definované operátory a metódy v doméne, aby ich ďalej mohol použiť pre svoje fungovanie. Funkcia `seek_plan()` zabezpečuje hľadanie plánu, pričom jej priebeh bol popísaný v kapitole 4.6.1. Funkcia začína od zadaného stavu, ktorý sa pokúša buď vyriešiť priamo, alebo ďalej rozdeliť na podúlohy. V prípade rozdelenia na podúlohy, rekurzívne začne riešiť vzniknuté podúlohy, až kým nenarazí na základné operátory. Implementácia ukazuje, že fungovanie metódy ako takej je jednoduché, ale takisto je preň nevyhnutné dobre a správne zadefinovanie domény.

Príklad výstupu programu na testovacej úlohe:

```
('move', 'b', 'c'),
('move', 'a', 'b')
```

Kapitola 6

Porovnanie metód plánovania

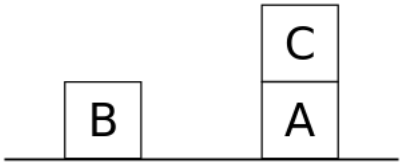
Hlavným cieľom tejto práce je porovnanie zvolených plánovacích metód, ktoré boli v predchádzajúcich kapitolách predstavené a definované. Ich implementácie boli použité na vzájomné porovnanie a testovanie na rôznych úlohách, ktoré pomôžu poukázať na vlastnosti metód. Ako vstupné úlohy boli zvolené domény typické pre automatizované plánovanie, ktoré sú často využívané v literatúre. V tejto kapitole budú predstavené úlohy, na ktorých sú metódy testované ako aj ich riešenie jednotlivými úlohami. Nakoniec budú porovnané výsledky, ktoré dosiahli, či boli schopné nájsť správny a efektívny plán a takisto porovnanie času potrebného na vyriešenie zadanej úlohy. Okrem toho je potrebné metódy porovnať z hľadiska náročnosti implementácie, keďže aj to je dôležitým faktorom ich využiteľnosti.

Z doterajších informácií, poskytnutých o zvolených prístupoch je možné vidieť, že nie všetky budú schopné správne alebo efektívne riešiť všetky plánovacie úlohy. Tie je možné rozdeliť do skupín, podľa toho o aký typ úlohy ide. Typickými skupinami sú napríklad konfliktné ciele, čiastočne usporiadané ciele, alebo domény, ktoré obsahujú veľké množstvo objektov, s ktorými bude metóda pracovať. Každá z metód má vlastnosti, ktoré jej dávajú predpoklady na riešenie konkrétnych typov úloh. Takisto platí opak, kedy niektoré prístupy nebudú schopné nájsť plán pre zadaný problém [10], aj keď existuje. Využitie rôznych metód v praxi preto závisí od toho, aké typy úloh sú schopné riešiť a podľa toho je nutné správne zvoliť metódu, s ohľadom na zadanú úlohu, ktorú je potrebné riešiť. Zámerom tejto práce je pre jednotlivé metódy zhodnotiť v akých úlohách je vhodné ich využitie.

6.1 Známe problémy a limitácie metódy STRIPS

Existujú vlastnosti metódy STRIPS, ktoré sú o nej známe a ktoré prinášajú problémy pre jej schopnosti riešenia problémov [10]. V tejto kapitole budú popísané problémy, o ktorých je známe že vznikajú a neskôr pri testovaní na vstupoch bude ukázané, ako to prakticky vyzerá.

Prvou chybou, ktorá je známa, je problém nazývaný Sussmanova anomália, ktorej priebeh je popísaný na podklade [5], pričom konkrétne zadanie úlohy bolo pozmenené. Typický príklad je možné ukázať na jednoduchej doméne `Simple Blocks` 6.4.1, v ktorej je úlohou pomocou robota presúvať kocky na podložke do zadaných tvarov a nastavení. Existuje príklad, ktorého počiatočný stav je možné vidieť na obrázku 6.1, na ktorom je možné anomáliu opísať. Ide o jav, kedy plánovač vyberá spomedzi dvoch alebo viacerých krokov, ktoré je možné spraviť v jednu chvíľu. Nevie vyhodnotiť, že aj napriek tomu, že splní jeden z podcieľov, týmto zablokuje a predĺži plán potrebný na jej splnenie.



Obr. 6.1: Sussmanova anomália [4]

Na príklade je možné vidieť, že existujú dve veže kociek, pričom prvá pozostáva z kocky `B` a druhá pozostáva z kociek `C` a `A`. Cieľom je vytvoriť vežu zo všetkých kociek, pričom ich poradie od podložky je vzostupné `C`, `B` a navrchu veže `A`. Plánovač sa snaží o splnenie 3 predikátov, pričom ide o `ontable(C)`, `on(B, C)` a `on(A, B)`. Z počiatočného stavu je možné priamo splniť posledné dva predikáty. Najpriamejší prístup by bolo presunúť kocku `C` a následne na ňu postupne poskladať kocky. Problém nastane, pokiaľ sa metóda snaží ako prvý predikát splniť `on(B, C)`, pretože vtedy síce splní jeden z podcieľov, ale nemôže pokračovať ďalej v riešení, bez odstránenia tohto splneného podcieľa. Podobný prípad môže nastať, keď plánovač najprv uvoľní kocku `A` zložením kocky `C` na podložku, ale rozhodne sa ako ďalší predikát splniť `on(A, B)`. Vtedy opäť dôjde k splneniu jedného z podcieľov, ale nie je možné splniť celý cieľ, pokým tento nebude odstránený. Táto anomália nemusí nastať, pokiaľ metóda zvolí také poradie predikátov, ktoré nespôsobí vzájomné rušenie podcieľov. Takisto nejde o chybu, ktorá by spôsobovala nefunkčnosť metódy, skôr ide o spomalenie plánovania [10] a skúmanie cesty, ktorá na prvý pohľad nemôže fungovať.

Ďalší problém, ktorý pri metóde nastáva a je závažnejší, je neschopnosť nájsť plán aj keď existuje, ktorý uvádza aj publikácia [10]. Problém vzniká napríklad v prípadoch, kedy plánovač zvolením konkrétneho postupu operátorov vyrieši čiastočný plán, no kvôli tomu nie je schopný prehľadať inú časť grafu, v ktorej sa nachádza riešenie ostatných podcieľov. Vzhľadom na to, že STRIPS po nájdení riešenia pre nejaký predikát toto riešenie berie ako záväzné, v tomto bode riešenia už metóda nie je schopná návratu a hľadania alternatívy. Namiesto toho je celá úloha nevyriešená.

Môžu nastať prípady, kedy metóda vráti neplatný plán, ktorý je chybný [10]. Môže sa to stať v prípade, že pri spätnom prehľadávaní metóda zruší splnenie nejakého podcieľa, ktorý už je vo finálnom pláne splnený. Vtedy pokračuje ďalej a nesnaží sa spätne ho splniť, pretože ho považuje za vyriešený. Namiesto toho hľadá riešenie ostatných podcieľov. Vtedy vrátený plán nie je možné považovať za validný.

6.2 Známe problémy a limitácie metódy Graphplan

Na rozdiel od STRIPS, pokiaľ Graphplan nájde nejaký plán, bude sa jednať o validné riešenie. Takisto platí, že pokiaľ existuje riešenie nejakej úlohy, Graphplan je schopný ho nájsť [3]. Tieto vlastnosti hovoria o jeho lepšej plánovacej schopnosti ako pri predchádzajúcej metóde. Je teda možné predpokladať, že na zložitejších príkladoch bude mať Graphplan lepšiu úspešnosť ako STRIPS.

Najväčšou nevýhodou tohto prístupu je samotná tvorba plánovacieho grafu, pričom sa konkrétne jedná o hľadanie vzájomne vylučujúcich sa akcií. Tento proces trvá najdlhšie z celého procesu tvorby grafu [3]. Napriek napriek tomu ide o súčasť algoritmu, ktorá jej umožňuje vyhýbať sa cestám v grafe, ktoré nevedú ku konečnému riešeniu. Čas strávený pri tvorbe mutex akcií sa vráti pri spätnom prehľadávaní grafu, kedy je výber aplikovateľ-

ných akcií značne zmenšený. Implementácia tejto metódy je pravdepodobne najzložitejšia zo všetkých troch porovnávaných, no stále nejde o príliš zložitý proces.

6.3 Známe problémy a limitácie metódy HTN

Pri hierarchickom plánovaní bolo už niekoľkokrát načrtnuté, aké dôležité je poskytnúť dobre definovanú doménu. Toto je určite najväčšia slabina metódy HTN, kedy bez správne a kvalitne zadanej domény, nie je plánovač schopný nájsť validný plán. Miera, v ktorej je potrebné poskytnúť informácie plánovača, zahŕňa informácie, ktoré niekedy obsahujú veľkú časť samotného riešenia problémov v zadanej doméne [8]. Je možné vidieť, že toto dáva HTN metóde výhodu oproti ostatným metódam. Umožňuje efektívnejšie riešenie, nemusí hľadať tak veľa rôznych možností aplikovania rôznych operátorov a takisto je rýchlejší oproti ostatným metódam. Toto je samozrejme z pohľadu aplikovania metódy v praxi výhodou a je to žiaduce pre aplikovateľné programy. Avšak v rámci súťažného automatizovaného plánovania je tento prístup kontroverzný. Najznámejším incidentom je diskvalifikácia plánovača SHOP [20] na plánovacej súťaži v roku 2000, pričom plánovač nebol schopný riešenia zadaných úloh bez podrobnejšej špecifikácie domény. Tento fakt bol pripísaný spôsobu zapisovania domény a kritizovaním jej nedostatočne bohatého popisu. [8]

Je preto potrebné brať do úvahy, že potreba podrobne definovanej metódy komplikuje inak priamočiaru implementáciu metódy. Samotné fungovanie nie je komplikované, ale čas ušetrený v tejto oblasti je potrebné vložiť do definovania testovacích domén. Tak tomu bolo aj pri testovaní v tejto práci, pričom popis domény pomocou stavov je podobne komplikovaný ako definovanie v jazyku PDDL.

6.4 Testovanie metód na zvolených plánovacích úlohách

Táto podkapitola sa venuje samotnému testovaniu metód na rôznych problémoch, ktoré sa v plánovaní riešia. Ide o výber domén, ktoré sa zameriavajú na rôzne typy plánovacích problémov, pričom ide aj o klasické úlohy často sa vyskytujúce v literatúre, ako aj o úlohy vyskytujúce sa na plánovacích súťažiach [13]. Vo väčšine úloh nejde o príliš komplikované zadania. Porovnanie bolo zamerané na menšie úlohy, ktoré ukážu výkonnosť a spoľahlivosť metód na niektorých typoch a druhoch úloh. Je možné tieto predpoklady potom aplikovať na podobné zadania s väčšou náročnosťou v prípade, že je metóda schopná zvládať také úlohy. Ako bude možné pozorovať z výsledkov, základné implementácie zvolených metód majú isté obmedzenia, ktoré bránia v riešení niektorých úloh.

Domény, ktoré sa typicky v plánovaní riešia a vyskytujú, boli v tejto práci rozdelené do niekoľkých skupín, podľa zamerania problematiky. Vo väčšine prípadov je možné hovoriť o prelínaní týchto skupín, kedy niektoré úlohy naraz spadajú do viacerých.

Zvolené úlohy môžu byť rozdelené do nasledujúcich kategórií: čiastočné usporiadanie cieľov, konfliktné ciele a veľký počet objektov v doméne. Úlohy s čiastočným usporiadaním cieľov sú také úlohy, kde je možné jednotlivé časti cieľa rozdeliť do podcieľov, ktoré je možné riešiť nezávisle od seba. Vtedy je možné, aby riešenie jedného z podcieľov nemalo žiadny alebo minimálny vplyv na riešenie ostatných podcieľov. Konfliktné ciele obsahujú také úlohy, pri ktorých nastáva stav taký, že na splnenie dvoch alebo viacerých predikátov je potrebné predtým splniť navzájom konfliktné podmienky. Vtedy je potrebné správne usporiadanie riešenia podcieľov tak, aby došlo k čo najefektívnejšiemu riešeniu problému a plánovač našiel spôsob ako postupne splniť konfliktné podmienky. Úlohy s veľkým počtom objektov

sa vyznačujú väčšou výpočetnou náročnosťou ako iné typy úloh. Pri niektorých metódach počet objektov a operátorov v doméne spomalí jej schopnosť riešenia úloh. Niektoré metódy majú nástroje na prechádzanie veľkému spomaleniu a nevplýva to na ich funkčnosť.

Domény používané na testovanie boli inšpirované z niekoľkých zdrojov. [12], [14] a [2] Vzhľadom na to, že väčšina domén je často používaná a opakovaná, ich zadania sa od seba rôzne líšia aj v prípade, že sa jedná o tú istú doménu. Každá z nich pochádza z myšlienky nejakého procesu, ktorý je možné vyriešiť automatizovaným plánovaním a samotné detaily, ako obmedzenia, operátory a objekty je možné ľubovoľne zadefinovať. Testovacie domény sú inšpirované a upravené z literatúry [10], [9], [24] a takisto aj použitých implementácií, ktoré ich využívali na testovanie. Pri testovaní bol zvolený jeden príklad, ktorý postupne riešili všetky metódy. Kvôli dôveryhodným výsledkom bolo potrebné jeden test spúšťať opakovane, vzhľadom na niekedy veľké rozpätie časovej náročnosti pre jednotlivé metódy. Preto boli úlohy testované opakovane a finálny čas je uvedený priemernou hodnotou. Pre každý test bola spustená každá metóda postupne, pričom bol meraný čas potrebný na vyriešenie zadanej úlohy. Do toho nebol počítaný čas potrebný pre zadanie a spracovanie vstupu. Na výstupe boli skúmané a porovnávané výsledné plány, ako kvôli správnosti nájdených výsledkov, tak kvôli porovnaniu ich fungovania. Pri niektorých testoch bol podrobnejšie sledovaný postup metódy a to prípadným krokovaním jeho riešenia, v prípade, že sa metóda správala inak než bolo očakávané, alebo prejavovala svoje špecifické vlastnosti.

Rozdiel v zadaní domény pre HTN oproti ostatným metódam spočíva v definovaní zložených operátorov pre zadanú doménu. Takisto sú definované vstupné podmienky zložených operátorov a spôsob akým tvoria hierarchiu operátorov.

6.4.1 Simple blocks domain

Ako prvá doména bola zvolená pravdepodobne najjednoduchšia doména, s akou je možné sa v plánovaní stretnúť. Jej použitie je dobré na ukázanie fungovania metód, ako aj na poukázanie ich schopnosti riešiť jednoduché a priamočiare úlohy.

Podstatou tejto domény je rameno robota, ktoré dokáže presúvať kocky, ktoré sa v doméne vyskytujú. Môže sa tu nachádzať nenulový počet kociek, ktoré ležia na stoloch. Tých môže takisto byť väčší počet, a úlohou robota je presúvanie kociek z ich pôvodných pozícií na stole, na inú zvolenú pozíciu. Kocky nemôžu ležať na sebe navzájom, musia byť na ľubovoľnom stole a rameno môže presúvať naraz práve jednu kocku.

Predikát `(on ?b ?t)` popisuje polohu bloku `?b` na stole `?t`. Operátor `move` presúva blok `?b` medzi pôvodnou polohou `?t1` na cieľovú polohu `?t2`. Na jeho vykonanie je potrebné, aby sa blok nachádzal na pôvodnom mieste. Grafickú vizualizáciu tejto domény je možné vidieť na obrázku 6.1.

Zadanie domény v PDDL:

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block table)
  (:predicates
    (on ?b - block ?t - table))
  (:action move
     :parameters (?b ?t1 ?t2)
     :precondition (and (on ?b ?t1) (not (on ?b ?t2)))
     :effect (and (on ?b ?t2) (not (on ?b ?t1)))))
```

HTN v tejto doméne má ako jediný zložený operátor úlohu presunu všetkých kociek v doméne. Ten je priamo rozložiteľný na operátor `move()`.

Ide o jednoduchú doménu, v ktorej nie je možné vytváranie komplikovaných úloh. Možnosťou je ale presúvanie väčšieho množstva objektov. Cieľom tejto testovacej úlohy je sledovať, akým spôsobom sú jednotlivé metódy schopné pracovať v nenáročnom prostredí. Z pohľadu praktického využitia je to zaujímavé kvôli mechanickým úlohám, kde je potrebné rýchle riešenie a nie je potrebné vytvárať zložité pomocné prostriedky pri plánovaní.

Zadaná úloha

Zadanie úlohy v PDDL:

```
(define (problem move-blocks-between-tables)
    (:domain blocksworld)
  (:objects
  a b c d e f - block
  table1 table2 - table)
  (:init (on a table2) (on b table1) (on c table2)
       (on d table1) (on e table2))
  (:goal (and (on a table1) (on b table2)
   (on c table1) (on d table2) (on e table1))))
```

Porovnanie metód na zadanej úlohe

Túto úlohu dokážu vyriešiť všetky domény bez problémov. Preto bola zadaná ako testovacia a neukazuje špeciálne zameranie plánovacieho problému. Na časových údajoch je možné sledovať, že čas potrebný na riešenie takto jednoduchej úlohy sa postupne znižuje v závislosti od zložitosti metódy, tak ako bolo predpokladané. Teda STRIPS má výsledky najpomalšie, a HTN rieši úlohu najrýchlejšie.

STRIPS	Graphplan	HTN
0.0012s	0.00073s	0.0001s

Tabuľka 6.1: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Simple Blocks

Nájdený plán:

```
move(d, table1, table2) -> move(b, table1, table2) ->
move(e, table2, table1)-> move(c, table2, table1) ->
move(a, table2, table1)
```

Všetky metódy našli rovnaký finálny plán, ktorý je vyššie zobrazený.

Už pri takto malej úlohe je možné vidieť, že priamočiary prístup metódy STRIPS, ktorá skúša všetky možnosti, ktoré má k dispozícii v každom kroku, nestíha oproti ostatným metódam, ako je možné vidieť v tabuľke 6.1. Tie sa snažia eliminovať nepriechodné cesty pred ich skúmaním.

Pri jednoduchých úlohách ako je táto, môže v praxi postačovať riešenie pomocou STRIPS vzhľadom na menšiu implementačnú náročnosť ako je Graphplan a menšiu potrebu špecifikovania domény ako HTN. Aj napriek tomu, pokiaľ ide o efektivitu, už od samého

začiatku testovania je zrejmé, že metóda STRIPS nebude dosahovať tak kvalitné výsledky ako ostatné.

Na výsledných plánoch je možné sledovať to, že všetky metódy vytvorili úplne usporiadané plány. Teda aj metóda Graphplan, ktorá tvorí čiastočne usporiadané plány dokáže vytvoriť iba úplne usporiadaný plán a nevie využiť svoju výhodu grafu s vrstvami, kde by bolo možné vykonávať niektoré akcie nezávislo v jednom čase. To isté platí pre výsledok HTN, ktorý vytvorí plán s akciami ktoré sa cyklicky opakujú, teda pre niekoľko predikátov sa vykoná tá istá postupnosť krokov na jeho splnenie.

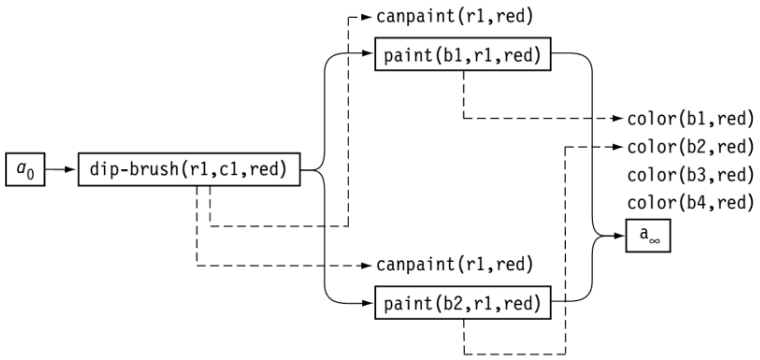
6.4.2 Paint domain

Jednou zo zvolených domén obsahujúcich čiastočne usporiadané ciele je práve doména `Paint` [10]. Obsahuje niekoľko blokov, ktoré je potrebné natrieť zvolenou farbou pomocou štetcov, ktoré môžu byť použité práve raz. Vyskytuje sa tu obmedzenie použitia nejakého objektu iba jedenkrát, čo pre plánovač znamená hľadanie nejakého voľného objektu, ktorý by mohol zastať túto úlohu.

Okrem toho je v doméne 7 predikátov, ale naopak operátory sú iba 2, čiže sa nejedná o veľkú doménu. Je na nej možné ukázať, ako sú metódy schopné vysporiadať sa s jednoduchou doménou so zameraním na čiastočné usporiadanie cieľov.

Grafická ukážka domény na obrázku 6.2 pochádza z pôvodného zdroja tejto domény [10] a je možné na nej ukázať, ako doména funguje na jednoduchom príklade. Počiatočný stav je popísaný pomocou `a0`, pričom je potrebné v prvom rade zvoliť potrebnú farbu na štetec `r1`, ktorú bude aplikovať na zadané bloky. Následne pre jednotlivé bloky vykoná operáciu maľovania, kde aplikuje predtým zvolenú farbu z štetca `r1` a dosiahne stav namaľovaných blokov. Farby, ktoré je potrebné aplikovať sa môžu pre každý blok líšiť a preto je potrebné v prvom kroku zvoliť toľko štetcov, koľko rôznych farieb je potrebné využiť. Následne sú zvolené konkrétne štetce pre tie bloky, ktoré majú cieľový stav farby príslušného štetca.

Pre HTN sú navyše definované zložené operátory. Ten s najvyšším postavením v hierarchii reprezentuje odbavenie všetkých blokov v doméne. Rozkladá sa na operátor reprezentujúci odbavenie jedného konkrétneho bloku. Operátor jedného bloku sa potom rozkladá na sekvenciu jeho maľovania, ktorá pozostáva z operátora `dip_brush()` a `paint()`, ktoré sú aplikované na konkrétny blok, štetec a farbu.



Obr. 6.2: Príklad v doméne Paint, ktorý ukazuje problém natierania blokov na zvolenú farbu, pričom je potrebná konkrétna postupnosť operátorov tak, aby bolo cieľ možné dosiahnuť. [10]

Zadaná úloha

Zadaná úloha obsahuje štyri bloky a štetce, ktoré sa dajú nafarbiť určitou zvolenou farbou. Pri riešení tejto úlohy je potrebné hľadať voľný objekt štetca, ktorý by mohol byť využitý na potrebnú akciu.

Porovnanie metód na zadanej úlohe

Jednoduchú úlohu s čiastočným usporiadaním riešia všetky metódy, pričom je možné sledovať, že Graphplan je najlepšie prispôsobený na prácu s takýmto typom úloh. Spôsob, akým funguje jeho riešenie mu umožňuje nájsť riešenia všetkých čiastočných úloh a tak nájsť najefektívnejšie riešenie, ktoré má výhodu čiastočného usporiadania.

STRIPS	Graphplan	HTN
0.27553s	0.05116s	0.00016s

Tabuľka 6.2: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Paint

Plán nájdený metódami STRIPS a HTN:

```
dip-brush(brush1, can1, red) -> paint(block1, brush1, red) ->
dip-brush(brush3, can2, blue) -> paint(block4, brush3, blue) ->
dip-brush(brush2, can1, red) -> paint(block2, brush2, red) ->
dip-brush(brush4, can2, blue) -> paint(block3, brush4, blue)
```

V tomto testovaní všetky metódy našli riešenie, pričom Graphplan našiel čiastočne usporiadaný plán umožňujúci efektívnejší postup, v ktorom zoskupuje rovnaké operátory nezávisle pracujúce do jednej úrovne. Preto v jednej úrovni môže nastať aplikovanie všetkých `dip-brush()` operátorov a v tej nasledujúcej `paint()` operátorov.

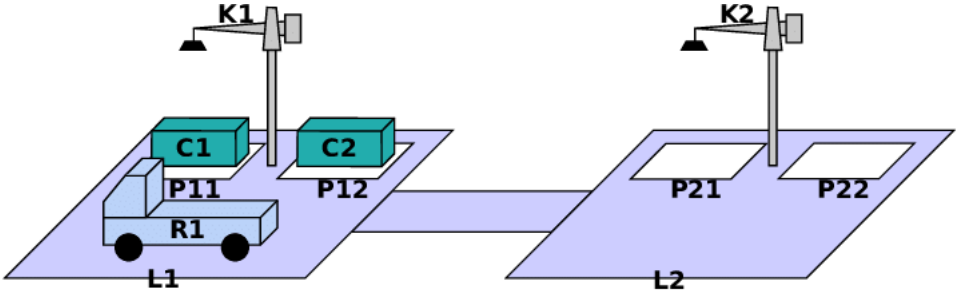
STRIPS nájde riešenie úlohy a potrebuje dlhší čas ako HTN, keďže musí v každom kroku postupne prejsť aplikovateľné operátory, pričom výsledky je možné pozorovať v tabuľke 6.2. V jednoduchej úlohe tohto typu je schopný nájsť validné riešenie.

HTN využíva znalosti o doméne na rýchle aplikovanie zložených operátorov a nájdenie riešenia. Keďže ide o rozkladanie abstraktných úloh, v doméne je zloženým operátorom definovaná postupnosť operátorov `dip-brush(brush, can, color)` a `paint(block, brush, color)`. Je možné vidieť, že je vytvorený plán pozostávajúci z týchto opakujúcich sa krokov, pričom to isté riešenie našiel aj STRIPS.

6.4.3 Dock Worker Robot domain

Komplikovanejšou doménou, ktorá sa typicky vyskytuje v literatúre [10], je doména `Dock-Worker-Robot`, ktorá pozostáva z prístavu, v ktorom sa pomocou ramien robotov presúvajú kontajnery. Jej grafickú ukážku je možné vidieť na obrázku 6.3, pričom je tu možné vidieť objekt auta `R1`, ktorý je v tejto práci vynechaný. Ako je možné vidieť, ide o skomplikovanie domény `Simple Blocks`, pričom sa jedná o rôzny počet lokalít, na ktorých sa môžu kontajnery nachádzať a takisto väčší počet robotov, s ktorými je možné pracovať.

Jedná sa o príklad s väčším počtom objektov, s ktorými je potrebné pracovať, ale dôležitou vlastnosťou tejto domény je, že sa v nej pracuje s úlohami s čiastočne usporiadanými cieľmi. V tejto doméne sa na každej lokalite nachádza jeden robot, ktorý sa môže presúvať medzi susednými lokalitami, na obrázku označenými ako `L1` a `L2`. Robot vie zdvihnúť kontajner z lokality kde sa nachádza a presunúť sa aj s ním. Vzhľadom na to, že je v doméne



Obr. 6.3: Grafická ukážka domény Dock-Worker-Robot, kde sú medzi susediacimi lokalitami pomocou ramien presúvané kontajnery. [15]

niekoľko robotov, je možné naraz pracovať s viacerými presunmi, oddelene, bez vzájomného ovplyvňovania.

HTN disponuje informáciami rozloženia tejto úlohy na podúlohy. Najprv sa pozerá na vyriešenie úlohy pre všetky kontajnery, tento operátor sa rozloží na úlohu pre každý jeden operátor zvlášť. Pre každý z nich je potom vygenerovaný operátor reprezentujúci sekvenciu presunu jedného bloku. Pozostáva z usporiadaného zoznamu jednoduchých operátorov `load()`, `move()`, `unload()` a `move()`. Keďže na jednej lokalite môže byť naraz práve jeden robot, táto podmienka je vynútená presunom robota späť na jeho pôvodné miesto po presune kontajnera.

Jedná sa o zložitejšiu doménu, vzhľadom na to, že počet objektov v nej sa zvýšil oproti predchádzajúcim problémom. Nielen to, ale jedná sa aj o počet predikátov, s ktorými sa v nej pracuje, ktorých zvyšujúci sa počet prispieva k zložitosti úloh keďže dochádza k širšiemu vetveniu možných riešení. Dochádza k nemu kvôli tomu, že samotné podmienky a efekty operátorov obsahujú viac predikátov, ako v predchádzajúcich úlohách a teda je potrebné ich pri riešení zobrať do úvahy.

Zadaná úloha

Úloha v tejto doméne nie je komplikovaná, ale je na nej možné ukázať akým spôsobom sa jednotlivé metódy dokážu vysporiadať s čiastočne usporiadanými cieľmi, ktoré je výhodné riešiť každý osobitne.

Zadanie úlohy v PDDL (presné zadanie bolo zjednodušené z rozmerových dôvodov):

```
(define (problem dwr)
  (:domain dock-worker-robot)
  (:objects
   robr robq robs - robot
   loc1 loc2 loc3 - location
   conta contb contc - container)
  (:init
   (in conta loc1) (in contb loc2) (in contc loc3)
   (atl robr loc1) (atl robq loc2) (atl robs loc3))
  (:goal (and (in contb loc1)
  (in conta loc3) (in contc loc2))))
```

Porovnanie metód na zadanej úlohe

Všetky metódy zvládajú riešenie zadnej úlohy v tejto doméne. Na výsledkoch je možné pozorovať, že časové výsledky metód sú usporiadané rovnako, ako v predchádzajúcich testoch 6.3. Avšak v tomto prípade je potrebné sledovať plány, ktoré boli nájdené. Je možné vidieť, že aj napriek tomu že nebol najrýchlejší, Graphplan našiel najefektívnejší plán zo všetkých metód.

STRIPS	Graphplan	HTN
0.17867s	0.10528s	0.00002s

Tabuľka 6.3: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Dock-Worker-Robot

Plán nájdený metódou Graphplan:
Prvá úroveň grafu:

```
load(loc3, contc, robs), load(loc1, conta, robr), load(loc2, contb, robq)
```

Druhá úroveň grafu:

```
move(robq, loc2, loc1), move(robr, loc1, loc3), move(robs, loc3, loc2)
```

Tretia úroveň grafu:

```
unload(loc3, conta, robr), unload(loc1, contb, robq),
unload(loc2, contc, robs)
```

Plán nájdený metódou HTN:

```
load(loc1, conta, robr), move(robr, loc1, loc3),
unload(loc3, conta, robr), move(robr, loc3, loc1),
load(loc2, contb, robq), move(robq, loc2, loc1),
unload(loc1, contb, robq), move(robq, loc1, loc2),
load(loc3, contc, robs), move(robs, loc3, loc2),
unload(loc2, contc, robs), move(robs, loc2, loc3)
```

Plán nájdený metódami Graphplan a HTN je veľmi podobný. Ide o postupnosti operátorov `load()`, `move()` a `unload()`, pre každý kontajner, ktorý je potrebné presunúť. Graphplan opäť rozdeľuje prácu do jednotlivých úrovní, a teda stanovuje, že je možné vykonať všetky operátory `load()` v tom istom čase, keďže sú od seba vzájomne nezávislé. To isté platí aj pre ostatné operátory.

Riešenie metódy Graphplan je priamočiare, nevykonáva žiadne zbytočné kroky, ktoré by zdržovali riešenie. Vykoná presne tie kroky, na ktorých je možné vidieť, že tvoria najefektívnejšie riešenie úlohy. Jedná sa o vzorové riešenie a teda je možné zhodnotiť, že v prípade úloh s čiastočne usporiadanými cieľmi je vhodné využívať metódu prispôsobenú na hľadanie rozložiteľného riešenia, ako v prípade Graphplan metódy. Nielen, že nájde najefektívnejšie riešenie, ale takisto ho vie rozložiť do niekoľkých rovín, podľa toho, ako nezávislé tieto činnosti od seba sú. Tu je možné sledovať, že presun kontajnerov je možné rozdeliť na vzájomne nezávislé činnosti, pričom každú osobitne môže riešiť jeden robot. Metóda vedela nájsť toto oddelenie a tak aj ukázať, ako by bolo v praxi možné takýto problém vyriešiť v troch oddelených procesoch.

STRIPS našiel plán, ktorý je funkčný, ale nejedná sa o efektívny ani dobrý plán. Nie je zobrazený z rozmerových dôvodov, pretože sa jedná o zdĺhavé riešenie, ktoré obsahuje veľa prebytočných krokov a nie je priamočiare. Vyskytujú sa v ňom zbytočné kroky, ktoré pri využívaní tohto riešenia nie je potrebné vykonať, pretože nijakým spôsobom nebránia jeho postupu, ale takisto neprispievajú k riešeniu cieľového stavu. Ide o sekvenciu operátorov `load()` a `unload()`, ktorých striedanie v jednom bode plánovač využíva na presun prázdneho robota, ktorého polohu nie je potrebné meniť. Problém bol spôsobený nesprávnym poradím podcieľov, ktoré boli usporiadané tak, že pri splnení jedného z nich bol hneď ten nasledujúci zablokovaný a bolo potrebné rekurzívne ho splniť pred pokračovaním riešenia. Slučka sa môže niekoľkokrát rekurzívne opakovať pre riešenie jediného predikátu. Toto môže nastať pre rôzne operátory, ktoré majú ako vstupné podmienky opačne platné predikáty. Riešenie nastalo až v bode, kedy boli podciele usporiadané nekonfliktným spôsobom. Metóda sa dostala do slučky z ktorej bolo potrebné vyjsť. Predikáty, ktoré sa snažila splniť pri rekurzívnom riešení zavádzala opakovane do svojho riešenia, aplikovaním tých istých protichodných akcií. Pri rekurzívnom riešení posledného cieľového predikátu, v ktorom sa snažila o umiestnenie posledného kontajnera na príslušnú lokalitu v nesprávnom poradí volila podciele na splnenie vstupných podmienok operátora, ktorý v tom bode skúmala. Aj napriek tomu, v konečnom dôsledku našla plán, ktorý by splnil zadanú úlohu. Táto situácia môže nastať aj pre inú dvojicu operátorov.

HTN našlo validný plán, ktorý splní zadanú úlohu, ale obsahuje niektoré nepotrebné kroky. Tie vyplývajú z definovania zloženého operátora, ktorý táto metóda využíva. Pri presune kontajnera na cieľové miesto následne robota premiestni späť na jeho predchádzajúcu polohu. V tomto prípade ide o nadbytočný krok, ktorý nie je pre riešenie zadanej úlohy nijako prínosný. V prípade, že by z jednej lokality bol potrebný presun viacerých objektov, tieto kroky by boli praktické a predišlo by sa zbytočnému hľadaniu voľného robota. Pokiaľ by bola doména iným spôsobom definovaná, bolo by možné tomuto kroku predísť a miesto toho aplikovať presun robota až v momente ako je potrebný. Otázka definovania domény tu ukazuje, že to, akým spôsobom je jej fungovanie pochopené, vplýva na efektivitu nájdených plánov. Pre niektoré úlohy v tejto doméne by tento prístup ušetril potrebný čas hľadania riešenia. Pri zadanej úlohe tomu tak nie je, i keď nájdený plán nie je natoľko neefektívny ako plán metódy STRIPS.

6.4.4 Aircargo domain

Podobná doména, ako je `Dock-Worker-Robot` je letecká doména, kde pomocou lietadiel dochádza k presunu kontajnerov medzi rôznymi letiskami a je možné ju nájsť napríklad v [24]. Takisto ide o doménu, kde sa pracuje s úlohami s čiastočne usporiadanými cieľmi. Jej fungovanie je možné si predstaviť podobne, ako v predchádzajúcej úlohe. Existuje niekoľko letísk, medzi ktorými sa presúvajú nákladné lietadlá. Tie je možné použiť na presun kontajnerov medzi letiskami. Cieľom je presun kontajnerov medzi lokalitami tak, ako je zadané v úlohe.

HTN, tak ako v predchádzajúcich úlohách, má ako najvyššie postavený operátor riešenie všetkých kontajnerov existujúcich v doméne. Tento operátor je rozložený na odbavenie každého z nich konkrétne, pričom sa ďalej rozdeľuje na sekvenciu presunu práve jedného kontajnera. Tá pozostáva z `load()`, `fly()`, `unload()` a `fly()`.

Navyše je tu zaujímavé, že narozdiel od robotov, je lietadlo schopné presunúť niekoľko kontajnerov naraz a tak ušetriť zbytočné operátory opakovaných presunov. Pokiaľ vie metóda nájsť toto riešenie, ide o značne veľké šetrenie času pri krokoch finálneho plánu. Táto

doména má menej predikátov a menšie obmedzenia presunu, preto v niektorých ohľadoch ide o jednoduchšie aplikovanie operátorov. Z toho hľadiska bola zvolená úloha s viacerými objektmi, aby bolo testované nielen čiastočné usporiadanie úloh, ale takisto aj väčšie množstvo objektov, čiže širšie vetvenie prehľadávacieho priestoru. V tomto testovaní teda ide o porovnávanie na úlohe s kombinovaným zameraním.

Zadanie domény v PDDL:

```
(define (domain air-cargo)
  (:requirements :strips)
  (:predicates (in ?obj ?place) (at ?obj ?place) (cargo ?obj)
           (plane ?obj) (airport ?obj))
  (:action load
     :parameters (?c ?p ?a)
     :precondition (and (at ?c ?a) (at ?p ?a) (cargo ?c) (plane ?p)
     (airport ?a))
     :effect (and (in ?c ?p) (not (at ?c ?a))))
  (:action unload
     :parameters (?c ?p ?a)
     :precondition (and (in ?c ?p) (at ?p ?a) (cargo ?c) (plane ?p)
     (airport ?a))
     :effect (and (at ?c ?a) (not (in ?c ?p))))
  (:action fly
     :parameters (?p ?from ?to)
     :precondition (and (at ?p ?from) (plane ?p) (airport ?from)
     (airport ?to))
     :effect (and (at ?p ?to) (not (at ?p ?from)))))
```

Zadaná úloha

V tejto úlohe sa pracuje s 10 kontajnermi, ktoré sa nachádzajú na rôznych letiskách a je potrebné ich medzi nimi previezť. Predikátov v tejto doméne je 5, čiže podobne, ako v náročnejších úlohách, ale vstupné podmienky operátorov sú menej náročné ako v iných doménach.

Porovnanie metód na zadanej úlohe

Pripravený problém boli schopné vyriešiť všetky metódy s rôznou časovou úspešnosťou, aj s ohľadom na efektivitu plánu. Nastáva tu podobná situácia ako v prípade `Dock-Worker-Robot` domény, kedy sa efektivita nájdených plánov od seba značne odlišuje.

Opäť najlepší výsledok prinieslo riešenie pomocou metódy Graphplan, ktorá síce netrvá najkratšie, ale našla najefektívnejší možný plán. Zozbierala všetky kontajnery z jedného letiska, ktoré smerovali na to isté finálne miesto a potom ich poslala na cieľovú stanicu. Následne ich tam vyložila, pričom je možné, aby tieto lietadlá od seba fungovali nezávislo. Môžu pracovať na oddelených lokalitách, vykonať potrebnú činnosť a zároveň sa navzájom neobmedzovať. Plán nájdený touto metódou je zároveň aj vzorovým a teda najefektívnejším riešením úlohy.

Pre HTN platí opäť to isté, a teda našla riešenie najrýchlejšie, ako je možné vidieť v tabuľke 6.4, no nejedná sa o ten najlepší plán. Definovanie domény v tom takisto zohráva svoju úlohu, preto v prípade inak definovaných zložených operátorov by bolo možné nájsť

STRIPS	Graphplan	HTN
5.04645s	2.46449s	0.00012s

Tabuľka 6.4: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Aircargo

efektívnejšie riešenie bez zbytočných presunov. Takisto je v tomto prípade možné zlepšiť jeho fungovanie lepším definovaním zozbierania objektov so spoločnou počiatočnou a cieľovou pozíciou. V tomto prípade ide o funkčné riešenie, ktoré je ale pri jeho vykonaní pomalšie a neefektívne. Pokiaľ je prioritou nájsť efektívne riešenie, oproti rýchlo nájdenému, tak je potrebná lepšia a podrobnejšia špecifikácia zadanej domény. Opäť sa bude jednať o väčšiu pomoc pre plánovač než pri ostatných metódach, ktoré takúto výhodu pri svojom fungovaní nemajú.

STRIPS našlo plán, ktorý pri rekurzii narazil na slučku, tak ako v prípade `Dock-Worker-Robot` a kvôli tomu do svojho riešenia pridal nepotrebné kroky. Ide o ten istý prípad ako v doméne `Dock-Worker-Robot` a postup riešenia problému tejto metódy je taký istý ako predtým spomínaný. Nejde o chybné fungovanie algoritmu, ale práve o jeho princíp, ktorý sa v tomto prípade preukazuje neefektívnym riešením, ktoré navyše aj trvá o niečo dlhšie ako pri ostatných metódach.

Z hľadiska samotných nájdených plánov je výsledok podobný tomu z `Dock-Worker-Robot` domény.

6.4.5 Dinner domain

Doména `Dinner` je klasickým príkladom úlohy s konfliktnými cieľmi. Jej cieľom je pripraviť večeru, zabaliť darček a upratať dom. Nie je potrebné dodržať presné poradie, ale jednotlivé akcie vzájomne rušia svoje efekty na doménu a preto je potrebné nájsť správne poradie ich použitia. V pôvodnej doméne sú niektoré ciele definované ako nepravdivé pri výstupe. Vzhľadom na predpoklad uzatvoreného sveta, ktorý bol už v práci definovaný, nie je možné nejaký predikát označiť na výstupe za nepravdivý. Preto bolo potrebné túto doménu upraviť, pričom bola zachovaná jej podstata a význam.

Pri aplikovaní niektorého operátora sa zruší aplikovateľnosť iného, alebo dôjde k zrušeniu predtým už platného finálneho predikátu a je potrebné opäť nájsť riešenie tohto predikátu. Preto v tejto úlohe môže dochádzať k opakovanému aplikovaniu toho istého operátora. Zaujímavou vlastnosťou tejto domény je to, že v nej nie je možné vytvoriť veľa rôznych úloh a to z toho dôvodu, že sa tu pracuje vždy práve s jedným objektom. Ten reprezentuje vykonanie nejakej činnosti alebo splnenie niektorej z úloh v doméne. Teda nejde o prácu s nejakým objektom, na ktorý sa nejako aplikujú operátory, ktoré zmenia jeho postavenie v doméne. Ide o reprezentáciu vykonanej akcie, ktorá nastala v doméne a je potrebné, aby niektoré stavy boli platné predtým, ako môže dojsť k vykonaniu iných. Napríklad v prípade operátora `cook(x)`, kedy výsledkom tohto operátora je večera, pred jeho vykonaním musí byť upratané a musí platiť predikát `clean(x)`. Po jeho aplikovaní sa predikát `clean(x)` stane neplatným a pokiaľ je v cieľovom stave tento predikát pravdivý, je potrebné opätovné jeho splnenie.

Pre HTN bolo v tomto prípade potrebné definovať dva zložené operátory. Ide o vyriešenie všetkých aktivít v doméne, a následne vyriešenie jednej konkrétnej z nich.

Zadanie domény v PDDL:

```
(define (domain birthday-dinner)
 (:requirements :strips :typing)
```

```
(:types  activity)
(:predicates (clean ?x) (dinner ?x)
    (quiet ?x) (present ?x) (garbage ?x))
 (:action cook
    :parameters (?x)
    :precondition (and (clean ?x))
    :effect (and (dinner ?x) (not (clean ?x))))
 (:action wrap
    :parameters (?x)
    :precondition (and (quiet ?x))
    :effect (and (present ?x)))
 (:action carry
    :parameters (?x)
    :precondition (and (garbage ?x))
    :effect (and (not (clean ?x))) )
 (:action dolly
    :parameters (?x)
    :precondition (and (garbage ?x))
    :effect (and (clean ?x))))
```

Zadaná úloha

Zadaná úloha je typickým problémom tejto domény, kde sú na začiatku riešenia platné predikáty `garbage(x)` a `quiet(x)`. Cieľovým stavom je, aby platili predikáty `present(x)`, `dinner(x)` a `clean(x)`. Ako bolo už vyššie spomenuté, niektoré operátory rušia platnosť predikátov, ktoré vyžadujú aby boli platné pred ich aplikovaním. Sledované je, akým spôsobom budú metódy schopné nájsť platnú postupnosť akcií, ktoré by zadanie splnili.

Porovnanie metód na zadanej úlohe

Prvou testovanou metódou bol STRIPS, ktorý nebol schopný nájsť riešenie úlohy. Ako prvé kroky sú splnené cieľové predikáty `present(x)` a následne `clean(x)`. Tie vedel správne splniť, pričom dovtedy nenarazil na žiadne problémy. Následne, pri snahe splniť predikát `dinner(x)` pomocou akcie `cook(x)` zrušil proces hľadania riešenia, pretože ide o jediný operátor spĺňajúci cieľový predikát, pričom jeho aplikovaním dôjde k zrušeniu cieľového predikátu `clean(x)`. Z toho dôvodu plánovač zrušil prehľadávanie, pretože nedovolí aplikovanie operátora, ktorý vyústi v predikát v priamom rozpore s cieľovými predikátmi a zároveň neexistuje iný aplikovateľný operátor.

STRIPS	Graphplan	HTN
-	0.00205s	0.0002s

Tabuľka 6.5: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Dinner

Plán nájdený metódami Graphplan HTN:

```
wrap(x), dolly(x), cook(x), dolly(x)
```

Graphplan a HTN našli rovnaký plán, ktorý je validný a spĺňa zadanú úlohu. Graphplan našiel čiastočne usporiadaný cieľ, pričom je možné zamieňať poradie niektorých krokov

bez vplyvu na finálny výsledok. Riešenie nájdené metódou HTN je rovnaké, pričom ide o úplne usporiadaný plán. V takomto type úlohy je možné hovoriť o zhruba podobnej efektivite pri týchto dvoch metódach, pričom HTN má opäť o niečo navrch v časovej efektivite, ako je možné pozorovať na výsledkoch v tabuľke 6.5.

Z tohto testovania je možné vyvodiť záver, že na príklady s konfliktnými cieľmi nie je vhodné využiť metódu STRIPS, pretože je pravdepodobné, že zastaví svoje riešenie pred jeho dokončením. Vychádza to z obmedzenia tejto implementácie, kedy neaplikuje operátor, ktorý by aplikoval operátor, ktorý vytvorí predikát konfliktný s nejakým cieľovým. V prípade, že by toto obmedzenie nebolo implementované, metóda nájde plán, ktorý avšak nemusí byť validný. Keďže v tejto konkrétnej úlohe je potrebné splnenie toho istého cieľového predikátu dvakrát, STRIPS po splnení cieľového predikátu tento vymaže a nezaoberá sa ním po jeho splnení. Preto by úlohu považoval za splnenú aj v prípade, kedy by tomu tak nebolo.

6.4.6 Competition blocks domain

Doména, ktorá je podobná `Simple Blocks`, sa vyskytuje na plánovacích súťažiach a bola inšpirovaná z tohto zdroju [12]. Ide o jej rozšírenie, ktoré obsahuje viac operátorov a takisto aj väčší počet predikátov, s ktorými je možné pracovať.

Úlohy v tejto doméne sú podobné, ako v `Simple Blocks`, pričom ide o rôzne zostavenia ľubovoľného počtu kociek, ktoré môžu ležať na stole alebo na inej kocke. Úlohou je z pôvodného zoskupenia kociek ich presun na zadané cieľové pozície. Táto doména obsahuje väčší počet predikátov, z čoho je možné usúdiť, že náročnosť riešenia príkladov v tejto doméne bude o značne väčšia. Metódy sa musia vysporiadať s väčším množstvom možných ciest a riešení, ktoré musia preskúmať.

Doména obsahuje štyri operátory, pričom ide o `pickup(block)` a `putdown(block)`, ktoré reprezentujú dvíhanie ľubovoľnej kocky zo stola. Ďalšie dva operátory sú `stack(block1, block2)` a `unstack(block1, block2)`, ktorých výsledkom je zdvihnutie kocky z inej kocky alebo položenie na ľubovoľnú kocku. Predikátov je v doméne 5, pričom popisujú polohu kocky, či je jej povrch voľný pre uloženie ďalšej kocky navrch a či rameno robota drží kocku alebo je voľné.

HTN má v tejto doméne definované 4 zložené operátory. Tak ako v predchádzajúcich úlohách, ako prvý je definovaný operátor pre spracovanie všetkých blokov. Ten sa rozkladá na spracovanie jedného konkrétneho bloku. Operátor pre jeden blok je ďalej možné rozložiť dvoma rôznymi spôsobmi. Ten prvý popisuje situáciu, kedy je blok potrebné zdvihnúť. Operátor pre túto situáciu potom priamo aplikuje jednoduchý operátor zdvihnutia, podľa aktuálnej pozície bloku, teda či sa nachádza na inom bloku alebo na stole. Druhý spôsob rozloženia je podobný, pričom sa zaoberá umiestnením bloku na nejaké miesto, v závislosti od jeho aktuálnej polohy.

Zadaná úloha

Úloha zadaná v tejto doméne je prestavanie veže 10 kociek do opačného poradia ako bola v pôvodnom rozložení. Jedná sa o problém, ktorý ukazuje schopnosť metód pracovať v zložitejšej doméne ako v ostatných úlohách.

Porovnanie metód na zadanej úlohe

Zadanú úlohu metóda STRIPS nie je schopná riešiť v primerane dlhom čase. Táto doména je pre metódu priveľmi rozsiahla a teda nevie nájsť riešenie v čase, ktorý by sa aspoň blížil ostatným metódam. Dôvodom je množstvo operátorov a objektov zavedených v doméne a úlohe. Z toho je možné vidieť, že na náročnejšie úlohy táto metóda nie je vhodným prostriedkom riešenia.

STRIPS	Graphplan	HTN
-	20.66623s	0.00003s

Tabuľka 6.6: Trvanie riešenia úlohy pre jednotlivé metódy v sekundách v doméne Competition Blocks

Riešenie metódou Graphplan je schopné nájsť validný plán, ktorý je aj najrýchlejším riešením. Ako je možné vidieť, čas potrebný na nájdenie riešenia je naozaj veľký a neporovnateľný oproti metóde HTN, ako ukazuje tabuľka 6.6, a takisto aj oproti dĺžke trvania pri ostatných testovaniach. Aj keď ide o zdanlivo jednoduchú úlohu, jej prehľadávací priestor, ktorý vzniká pri jej riešení v metóde Graphplan, je natoľko rozvetvený, že zaberie veľa času. Pri tejto metóde je potrebné mať na pamäti, že aj keď sú operátory na jednotlivých vrstvách nezávislé, spájajú sa do jedného pomyselného uzla. Aj napriek tomu je potrebné kontrolovať všetky možné mutex akcie a predikáty, ktoré by mohli nastať, a to už v prípade veľkého množstva objektov zaberie značný čas, ktorý sa ukáže na výsledkoch. Metóda Graphplan nie je vhodná na úlohy, kde sa pracuje s väčším množstvom objektov, alebo aj v prípade veľkého počtu operátorov a predikátov v úlohe.

HTN v tejto úlohe nemá problém nájsť riešenie. Keďže aj na prvý pohľad je možné vidieť, ako by mal prebiehať presun jednej kocky, definovanie tejto postupnosti v doméne prinesie obrovskú časovú výhodu pre jej výkonnosť. Nájdený plán je zhodný s tým, ktorý našiel Graphplan a je validný. Opäť sa tu ukazuje otázka, ktorá sprevádza tento typ plánovacích metód, a teda či nie je podrobná definícia domény príliš veľkou pomocou v riešení úloh. V prípade praktického aplikovania metódy je veľká výhoda, pokiaľ je možné ten istý problém vyriešiť omnoho rýchlejšie aj za cenu podrobného definovania toho problému. Pokiaľ sa bude jednať o doménovo závislé plánovanie, kedy je cieľom riešiť jednu konkrétnu doménu, je určite vhodné investovanie času do implementácie hierarchického plánovača.

6.5 Zhrnutie hodnotenia a porovnania metód na rôznych typoch úloh

Po testovaní všetkých metód na zvolených príkladoch je možné zhodnotiť ich vhodnosť a aplikovateľnosť na rôzne druhy problémov, ktoré boli demonštrované zadanými úlohami. Špecifické správanie metód bolo podrobne popísané v predchádzajúcich podkapitolách. Nasledujúca časť sa zaoberá celkovým zhodnotením vhodnosti metód na rôzne typy úloh.

Hodnotenie a analyzovanie metódy STRIPS

STRIPS ako metóda nie je implementačne náročná. Z výsledkov testovania je jasné, že práve to je jednou z jeho nevýhod a slabín. V prípade jednoduchých domén a úloh bol schopný nájsť riešenie a niekedy aj to najefektívnejšie. Vo viacerých prípadoch došlo k situácii, kedy nenašiel existujúce riešenie z dôvodu implementačných obmedzení. Stalo sa tak v prípade

domény `Dinner` a `Competition Blocks`. V každej z týchto domén sa prejavila iná jeho nevýhoda, ktorá zmenšuje skupinu domén, ktoré vie úspešne riešiť.

V prípade domény `Dinner`, ktorá bola ukážkou konfliktných cieľov, je v tejto implementácii zavedené obmedzenie aplikovania operátora, pokiaľ vytvorí predikát konfliktný s niektorým cieľovým predikátom. Samozrejme z pohľadu iných úloh ide o bezpečnostný prvok zabezpečujúci, že nebudú aplikované operátory, ktoré by aktuálny stav vzďaľovali od cieľového. V prípade domén s konfliktnými cieľmi však nastane nemožnosť riešenia takýchto úloh, preto na ne nie je vhodná metóda STRIPS.

Pri rozšírenej doméne `Competition Blocks` nastala iná situácia, kedy metóda nevedela v primeranom čase nájsť riešenie úlohy. Veľkosť domény a úlohy spôsobila zdanlivé zacyklenie metódy. V skutočnosti sa metóda pokúša o aplikovanie všetkých možností riešenia pre každý čiastočný cieľ. Z toho dôvodu je v náročnejších úlohách táto metóda nevhodná, keďže nie je možné spoľahlivo určiť, v akom čase sa metóde podarí nájsť riešenie.

Zaujímavá situácia nastala aj v prípade domén s čiastočne usporiadanými cieľmi, kedy nájdené plány boli funkčné, ale obsahovali kroky navyše. Preto nešlo o optimálne riešenia spomedzi skúmaných metód.

Pokiaľ je potrebné riešiť jednoduché domény, ktoré sami o sebe nemajú žiadne zvláštne prvky alebo špecifikácie, metóda STRIPS je vhodným prostriedkom na ich riešenie. Nie je potrebná komplikovaná implementácia, a metóda poskytne potrebné výsledky. Najlepšími príkladmi sú problémy, kde sa jedná o často opakované až priam mechanické úkony. I keď zvláda aj náročnejšie domény, keďže je tu možnosť nájdenia nie veľmi efektívneho finálneho plánu, je vhodnejšie ju využívať pri jednoduchých úlohách.

Hodnotenie a analýza metódy Graphplan

Z testovania a porovnávania metód je možné usúdiť, že metóda Graphplan je prijateľným kandidátom na riešenie širokého spektra domén. Bola schopná vyriešiť všetky testované úlohy, pričom v niekoľkých z nich našla ako jediná najefektívnejší plán. Časová náročnosť tejto metódy bola vo väčšine prípadov najväčšia, čo je pochopiteľné z jej fungovania. Čas potrebný na vytvorenie grafu celej zadanej úlohy, spolu s mutex operátormi a predikátmi je sám o sebe veľký. Podľa toho, ako náročná úloha je a čím viac objektov sa v nej nachádza, tým je tento čas väčší. Na druhej strane, vďaka precíznej príprave plánovacieho grafu, je možné nájsť najlepší možný plán, aký existuje. To pri ostatných metódach nie je vždy pravda.

Zlé výsledky bolo možné pozorovať najmä v úlohách, ktoré mali v zadaní väčší počet objektov. Išlo napríklad o `Competition Blocks` doménu, ktorá je dobrým príkladom domény, ktorú metóda vyrieši, ale nebude na ňu najvhodnejšia.

Naopak veľmi dobré výsledky bolo možné pozorovať pri úlohách s čiastočne usporiadanými cieľmi. Pri všetkých výsledkoch bolo vidno, že metóda nepotrebovala oveľa dlhší čas na nájdenie riešenia ako ostatné porovnávané a zároveň vždy našla lepší plán. Vrstvový spôsob, akým je tvorený plánovací graf, umožňuje práve nezávislé riešenie podcieľov. Vďaka tomu je možné nájsť v každej vrstve grafu niekoľko operátorov, pre ktoré platí, že každý z nich sa snaží o riešenie iného podcieľa a je možné ich aplikovať v ľubovoľnom poradí. Zároveň, aby daný podcieľ splnený bol, je potrebné medzi vrstvami dodržať postupnosť nájdených operátorov. Štruktúra plánovacieho grafu toto umožňuje a podporuje a vďaka tomu je Graphplan schopný nájsť efektívne a dobré plány, ktoré môžu byť čiastočne usporiadané. To je výhodné vo všetkých prípadoch, keďže v praxi pri vykonávaní nejakého čias-

točne usporiadaného plánu môže dojsť k šetreniu času strávenému čakaním na dokončenie predchádzajúceho kroku, ak je možné nejaký iný vykonávať v ten istý čas.

Metóda je takisto schopná riešiť úlohy s konfliktnými cieľmi, takže je možné ju využiť vo veľa rôznych doménach.

Pre jej najlepšie využitie sú ideálne domény s komplikovanou štruktúrou alebo úlohy s väčšími obmedzeniami. Metóda je schopná ich dobre riešiť a zároveň nájsť vhodné riešenie. Naopak je menej vhodná vo veľkých úlohách, kedy jej proces plánovania bude dlhší ako pri ostatných metódach a teda je vhodné zváženie jej využitia.

Hodnotenie a analýza metódy HTN

Hierarchické plánovanie využíva odlišný prístup ako predchádzajúce metódy a vlastnosť, ktorá už bola niekoľkokrát spomínaná, je potreba lepšej špecifikácie domény. Ide o potrebu omnoho podrobnejšieho definovania a uchopenia vstupnej domény. Tento fakt je zároveň výhodou aj nevýhodou pre využitie metódy. Pochopiteľne ide o dobrý nástroj, ktorý zvyšuje efektivitu nájdeného plánu a samozrejme znižuje čas na to potrebný. Podľa toho, ako dobre je špecifikovaná táto doména, je možné vylepšiť fungovanie tejto metódy, a tak s jej výkonnosťou v rôznych doménach manipulovať. Avšak je potrebné spomenúť, že niekedy potrebná špecifikácia pripomína doménovo závislé plánovače, ktoré sú schopné riešiť úlohy len v jednej konkrétnej doméne. Nejde o tak podrobne prispôsobené riešenie, avšak úroveň, do akej je efektívne uchopená doména a to, ako je možné narábať jednoduchými operátormi v nej, má veľký vplyv na výsledný plán a jeho vhodnosť. Zároveň ide o implementačne náročnú činnosť, kedy pre každú osobitnú úlohu je potrebné do hĺbky pochopiť jej fungovanie a predpokladať najlepší možný spôsob rozloženia úlohy na časti. Je možné teda povedať, že istú časť riešenia vytvorí samotný programátor pri definovaní domény. V prípade ostatných metód je potrebné poznať operátory a ich podmienky a zvyšná činnosť je ponechaná samotnej metóde na vyriešenie. Pri HTN je nevyhnutné špecifikovať proces rozkladania zložitých operátorov čo najefektívnejšie, aby implementácia touto metódou mohla správne fungovať. Jej najväčšia výhoda spočíva práve v bohatých informáciách o doméne v podobe hierarchie operátorov a pokiaľ nie sú dobre špecifikované, je možné nájsť nesprávne riešenia, prípadne sa zníži efektivita tejto metódy.

Potrebné je uviesť, že táto metóda sa najviac podobá ľudskému rozmýšľaniu, kedy je veľký problém rozložený na menšie, uchopiteľné úlohy, ktoré sú riešené samostatne. Vďaka tomu je táto metóda schopná riešiť komplexnejšie domény než ostatné metódy. Ukážková doména presunu na letisko je toho dobrým príkladom, kedy nejde o mechanickú činnosť, ale práve o rozkladanie problému. Pokiaľ je zadaná doména komplexná a je ťažké ju vyjadriť v obmedzenom formalizme ako je PDDL, prístup HTN metódy umožňuje riešenie aj takýchto úloh. Zároveň je možné riešiť klasické plánovacie úlohy, aké tu boli ukázané a testované. Vo výsledkoch je možné pozorovať, že celkovo sa tejto doméne darí produkovať efektívne alebo aspoň správne riešenia za veľmi krátky čas. V úlohách s čiastočným usporiadaním cieľov nájdené plány obsahovali nadbytočné kroky, ktoré ale boli spôsobené špecifikáciou domény. Tu je preto možné vidieť, ako môže definovanie domény pomôcť alebo uškodiť finálnemu riešeniu.

Pri jednoduchých úlohách je metóda HTN príliš silným nástrojom, pre ktorý je potrebná zbytočná práca navyše pri definovaní domény, pričom nájdenie plánu nebude zložitý proces. Pri testovaných úlohách bolo možné sledovať, že HTN prináša dobré výsledky, ale na úkor náročnej prípravy domény. Preto je táto metóda vhodnejšia na menej typické a mechanické úlohy. Rovnaký výsledok je možné dosiahnuť menej náročnou prípravou domény a inou me-

tódou. Prístup HTN je vhodné voliť pokiaľ sa jedná o zložitú úlohu, ktorú je možné rozložiť na nejaké menšie časti, pretože vtedy je aj definícia domény uľahčená a viac priamočiara ako hľadanie zloženého operátora aplikovateľného na zadaný problém.

Nie je možné popierať skvelé výsledky tejto domény, keďže má najvyrovnanejšiu úspešnosť spomedzi skúmaných metód. Aj v prípadoch, kedy našla menej efektívny plán, čas hľadania bol vždy medzi najmenšími. Preto na takmer ľubovoľnú úlohu bude možné upraviť doménu tak, aby ju táto metóda vedela vyriešiť. Pokiaľ však nie je možné tráviť čas špecifikáciou a hlbším pochopením úlohy, táto metóda nebude vhodná.

6.6 Zlepšenie fungovania jednotlivých metód

Je možné vidieť, že metódy majú v niektorých úlohách problém efektívne alebo správne nájsť riešenie. Okrem základného fungovania metód je možné ich implementáciu obohatiť a nejakým spôsobom kompenzovať ich nedostatky. V tejto práci neboli vylepšenia brané do úvahy pri porovnávaní a teda neboli ani využívané. Cieľom bolo porovnať základné metódy. Možnosť ďalšieho porovnávania a pozorovania metód je po aplikovaní zlepšovacích prvkov implementácie metód. Nižšie sú spomenuté dva konkrétne spôsoby, ktoré je možné využiť. V prvom rade ide o rozšírenie definície domény, v ktorej je možné jednotlivým objektom priradiť typ. Takisto je potom operátorom priradený typ pri definovaní parametrov. Teda každý parameter dostane typ, ktorým musí mať objekt, ktorý chce aplikovať namiesto daného parametra.

Napríklad na doméne `Simple Blocks` je možné objekty rozdeliť na bloky a na stoly. Toto priradenie je potrebné definovať v doméne a takisto v úlohe, kde konkrétne objekty s menom dostanú priradený typ. Túto informáciu môžu všetky metódy využiť na zlepšenie fungovania. Namiesto skúšania aplikovania všetkých objektov do všetkých parametrov, sú skúšané objekty daného typu na miesto príslušných parametrov. Príklad takejto domény v jazyku PDDL je možné vidieť nižšie.

```
(define (domain blocksworld)
  (:requirements :strips :typing)
  (:types block table)
  (:predicates
    (on ?b - block ?t - table))
  (:action move
     :parameters (?b - block ?t1 ?t2 - table)
     :precondition (and (on ?b ?t1) (not (on ?b ?t2)))
     :effect (and (on ?b ?t2) (not (on ?b ?t1)))))
```

Druhý návrh zlepšenia metód sú heuristiky [18], ktoré by vylepšovali ich fungovanie. V tomto prípade ide o celú škálu vylepšení, ktoré môžu pomôcť s rôznymi obmedzeniami jednotlivých metód. Hľadanie vhodných heuristík pre jednotlivé metódy je samo o sebe proces, kde je potrebné porovnávať a hľadať najlepšie riešenia. Je pritom potrebné nájsť takú heuristiku, ktorá je aplikovateľná na doménovo nezávislé riešenia metód a takú, ktorá pomôže kompenzovať slabé stránky metód.

Kapitola 7

Záver

Cieľom práce bolo naštudovanie, predstavenie a porovnanie klasických metód plánovania. V prvej časti práce bola predstavená problematika automatizovaného plánovania a takisto jednotlivé zvolené metódy. Bolo popísané ich fungovanie, ako aj ich vlastnosti a vzájomné rozdiely medzi nimi. Následne boli zvolené implementácie metód, ktoré reprezentovali jednotlivé prístupy, na ktorých boli potom porovnávané a analyzované úlohy. Na to bolo potrebné ich nájsť a definovať tak, aby ukázali vlastnosti a fungovanie metód na rôznych druhoch problémov.

Na zvolených úlohách boli testované všetky metódy a následne došlo k ich porovnaniu. Na konci testovania boli zhrnuté zistené poznatky o správaní metód pri rôznych problémoch. Bolo možné nájsť a na výsledkoch ukázať vopred známe problémy a nedostatky metód, ako aj analyzovať ich správanie. Pozorovania boli podložené samotným výsledným správaním metód pri testovaní. Zvolené úlohy boli rôznych rozsahových náročností, pričom bolo sledované správanie vzhľadom na tento faktor. Bolo možné identifikovať nedostatky a nevhodné použitie metód a takisto nájsť najlepšie typy úloh pre jednotlivé metódy.

Pri prípadnom ďalšom porovnávaní by bolo možné aplikovať vylepšenia jednotlivých metód, ktoré by kompenzovali známe nedostatky a tak zlepšili ich správanie. Takisto je možné do porovnania pridať ďalšie metódy, ktoré sú využívané a sledovať ich správanie oproti tu zvoleným metódam.

Literatúra

[1] Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Nau, D. S. et al. SHOP2: An HTN Planning System. *CoRR*. 1. vyd. 2011, abs/1106.4869, č. 1. Dostupné z: http://arxiv.org/abs/1106.4869.

[2] Becker, K. Artificial Intelligence Planning with STRIPS, A Gentle Introduction. *Primary Objects* [online]. Jún 2015 [cit. 2022-04-10]. Dostupné z: http://www.primaryobjects.com/2015/11/06/artificial-intelligence-planning-with-strips-a-gentle-introduction/.

[3] Blum, A. L. a Furst, M. L. Fast planning through planning graph analysis. *Artificial Intelligence*. 1. vyd. 1997, zv. 90, č. 1, s. 281–300. DOI: https://doi.org/10.1016/S0004-3702(96)00047-1. ISSN 0004-3702. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0004370296000471.

[4] contributors, W. *Sussman anomaly — Wikipedia, The Free Encyclopedia*. 2022. [Online; accessed 1-May-2022]. Dostupné z: https://en.wikipedia.org/w/index.php?title=Sussman_anomaly&oldid=1067921515.

[5] Coppin, B. *Artificial Intelligence Illuminated*. 1st. USA: JONES AND BARTLETT PUBLISHERS, 2004. ISBN 0-7637-3230-3.

[6] Fikes, R. E. a Nilsson, N. J. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*. 1st. 1971, zv. 2, č. 3, s. 189–208. DOI: https://doi.org/10.1016/0004-3702(71)90010-5. ISSN 0004-3702. Dostupné z: https://www.sciencedirect.com/science/article/pii/0004370271900105.

[7] Fox, M. a Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *CoRR*. 1. vyd. 2011, abs/1106.4561, č. 1. Dostupné z: http://arxiv.org/abs/1106.4561.

[8] Georgievski, I. a Aiello, M. HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*. 1. vyd. 2015, zv. 222, č. 222, s. 124–156. DOI: https://doi.org/10.1016/j.artint.2015.02.002. ISSN 0004-3702. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0004370215000247.

[9] Ghallab, M., Nau, D. S. a Traverso, P. *Automated planning and acting*. 1st. New York, NY: Cambridge university press, [2016]. ISBN 978-1-107-03727-4.

[10] Ghallab, M., Nau, D. S. a Traverso, P. *Automated planning: theory and practice*. 1st. San Francisco: Morgan Kaufmann, c2004. ISBN 1-55860-856-7.

[11] Höller, D., Behnke, G., Bercher, P., Biundo, S., Fiorino, H. et al. HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems. *Proceedings of the AAAI Conference on Artificial Intelligence*. 1. vyd. Apr. 2020, zv. 34, č. 06, s. 9883–9891. DOI: 10.1609/aaai.v34i06.6542. Dostupné z: https://ojs.aaai.org/index.php/AAAI/article/view/6542.

[12] ICAPS. Domains. *International Conference on Automated Planning and Scheduling* [online]. Október 2013 [cit. 2022-04-10]. Dostupné z: http://www.plg.inf.uc3m.es/ipc2011-learning/Domains.html.

[13] ICAPS. Welcome to ICAPS. *International Conference on Automated Planning and Scheduling* [online]. 2021 [cit. 2022-04-10]. Dostupné z: https://www.icaps-conference.org/.

[14] Kjellerstrand, H. *GitHub* [online]. August 2013 [cit. 2022-04-10]. Dostupné z: https://github.com/hakank/hakank/tree/master/pddl.

[15] Lallement, R., Silva, L. de a Alami, R. HATP: An HTN Planner for Robotics. *CoRR*. 1. vyd. 2014, abs/1405.5345, č. 1. Dostupné z: http://arxiv.org/abs/1405.5345.

[16] Lekavý, M. a Návrat, P. Expressivity of STRIPS-Like and HTN-Like Planning. In: Nguyen, N. T., Grzech, A., Howlett, R. J. a Jain, L. C., ed. *Agent and Multi-Agent Systems: Technologies and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 121–130. ISBN 978-3-540-72830-6.

[17] Lifschitz, V. ON THE SEMANTICS OF STRIPS. In: GEORGEFF, M. P. a LANSKY, A. L., ed. *Reasoning About Actions and Plans*. Morgan Kaufmann, 1987, s. 1–9. DOI: https://doi.org/10.1016/B978-0-934613-30-9.50004-4. ISBN 978-0-934613-30-9. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9780934613309500044.

[18] Minton, S., Bresina, J. L. a Drummond, M. Total-Order and Partial-Order Planning: A Comparative Analysis. *CoRR*. 1. vyd. 1994, abs/cs/9412103, č. 1. DOI: 10.48550/ARXIV.CS/9412103. Dostupné z: http://arxiv.org/abs/cs/9412103.

[19] Nau, D. Pyhop, version 1.2.2. *Bitbucket* [online]. Máj 2013. Revidované 7.8.2021 [cit. 2022-04-10]. Dostupné z: https://bitbucket.org/dananau/pyhop/src/master.

[20] Nau, D., Cao, Y., Lotem, A. a Munoz Avila, H. SHOP: Simple Hierarchical Ordered Planner. *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2*. 1. vyd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. 1999, č. 1, s. 968–973. IJCAI'99.

[21] Nau, D., Goldman, R. P. a Contributors. SHOP3 Manual. *GitHub* [online]. Máj 2013 [cit. 2022-04-10]. Dostupné z: ttps://shop-planner.github.io/.

[22] Nirwan, D. Planning Graph for AI Planning in Python. *GitHub* [online]. Február 2021 [cit. 2022-04-10]. Dostupné z: https://github.com/debbynirwan/planning_graph.

[23] Nirwan, D. *Improving Classical AI Planning Complexity with Planning Graph* [online]. Towards Data Science, február 2021 [cit. 2022-04-10]. Dostupné z:

https://towardsdatascience.com/improving-classical-ai-planning-complexity-with-planning-graph-c63d47f87018.

[24] Russell, S. J., Norvig, P. a Davis, E. *Artificial intelligence: a modern approach.* 3rd ed. Boston: Pearson, c2010. ISBN 978-0-13-207148-2.

[25] Sacerdoti, E. D. *A structure for plans and behavior.* SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER, august 1975. Dostupné z: https://apps.dtic.mil/sti/pdfs/ADA458657.pdf.

[26] Tansey, W. Stanford Research Institute Problem Solver (STRIPS). *GitHub* [online]. September 2011. Revidované 2.4.2020 [cit. 2022-04-10]. Dostupné z: https://github.com/tansey/strips.