Univerza v Ljubljani
Fakulteta za računalništvo in informatiko
Fakulteta za matematiko in fiziko

Polona Tomašič

Eksperimentalna primerjava algoritmov planiranja POP in GRAPHPLAN

diplomsko delo
na interdisciplinarnem univerzitetnem študiju
računalništva in matematike

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00045/2013
Datum: 04.02.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko ter Fakulteta za matematiko in fiziko izdaja naslednjo nalogo:

Kandidat: **POLONA TOMAŠIČ**

Naslov: **EKSPERIMENTALNA PRIMERJAVA ALGORITMOV PLANIRANJA POP IN GRAPHPLAN**

AN EXPERIMENTAL COMPARISON OF PLANNING ALGORITHMS POP AND GRAPHPLAN

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Najbolj znana pristopa k planiranju z delno urejenostjo sta POP in GRAPHPLAN. GRAPHPLAN, ki temelji na t.i. planirnih grafih, velja za časovno bolj učinkovitega. Po drugi strani pa je predstavitev planov, ki jih generira algoritem POP, včasih primernejša in bolj pregledna. Naloga je primerjati oba algoritma na izbranih domenah planiranja glede na njuno časovno zahtevnost. Pri tem uporabite kompaktni implementaciji teh algoritmov v prologu v 4. izdaji knjige I. Bratko, Prolog Programming for Artificial Intelligence. Obenem popravite program POP tako, da bo pravilno upošteval medsebojno izključevanje akcij. Preučite tudi možnosti za bolj učinkovito izvedbo teh programov.

Mentor:

akad. prof. dr. Ivan Bratko

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnerič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 1

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Polona Tomašič, z vpisno številko **63080453**, sem avtorica diplomskega dela z naslovom:

Eksperimentalna primerjava algoritmov planiranja POP in GRAPHPLAN

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom akad. prof. dr. Ivana Bratka,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 14. maj 2013 Podpis avtorja:

Zahvaljujem se mentorju, akad. prof. dr. Ivanu Bratku, za predloge in strokovno pomoč pri izdelavi diplomske naloge ter za vse znanje, ki sem ga od njega prejela v času študija.

Zahvala gre tudi družini in fantu Klemenu za vso podporo in spodbudo tekom študija in v času nastajanja diplomske naloge.

Kazalo

Povzetek

Planiranje je področje umetne inteligence in predmet številnih raziskav že več kot štiri desetletja. Z gradnjo zaporedij akcij, ki privedejo do želenega cilja, zvišuje avtonomijo in fleksibilnost pametnih sistemov.

V tej diplomski nalogi smo se ukvarjali z dvema dobro znanima pristopoma k planiranju z delno urejenostjo. Algoritem GRAPHPLAN, ki velja za enega najučinkovitejših planerjev, zgradi t.i. planirni graf po metodi veriženja naprej. Na drugi strani algoritmi za planiranje z delno urejenostjo, med katere sodi tudi algoritem POP, uporabljajo veriženje nazaj. Njihova glavna prednost je v tem, da nikoli ne preizkušajo akcij, ki nimajo vpliva na dosego končnega cilja. Na podlagi praktičnih poskusov na izbranih domenah planiranja z več roboti smo pokazali, da je algoritem GRAPHPLAN učinkovitejši od algoritma POP. Pri tem smo uporabili kompaktni implementaciji teh dveh algoritmov v prologu v 4. izdaji knjige I. Bratko, *Prolog Programming for Artificial Intelligence* [4]. Program POP je bilo potrebno popraviti tako, da pravilno upošteva medsebojno izključevanje akcij. Predlagali smo dva načina odpravljanja te težave. Preučili smo tudi možnosti izboljšav in razširitev teh dveh pristopov k planiranju z delno urejenostjo: vmesni cilji, dvosmerno iskanje, ...

Ključne besede: planiranje, algoritem za planiranje GRAPHPLAN, algoritem za planiranje POP, delno urejeni plani, domene planiranja z več roboti

Abstract

Planning has been an area of research in artificial intelligence for over four decades. It increases autonomy and flexibility of intelligent systems through the construction of sequences of actions to achieve their goals.

In this thesis we take a look at two well known approaches to partial-order planning. The GRAPHPLAN system, which is one of the most efficient planning systems, builds a "planning graph" in a forward chaining manner. On the other hand, partial-order planners, such as POP, are goal driven. Their significant advantage over forward-chaining is that they never consider actions that are not relevant to the goal. We provide empirical evidence in favor of algorithm GRAPHPLAN, showing that it outperforms the partial-order planner, POP, on a variety of planning problems. For these two approaches we used implementations in Prolog, that can be found in Bratko's *Prolog Programming for Artificial Intelligence*, 4^{th} edition [4]. We tested them in a multi-agent planning domains. The algorithm POP needed to be fixed in order to handle correctly mutually exclusive actions. We proposed two different approaches for fixing that problem. We also considered different options for extending and performance enhancing the original algorithms: intermediate goals, bidirectional search in GRAPHPLAN, ...

Keywords: planning, planning algorithm GRAPHPLAN, planning algorithm POP, partially ordered plans, multi-agent planning systems

Poglavje 1

Uvod

Planiranje ima dolgo zgodovino v umetni inteligenci. Osnovni problem planiranja je sestavljen iz začetnega stanja, ciljnih pogojev in možnih akcij. Naloga je poiskati zaporedje akcij, ki nas iz začetnega stanja pripelje v končno stanje, v katerem so izpoljneni vsi ciljni pogoji.

Obstaja mnogo različnih algoritmov za planiranje. Nekateri temeljijo na preiskovanju prostora stanj z veriženjem naprej in možno uporabo hevristike, drugi na metodi sredstev in ciljev z veriženjem nazaj. V okviru diplomske naloge smo naredili kratek pregled izbranih algoritmov. Osredotočili smo se na dva pristopa k planiranju z delno urejenostjo. To sta POP (angl. *Partial-Order Planner*) in GRAPHPLAN. Slednji temelji na t. i. planirnih grafih. Njuno delovanje smo testirali na izbranih domenah planiranja z več roboti. Pri tem smo uporabili kompaktni implementaciji teh algoritmov v prologu v 4. izdaji knjige I. Bratko, Prolog Programming for Artificial Intelligence [4]. Program POP smo popravili tako, da pravilno upošteva medsebojno izključevanje akcij. Napako v programu smo našli s pomočjo podrobne analize primerov, ko program ni vrnil prave rešitve. Preučili smo tudi možnosti izboljšav in razširitev teh dveh pristopov planiranja. Napisali smo programa za dve različni možni pohitritvi algoritmov. Rezultati testiranj so pokazali v prid algoritmu GRAPHPLAN, ki tudi v splošnem velja za učinkovitejšega. Kljub temu ima program POP svoje prednosti. V primeru sveta z veliko nepomembnimi podatki, ki ne vplivajo na končni cilj, se POP zaradi veriženja nazaj

lahko izkaže veliko bolje od GRAPHPLANA. Slednji uporablja veriženje naprej, kar pomeni, da upošteva vse nepomembne podatke, zaradi katerih se planirni graf hitro povečuje.

Struktura diplomskega dela:

- V Poglavju 2 na kratko opišemo izbrane domene planiranja z več roboti. To so svet kock, pravokotna mreža, pravokotna mreža z ovirami, skladišče s paletami ter skladišče s paletami in vozički.
- V Poglavju 3 predstavimo razliko med planiranjem s preiskovanjem prostora stanj ter planiranjem po principu sredstev in ciljev. V nadaljevanju opišemo izbrane algoritme za planiranje.
- V Poglavju 4 opišemo dve verziji popravljenega programa POP.
- V Poglavju 5 naredimo pregled možnih izboljšav in razširitev algoritmov GRAPHPLAN in POP.
- V Poglavju 6 predstavimo rezultate testiranj izbranih algoritmov na enostavnih domenah planiranja.
- V Poglavju 7 predstavimo rezultate testiranj algoritmov GRAPHPLAN in POP na zahtevnejših domenah planiranja. Podamo tudi rezultate testiranja dometa teh dveh algoritmov na posebnih primerih domene pravokotna mreža. Na koncu naredimo primerjavo algoritmov z verzijo, ki uporablja vmesne cilje.
- V Poglavju 8 sledijo sklepi diplomskega dela in možno nadaljnje delo.

Poglavje 2

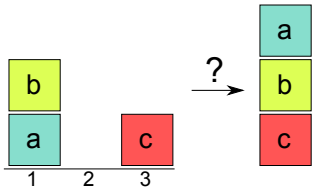
Definicije izbranih domen planiranja z več roboti

Za potrebe testiranja zmogljivosti različnih algoritmov planiranja smo v programskem jeziku prolog definirali različne domene. Začeli smo z enostavnimi, kot so svet kock, roboti na pravokotni mreži in roboti na pravokotni mreži z ovirami. Kasneje smo podrobneje analizirali delovanje dveh algoritmov, in sicer POP in GRAPHPLAN. Ta dva smo testirali tudi na dveh nekoliko bolj zahtevnih domenah z roboti.

2.1 Enostavne domene

2.1.1 Svet kock

Imamo kocke, ki so označene z malimi črkami abecede. Mesta v prostoru so oštevilčena. Želimo, da kocke navidezni robot prestavi iz začetnega stanja v želeno končno stanje. Če ima kocka, ki jo želimo prestaviti, na sebi neko drugo kocko, moramo najprej prestaviti slednjo. Primer začetnega in končnega stanja vidimo na sliki 2.1. Začetno stanje bi v tem primeru zapisali kot seznam: `[on(a,1), on(b,a), on(c,3), clear(b), clear(2), clear(c)]`. Cilje zapišemo na enak način: `[on(a,b), on(b,c)]`.



Slika 2.1: Primer začetnega in končnega stanja v svetu kock.

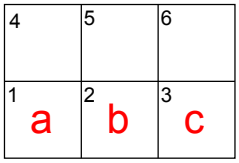
Stanje sveta je opisano z literali oblike:

- `on(A,B)`, kar pomeni, da je kocka `A` na kocki ali prostoru `B`;
- `clear(A)`, kar pomeni, da na kocki `A` ni nobene druge kocke oziroma, da je prostor `A` prazen.

Edina možna oblika akcij v tej domeni je `move(A,B,C)`, ki pravi, da kocko `A` prestavimo s kocke/prostora `B` na kocko/prostor `C`.

2.1.2 Roboti na pravokotni mreži

Imamo robote označene z malimi črkami in polja mreže označena s številkami. Roboti se lahko premikajo po sosednjih poljih mreže v smereh gor, dol, levo in desno. Želimo, da se roboti prestavijo iz začetnega stanja v želeno končno stanje. Začetno stanje s slike 2.2 zapišemo kot seznam: `[at(a,1), at(b,2), at(c,3), clear(4), clear(5), clear(6)]`.



Slika 2.2: Roboti `a`, `b` in `c` na pravokotni mreži velikosti 3x2.

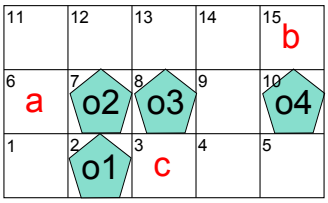
Stanje sveta je opisano z literali oblike:

- `at(A,B)`, kar pomeni, da je robot `A` na polju `B`;
- `clear(A)`, kar pomeni, da je polje `A` prazno.

Edina možna oblika akcij v tej domeni je `move(A,B,C)`, ki pravi, da se robot `A` prestavi s polja `B` na polje `C`.

2.1.3 Roboti na pravokotni mreži z ovirami

Definicija domene je identična pravokotni mreži, s to razliko, da v prostor dodamo ovire. Robot mora ovire na svoji poti obiti.

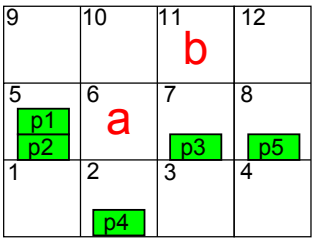


Slika 2.3: Roboti `a`, `b` in `c` ter ovire `o1, ..., o4` na pravokotni mreži velikosti 5x3.

2.2 Zahtevnejše domene

2.2.1 Skladišče s paletami

Imamo robote na pravokotni mreži. Celice mreže so označene s številkami. Roboti so označeni z malimi črkami abecede in vsak zaseda eno celico na mreži. Lahko se premikajo na sosednja polja v smereh levo, desno, gor in dol. Na določenih celicah mreže so postavljene palete, ki jih je potrebno prenesti na določene druge celice. Robot lahko nase naloži paleto s sosednje celice in se skupaj z njo tudi premika po mreži. Prav tako lahko paleto odloži na sosednjo celico. Na eni celici so hkrati lahko naložene največ tri palete.



Slika 2.4: Robota `a` in `b` ter palete `p1,...,p5` na pravokotni mreži velikosti 4x3.

Stanje sveta je opisano z literali oblike:

- `at(A,B)`, kar pomeni, da je robot ali paleta `A` na celici `B`;
- `on(P,R)`, kar pomeni, da ima robot `R` nase naloženo paleto `P`;
- `empty(R)`, kar pomeni, da robot `R` nima naložene nobene palete;
- `num_pal(C,N)`, kar pomeni, da celica `C` vsebuje `N` palet.

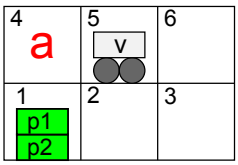
Možne oblike akcij v tej domeni:

- `move(R,C1,C2)`, kar pomeni, da se robot `R` premakne s celice `C1` na sosednjo celico `C2`;
- `pick(R,P,C1,C2,N_Before,N_After)`, kar pomeni, da robot `R`, ki je na celici `C2`, nase naloži paleto `P`, ki je na celici `C1`. Pri tem se število palet na celici `C1` spremeni iz `N_Before` na število `N_After`;
- `drop(R,P,C1,C2,N_Before,N_After)`, kar pomeni, da robot `R`, ki je na celici `C1`, odloži paleto `P` na celico `C2`. Pri tem se število palet na celici `C2` spremeni iz `N_Before` na število `N_After`.

2.2.2 Skladišče s paletami in vozički

V tej domeni skladišču s paletami dodamo vozičke. Definicija domene je enaka prejšnji s to razliko, da roboti lahko v voziček naložijo do tri palete in jih transportirajo po prostoru s potiskanjem vozička. Nato palete ponovno dvignejo iz vozička nase. Gre za dodatek, ki naj bi olajšal delo robotom in s tem zmanjšal število potrebnih akcij za dosego cilja. Da robot lahko naloži paleto v ali iz vozička ter da

lahko voziček potiska po prostoru, morata biti celici robota in vozička sosednji.



Slika 2.5: Robot `a`, paleti `p1` in `p2` ter voziček `v` na pravokotni mreži velikosti 3x2.

Prostorskim literalom iz prejšnje domene dodamo še dva:

- `in(P,V)`, kar pomeni, da je paleta `P` v vozičku `V`;
- `num_pal_trol(V,N)`, kar pomeni, da voziček `V` vsebuje `N` palet.

Akcijam iz prejšnje domene dodamo še tri nove:

- `upload(R,P,V,C1,C2,N_Before,N_After)`, kar pomeni, da robot `R`, ki je na celici `C1`, odloži paleto `P` v voziček `V`, ki je na celici `C2`. Pri tem se število palet v vozičku spremeni iz `N_Before` na število `N_After`;
- `download(R,P,V,C1,C2,N_Before,N_After)`, kar pomeni, da robot `R`, ki je na celici `C2`, dvigne paleto `P` iz vozička `V`, ki je na celici `C1`. Pri tem se število palet v vozičku spremeni iz `N_Before` na število `N_After`;
- `push(R,V,C1,C2,C3)`, kar pomeni, da robot `R`, ki je na celici `C1`, potisne voziček `V`, ki je na celici `C2`, na celico `C3`. Pri tem se robot prestavi na celico `C2`.

Poglavje 3

Planiranje

Planiranje je načrtovanje zaporedij akcij, ki iz začetnega stanja privedejo do končnega, v katerem so izpolnjeni vsi ciljni pogoji. Z izvedbo akcije spremenimo trenutno stanje. Ne spremenimo ga v celoti, ampak le določene komponente. Stanje sveta opišemo s seznamom prostorskih literalov, ki so v danem trenutku veljavni. Prostorski literali opisujejo relacije med objekti v prostoru. Vsaka akcija je definirana s predpogoji in učinki. Predpogoji so relacije, ki morajo biti veljavne, da se akcija lahko izvede. Akcija ima dve vrsti učinkov. Prve so relacije, ki po izvedbi akcije postaneje resnične, druge se po izvedbi uničijo.

Planiranje v širšem smislu pomeni reševanje problemov v prostoru stanj. V ožjem smislu nanj gledamo kot na planiranje po principu *sredstev* in *ciljev* (angl. *Means-ends planning*). V prvem primeru skonstruiramo preiskovalno drevo. Iz začetnega vozlišča, ki ustreza začetnemu stanju, izpeljemo naslednike. To počnemo, dokler ne pridemo do ciljnega vozlišča, ki ustreza ciljnim pogojem. Pri planiranju po principu sredstev in ciljev si najprej postavimo cilj, ga ustrezno zapišemo in nato poiščemo akcije, ki pripeljejo do tega cilja. Pri tem je potrebno paziti, da so pred izvajanjem določenih akcij izpolnjeni vsi predpogoji.

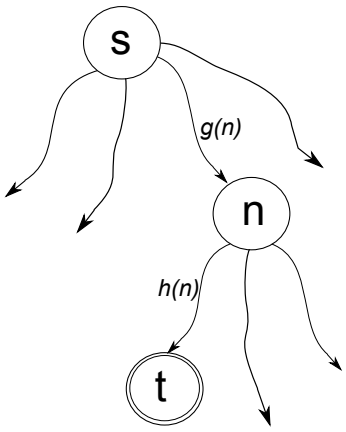
Metode planiranja, opisane v tem poglavju, sledijo opisom metod v knjigi [4], razen v primerih, ko je to posebej navedeno.

3.1 Hevristično preiskovanje

Problem je podan s prostorom stanj, začetnim stanjem in končnimi pogoji. Slednjim ustrezajo ciljna vozlišča. Prostor stanj je usmerjen graf, katerega vozlišča predstavljajo različna stanja prostora in povezave predstavljajo možne akcije. Problem planiranja prevedemo na iskanje poti v tem grafu.

3.1.1 Najprej najboljši (Best-first search)

Po grafu preiskujemo v najbolj obetavni smeri. Vsak premik iz vozlišča v njegovega naslednika ima določeno ceno. Naj bo `s` začetno vozlišče in `t` ciljno vozlišče. Pri gradnji drevesa, katerega koren je vozlišče `s`, vozlišča ocenimo s hevristično funkcijo f. Ta vsakemu vozlišču priredi neko realno število. Funkcija f je izbrana tako, da imajo bolj obetavna vozlišča nižjo vrednost. $f(n)$ je ocena cene najboljše rešitve, tj. cena poti od začetnega vozlišča `s` do ciljnega vozlišča `t`, ki gre skozi vozlišče `n`. Zapišemo jo kot vsoto $f(n) = g(n) + h(n)$. Pri tem je $g(n)$ cena optimalne poti od `s` do `n`, $h(n)$ pa ocena cene optimalne poti od `n` do `t`, kar je razvidno s slike 3.1. Za $g(n)$ vzamemo kar ceno poti med `s` in trenutnim vozliščem `n`. $h(n)$ je resnično hevristična funkcija, saj del sveta med `n` in `t` še ni raziskan.



Slika 3.1: Predstavitev funkcij $g(n)$ in $h(n)$ za vozlišče `n`.

Proces iskanja najcenejše poti je sestavljen iz množice med seboj tekmujočih podprocesov, kjer vsak raziskuje svojo pot. Med vsemi podprocesi je naenkrat aktiven le en - tisti, ki trenutno preiskuje najobetavnješo pot, tj. pot z najmanjšo vrednostjo funkcije f. Ko f ocena trenutnega podprocesa ni več najboljša, se aktivira podproces, ki je v tistem trenutku najobetavnejši.

3.1.2 Algoritem A*

Algoritem A* je najbolj znan algoritem v umetni inteligenci in se uporablja za iskanje poti v grafu. Vrne najcenejšo pot od začetnega vozlišča do enega izmed ciljnih vozlišč. Je nadgradnja Dijkstrinega algoritma in ima zaradi uporabe hevristike v primerjavi z Dijkstrinim algoritmom manjšo časovno zahtevnost.

A* temelji na prioritetnem preiskovanju. V grafu se usmeri na pot s trenutno najmanjšo hevristično ceno. Ves čas hrani urejeno prioritetno vrsto s cenami alternativnih poti. Algoritem uporablja hevristično oceno "razdalja + cena", katero označimo z $f(x)$. Ali algoritem najde optimalno pot, je odvisno od hevristike.

Izrek 3.1 (Hart, Nilsson, Raphael - 1968) *A* je popoln, če za vsa vozlišča n v prostoru stanj velja:*

$$h(n) \leq h^*(n),$$

kjer je $h^*(n)$ cena optimalne rešitve od n do cilja, $h(n)$ je ocena $h^*(n)$.

Preiskovalni algoritem je popoln, če v vsakem primeru najde optimalno rešitev. Ta je zagotovljena, če velja izrek o popolnosti.

3.2 Metoda sredstev in ciljev s popolnoma urejenimi plani

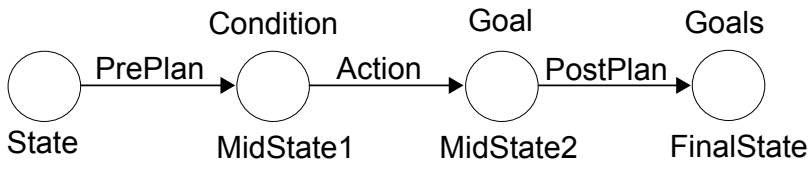
Pri metodi sredstev in ciljev se osredotočimo na cilj in poiščemo zaporedje akcij, ki privedejo do cilja. Sredstva so razpoložljive akcije za doseganje ciljev.

Poznamo začetno stanje `State` in seznam ciljev `Goals`. S `FinalState` označimo

stanje, v katerem so vsi cilji doseženi.

POSTOPEK:
Če so vsi cilji iz `Goals` resnični v stanju `State`, potem je `FinalState` kar enak stanju `State`. Če ne, se držimo naslednjih korakov:

1. Izberemo še ne uresničen cilj `Goal` iz seznama ciljev `Goals`.
2. Poiščemo akcijo `Action`, ki kot posledico doda `Goal` v trenutno stanje.
3. Da lahko izvedemo akcijo `Action`, moramo zagotoviti veljavnost predpogojev `Condition` (`PrePlan`), kar nas privede v stanje `MidState1`.
4. V stanju `MidState1` izvedemo akcijo `Action` in preidemo v stanje `MidState2`, v katerem je cilj `Goal` izpolnjen.
5. V stanju `MidState2` se lotimo izpolnjevanja preostalih ciljev s seznama `Goals` (`PostPlan`), kar nas privede v ciljno stanje `FinalState`.



Slika 3.2: Princip planiranja po metodi sredstev in ciljev (STRIPS).

3.2.1 Ščitenje ciljev

Izkaže se, da v bolj kompleksnih domenah metoda sredstev in ciljev ne zagotavlja optimalnega plana za reševanje problema. Algoritem ima velikokrat na izbiro več različnih akcij za dosego istega cilja. Več možnosti pomeni večjo kombinatorično zahtevnost. Pri tem obstaja nevarnost, da že dosežen cilj tekom reševanja preostalih ciljev uničimo. Algoritem se lahko celo zacikla.

Da bi se ognili tem problemom, se držimo pravila: "ne podri tistega, kar si že naredil". To lahko dosežemo s hranjenjem seznama ciljev, ki so že doseženi, in z izogibanjem uporabe akcij, ki uničijo že dosežene cilje.

3.2.2 Regresija ciljev

S ščitenjem ciljev močno izboljšamo delovanje algoritmov, vendar s tem niso odpravljene vse pomankljivosti. Problem leži v lokalnosti doseganja ciljev, kar nam mnogokrat onemogoča poiskati najkrajši rešitveni plan. Temu rečemo tudi *Sussmanova anomalija*. Planiranje z regresijo ciljev pomeni, da se problema lotimo globalno - poskušamo doseči vse cilje vzporedno, kar omogoča iskanje optimalnih planov.

IDEJA:
Naj bo `Goals` seznam vseh ciljev, ki jih želimo doseči v nekem stanju `S`. Izberemo enega izmed ciljev in poiščemo akcijo `A`, s katero bi dosegli ta cilj. Naj bo `S0` stanje tik pred stanjem `S`, torej z akcijo `A` v stanju `S0` dosežemo stanje `S`, v katerem so doseženi vsi cilji iz `Goals`.

Vprašanje je, kaj vse bi moralo veljati v stanju `S0`, da bi bili po izbrani akciji `A` vsi cilji doseženi. Iščemo seznam ciljev `Goals0` v stanju `S0`, za katere mora veljati:

1. Akcija `A` mora biti izvedljiva v stanju `S0`, torej morajo biti predpogoji akcije `A` vključeni v `Goals0`.
2. Za vsak cilj `G` iz `Goals` mora veljati ali
   - akcija `A` doda `G` v `Goals` ali
   - cilj `G` je v `Goals0` in ga akcija `A` ne izbriše.

Regresija ciljev iz končnih ciljev, pogojev za akcijo in njenih učinkov izračuna nove cilje. Če uresničimo vse nove cilje, z izbrano akcijo dosežemo vse prejšnje cilje. Postopek rekurzivno ponavljamo, dokler ne dobimo množice ciljev, ki so izpolnjeni že v začetnem stanju.

Algoritem na i-tem nivoju:

1. če so cilji `Goals(i)` izpolnjeni v začetnem stanju, končamo in izvedemo akcije v obratnem vrstnem redu;
2. sicer izberemo cilj iz `Goals(i)`, poiščemo akcijo `A`, ki doda ta cilj in regresiramo cilje po formuli:
   `Goals(i+1) = Goals(i)` $\cup$ `Cond(A) - Adds(A)`

Včasih se zgodi, da določenih kombinacij ciljev zaradi protislovja ni mogoče doseči. V takem primeru poskušamo najti rešitev po drugi poti. Paziti je potrebno tudi na to, da akcija `A` ne izbriše trenutnih ciljev - veljati mora `del(A)` $\cap$ `Goals(i) = []`. Za zagotavljanje najkrajše rešitve se lahko uporablja iterativno poglabljanje.

3.3 Kombinacija metode sredstev in ciljev z regresijo ciljev ter algoritma A*

Metoda sredstev in ciljev z regresijo ciljev je neinformirana metoda, kar pomeni, da pri izbiri alternativ ne upošteva nobenih lastnosti domene. Obstaja način, ki v to metodo vpelje neko hevristiko, tako da algoritem deluje po principu prioritetnega iskanja. Potrebno je definirati:

1. relacijo naslednika v prostoru stanj `s(Vozlišče1, Vozlišče2, Cena)`,
2. ciljna vozlišča z relacijo `goal(Vozlišče)`,
3. hevristično funkcijo `h(Vozlišče, HOcena)` in
4. začetno vozlišče preiskovanja.

Vozlišča v prostoru stanj predstavljajo množice ciljev. Zanje velja:

1. Vozlišče `Cilji2` je naslednik vozlišča `Cilji1`, če obstaja akcija `A`, za katero velja:
   - `A` doda nek cilj v `Cilji1`,
   - `A` ne izbriše nobenega cilja iz `Cilji1` in
   - z regresijo ciljev `Cilji1` po akciji `A` dobimo novo množico ciljev `Cilji2`.
2. Ciljna vozlišča so tista, katerih cilji so izpolnjeni že v začetnem stanju sveta.

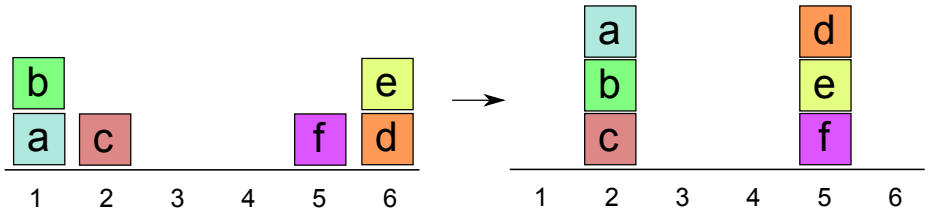
3. Za hevristično funkcijo vzamemo število še ne doseženih ciljev vozlišča. To je precej groba ocena, vendar algoritem deluje veliko bolje, kot če hevristike ne bi bilo.
4. Začetno vozlišče preiskovanja je množica ciljev, ki jih želimo doseči.
5. Cena vseh povezav je 1.

3.4 Delno urejeni plani (POP)

Začetki nelinearnega planiranja - v smislu istočasnega izvajanja neodvisnih akcij - segajo v leto 1975. Takrat sta Earl Sacerdoti in Austin Tate predstavila vsak svoj algoritem za delno urejeno planiranje. Planiranje na osnovi programov NOAH [9] in NONLIN [10] se je obdržalo 20 let, vse do razvoja GRAPHPLANA.

Planiranje po principu sredstev in ciljev upošteva vsa možna zaporedja akcij in vrača popolnoma urejene plane. Alogritem POP (angl. *Partial-Order Planner*) upošteva, da so nekatere akcije med seboj neodvisne in da vrstni red izvajanja teh akcij ni pomemben. POP planira s pomočjo regresije ciljev. Gradi prostor delno urejenih planov in išče vozlišče, v katerem so vsi cilji in predpogoji akcij uresničeni. Pri gradnji rešitvenega plana upošteva, da so nekatere akcije med seboj neodvisne in se lahko izvedejo istočasno.

Primer iz sveta kock s slike 3.3:



Slika 3.3: Primer problema iz sveta kock, ki se ga da rešiti z delno urejenimi plani.

Cilj je zgraditi dva stolpca iz dveh ločenih kupov kock. Stolpca sta lahko zgrajena neodvisno drug od drugega. Za vsak stolpec kreiramo en plan:

- `Plan1=[move(b,a,c), move(a,1,b)]`

- `Plan2=[move(e,d,f), move(d,6,e)]`

Ta dva plana sta med seboj popolnoma neodvisna. Pomembna sta le vrstna reda akcij znotraj vsakega plana posebej. Ni pomembno, ali se najprej izvede `Plan1` ali `Plan2`. Lahko se njuno izvajanje celo izmenjuje.

Načeloma algoritmi za planiranje vračajo celoten vrstni red izvajanja akcij v planu. Zato bi v našem primeru preiskali vseh 24 permutacij teh štirih akcij. V resnici so pomembne le 4 možnosti - 2 permutaciji za vsakega od planov. To dejstvo upoštevajo algoritmi z delno urejenimi plani, ki v posebnih primerih, ko vrstni red akcij ni pomemben, tega pustijo nedefiniranega. Tako rešitveni plani namesto popolnoma urejenih zaporedij akcij vsebujejo delno urejene množice akcij.

Vsak delno urejeni plan je definiran z:

- množico akcij {$\mathtt{A}_i$,$\mathtt{A}_j$, ...},
- množico omejitev (omejitve tipa `before(A1,A2)`, kar pomeni, da se mora akcija `A1` zgoditi pred akcijo `A2`) in
- množico vzročnih povezav.

Vzročne povezave so oblike `causes(`$\mathtt{A}_i$`,P,`$\mathtt{A}_j$`)`. Njihova interpretacija je, da akcija $\mathtt{A}_i$ omogoči pogoj `P` za akcijo $\mathtt{A}_j$. Akcija $\mathtt{A}_i$ se mora zgoditi pred $\mathtt{A}_j$. Namen uporabe vzročnih povezav je zaščita pogoja `P` v časovnem intervalu med akcijama `A` in `B`. Če je kakšna akcija v konfliktu z vzročno povezavo `causes(A,P,B)`, jo je potrebno izvesti pred `A` ali po akciji `B`. Akcija je v konfliktu, kadar negira pogoj `P`.

Plan je *konsistenten*, če ne obstajajo cikli znotraj omejitev in ni konfliktov med akcijami in vzročnimi povezavami.

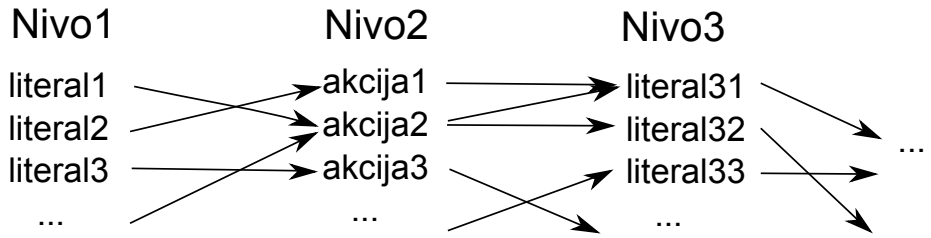
3.5 Planirni grafi (GRAPHPLAN)

GRAPHPLAN še danes velja za enega najučinkovitejših algoritmov za generiranje delno urejenih planov. Zasnovala sta ga Američana Avrim Blum in Merrick Furst leta 1995 [2]. Metoda GRAPHPLAN generira delno urejene plane, ki vsebujejo

akcije, razvrščene po nivojih. Akcije na istem nivoju lahko izvedemo v poljubnem vrstnem redu ali istočasno.

Naj bodo za dosego določenega cilja potrebne akcije `A1, A2` in `A3`. Pri tem naj velja, da se mora akcija `A1` zgoditi pred akcijo `A3`. GRAPHPLAN te tri akcije razvrsti na dva nivoja. Možna sta dva plana: `Plan1=[[A1,A2],[A3]]` in `Plan2=[[A1],[A2,A3]]`. Prvi plan pravi, da najprej izvedemo akciji s prvega seznama, in sicer v poljubnem vrstnem redu ali hkrati, nato še akcijo `A3`. `Plan2` pravi, da najprej izvedemo akcijo `A1`, nato v poljubnem vrstnem redu ali hkrati še preostali dve akciji.

Algoritem GRAPHPLAN uporablja planirne grafe. V grafu so vozlišča organizirana po nivojih. Prvi nivo vsebuje prostorske literale, ki opisujejo začetno stanje. Drugi nivo predstavlja vse akcije, katerih predpogoji so izpolnjeni na prvem nivoju. Tretji nivo predstavlja vse možne učinke akcij iz drugega nivoja. Tako si v urejenem časovnem zaporedju izmenjujoče sledijo nivoji literalov in akcij, kar je razvidno s slike 3.4.



Slika 3.4: Zgradba planirnega grafa.

Poleg običajnih akcij obstajajo še *virtualne akcije*, ki ohranjajo prostorske literale z enega na drug nivo. Če obstaja literal, ki je resničen na nekem nivoju in nanj nima vpliva nobena akcija z naslednjega akcijskega nivoja, potem bo ta literal resničen tudi na naslednjem nivoju, ki opisuje novo stanje. Akcija, ki ohranja tak literal `P` je `persist(P)` in ima za učinek spet literal `P`.

Nekonsistentna literala (ang. *inconsistent literals*) sta literala, ki ne moreta biti hkrati resnična - sta v protislovju. Primer sta `P` in ¬`P`.

Akciji na istem nivoju planirnega grafa sta *medsebojno izključujoči* (ang. *mutually exclusive* ali MUTEX), če velja:

1. njuni pogoji so nekonsistentni,
2. njuni učinki so nekonsistentni ali
3. učinki ene akcije so nekonsistentni s pogoji druge akcije.

Pri kreiranju domene je potrebno definirati vse možne nekonsistentne literale. To storimo s stavki oblike `inconsistent(A,B)`.

Zaradi možnih neskladnosti v planirni graf vpeljemo *indikatorje*. To so Boolove spremenljivke, katere priredimo vsaki akciji in vsakemu literalu. Povedo nam, ali je določena akcija oziroma literal prisotna v končnem planu. Planirni graf še ne predstavlja končnega plana. Tega je potrebno izluščiti iz grafa z dodeljevanjem vrednosti indikatorjem tako, da so ciljni literali resnični. Torej velja, da so literali, ki se pojavijo na določenem nivoju, le potencialno resnični. Prav tako velja za akcije, saj ni nujno, da se pojavijo v končnem planu.

Poglavje 4

Odpravljanje napak v programu POP

Vsi programi, ki so bili uporabljeni za raziskave diplomskega dela, so povzeti iz knjige *Prolog programming for Artificial Intelligence* [4]. Tekom testiranja algoritmov na različnih domenah z več roboti smo odkrili napako v programu POP. Izkazalo se je, da ta ni vedno vrnil prave rešitve. Da bi ugotovili, kje v programu je napaka, smo analizirali nekaj primerov iz domene skladišče s paletami in vozički, kjer je planer vrnil napačen plan.

Napaka je bila povsod enakega tipa - rešitveni plan je vseboval istočasno izvajanje akcij, ki se po vsej logiki ne bi smele zgoditi hkrati. Nekaj primerov:

- dva različna robota sta hkrati jemala dve različni paleti iz istega vozička,
- dva različna robota sta hkrati nalagala dve različni paleti v isti voziček,
- robot je hkrati dvignil paleto s sosednje celice in se premaknil na neko drugo sosednjo celico,
- robot je hkrati dvignil paleto s sosednje celice in potisnil voziček.

Kodo planerja POP smo popravili na dva načina.

4.1 1. verzija popravljenega programa POP

Prva ideja, kako bi rešili problem nekosistentnih akcij, je bila v dopolnitvi definicije domene in minimalnem posegu v kodo programa POP. Program smo popravili tako, da ta za vsako na novo dodano akcijo preveri, ali je nekonsistentna z akcijami, ki so že v rešitvenem planu. Če najdemo izključujoči akciji, dodamo omejitev, ki pravi, da se akciji ne smeta izvesti sočasno. Del dopolnjene kode programa POP je podan v dodatku A.1.

V vsaki od domen je bilo potrebno definirali vse pare izključujočih se akcij. Potrebna je bila posebna pozornost, pri kakšnih vrednostih komponent akcij so te nekonsistentne. Stavki, ki smo jih dodali, so oblike `inconsistent( Akcija1, Akcija2)`. Nekaj primerov iz domene skladišča s paletami in vozički:

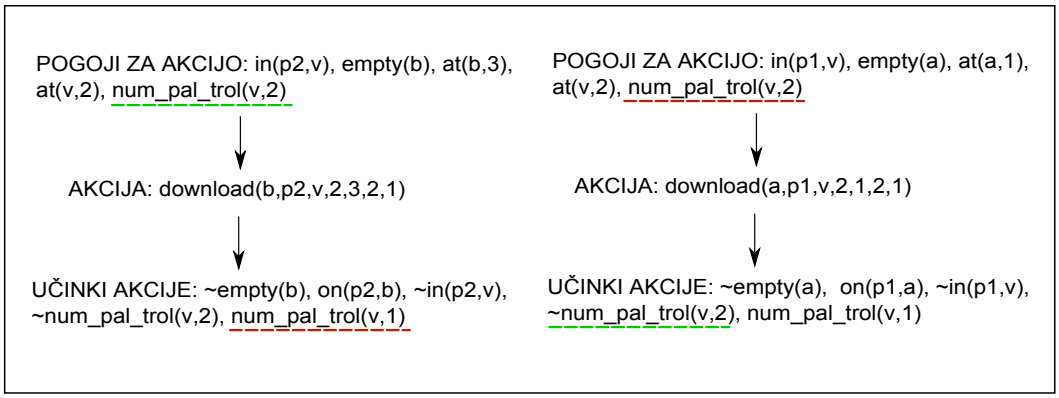
- `inconsistent(upload(R1,P1,V,_,_,_,_), upload(R2,P2,V,_,_)):-` `R1\== R2.`
- `inconsistent(pick(R,_,_,_,_,_), push(R,_,_,_,_)).`
- `inconsistent(pick(R,_,_,_,_,_), move(R,_,_)).`

Na tak način je bila napaka POP planerja popravljena in je ta vračal pravilne rešitve. Slabost tega popravka je v nujni dopolnitvi definicije domene. Pri snovanju domene je potrebno preučiti vse možne nekonsistentne akcije. Bolj kot je domena kompleksna, večje je število stavkov, ki jih moramo dodati. Pri tem se kaj hitro zgodi, da katerega izmed parov nekonsistentnih akcij spregledamo. Torej ta popravek v splošnem ni zelo uporaben.

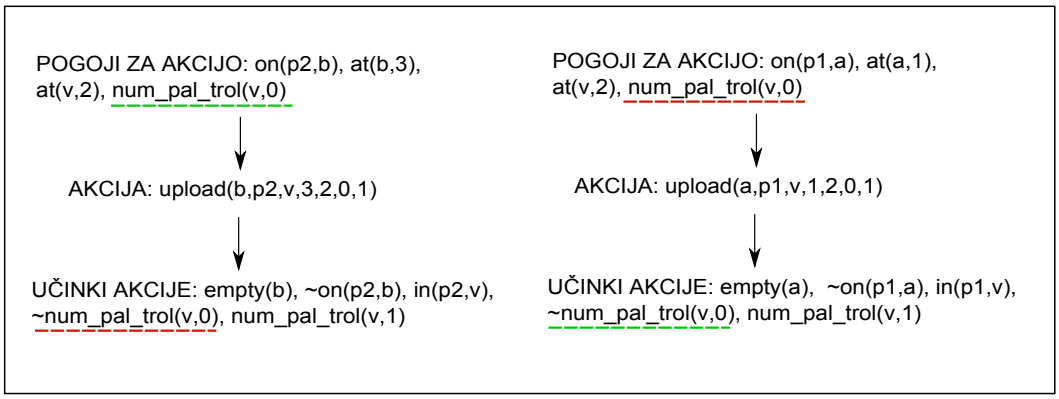
4.2 2. verzija popravljenega programa POP

V poglavju o planirnih grafih na strani 17 [4] smo spoznali, kdaj sta akciji izključujoči in se ne smeta zgoditi hkrati. Poznamo tri primere: (1) ko imata akciji nekonsistentne predpogoje, (2) ko imata akciji nekonsistentne učinke in (3) ko so predpogoji ene akcije nekonsistentni z učinki druge akcije. Da bi lažje ugotovili, kje POP program iz knjige zataji, smo analizirali pare akcij, ki se ne bi smele

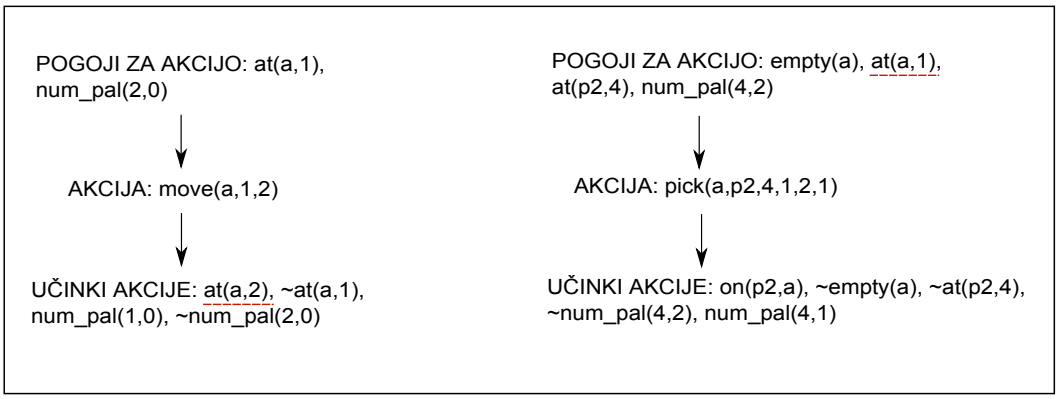
zgoditi hkrati, a jih je program vseeno izvajal istočasno. Razčlenili smo akcije na predpogoje in učinke, kar je razvidno s slik 4.1 - 4.4.



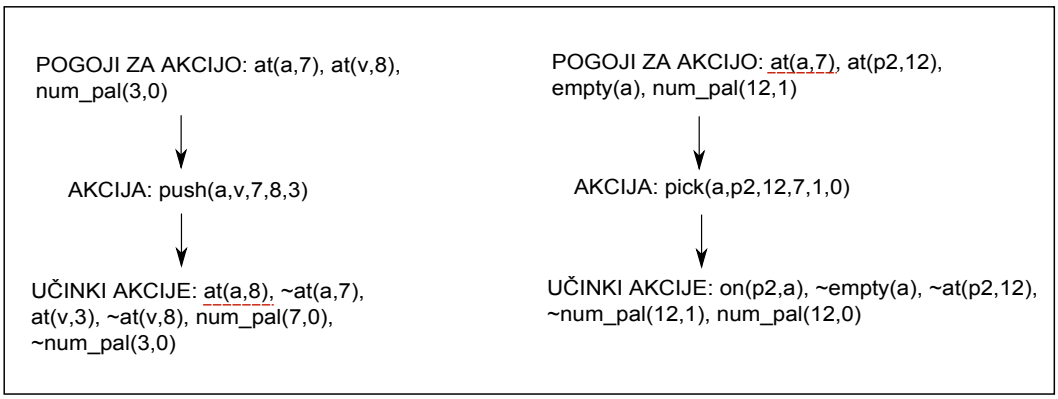
Slika 4.1: Razčlenitev akcij `download(b,p2,v,2,3,2,1)` in `download(a,p1,v,2,1,2,1)` na predpogoje in učinke.



Slika 4.2: Razčlenitev akcij `upload(b,p2,v,3,2,0,1)` in `upload(a,p1,v,1,2,0,1)` na predpogoje in učinke.



Slika 4.3: Razčlenitev akcij `move(a,1,2)` in `pick(a,p2,4,1,2,1)` na predpogoje in učinke.



Slika 4.4: Razčlenitev akcij `push(a,v,7,8,3)` in `pick(a,p2,12,7,1,0)` na predpogoje in učinke.

Pri vseh štirih primerih parov akcij smo opazili naslednje:

- akciji nimata nekonsistentnih predpogojev,
- akciji nimata nekonsistentnih učinkov in
- predpogoj ene akcije je nekonsistenten z učinkom druge akcije (nekonsistentni literali so na slikah podčrtani z zeleno ali rdečo črtkano črto).

Tokrat smo kodo POP planerja iz knjige dodelali tako, da ta preveri:

- Ali je kateri od predpogojev na novo dodane akcije nekonsistenten z učinkom katerekoli akcije, ki je že v planu. Če je, planer doda omejitev, tako da se ti dve akciji ne izvedeta hkrati.
- Ali je kateri od učinkov na novo dodane akcije nekonsistenten s predpogojem katerekoli akcije, ki je že v planu. Če je, planer doda omejitev, tako da se ti dve akciji ne izvedeta hkrati.

Del dopolnjene kode programa POP je podan v dodatku A.2. Druga verzija popravljenega POP programa je sicer v praksi precej počasnejša od prve verzije. V povprečju se čas izvajanja podaljša za 60 odstotkov. Velika prednost pa je v tem, da pri implementaciji domene ni potrebno definirati vseh možnih izključujočih se akcij. Program POP te odkrije sam na podlagi nekonsistentnih literalov.

4.3 POP z minimalnim številom korakov

POP planer, povzet iz knjige [4], je napisan tako, da vrača rešitvene plane z minimalnim številom akcij. Minimalno število akcij ne zagotavlja nujno najboljše rešitve v pogledu časa izvajanja programa. Predpostavimo, da so za rešitev problema potrebne minimalno 3 akcije. Naj obstajata dva rešitvena plana. V prvem se te tri akcije izvedejo zaporedno v treh korakih. V drugem planu se vse tri akcije izvedejo istočasno v enem koraku. Očitno je, da je drugi plan boljši, saj potrebuje manj časa za izvedbo.

V upanju, da bi izvajanje POP programa pohitrili, smo obe delujoči verziji spremenili tako, da vračata rešitvene plane z minimalnim številom korakov. Seveda obstaja nevarnost, da planer na vsakem koraku izvede veliko število akcij. Da bi se temu izognili, smo podali omejitev, da se na vsakem časovnem koraku hkrati izvede največ toliko akcij, kolikor je robotov v domeni. V poglavju z rezultati na strani 56 so zapisana opažanja, katera od teh verzij je bolj učinkovita. Koda spremenjenega programa POP je podana v dodatku A.3.

Poglavje 5

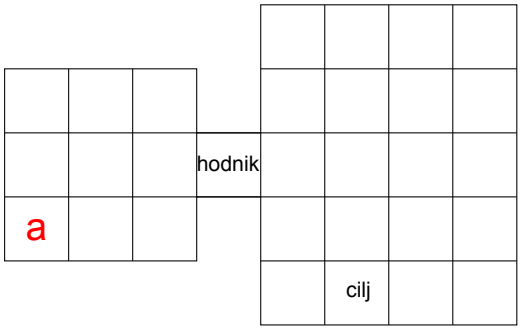
Izboljšave in razširitve algoritmov za planiranje POP in GRAPHPLAN

Obstajajo različne verzije algoritmov GRAPHPLAN in POP. Nekatere izvajanje programov pohitrijio, druge razširijo program tako, da ta deluje na bolj kompleksno definiranih domenah. V okviru diplomske naloge smo se lotili dveh izboljšav - planiranje s podanim vmesnim ciljem in dvosmerni GRAPHPLAN. Ogledali smo si tudi nekatere druge možne razširitve programov, ki smo jih našli v literaturi. Nobena od teh razširitev ni trivialna, zato se nismo lotili njihove implementacije.

5.1 Planiranje s podanim vmesnim ciljem

Ena od možnih pohitritev algoritmov je definiranje vmesnega cilja. Velikokrat že vnaprej vemo, da mora biti v nekem trenutku izvajanja plana določen prostorski literal resničen. Za primer lahko vzmamemo dva prostora, povezana z ozkim hodnikom, kot na sliki 5.1. Robot `a` je v enem od prostorov in mora priti do celice označene z besedo `cilj`, ki je v drugem prostoru. Predpostavimo, da smo tudi mi v istem prostoru kot robot. Ne vidimo celice s ciljem, vendar vemo, da bo moral robot za dosego cilja skozi hodnik. Torej robotu pomagamo in ga usmerimo proti

hodniku.



Slika 5.1: Primer domene, v kateri bi pri planiranju za dosego cilja `at(a,cilj)` definirali vmesni cilj `at(a, hodnik)`.

Z definiranjem vmesnega cilja problem razdelimo na dva podproblema, ki ju rešujemo neodvisno:

1. podproblem: iskanje plana iz začetnega stanja prostora do vmesnega cilja,
2. podproblem: iskanje plana iz stanja prostora, v katerem je vmesni cilj resničen, do končnega cilja.

Končni plan dobimo s konjunkcijo planov, ki ju dobimo z reševanjem podproblemov. Pričakujemo, da je reševanje podproblemov hitrejše od iskanja iz začetnega stanja do končnega cilja v enem kosu.

V splošnem vmesni cilj ne definira enega samega vmesnega stanja, ampak celo množico stanj sveta. Formalno vmesni cilj definiramo kot množico stanj sveta, pri čemer je stanje sveta element množice, če in samo če je vmesni cilj v njem resničen [1]. V kompleksnejših domenah je število stanj prostora, ki vsebujejo vmesni cilj, lahko zelo veliko. Posledično je v takih domenah vprašljiva učinkovitost algoritmov s podanim vmesnim ciljem.

Mi smo program za planiranje s podanim vmesnim ciljem implementirali tako, da deluje le na enostavni domeni pravokotna mreža brez ovir z enim robotom. V tej domeni vmesni cilj definira samo eno vmesno stanje prostora. Edina dopustna oblika vmesnega cilja je `at(A,X)`, pri čemer v začetnem stanju prostora velja `at(A,1)`. Vmesno stanje prostora definiramo kot: začetno stanje `S` + {`at(A,X), clear(1)`} - {`at(A,1), clear(X)`}.

Koda programa za planiranje z vmesnim ciljem je podana v dodatku B.1. Rezultati testiranj so zapisani na strani 56.

5.2 Dvosmerni GRAPHPLAN

Ena od idej, kako pohitriti izvajanje algoritma GRAPHPLAN, je v istočasni:

- gradnji planirnega grafa na običajen način z veriženjem naprej in
- gradnji planirnega grafa z regresijo ciljev z veriženjem nazaj.

V upanju, da bi pohitrili obstoječi algoritem GRAPHPLAN [4], smo v okviru praktičnega dela diplomske naloge spisali program, ki naj bi delal po tem principu.

Ideja algoritma:

1. Če je množica ciljev dosežena že v začetnem stanju prostora, končaj.
2. Sicer iz obeh smeri (iz začetnega stanja in od ciljev nazaj) zgradi nova dva nivoja planirnih grafov.
   (a) Če se novi nivo literalov z desne strani ujema (ujemanje je definirano kasneje):
      - s prejšnjim nivojem literalov z leve strani (za primer, ko končni plan vsebuje liho število akcij) ali
      - z novim nivojem literalov z leve strani (za primer, ko končni plan vsebuje sodo število akcij),

      iz planirnih grafov s pomočjo `clpfd` knjižnice sestavi rešitveni plan in končaj.
   (b) Sicer ponovi točko 2.

Postopek izgleda enostavno, vendar naleti na dva bistvena problema:

1. Za vzvratno gradnjo planirnega grafa od ciljev proti začetnemu stanju ne moremo uporabiti enakega postopka kot pri veriženju naprej. Razloga sta dva. Prvi je v nedefiniranem končnem stanju sveta, saj poznamo le nekaj prostorskih literalov. Drugi je v potrebi po definiranju inverznih akcij. Rešitev tega problema je v regresiji ciljev. Za gradnjo planirnega grafa od ciljev nazaj

uporabimo metodo sredstev in ciljev z regresijo ciljev. Pri tem nivo literalov sestavljajo predpogoji možnih akcij. Med možne akcije spadajo tudi virtualne akcije, ki ohranjajo cilje.

2. Pojavi se vprašanje, kako preveriti, ali se dva nivoja literalov ujemata. Seveda ne želimo, da vsebujeta povsem enake literale. Ujemanje smo v našem primeru definirali kot ujemanje pozitivnih indikatorjev. Bolj natančno: vsi literali, ki imajo vrednost indikatorja 1 na nivoju, ki je dobljen z regresijo ciljev, morajo imeti prav tako vrednost indikatorja 1 na nivoju, ki je dobljen z algoritmom GRAPHPLAN od začetnega stanja proti ciljem. Pri tem ni nujno, da to velja tudi v obratni smeri, saj GRAPHPLAN generira tudi nepomembne literale, ki nimajo vpliva na končni cilj, kljub temu pa imajo lahko vrednost indikatorja 1.

Čeprav so posamezne funkcije našega programa delovale pravilno, se je celoten algoritem pri vseh primerih, ki smo jih želeli testirati, nekje zaciklal. Program smo testirali za različne primere na različnih domenah, vendar v nobenem od teh primerov ni vrnil nobene rešitve. Tako žal nismo mogli primerjati čase dvosmernega GRAPHPLANA z enosmerno verzijo. Koda dvosmernega GRAPHPLANA je podana v dodatku B.2. Program smo večkrat pregledali in analizirali, vendar napake nismo znali odpraviti. Vse kaže, da je napaka v funkciji `goalActs(Goal/T, AllActs,[], ActsThatAchieveGoal)`. Če smo temu pravilu dodali stavek za izpis poljubne besede, se je ta beseda med delovanjem programa ves čas izpisovala.

5.3 POP z različno dolgo trajajočimi akcijami

Običajni modeli planiranja niso primerni za časovno planiranje na domenah, kjer različne akcije potekajo različno dolgo časa. Akcije se v teh domenah lahko izvajajo hkrati. Cilj je poiskati rešitveni plan, ki za izvajanje potrebuje najmanj časa.

Program TANDOR, opisan v [6], vsak literal opremi s časovno točko, v kateri postane resničen, in vsaki akciji dodeli časovni interval, v katerem akcija poteka. Pogoji in učinki akcije se uresničijo v različnih časovnih točkah intervala. To omogoča prekrivanje akcij tudi, ko se njihovi predpogoji in učinki nanašajo na iste

prostorske literale, saj so ti opremljeni s časovnimi točkami.

V prvem stadiju TANDOR zgradi časovni planirni graf (TPG), v katerem se negativne učinke akcij ignorira. V drugem stadiju s pomočjo regresije zgradi prostor časovnih planov. S pomočjo informacij v časovnem planirnem grafu izračuna cene povezav v prostoru. Nato s preiskovanjem poišče rešitev z minimalno časovno ceno. Implementacije te različice programa POP se nismo lotili.

5.4 Program za nadzor izvajanja POP plana

Tokrat ne gre za razširitev algoritma POP, temveč izrabljanje njegovih dobrih lastnosti v namen učinkovitejšega nadzora nad izvajanjem plana. Med nadzorom izvajanja (angl. *execution monitoring*) preverimo, če se dejansko stanje sveta in stanje predvideno po planu ujemata. Modeli sveta in učinki akcij so v realnosti velikokrat nepopolni. Stanje sveta se med izvedbo plana lahko spremeni tako, da se razlikuje od tistega, ki ga je predvidel rešitveni plan. Naloga programov za nadzor izvajanja je, da popravijo obstoječi plan oziroma zgenerirajo novega. V [7] je opisano, kako program za nadzor izvajanja s pomočjo regresiranja ciljev po akcijah delno urejenga plana zelo učinkovito poišče alternativni plan. Pri tem pomembno vlogo igra struktura delno urejenih planov, ki predstavlja družino planov, iz katere dobimo veliko število popolnoma urejenih planov. Glede na trenutno stanje sveta lahko preklapljamo med različnimi linearizacijami delno urejenega plana in s tem privarčujemo na času za iskanje nove poti.

Za potrebe naše diplomske naloge nismo implementirali in uporabljali programa za nadzor izvajanja, saj dobljenih rešitvenih planov nismo realizirali.

5.5 Razširitev GRAPHPLANA za delo z negotovostjo

Klasični planerji po večini delujejo le na deterministčnih domenah. V realnosti velikokrat ne poznamo celotnega stanja prostora. Prostorski literali so resnični z neko verjetnostjo. Lahko se zgodi, da poznamo stanje sveta, vendar so akcije

verjetnostne in imajo lahko različne učinke.

Dve možni razširitvi GRAPHPLANA za delovanje na domenah z verjetnostnimi akcijami sta opisani v [3]. Želena rešitev je pogojni plan z največjo verjetnostjo uspeha pri uresničevanju zastavljenih ciljev. Namesto iskanja plana, ki vsebuje zaporedje akcij, algoritem išče pogojni plan, ki nam pove, katero od akcij izvesti naslednjo glede na zgodovino učinkov akcije.

Implementacije te razširitve algoritma GRAPHPLAN se v naši diplomski nalogi nismo lotili.

5.6 GRAPHPLAN s hevristiko za izločevanje nepomembnih podatkov

Povečanje učinkovitosti algoritmov lahko dosežemo z izločevanjem nepomembnih podatkov - literalov in akcij, ki ne vplivajo na cilj. Zato se mnogi algoritmi planiranja lotijo od ciljev proti začetnemu stanju (*veriženje nazaj*). Pri tem generirajo le potencialno uporabne akcije in prostorske literale.

GRAPHPLAN, ki je eden najučinkovitejših algoritmov za planiranje, uporablja *veriženje naprej*. V planirni graf vključuje vse možne akcije, tudi tiste, ki nikoli ne prispevajo k uresničitvi ciljev. V prisotnosti velike količine nepomembnih informacij se časovna zahtevnost GRAPHPLANA lahko močno poveča.

Ideja za odstranitev nepomembnih podatkov, opisana v [8], je v združevanju veriženja naprej in veriženja nazaj. Z izgradnjo AND-OR dreves od ciljev proti začetnemu stanju zgeneriramo akcije in literale, ki so potencialno uporabni. Nato z GRAPHPLANOM zgradimo planirni graf, v katerem ignoriramo vse literale in akcije, ki se ne pojavijo v AND-OR drevesu. Iz okrnjenega planirnega grafa dobimo končni rešitveni plan. Uporaba te hevristike ne zagotavlja rešitve, saj so učinki akcij in stanje sveta popolnoma ignorirani. Kljub temu se dobro obnese v veliko domenah in je računsko zelo učinkovita.

Podobna ideja za izločevanje nepomembnih podatkov je opisana v [5]. Temelji na izgradnji planirnega grafa v nasprotni smeri s pomočjo regresije ciljev. Dobljeno strukturo uporabi za vodenje pri izgradnji običajnega planirnega grafa z veriženjem

naprej. Pri vzvratnem planiranju se generirajo le akcije, ki dejansko lahko prispevajo k uresničitvi ciljev.

Tudi implementacije te različice algoritma GRAPHPLAN se v naši diplomski nalogi nismo lotili. Spominja na dvosmerni GRAPHPLAN, ki smo ga implementirali, vendar ta žal ne deluje.

Poglavje 6

Rezultati testiranj algoritmov na enostavnih domenah planiranja z roboti

V tem poglavju so opisani rezultati poskusov planiranja na enostavnih domenah. Uporabljeni pristopi planiranja so:

- algoritem A*,
- metoda sredstev in ciljev z regresijo ciljev,
- kombinacija metode sredstev in ciljev z regresijo ter algoritma A*,
- algoritem POP,
- planirni grafi (GRAPHPLAN).

Vsi programi, ki so bili uporabljeni, so povzeti iz knjige *Prolog programming for Artificial Intelligence* [4], kjer so tudi natančno opisani. POP planer iz knjige ne vrača pravilnih planov, zato smo uporabili našo popravljeno verzijo programa, in sicer drugo verzijo, ki je opisana na strani 19.

Za potrebe delovanja algoritma A*, je bilo potrebno za vsako domeno definirati prostor stanj in hevristično funkcijo. Uporabili smo že napisan program `action_to_state_space.pl`, ki je iz definicij akcij zgeneriral prostor stanj. Vsako vozlišče v prostoru stanj ima obliko `state(State0, Goals, Action)`. `State0`

predstavlja trenutno stanje sveta, `Goals` vsebuje množico ciljev, ki jih želimo doseči, ter `Action` predstavlja eno izmed možnih akcij v trenutnem stanju. Cena za prehod med vozlišči je povsod enaka 1.

Hevristična funkcija za A* v svetu kock je napisana tako, da vsakemu od ciljev dodeli ceno. Ta je odvisna od oblike cilja:

- Če je cilj oblike `clear(_)`, je cena 1. Za dosego cilja je potrebno izpeljati vsaj eno akcijo.
- Če je cilj oblike `on(A,B)`, algoritem preveri, katera od treh možnosti je resnična v trenutnem stanju sveta:
  1. Oba prostorska literala `clear(A)` in `clear(B)` sta resnična. Cilju dodelimo ceno 1.
  2. Samo en od prostorskih literalov `clear(A)` in `clear(B)` je resničen. Cilju dodelimo ceno 2.
  3. Noben od prostorskih literalov `clear(A)` in `clear(B)` ni resničen. Cilju dodelimo ceno 3.

Končna hevristična ocena vozlišča state(State0, Goals, Action) je vsota cen ciljev iz `Goals`, ki še niso uresničeni po izvedbi akcije `Action`.

Hevristična funkcija za A* v domeni pravokotna mreža (z ovirami) je definirana kot manhattanska razdalja. Tako kot pri svetu kock, tudi tukaj vsakemu od še ne doseženih ciljev dodelimo ceno:

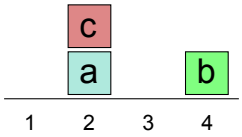
- Če je cilj oblike `clear(_)`, dodelimo ceno 1, saj včasih zadošča le en korak, da polje izpraznimo.
- Če je cilj obike `at(A,B)`, poiščemo trenuten položaj robota `A` in izračunamo manhattansko razdaljo med trenutnim položajem in poljem `B`. Razdalja je cena tega cilja.

Končna hevristična ocena vozlišča state(State0, Goals, Action) je vsota cen ciljev iz `Goals`, ki še niso uresničeni po izvedbi akcije `Action`.
Ta hevristika na pravokotni mreži z ovirami ne upošteva položajev ovir, zato ni tako učinkovita kot v primeru prazne mreže.

6.1 Svet kock

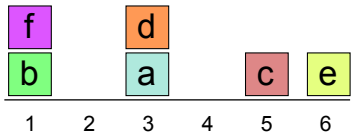
Algoritmi so primerjani na treh različnih začetnih stanjih prostora. Za vsakega izmed stanj prostora sta izbrani dve različni množici ciljev, ki jih s planiranjem želimo doseči.

1. Primer: začetno stanje prostora je razvidno s slike 6.1. Množici ciljev:
   - Cilji 1: `[on(a,4)]`
   - Cilji 2: `[on(b,c), clear(a)]`



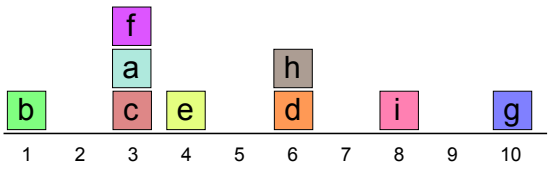
Slika 6.1: Začetno stanje prostora za prvi primer iz sveta kock.

2. Primer: začetno stanje prostora je razvidno s slike 6.2. Množici ciljev:
   - Cilji 1: `[on(b,c), on(a,e)]`
   - Cilji 2: `[clear(6), on(f,c), on(c,2)]`



Slika 6.2: Začetno stanje prostora za drugi primer iz sveta kock.

3. Primer: začetno stanje prostora je razvidno s slike 6.3. Množici ciljev:
   - 1. primer: `[on(i,2), on(h,4)]`
   - 2. primer: `[clear(6), on(a,d)]`



Slika 6.3: Začetno stanje prostora za tretji primer iz sveta kock.

Tabela 6.1: Časi izvajanja algoritmov za planiranje v domeni svet kock (v milisekundah).

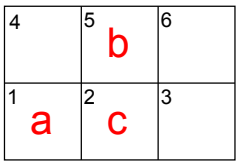
	svet kock 1		svet kock 2		svet kock 3	
	cilji 1	cilji 2	cilji 1	cilji 2	cilji 1	cilji 2
Algoritem A*	30	160	>1 h	>1 h	>1 h	>1 h
Regresija ciljev	3	2	45	36	57	16825
Regresija ciljev in A*	30	6	48725	4490	25796	>1 h
POP	218	74	843000	29000	101000	>1 h
GRAPHPLAN	208	265	3255	3780	37253	Out of Memory

Komentar: Iz tabele 6.1 je razvidno, da je najučinkovitejši pristop planiranja v svetu kock metoda sredstev in ciljev z regresijo ciljev. Algoritem A* se izkaže za najbolj potratnega. Vrača zelo dolge plane z veliko nepotrebnimi akcijami. Mogoče je, da bi bil z drugačno izbiro definicije prostora stanj in hevristične funkcije algoritem A* učinkovitejši. Prve tri metode dajo popolnoma urejene plane, medtem ko GRAPHPLAN in POP vračata delno urejene plane, kar omogoča istočasno izvajanje določenih akcij.

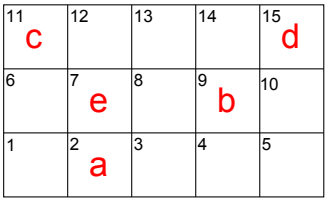
6.2 Pravokotna mreža

Algoritmi so primerjani na dveh različnih začetnih stanjih prostora. Za vsakega izmed stanj prostora sta izbrani dve različni množici ciljev, ki jih s planiranjem želimo doseči.

1. Primer: začetno stanje prostora je razvidno s slike 6.4. Množici ciljev:
   - Cilji 1: `[at(a,3), at(c,6)]`
   - Cilji 2: `[at(b,3), clear(1), at(c,4)]`
2. Primer: začetno stanje prostora je razvidno s slike 6.5. Množici ciljev:
   - Cilji 1: `[at(d,6)]`
   - Cilji 2: `[at(c,8), at(b,14), at(a,11)]`



Slika 6.4: Začetno stanje prostora za prvi primer robotov na pravokotni mreži.



Slika 6.5: Začetno stanje prostora za drugi primer robotov na pravokotni mreži.

Tabela 6.2: Časi izvajanja algoritmov za planiranje v domeni pravokotna mreža (v milisekundah).

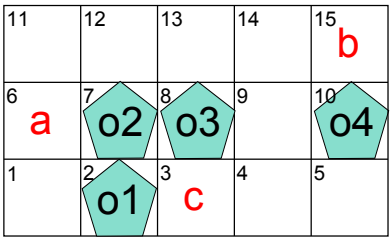
	mreža 1		mreža 2	
	cilji 1	cilji 2	cilji 1	cilji 2
Algoritem A*	237	104	267370	312345
Regresija ciljev	7	580	30049	>1 h
Regresija ciljev in A*	5	153	1969	4670
POP	53	1267	1119	>1 h
GRAPHPLAN	145	184	9020	6191

Komentar: Časi izvajanja algoritmov iz tabele 6.2 so primerljivi za vse pristope planiranja. Vendar se na večji mreži že opazijo razlike. Edina dva algoritma, na katera večja dimenzija mreže in večje število ciljev bistveno ne vpliva, sta GRAPHPLAN in kombinacija metode sredstev in ciljev z regresijo ciljev ter algoritma A*. V tej domeni se A* izkaže boljše kot v svetu kock. To gre pripisati dobro definirani hevristični funkciji (manhattenska razdalja).

6.3 Pravokotna mreža z ovirami

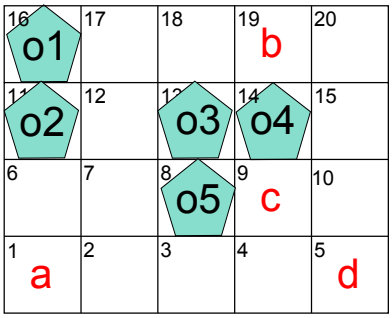
Algoritmi so primerjani na dveh različnih začetnih stanjih prostora. Za vsakega sta izbrani dve različni množici ciljev, ki jih s planiranjem želimo doseči.

1. Primer: začetno stanje prostora je razvidno s slike 6.6. Množici ciljev:
   - 1. primer: `[at(c,5)]`
   - 2. primer: `[at(a,9), at(b,3)]`



Slika 6.6: Začetno stanje prostora za prvi primer robotov na pravokotni mreži z ovirami.

2. Primer: začetno stanje prostora je razvidno s slike 6.7. Množici ciljev:
   - 1. primer: `[at(c,5), at(b,10)]`
   - 2. primer: `[at(a,12), at(c,6)]`



Slika 6.7: Začetno stanje prostora za drugi primer robotov na pravokotni mreži z ovirami.

Tabela 6.3: Časi izvajanja algoritmov za planiranje na domeni pravokotna mreža z ovirami (v milisekundah).

	ovire 1		ovire 2	
	cilji 1	cilji 2	cilji 1	cilji 2
Algoritem A*	39536	446	>1 h	>1 h
Regresija ciljev	1232	955397	24645	>1 h
Regresija ciljev in A*	268	18772	146	48469
POP	678	49671	6917	79373
GRAPHPLAN	230	11079	535	1170

Komentar: V tabeli 6.3 so zbrani časi izvajanja algoritmov v domeni pravokotna mreža z ovirami. Rezultati so zelo primerljivi tistim iz domene brez ovir, saj sta si domeni zelo podobni. Edina razlika je v hevristični funkciji pri algoritmu A*. Ta ne upošteva položajev ovir, kar se kaže v časih izvajanja. Algoritem A* je zaradi tega na tej domeni praktično neuporaben. Najboljše se tudi v tej domeni odreže GRAPHPLAN.

6.4 Ugotovitve

Po dobljenih rezultatih testiranja na enostavnih domenah bi lahko sklepali, da sta najboljša pristopa planiranja algoritem GRAPHPLAN in metoda, ki je kombinacija metode sredstev in ciljev z regresijo ciljev ter algoritma A*. V kombinirani metodi so združene najboljše lastnosti obeh zapisanih algoritmov. Vidi se tudi, kako pomembna je pri planiranju uporaba hevristike.

Poskusi z algoritmom A* morda ne kažejo prave slike, saj je pri tej metodi zelo pomemben izbor hevristične funkcije ter izbira zapisa prostora stanj.

Pri izbiri algoritma je poleg vsega pomembna domena, na kateri želimo algoritem uporabiti. Uporaba planirnih grafov je učinkovita na pravokotni mreži z ali brez ovir, v svetu kock pa GRAPHPLAN dosega slabše rezultate.

Poglavje 7

Primerjava algoritmov POP in GRAPHPLAN na zahtevnejših domenah planiranja z več roboti

Eden od ciljev diplomske naloge je bil podrobnejša analiza oziroma primerjava algoritmov Graphplan in POP. Želeli smo tudi preveriti, katera od štirih verzij popravljenega programa POP je najučinkovitejša. Tako smo na domenah skladišče s paletami ter skladišče s paletami in vozički testirali vseh pet algoritmov.

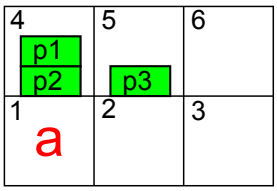
Opomba: zaradi boljše preglednosti smo v tem poglavju ponekod uporabili krajše oznake imen algoritmov:

- GP: algoritmem GRAPHPLAN,
- POP1: prva verzija popravljenega programa POP s strani 18,
- POP1_min_korakov: spremenjeni algoritem POP1, ki vrača rešitvene plane z minimalnim številom korakov,
- POP2: druga verzija popravljenega programa POP s strani 19,
- POP2_min_korakov: spremenjeni algoritem POP2, ki vrača rešitvene plane z minimalnim številom korakov.

7.1 Skladišče s paletami

Algoritem GRAPHPLAN ter vse štiri verzije programa POP so testirani na treh različnih začetnih stanjih prostora. Za vsakega izmed stanj prostora so izbrane tri različne množice ciljev.

1. Primer: začetno stanje prostora je razvidno s slike 7.1



Slika 7.1: Začetno stanje prostora za prvi primer iz domene skladišče s paletami.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[at(a,5)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[move(a,1,2)]`
    `2:[pick(a,p3,5,2,1,0)]`
    `3:[move(a,2,5)]`
- Cilji 2: `[num_pal(4,0)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[pick(a,p2,4,1,2,1)]`
    `2:[drop(a,p2,1,2,0,1)]`
    `3:[pick(a,p1,4,1,1,0)]`
- Cilji 3: `[at(p2,3),num_pal(5,0)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[pick(a,p2,4,1,2,1)]`
    `2:[move(a,1,2)]`
    `3:[drop(a,p2,2,3,0,1)]`
    `4:[pick(a,p3,5,2,1,0)]`

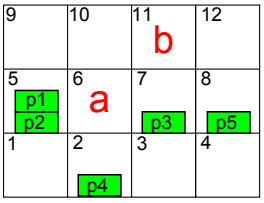
Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.1.

Tabela 7.1: Skladišče s paletami 1. primer: časi izvajanja v milisekundah za vseh pet pristopov planiranja.

Mn. ciljev:	`[at(a,5)]`	`[num_pal(4,0)]`	`[at(p2,3), num_pal(5,0)]`
GP	208	2879	32000
POP1	104	1253	2000
POP1_min_korakov	236	3247	15783
POP2	243	1775	3000
POP2_min_korakov	240	2560	5187

Komentar: Časi izvajanja programov so za vse tri primere zelo podobni. Izstopa le 3. primer, za katerega algoritem GRAPHPLAN ter obe verziji algortima POP z minimalnim številom korakov potrebujejo bistveno več časa.

2. Primer: začetno stanje prostora je razvidno s slike 7.2



Slika 7.2: Začetno stanje prostora za drugi primer iz domene skladišče s paletami.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[at(a,1)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[pick(a,p4,2,6,1,0)]`
    `2:[move(a,6,2)]`
    `3:[move(a,2,1)]`
- Cilji 2: `[at(p3,10), at(p4,3)]`

  - Rešitev algoritmov GP, POP1_min_korakov in POP2_min_korakov:

    `1:[pick(b,p3,7,11,1,0), pick(a,p4,2,6,1,0)]`

    `2:[drop(b,p3,11,10,0,1), move(a,6,2)]`

    `3:[drop(a,p4,2,3,0,1)]`

  - Algoritma POP1 in POP2 rešitve ne vrneta več kot 10 minut.

- Cilji 3: `[at(p5,4)]`

  - Rešitev algoritmov POP1 in POP2:

    `1:[move(b,11,12)]`

    `2:[pick(b,p5,8,12,1,0)]`

    `3:[move(b,12,8)]`

    `4:[drop(b,p5,8,4,0,1)]`

  - Rešitev algoritmov POP1_min_korakov in POP2_min_korakov:

    * 1. verzija:

      `1:[pick(a,p4,2,6,1,0), pick(b,p3,7,11,1,0)]`

      `2:[move(a,6,2), drop(b,p3,11,10,0,1)]`

      `3:[drop(a,p4,2,3,0,1)]`

    * 2. verzija:

      `1:[pick(a,p4,2,6,1,0), pick(b,p3,7,11,1,0)]`

      `2:[move(a,6,2)]`

      `3:[drop(a,p4,2,3,0,1), drop(b,p3,11,10,0,1)]`

    * 3. verzija:

      `1:[pick(a,p4,2,6,1,0)]`

      `2:[move(a,6,2), pick(b,p3,7,11,1,0)]`

      `3:[drop(a,p4,2,3,0,1), drop(b,p3,11,10,0,1)]`

  - Algoritem GRAPHPLAN rešitve ne vrne več kot 10 minut.

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.2.

Komentar: Na tem primeru začetnega stanja prostora se že vidijo razlike med GRAPHPLANOM in POP programi. Zanimivo je, da se časi izvajanja tako razlikujejo od primera do primera. Na 1. in 3. primeru so vse verzije planerja POP občutno hitrejše od GRAPHPLANA, medtem ko je slednji

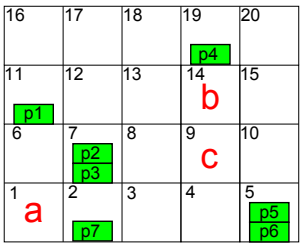
Tabela 7.2: Skladišče s paletami 2. primer: časi izvajanja v milisekundah za vseh pet pristoov planiranja

Mn. ciljev:	`[at(a,1)]`	`[at(p3,10), at(p4,3)]`	`[at(p5,4)]`
GP	15000	4000	>10 min
POP1	137	> 10 min	4000
POP1_min_korakov	60	9 min 48 s	33000
POP2	96	> 10 min	6000
POP2_min_korakov	109	8 min 25 s	36000

občutno hitrejši od ostalih na 2. primeru. Kar kaže na to, da se različni algoritmi problemov lotijo na različne načine.

Pri tretjem primeru opazimo razliko v rešitvi med algoritmoma POP1 in POP2 ter obema verzijama istih programov z minimalnim številom korakov. Prva dva vrneta rešitev, ki sicer vsebuje eno akcijo manj, vendar traja en časovni korak dlje.

3. Primer: začetno stanje prostora je razvidno s slike 7.3



Slika 7.3: Začetno stanje prostora za tretji primer iz domene skladišče s paletami.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[at(a,4)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[pick(a,p7,2,1,1,0)]`
    `2:[move(a,1,2)]`

3:[move(a,2,3)]

4:[move(a,3,4)]

– Pri GRAPHPLANU prolog odgovori, da je prišlo do sistemske napake.

- Cilji 2: [at(p1,7)]
  - Rešitev algoritmov GP, POP1_min_korakov in POP2_min_korakov:

    1:[move(c,9,8), move(a,1,6)]

    2:[pick(c,p2,7,8,2,1), pick(a,p1, 11,6,1,0)]

    3:[drop(a,p1,6,7,1,2)]
  - Rešitev algoritmov POP1 in POP2:

    1:[move(a,1,6)]

    2:[pick(a,p1, 11,6,1,0)]

    3:[drop(a,p1,6,7,2,3)]
- Cilji 3: [num_pal(5,0)]
  - Rešitev algoritmov GP, POP1_min_korakov in POP2_min_korakov:

    1:[move(c,9,4), move(b,14,15)]

    2:[pick(c,p6,5,4,2,1), move(b,15,10)]

    3:[pick(b,p5,5,10,1,0)]
  - Rešitev algoritmov POP1 in POP2:
    * 1. verzija:

      1:[move(c,9,4)]

      2:[move(b,14,9), pick(c,p6,5,4,2,1)]

      3:[move(b,9,10)]

      4:[pick(b,p5,5,10,1,0)]
    * 2. verzija:

      1:[move(c,9,4)]

      2:[move(b,14,9)]

      3:[move(b,9,10), pick(c,p6,5,4,2,1)]

      4:[pick(b,p5,5,10,1,0)]

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.3.

Tabela 7.3: Skladišče s paletami 3. primer: časi izvajanja v milisekundah za vseh pet pristopov planiranja

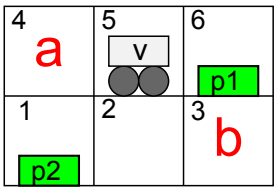
Mn. ciljev:	`[at(a,4)]`	`[at(p1,7)]`	`[num_pal(5,0)]`
GP	error	13000	15000
POP1	200	4000	72000
POP1_min_korakov	346	112000	8000
POP2	252	7422	112000
POP2_min_korakov	150	48000	3300

Komentar: Na tem primeru skladišča s paletami opazimo podobnost delovanja GRAPHPLANA in obeh različic POP programa z minimalnim številom korakov - POP1_min_korakov in POP2_min_korakov. Vsi trije algoritmi imajo podobne čase izvajanja in vračajo enake rešitvene plane. Pri drugem primeru opazimo, da sta v rešitvenem planu dodani dve nepotrebni akciji. To ni neobičajen pojav, saj so ti trije programi optimizirani za vračanje rešitev, ki vsebujejo minimalno število korakov, število akcij pa pri tem ni pomembno. Podobno opazimo v tretjem primeru. Rešitveni plan programov POP1 in POP2 vsebuje 4 korake, čeprav optimalni plan potrebuje le 3. Razlaga je podobna tisti iz drugega primera. POP1 in POP2 sta optimizirana za vračanje rešitvenih planov z minimalnim številom akcij. Pri tem število korakov ni pomembno. Neoptimalne rešitve pomenijo daljše izvajanje programa, kar se vidi v časih iz tabele 7.3.

7.2 Skladišče s paletami in vozički

Algoritem GRAPHPLAN ter vse štiri verzije programa POP smo testirali na različnih začetnih stanjih domene skladišče s paletami in vozički. V nadaljevenju so predstavljeni rezultati za tri izmed teh začetnih stanj. Za vsakega so izbrane tri različne množice ciljev.

1. Primer: začetno stanje prostora je razvidno s slike 7.4



Slika 7.4: Začetno stanje prostora za prvi primer iz domene skladišče s paletami in vozički.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[num_pal(6,0), num_pal(1,0), empty(a), empty(b)]`
  - Rešitev vseh algoritmov je enaka:
    `1:[pick(b,p1,6,3,2,0), pick(a, p2,1,4,1,0)]`
    `2:[upload(a,p2,v,4,5,0,1), drop(b,p1,3,2,0,1)]`
- Cilji 2: `[at(a,3)]`
  - Rešitev GRAPHPLANA:
    `1:[pick(b,p1,6,3,1,0), pick(a,p2,1,4,1,0)]`
    `2:[move(b,3,6), move(a,4,1)]`
    `3:[move(a,1,2)]`
    `4:[move(a,2,3)]`
  - Rešitev programov POP1 in POP2:
    `1:[pick(b,p1,6,3,1,0), push(a,v,4,5,2)]`
    `2:[move(a,5,6)]`
    `3:[push(b,v,3,2,5)]`
    `4:[move(a,6,3)]`
  - Rešitev programov POP1_min_korakov in POP2_min_korakov:
    - 1. verzija:
      `1:[pick(b,p1,6,3,1,0), pick(a,p2,1,4,1,0)]`
      `2:[move(b,3,6), move(a,4,1)]`
      `3:[move(a,1,2)]`
      `4:[move(a,2,3)]`
    - 2. verzija:

1:[pick(b,p1,6,3,1,0), pick(a,p2,1,4,1,0)]
2:[move(a,4,1)]
3:[move(b,3,6), move(a,1,2)]
4:[move(a,2,3)]

∗ 3. verzija:

1:[pick(b,p1,6,3,1,0), pick(a,p2,1,4,1,0)]
2:[move(a,4,1)]
3:[move(a,1,2)]
4:[move(b,3,6), move(a,2,3)]

- Cilji 3: [in(p1,v), in(p2,v), at(v,2)]
  - Rešitev GRAPHPLANA:

    1:[pick(a,p2,1,4,1,0), pick(b,p1,6,3,1,0)]
    2:[upload(a,p2,v,4,5,0,1), move(b,3,6)]
    3:[upload(b,p1,v,6,5,1,2)]
    4:[push(a,v,4,5,2)]

  - Ostali programi ne vrnejo rešitve več kot 10 minut.

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.4.

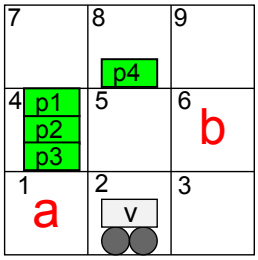
Tabela 7.4: Skladišče s paletami in vozički 1. primer: časi izvajanja v milisekundah za vseh pet pristopov planiranja.

Mn. ciljev:	[num_pal(6,0), num_pal(1,0), empty(a), empty(b)]	[at(a,3)]	[in(p1,v), in(p2,v), at(v,2)]
GP	200	90000	8 min
POP1	12000	5000	> 10 min
POP1_min_korakov	3500	1000	> 10 min
POP2	22000	8600	> 10 min
POP2_min_korakov	1688	900	> 10 min

Komentar: Opazimo, da je v 1. in 3. primeru, ko množica ciljev vsebuje 3

oziroma 4 elemente, GRAPHPLAN precej hitrejši od različic algoritma POP. Večje število ciljev, ki jih želimo doseči, močno upočasni delovanje algoritmov POP. V 2. primeru, ko je bil cilj le en, so bili ti bistveno hitrejši od GRAPHPLANA. Ponovno vidimo, da se različni planerji lotijo planiranja na različne načine, saj se rešitveni plan algoritmov POP1 in POP2 precej razlikuje od planov preostalih planerjev.

2. Primer: začetno stanje prostora je razvidno s slike 7.5



Slika 7.5: Začetno stanje prostora za drugi primer iz domene skladišče s paletami in vozički.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[at(a,1), at(v,3)]`
  - Rešitev GRAPHPLANA:

    `1:[push(a,v,1,2,3)]`

    `2:[move(a,2,1)]`
  - Rešitev vseh POP algoritmov:

    `1:[move(b,6,5)]`

    `2:[push(b,v,5,2,3)]`
- Cilji 2: `[at(b,8), num_pal(2,0)]`
  - Rešitev GRAPHPLANA:

    `1:[push(a,v,1,2,3), move(b,6,5)]`

    `2:[pick(b,p4,8,5,1,0), move(a,2,1)]`

    `3:[move(b,5,8)]`
  - Rešitev algoritmov POP1 in POP2:

        * 1. verzija:

          1:[move(b,6,5), push(a,v,1,2,3)]

          2:[pick(b,p4,8,9,1,0)]

          3:[move(b,5,8)]

          4:[move(a,2,5)]

        * 2. verzija:

          1:[move(b,6,5)]

          2:[pick(b,p4,8,9,1,0), push(a,v,1,2,3)]

          3:[move(b,5,8)]

          4:[move(a,2,5)]

        * 3. verzija:

          1:[move(b,6,5)]

          2:[pick(b,p4,8,9,1,0)]

          3:[move(b,5,8), push(a,v,1,2,3)]

          4:[move(a,2,5)]

    – Rešitev algoritmov POP1_min_korakov in POP2_min_korakov:

        * 1. verzija:

          1:[push(a,v,1,2,3), move(b,6,5)]

          2:[pick(b,p4,8,5,1,0), move(a,2,1)]

          3:[move(b,5,8)]

        * 2. verzija:

          1:[push(a,v,1,2,3), move(b,6,5)]

          2:[pick(b,p4,8,5,1,0)]

          3:[move(b,5,8), move(a,2,1)]

        * 3. verzija:

          1:[move(b,6,5)]

          2:[push(a,v,1,2,3), pick(b,p4,8,5,1,0)]

          3:[move(b,5,8), move(a,2,1)]

- Cilji 3: [at(p1,5), at(b,9)]

    – Rešitev GRAPHPLANA:

      1:[pick(a,p1,4,1,3,2), move(b,6,9)]

2:[push(a,v,1,2,3)]

3:[drop(a,p1,2,5,0,1)]

– Rešitev vseh POP algoritmov:
  * 1. verzija:
    1:[pick(a,p1,4,1,3,2), move(b,6,9)]
    2:[push(a,v,1,2,3)]
    3:[drop(a,p1,2,5,0,1)]
  * 2. verzija:
    1:[pick(a,p1,4,1,3,2)]
    2:[push(a,v,1,2,3), move(b,6,9)]
    3:[drop(a,p1,2,5,0,1)]
  * 3. verzija:
    1:[pick(a,p1,4,1,3,2)]
    2:[push(a,v,1,2,3)]
    3:[drop(a,p1,2,5,0,1), move(b,6,9)]

Tabela 7.5: Skladišče s paletami in vozički 2. primer: časi izvajanja v milisekundah za vseh pet pristopov planiranja

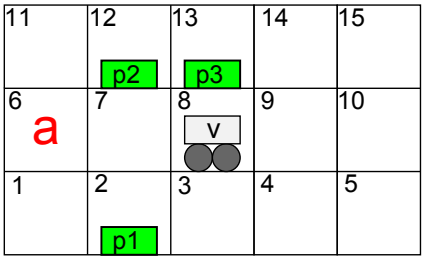
Mn. ciljev:	[at(a,1), at(v,3)]	[at(b,8), num_pal(2,0)]	[at(p1,5), at(b,9)]
GP	675	21000	19000
POP1	158	30000	9000
POP1_min_korakov	115	5881	11500
POP2	188	25000	8600
POP2_min_korakov	87	1465	4400

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.5.

Komentar: Časi vseh algoritmov so na vseh množicah ciljev precej podobni. Pri 1. primeru GRAPHPLAN porabi nekoliko več časa, vendar najde drugačno rešitev od ostalih planerjev. Pri 2. primeru ponovno opazimo, da algoritma POP1 in POP2 ne vrneta optimalnega rešitvenega plana glede na število ko-

rakov.

3. Primer: začetno stanje prostora je razvidno s slike 7.6



Slika 7.6: Začetno stanje prostora za tretji primer iz domene skladišče s paletami in vozički.

Množice ciljev in ustrezni rešitveni plani:

- Cilji 1: `[on(p1,a)]`
  - Rešitev vseh algoritmov je enaka:

    `1:[move(a,6,1)]`

    `2:[pick(a,p1,2,1,1,0)]`
- Cilji 2: `[in(p3,v)]`
  - Rešitev GRAPHPLANA:

    `1:[move(a,6,7)]`

    `2:[push(a,v,7,8,3)]`

    `3:[pick(a,p3,13,8,1,0)]`

    `4:[upload(a,p3,v,8,3,0,1)]`
  - POP planerji ne vrnejo rešitve več kot 10 minut.
- Cilji 3: `[at(p2,14)]`
  - Rešitev algoritmov GRAPHPLAN, POP1 in POP2:

    `1:[move(a,6,7)]`

    `2:[pick(a,p2,12,7,1,0)]`

    `3:[push(a,v,7,8,3)]`

    `4:[move(a,8,9)]`

    `5:[drop(a,p2,9,14,0,1)]`

– Algoritma POP1_min_korakov in POP2_min_korakov ne vrneta rešitve več kot 15 minut.

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.6.

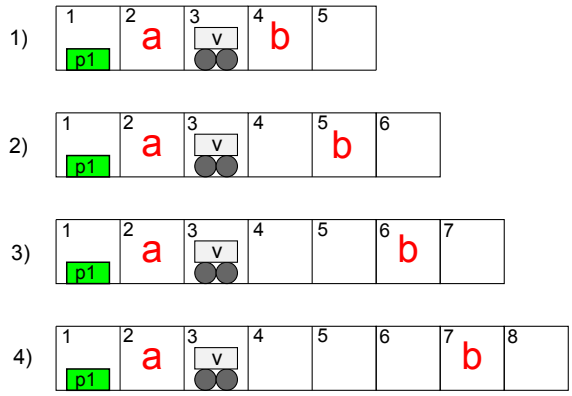
Tabela 7.6: Skladišče s paletami in vozički 3. primer: časi izvajanja v milisekundah za vseh pet pristopov planiranja

Mn. ciljev:	`[on(p1,a)]`	`[in(p3,v)]`	`[at(p2,14)]`
GP	180	33000	1 min 40 s
POP1	1500	$>$ 10 min	10 min 30 s
POP1_min_korakov	3500	$>$ 10 min	$>$ 15 min
POP2	3900	$>$ 10 min	9 min
POP2_min_korakov	3554	$>$ 10 min	$>$ 15 min

Komentar: Primer skladišča velikosti 5x3 pokaže slabost POP planerjev. Ti za dosego večine ciljev potrebujejo več kot deset minut. GRAPHPLAN je občutno hitrejši.

7.2.1 Domet algoritmov v domeni skladišče s paletami in vozički

Smisel domene z vozički je v tem, da roboti za transport palet uporabijo voziček. Med testiranjem algoritmov na tej domeni smo opazili, da imajo programi težave že pri zelo enostavnih ciljih. Predvsem različice planerja POP so za plane s petimi akcijami v povprečju potrebovale več kot 10 minut. Zato ni čudno, da tekom testiranja nismo naleteli na želeno zaporedje akcij oblike `pick(A,P,_,_,_,_)` `- upload(A,P,V,_,_,_,_) - push(A,V,_,_,_) -` `download(A,P,V,_,_,_,_) - drop(A,P,_,_,_,_)`.
Da bi od programov izsilili temu podobno zaporedje akcij, smo definirali štiri posebna začetna stanja prostora s slike 7.7. Doseči želimo prenos palete na skrajno desno celico mreže. Za dosego tega cilja sta nujno potrebni akciji `upload(a,p1,v,_,_,0,1)` in `download(b,p1,v,_,_,1,0)`.



Slika 7.7: Štiri posebna začetna stanja iz domene skladišče s paletami in vozički.

Množice ciljev in ustrezni rešitveni plani:

1. primer s slike. Cilj: `[at(p1,5)]`
   - Rešitev vseh algoritmov je enaka:

     `1:[pick(a,p1,1,2,1,0)]`

     `2:[upload(a,p1,v,2,3,0,1)]`

     `3:[download(b,p1,v,3,4,1,0)]`

     `4:[drop(b,p1,4,5,0,1)]`
2. primer s slike. Cilj: `[at(p1,6)]`
   - Rešitev GRAPHPLANA:

     `1:[pick(a,p1,1,2,1,0), move(b,5,4)]`

     `2:[upload(a,p1,v,2,3,0,1)]`

     `3:[download(b,p1,v,3,4,1,0)]`

     `4:[move(b,4,5)]`

     `5:[drop(b,p1,5,6,0,1)]`
   - Rešitev algoritmov POP1 in POP2:

     `1:[pick(a,p1,1,2,1,0)]`

     `2:[upload(a,p1,v,2,3,0,1)]`

     `3:[push(a,v,2,3,4)]`

     `4:[download(b,p1,v,4,5,0,1)]`

     `5:[drop(b,p1,5,6,0,1)]`
   - POP1_min_korakov in POP2_min_korakov ne vrneta rešitve več kot eno

uro.

3. primer s slike. Cilj: `[at(p1,7)]`
   - Rešitev GRAPHPLANA:

     `1:[pick(a,p1,1,2,1,0), move(b,6,5)]`

     `2:[upload(a,p1,v,2,3,0,1), move(b(5,4)]`

     `3:[download(b,p1,v,3,4,1,0)]`

     `4:[move(b,4,5)]`

     `5:[move(b,5,6)]`

     `6:[drop(b,p1,6,7,0,1)]`
   - POP planerji ne vrnejo rešitve več kot eno uro.
4. primer s slike. Cilj: `[at(p1,8)]`
   - Vsi algoritmi ne vrnejo rešitve več kot eno uro.

Časi izvajanja za vseh pet algoritmov so zbrani v tabeli 7.7.

Tabela 7.7: Posebni primeri domene skladišče s paletami in vozički: časi izvajanja v sekundah za vseh pet pristopov planiranja.

Mn. ciljev:	`[at(p1,5)]`	`[at(p1,6)]`	`[at(p1,7)]`	`[at(p1,8)]`
GP	3	236	2503	> 1 h
POP1	93	2310	> 1 h	> 1 h
POP1_min_korakov	161	> 1 h	> 1 h	> 1 h
POP2	87	1279	> 1 h	> 1 h
POP2_min_korakov	103	> 1 h	> 1 h	> 1 h

Komentar: časi izvajanja nam povedo, da je domena skladišče s paletami in vozički za vse planerje zelo zahtevna. Čeprav so bila začetna stanja prostora definirana tako, da niso dopuščala veliko različnih planov, so planerji potrebovali veliko časa za iskanje rešitve. Programi POP za plane z več kot petimi akcijami potrebujejo več kot eno uro. Razlog je v zahtevnosti akcij za upravljanje vozička. Te imajo veliko predpogojev in učinkov, kar posledično vpliva na čas delovanja algoritma.

Iz rezultatov je razbrati, da je GRAPHPLAN v splošnem močnejše orodje v primerjavi z vsemi različicami programa POP. Kljub boljšim časom,
GRAPHPLAN vrne rešitvene plane z več akcijami, kot bi bilo potrebno. To se

vidi v drugem primeru, ko POP1 in POP2 vrneta optimalni plan, ki vsebuje akcijo `push`. GRAPHPLAN se uporabi te akcije izogiba in vrne rešitev z večjim številom akcij.

7.3 Domet algoritmov na domeni pravokotna mreža

Rezultati na domeni skladišče so pokazali, da tako algoritem za planiranje GRAPHPLAN kot POP nista učinkovita pri sestavljanju planov z veliko akcijami. Zanima nas maksimalno število akcij, ki sta jih ta dva pristopa planiranja sposobna sproducirati v rešitvenemu planu. To najlažje vidimo na enostavnem primeru domene pravokotna mreža, kjer je edina možna akcija `move(A,C1,C2)`.

7.3.1 Linearna mreža

Izgled začetnega stanja prostora vidimo na sliki 7.8. Robot `a` je situiran na skrajni levi celici mreže. Zanima nas čas, ki ga porabi, da pride na skrajno desno celico. Število n smo tekom testiranja povečevali. Rezultati so zbrani v tabeli 7.8.

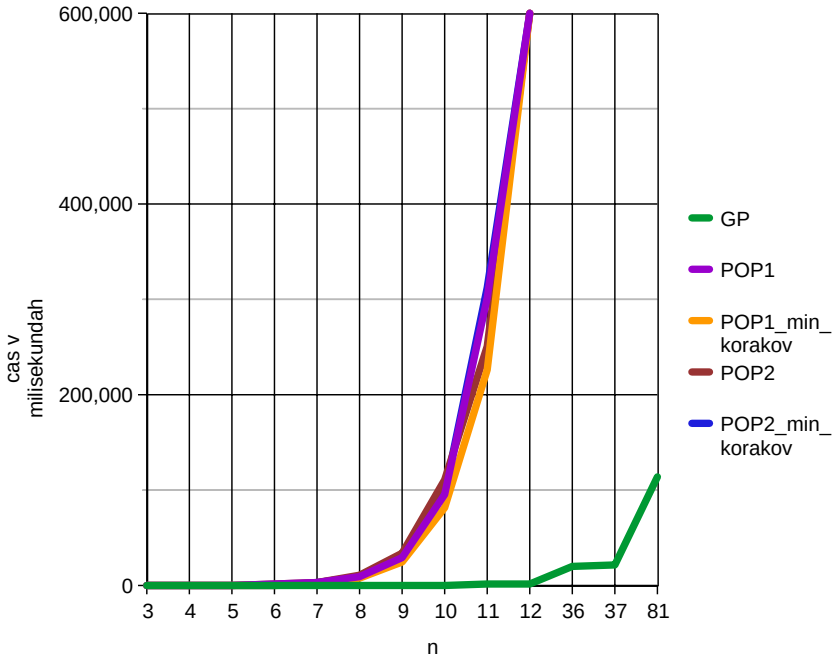
Slika 7.8: Skica linearne mreže velikosti nx1.

Tabela 7.8: Časi izvajanja algoritmov za dosego cilja `at(a,n)` na linearni mreži (v milisekundah).

n	4	5	6	7	8	9	10	11
št. akcij	3	4	5	6	7	8	9	10
GP	33	70	114	180	262	380	501	651
POP1	21	66	230	815	2722	8490	29791	94661
POP1_min_korakov	15	51	189	630	2148	7302	24791	81496
POP2	13	70	247	876	2592	10100	34073	111058
POP2_min_korakov	13	55	149	735	2407	8897	29796	97454

n	12	13	37	38	81	91
št. akcij	11	12	36	37	80	90
GP	837	958	20312	21757	114000	Out of GS
POP1	301581	> 10 min	/	/	/	/
POP1_min_korakov	226498	> 10 min	/	/	/	/
POP2	250102	> 10 min	/	/	/	/
POP2_min_korakov	311629	> 10 min	/	/	/	/

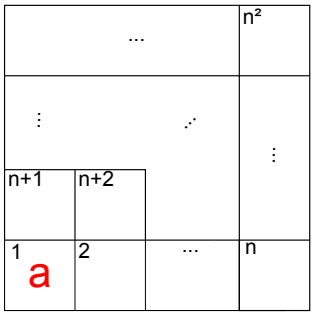
Komentar: Čas izvajanja algoritma GRAPHPLAN se sprva s povečanjem mreže za eno celico podaljša za približno 60%. Vendar procent začne padati. Med prehodom iz 36 na 37 akcij se čas podaljša le še za 7%. GRAPHPLAN 80 akcij izvede v dveh minutah. V primeru, ko bi potreboval 90 akcij, mu zmanjka prostora.
Čas algoritmov POP se z vsako dodano akcijo podaljša za več kot 200%. Za 12 akcij je potrebnih že več kot 10 minut. GRAPHPLAN je neprimeno zmogljivejši od programov POP, kar se dobro vidi iz diagrama na sliki 7.9.



Slika 7.9: Časi izvajanja v milisekundah v odvisnosti od n v domeni linearna mreža.

7.3.2 Kvadratna mreža

Izgled začetnega stanja prostora vidimo na sliki 7.10. Robot `a` je situiran na skrajni levi spodnji celici mreže. Zanima nas čas, ki ga porabi, da pride na skrajno desno zgornjo celico. Število n smo tekom testiranja povečevali. Rezultati so zbrani v tabeli 7.9.



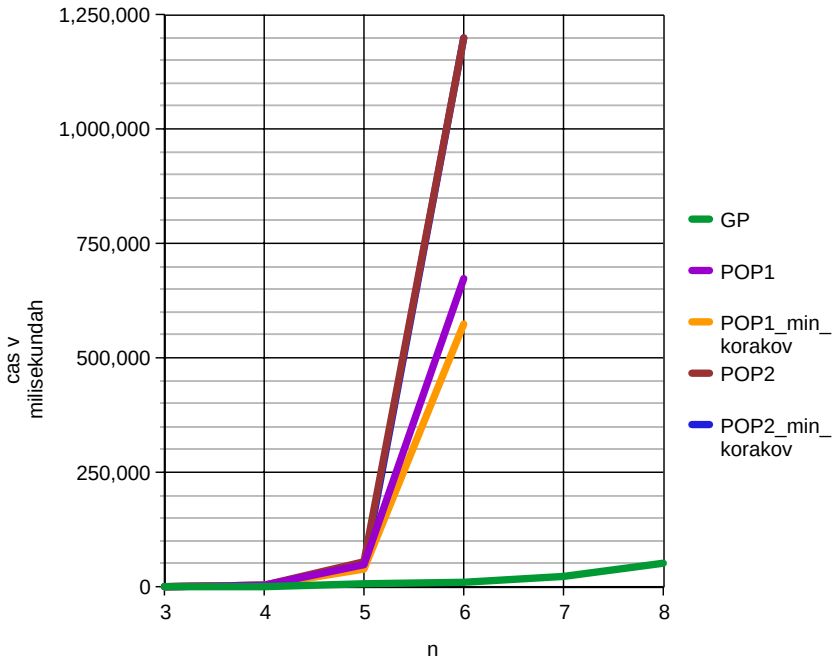
Slika 7.10: Skica kvadratne mreže velikosti nxn.

Tabela 7.9: Časi izvajanja algoritmov za dosego cilja `at(a,n²)` na kvadratni mreži (v milisekundah).

n	3	4	5	6	7	8
št. akcij	4	6	8	10	12	14
GP	335	1130	5325	10178	23556	50289
POP1	153	2528	49327	672068	/	/
POP1_min_korakov	134	1798	38511	574350	/	/
POP2	172	2105	54954	> 20 min	/	/
POP2_min_korakov	102	1696	43094	> 20 min	/	/

Komentar: V primerjavi z linearno mrežo ima robot na kvadratni mreži na vsakem koraku na voljo večje število možnih akcij. To ima pomemben vpliv na časovno zahtevnost. Na tem primeru domene algoritmi POP za plane z desetimi akcijami potrebujejo več kot 10 minut. Tudi GRAPHPLAN na tej domeni za isto število akcij kot na linearni mreži potrebuje precej več časa. Časi izvajanj programov v odvisnosti od n so podani tudi v diagramu na sliki 7.11.

Na tem mestu lažje opazimo tudi razliko med različicami programa POP. In sicer gledamo razliko med verzijama, ki vračata rešitvene plane z minimalnim številom akcij, in verzijama, ki vračata rešitvene plane z minimalnim številom korakov. Iz testov v domenah skladišče s paletami ter skladišče s paletami in vozički ni bilo mogoče sklepati, kateri od dveh pristopov je boljši. V določenih primerih se je bolje izkazal en, v drugih primerih pa drugi pristop. Na domenah linearna in kvadratna mreža se za učinkovitejšega pokaže POP, ki vrača rešitvene plane z minimalnim številom korakov.



Slika 7.11: Časi izvajanja v milisekundah v odvisnosti od n v domeni kvadratna mreža.

7.3.3 Delovanje programov s podanim vmesnim ciljem

Delovanje programa s podanim vmesnim ciljem smo testirali na domenah pravokotna in kvadratna mreža. Da bi bila podproblema primerljiva po zahtevnosti, smo za vmesni cilj izbrali literal, ki ga dosežemo približno na sredini izvajanja plana:

- Linearna mreža: vmesni cilj `at(a,⌈n/2⌉)`.

- Kvadratna mreža: vmesni cilj `at(a,n)`.

Domeni sta enostavni, zato iz vmesnega cilja ni težko razbrati vmesnega stanja prostora. Če je vmesni cilj `at(a, X)`, vmesno stanje prostora definiramo kot: začetno stanje `S` + {`at(a,X), clear(1)`} - {`at(a,1), clear(X)`}.
Rezultati testiranja so zbrani v tabelah 7.10 in 7.11 . Testirali smo le algoritma GRAPHPLAN in POP1, ki vrača rešitvene plane z minimalnim številom korakov.

Tabela 7.10: Časi izvajanja algoritmov za dosego cilja `at(a,n)` s podanim vmesnim ciljem `at(a,`$\lceil$`n/2`$\rceil$`)` na linearni mreži (v milisekundah).

n	4	5	6	7	8	9	10	11	12
št. akcij	3	4	5	6	7	8	9	10	11
GP	9	20	45	92	112	130	200	239	418
POP1_min_korakov	6	6	18	38	47	112	170	320	510

n	13	37	38	81	91
št. akcij	12	36	37	80	90
GP	535	6918	8025	79000	106000
POP1_min_korakov	1346	$>$ 10 min	/	/	/

Tabela 7.11: Časi izvajanja algoritmov za dosego cilja `at(a,`n^2`)` s podanim vmesnim ciljem `at(a,n)` na kvadratni mreži (v milisekundah).

n	3	4	5	6	7	8
št. akcij	4	6	8	10	12	14
GP	74	142	540	1284	2495	4643
POP1_min_korakov	10	63	338	1519	8683	49351

Komentar: Izvajanje programov se z uporabo vmesnega cilja močno pohitri. Čas izvajanja programa, ki vsebuje m akcij, ustreza enačbi:
*2 * čas izvajanja programa brez vmesnega cilja z m/2 akcij.*

Poglavje 8

Sklepi in nadaljnje delo

Na podlagi mnogih raziskav je znano, da ni pristopa planiranja, ki bi bil boljši od vseh ostalih. Niti ne obstaja hevristika, ki bi bila neodvisna od domene in bi bila boljša od vseh ostalih hevristik za vse probleme ali na vseh domenah. Vsak pristop planiranja ima svoje prednosti in slabosti. Čas izvajanja algoritma je odvisen od marsičesa. Določen program bo učinkovit na eni domeni planiranja in neuspešen na drugi. Morda se bo za določeno množico ciljev odrezal bolje kot nek drug planer, vendar bo za drugo množico ciljev iz istega začetnega stanja sveta veliko počasnejši od drugega algoritma. Do teh ugotovitev smo prišli tudi v naši diplomski nalogi. Časi izvajanja algoritmov za planiranje so močno odvisni od domene planiranja in od ciljev, ki jih želimo doseči.

Obstajajo primeri problemov, za katere algoritma GRAPHPLAN in POP vrneta različen rešitveni plan. To potrjuje, da imata algoritma različna pristopa k planiranju. Na podlagi vseh testiranj, ki smo jih opravili, lahko sklepamo, da pa je algoritem GRAPHPLAN vseeno nekoliko učinkovitejši od programa POP. Slednji je boljši le na določenih enostavnih primerih domene, kjer je potrebno doseči le en cilj (množica ciljev vsebuje le en literal).

Zanimiva je tudi primerjava različnih verzij programa POP. Prva popravljena verzija je sicer v večini primerov hitrejša od druge, vendar zahteva definiranje vseh nekonsistentnih akcij domene, kar je v kompleksnejših domenah zelo nerodno. Ugotovili smo tudi, da ni bistvene razlike med programom, ki vrača rešitvene plane z

minimalnim številom akcij, in programom, ki vrača rešitvene plane z minimalnim številom korakov. Izbira med tem dvema je odvisna od tega, kaj nam je v planu bolj pomembno.

Algoritme je možno izboljšati in razširiti na mnogo načinov. Lotili smo se pohitritve algoritmov z definiranjem vmesnega cilja. Gre za zelo poenostavljeno verzijo planiranja z vmesnim ciljem, vendar smo že na tej uspeli videti, da se hitrost izvajanja programov močno poveča. Žal je program, ki smo ga sestavili za dvosmerni GRAPHPLAN, nedelujoč. Zanimivo bi bilo primerjati čase z osnovnim algoritmom GRAPHPLAN.

Opazili smo, da je domena skladišče s paletami in vozički za oba algoritma precej zahtevna. Za plan s sedmimi akcijami noben od njiju ne vrne rešitve niti po preteku ene ure. V enostavni domeni, kot je linearna mreža, se opazi razlika v časovni zahtevnosti algoritmov. V tej domeni v času dveh minut GRAPHPLAN poišče plan z 80 akcijami, POP pa le z 10 akcijami.

Nadaljnje delo vidimo predvsem v pohitritvi in razširitvi algoritmov na različne načine in primerjavi le-teh na različnih domenah planiranja. Algoritme bi bilo smiselno testirati na še večjem številu različnih domen in večjem številu primerov problemov. Tako bi lažje prišli do morebitnih sklepov, za kakšne primere in domene je boljši en algoritem in za kakšne drug. Zelo zaželeno bi bilo tudi ugotoviti, zakaj se posebej pri programu POP kombinatorika tako hitro povečuje. Posebej izzivalen je primer z linearno mrežo. Verjetno bi bilo treba slediti izvajanju algoritma in opazovati, ali se kopičijo nesmiselne kombinacije akcij in ali se mogoče ponavljajo očitno ekvivalentni vzorci. Ta vpogled bi lahko pripeljal do globljega razumevanja in idej za izboljšave. Vsekakor je področje dovolj široko in dopušča še veliko raziskovalnega dela.

Literatura

[1] A. Barrett, D. S. Weld: "Partial-Order Planning: Evaluating Possible Efficiency Gains" *Artificial Intelligence*, vol. 67, št. 2, str. 71-112, 1994.

[2] A. Blum, M. Furst: "Fast Planning Through Planning Graph Analysis" *Artificial Intelligence*, vol. 90, str. 281–300, 1997.

[3] A. Blum, J. Langford, "Probabilistic Planning in the Graphplan Framework", v zborniku *5th European Conference on Planning*, 1999, str. 319-332.

[4] I. Bratko, *Prolog programming for Artificial Intelligence*, Faculty of Computer and Information Science - Ljubljana University, četrta izdaja, 2012.

[5] S. Kambhampati, E. Lambrecht and E. Parker, "Understanding and Extending Graphplan", v zborniku *Proceedings of the 4th European Conference on Planning: Recent Advances in AI Planning*, 1997, str. 260-272.

[6] E. Marzal, E. Onaindia, L. Sebastia, "An Incremental Temporal Partial-Order Planner", v zborniku *AIPS Workshop on Planning for Temporal Domains Foreword* (ur. M. Fox, A. Coddington), 2002, str. 26-32.

[7] C.Muise, S. A. McIlraith, J. C. Beck, "Monitoring the Execution of Partial-Order Plans via Regression", v zborniku *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI-11)*, 2011, str. 1975–1982.

[8] B. Nebel, Y. Dimopoulos, J. Koehler, "Ignoring Irrelevant Facts and Operators in Plan Generation", v zborniku *European Conference on Planning (ECP)*, 1997, str. 338-350.

[9] E. D. Sacerdoti, "The Nonlinear Nature of Plans", v zborniku *In Proceedings of the Fourth International Joint Conference on Artificial Intelligence (IJCAI-75)*, 1975, str. 206-214.

[10] A. Tate, *Project Planning Using a Hierarchic Non-linear Planner*, D.A.I. Research Report No. 25, Department of Artificial Intelligence, University of Edinburgh, 1976.

Dodatek A

Programska koda popravljenega programa POP

A.1 1. verzija popravljenega programa POP

Podan je del kode programa POP [4], v katerega smo dodali nekaj novih stavkov kode. Ta popravek skupaj s podanimi nekonsistentnimi akcijami v definiciji domene odpravi napako v programu POP.

```
s( pop( Acts, [Cond:Time | OpenPs0], TrueConds0, Fin),
pop( [Action1:Time1 | Acts], OpenPs, TrueConds, Fin)) :-
    effects( Action1, Effects),
    del( Cond, Effects, RestEffects),
    can( Action1, PreConds1),
    0 #< Time1,  Time1 #< Time,
%-------------------novo-----------------------------------------
    add_conds_acts(Action1, Time1, Acts),
%-------------------konec novega---------------------------------
    add_times( Time1, PreConds1, NewOpenPs),
    add_intervals( Time1, RestEffects, RestEffectsTimes, Fin),
    Time #=< Time2,
```

```
    add_conds( [ Cond:Time1/Time2 | RestEffectsTimes],
                 TrueConds0, TrueConds),
    conc( NewOpenPs, OpenPs0, OpenPs).

%-------------------novo-----------------------------------------------
% add_conds_acts(Action, Time, Acts)
% doloci casovne omejitve med akcijo Action in akcijami
% iz mnozice akcij Acts

add_conds_acts(Action, Time, []).
add_conds_acts(Action, Time, [Act:Time1| Acts]):-
        no_conflict_act(Action:Time, Act:Time1),
        add_conds_acts(Action, Time, Acts).

% no_conflict_act(Action:Time, Act:Time1)
% preveri, ali sta akciji Action in Act nekonsistentni
% ce sta, poskrbi, da casa Time in Time1 nista enaka

no_conflict_act(Action:Time, Act:Time1):-
        inconsistent(Action,Act),!,
        (Time#<Time1 ; Time1#<Time)
        ;
        true.
%-----------konec novega-----------------------------------------------
```

A.2 2. verzija popravljenega programa POP

Podan je del kode programa POP [4], v katerega smo dodali nekaj novih stavkov kode. Ta popravek brez podanih nekonsistentnih akcij v definiciji domene odpravi napako v programu POP.

```
s( pop( Acts, [Cond:Time | OpenPs0], TrueConds0, Fin),
   pop( [Action1:Time1 | Acts], OpenPs, TrueConds, Fin)) :-
   effects( Action1, Effects),
   del( Cond, Effects, RestEffects),
   can( Action1, PreConds1),
   0 #< Time1,  Time1 #< Time,
%----------------------novo-------------------------------------------
   add_conds_acts(PreConds1, Time1, Acts),
   add_conds_acts2(RestEffects, Time1, Acts),
%---------------------konec novega-----------------------------------
   add_times( Time1, PreConds1, NewOpenPs),
   add_intervals( Time1, RestEffects, RestEffectsTimes, Fin),
   Time #=< Time2,
   add_conds( [ Cond:Time1/Time2 | RestEffectsTimes],
                    TrueConds0, TrueConds),
   conc( NewOpenPs, OpenPs0, OpenPs).

%---------------------novo-------------------------------------------
% add_conds_acts(PreConds, Time, Acts)
% Preveri, ali so predpogoji PreConds nekonsistentni z ucinki
% akcij iz mnozice Acts. Ce so, postavi omejitve casa Time.

add_conds_acts(_, _, []).
add_conds_acts(PreConds, Time, [Act:Time1| Acts]):-
        effects(Act,Effects),
        no_conflict_act(PreConds, Time, Effects, Time1),
        add_conds_acts(PreConds, Time, Acts).

% no_conflict_act(PreConds, Time, Effects, Time1)
% Preveri, ali so prostorski literali iz mnozice PreConds
% nekonsistentni s prostorskimi literali iz mnozice Effects.
```

```
% Dodaj ustrezne omejitve casa Time.

no_conflict_act([], _,_, _).
no_conflict_act([Cond|RestConds], Time, Effects, Time1):-
        no_conflict_act2(Cond, Time, Effects, Time1),
        no_conflict_act(RestConds,Time,Effects, Time1).

% no_conflict_act2(Cond, Time, Effects, Time1)
% Preveri, ali je prostorski literal Cond nekonsistenten
% s prostorskimi literali iz mnozice Effects.
% Dodaj ustrezne omejitve casa Time.

no_conflict_act2(_,_,[],_).
no_conflict_act2(Cond,Time, [Effect|RestEffects],Time1):-
        no_conflict_act3(Cond, Time, Effect,Time1),
        no_conflict_act2(Cond, Time,RestEffects,Time1).

% no_conflict_act3(Cond, Time, Effect, Time1)
% Preveri, ali je prostorski literal Cond nekonsistenten
% s prostorskim literalom Effect.
% Dodaj ustrezne omejitve casa Time

no_conflict_act3(Cond,Time,Effect,Time1):-
        inconsistent(Cond,Effect),!,
        (Time#<Time1 ; Time1#<Time)
        ;
        true.

% add_conds_acts2(Effects,Time,Acts)
% Preveri, ali so ucinki Effects nekonsistentni s predpogoji
% akcij iz mnozice Acts. Ce so, postavi omejitve casa Time.
```

```
add_conds_acts2(_, _, []).
add_conds_acts2(Effects, Time, [Act:Time1| Acts]):-
        can(Act,PreConds),
        no_conflict_act_two(Effects, Time, PreConds, Time1),
        add_conds_acts2(Effects, Time, Acts).

% no_conflict_act_two(Effects, Time, PreConds, Time1)
% Preveri, ali so prostorski literali iz mnozice Effects
% nekonsistentni s prostorskimi literali iz mnozice PreCond.
% Dodaj ustrezne omejitve casa Time.

no_conflict_act_two([], _,_, _).
no_conflict_act_two([Effect|RestEffects], Time, PreConds, Time1):-
        no_conflict_act2_two(Effect,Time,PreConds, Time1),
        no_conflict_act_two(RestEffects,Time,PreConds, Time1).

% no_conflict_act2_two(Effect, Time, PreConds, Time1)
% Preveri, ali je prostorski literal Effect nekonsistenten
% s prostorskimi literali iz mnozice PreConds.
% Ddodaj ustrezne omejitve casa Time.

no_conflict_act2_two(_,_,[],_).
no_conflict_act2_two(Effect,Time, [Cond|RestConds],Time1):-
        no_conflict_act3_two(Effect, Time, Cond,Time1),
        no_conflict_act2_two(Effect, Time,RestConds,Time1).

% no_conflict_act3_two(Effect, Time, Cond, Time1)
% Preveri, ali je prostorski literal Effect nekonsistenten
% s prostorskim literalom Cond.
% Dodaj ustrezne omejitve casa Time.
```

```
no_conflict_act3_two(Effect,Time,Cond,Time1):-
        inconsistent(Effect,Cond),!,
        (Time#<Time1 ; Time1#<Time)
        ;
        true.
%-----------konec novega-----------------------------------
```

A.3 POP z minimalnim številom korakov

Podani so deli kode programa POP [4], ki smo jih spremenili z namenom, da program vrača rešitvene plane z minimalnim številom korakov.

```
plan( StartState, Goals, Plan) :-
    add_intervals( 0, StartState, TrueConds, Finish),
    add_times( Finish, Goals, OpenConds),
    EmptyPlan = pop( [ ], OpenConds, TrueConds, Finish),
%--------Spremenjeno, tako da vraca plane----------------
%--------- z minimalnim stevilom korakov-----------------
    Finish in 0..100,
    indomain( Finish),
    st_robotov(StartState,0,X),
    MaxActions in 1..X,
    depth_first( EmptyPlan, SolutionPath, Finish),
    once( indomain( MaxActions)),
% ----------Konec spremembe.-----------------------------
    conc( _, [Plan], SolutionPath).

%----------novo: stevilo robotov v domeni---------------
% st_robotov(State,NumRobotsBefore,NumRobotsAfter)
% NumRobotsAfter je stevilo robotov v domeni z zacetnim
```

```
% stanjem State. Pri klicu pravila postavimo
% NumRobotsBefore na 0.

st_robotov([],N,N).
st_robotov([Literal|RestOfState],NB,NA):-
        (member(at(A,_),[Literal]),
        robot(A),
        N is NB+1,
        st_robotov(RestOfState,N,NA))
        ;
        st_robotov(RestOfState,NB,NA).
%----------------konec novega-------------------------

depth_first( POP, [POP], _) :-
  POP = pop( _, [], _, _).
%---------Spremenjeno, tako da vraca plane-----------
%--------- z minimalnim stevilom korakov--------------
depth_first( First, [First | Rest], Finish) :-
  First = pop( _, _, _, Fin),
  ( Fin < Finish, !
    ;
    Second = pop( _, _, _, Fin2) ),
  s( First, Second),
  depth_first( Second, Rest, Finish).
%----------------Konec spremembe----------------------
```

Dodatek B

Programska koda razširjenih programov POP in GRAPHPLAN

B.1 Planiranje s podanim vmesnim ciljem

Podana je programska koda za planiranje z vmesnim ciljem. Kodo lahko dodamo v originalna programa GRAPHPLAN in POP ali jo shranimo kot samostojni prologov program, v katerem definiramo lokacijo pravila `plan`.

```
% plan2(State0, MidGoal, Goals, Plan)
% State0 je zacetno stanje prostora, Goals so koncni
% cilji, ki jih zelimo doseci in MidGoal je vmesni cilj.
% Zgeneriramo vmesno stanje, v katerem je vmesni
% cilj MidGoal resnicen. Nato zacetni problem
% razdelimo na dva podproblema, ki ju resujemo z
% algoritmom GRAPHPLAN ali POP. Resitvi podproblemov
% Plan1 in Plan2 zdruzimo v koncno resitev Plan.

plan2(State0, MidGoal, Goals, Plan):-
```

```
        vmesniState(State0,MidGoal,MidState),
        plan(State0,MidGoal,Plan1),
        plan(MidState,Goals,Plan2),
        conc(Plan1,Plan2,Plan).

% vmesniState(State, MidGoal, MidState)
% iz zacetnega stanja prostora State in
% vmesnega cilja MidGoal zgeneriramo vmesno
% stanje MidState, v katerem je MidGoal resnicen

vmesniState(S,MidGoal,MidState):-
        member(at(X,Y1), MidGoal),
        member(at(X,Y2), S),
        del(at(X,Y2),S,MidState0),
        del(clear(Y1),MidState0,MidState1),
        conc([at(X,Y1), clear(Y2)], MidState1,MidState).
```

Pri tem moramo opozoriti, da je izpis rešitvenega plana za GRAPHPLAN pravilne oblike, za program POP pa ne. Pri slednjem smo za izpis v definicijo pravila `plan2` dodali kodo:

```
show_pop(Plan1),
show_pop(Plan2).
```

Ta dva stavka izpišeta rešitvena plana podproblemov, iz katerih z združitvijo razberemo končni rešitveni plan.

B.2 Dvosmerni GRAPHPLAN

Podana je programska koda za dvosmerni GRAPHPLAN. Zaradi boljše preglednosti smo podali le tiste dele kode, ki se od originalnega programa GRAPHPLAN [4] razlikujejo ali pa so na novo dodani. Opozorilo: program ne deluje pravilno!

```
plan( StartState, Goals, Plan) :-
```

```
    findall( P/1, member( P, StartState), StartLevel1),
%------na novo dodan stavek ----------------------------
% zacetni nivo planirnega grafa od ciljev nazaj
    findall( G/1, member( G, Goals), StartLevel2),
    setof( action( A, PreCond, Effects), (effects( A, Effects),
            can( A, PreCond)), AllActions),
%-------spremenjen stavek-------------------------------
    graphplan( [StartLevel1],[StartLevel2], Goals, StartState,
                Plan, AllActions).

%--------------Zacetek sprememb v kodi--------------------------
% Robni primer-zacetno stanje
% Preverimo, ali so ze v zacetnem stanju izpolnjeni vsi cilji
graphplan( [StateLevel1|[]], [StateLevel2|[]], Goals1,
                Goals2, Plan, AllActs):-
    satisfied(StateLevel1, Goals1),
    extract_plan([[StateLevel1]],Plan)
        ;
% z desne strani izpeljemo nov ActionLevel1 in NewStateLev1
    expand(StateLevel1,ActionLevel1,NewStateLev1,AllActs),
% z leve strani izpeljemo nov ActionLevel2 in NewStateLev2
    expand2(StateLevel2,ActionLevel2,NewStateLev2,AllActs),
    graphplan([NewStateLev1,ActionLevel1,StateLevel1],
        [NewStateLev2,ActionLevel2,StateLevel2],Goals1,
        Goals2,Plan,AllActs).

graphplan( [StateLevel1, ActionLevel1_0,
                StateLevel1_0|PlanGraph1],
                [StateLevel2, ActionLevel2_0,
                StateLevel2_0|PlanGraph2],
                Goals1, Goals2, Plan, AllActs):-
```

```
% Preverimo, ali se ujema kateri izmed parov nivojev
    ((satisfied2(StateLevel1_0, StateLevel2);
    satisfied2(StateLevel1_0, StateLevel2_0)),
    extract_plan([StateLevel1,ActionLevel1_0,
          StateLevel1_0|PlanGraph1],Plan1),
    extract_plan([StateLevel2,ActionLevel2_0,
          StateLevel2_0|PlanGraph2],Plan2),
% Akcije iz Plan2 je potrebno izvesti v obratnem vrstnem redu
    reverse(Plan2,Plan3),
    conc(Plan1,Plan3,Plan),
    );
    expand( StateLevel1, ActionLevel1_1, NewStateLev1, AllActs),
    expand2( StateLevel2, ActionLevel2_1, NewStateLev2, AllActs),
    graphplan([NewStateLev1,ActionLevel1_1,StateLevel1,
          ActionLevel1_0, StateLevel1_0|PlanGraph1],
          [NewStateLev2,ActionLevel2_1, StateLevel2,
          ActionLevel2_0,StateLevel2_0|PlanGraph2],Goals1,
          Goals2,Plan,AllActs).
%-----------Konec sprememb v kodi-----------------
% ----------Zacetek na novo dodane kode----------------
% Preverimo, ali se nivoja ujemata
satisfied2(Level1,Level2):-
          findall(G/1, member(G/1, Level2), TrueLevel2),
          satisfied3(Level1,TrueLevel2).

% Preverimo, ali so vsi literali nivoja z leve strani, ki imajo
% vrednost indikatorja 1, prisotni tudi v nivoju z desne strani
% in imajo prav tako vrednost indikatorja 1.
satisfied3( _, []).
satisfied3( StateLevel, [G | Goals]) :-
    member( G, StateLevel),
```

```
    satisfied3( StateLevel, Goals).

% Generiranje novih dveh nivojev akcij in literalov z
% leve strani s pomocjo regresije ciljev.
expand2( StateLev, ActionLev, NextStateLev, AllActs) :-
    add_actions2(StateLev,AllActs, [], ActionLev1, [],
        NextStateLev1),
    findall( action(persist(P), [P],[P]), member(P/_, StateLev),
        PersistActs),
    add_actions(StateLev, PersistActs, ActionLev1, ActionLev,
        NextStateLev1, NextStateLev),
    mutex_constr( ActionLev),
    mutex_constr(NextStateLev).

% Poisci akcije, ki imajo literale trenutnega nivoja za
% ucinek. Shrani jih v akcijski nivo in njihove predpogoje
% v nov nivo literalov.
add_actions2([],_, ActLev,ActLev,NextLev, NextLev).
add_actions2([Goal/T|Goals], AllActs, ActLev0, ActLev,
        NextLev0, NextLev) :-
  goalActs(Goal/T, AllActs,[], ActsThatAchieveGoal),
  zdruziAkcije(ActLev0,ActsThatAchieveGoal),
  conc(ActLev0,ActsThatAchieveGoal, ActLevel0),
  dodajPredpogoje(ActsThatAchieveGoal,NextLev0,NextLev1),
  !,
  add_actions2(Goals, AllActs, ActLevel0, ActLev,
        NextLev1, NextLev).

% Poisci akcije, ki imajo Goal za ucinek
goalActs(_,[],ATAG,ATAG).
goalActs(Goal/T,[action(A, PreCond, Effects) |Acts],
```

```
        ATAG0, ATAG):-
    member(Goal, Effects),
    TA in 0..1,
    NewTA #= T-TA,
    posodobiOmejitve(ATAG0,[],NATAG, NewTA, NewTA ,NewNewTA),
    goalActs(Goal/T, Acts, [A/NewNewTA|NATAG], ATAG)
    ;
    goalActs(Goal/T, Acts, ATAG0, ATAG).

% Posodobi omejitve akcij
posodobiOmejitve([], NATAG, NATAG, _, NewTA, NewTA).
posodobiOmejitve([B/TB|RestATAG0], NewATAG0, NewATAG,
        TA, ChangingTA, NewTA):-
    NewNewTA in 0..1,
    NewNewTA #= ChangingTA-TB,
    NewTB in 0..1,
    NewTB #= TB-NewNewTA,
    NewATAG1=[B/NewTB|NewATAG0],
    posodobiOmejitve(RestATAG0, NewATAG1, NewATAG,
        TA, NewNewTA, NewTA).

% Izenaci indikatorje akcij, ki so ze v planu
zdruziAkcije([], A).
zdruziAkcije([A/TA|Acts], ATAG):-
        member(A/TB, ATAG),
        TA#=TB,
        zdruziAkcije(Acts, ATAG)
        ;
        zdruziAkcije(Acts, ATAG).

% Zberi vse predpogoje mnozice akcij
```

```
dodajPredpogoje([],NextLev,NextLev).
dodajPredpogoje([A/TA|Acts],NextLev0,NextLev):-
        can(A,PreCond),
        add_preconds(TA,PreCond,[],NL),
        conc(NL,NextLev0,NextL),
        dodajPredpogoje(Acts,NextL,NextLev).

% Pogojem doloci vrednosti indikatorjev
add_preconds(_,[], StateLev, StateLev).
add_preconds(TA, [P|Ps], StateLev0, ExpandedState):-
  (member(P/TP, StateLev0),
   StateLev=StateLev0,
   TA#=<TP
   ;
   TA#=<TP,
   StateLev=[P/TP|StateLev0],!
   ),
   add_preconds(TA,Ps, StateLev,ExpandedState).
%-----------Konec na novo dodane kode----------------
```