Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jakob Uršič

Planiranje zbiranja blaga v skladišču s hevrističnimi algoritmi

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Optimalno zbiranje specificiranega blaga v skladišču je zahteven kombinatorični problem. V svojem delu razvijte in implementirajte hevristični algoritem za robotsko zbiranje blaga, ki bo deloval za nastavljive konfiguracije skladišča in za enega ali več robotov. Posebej se posvetite definiranju hevurističnih ocen stanj skladišča med zbiranjem blaga. Predlagane hevristike eksperimentalno ovrednotite s poskusi na raznih problemih zbiranja blaga v skladišču in za razna števila robotov.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jakob Uršič, z vpisno številko **63100274**, sem avtor diplomskega dela z naslovom:

Planiranje zbiranja blaga v skladišču s hevrističnimi algoritmi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal 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 na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 30. avgusta 2014 Podpis avtorja:

Kazalo

Povzetek

Planiranje zbiranja blaga je smiseln proces v vsakem skladišču. V okviru te naloge je bil razvit simulator enostavnega skladišča, na katerem lahko s pomočjo algoritma A* generiramo plane za pobiranje vnaprej določene količine blaga za enega ali več robotov. Način dodeljevanja prioritete pri preiskovanju stanj je v veliki meri odvisen od hevrističnih ocen. Implementirali smo pet načinov za izračun hevrističnih ocen in jih preizkusili na primerih z različnimi lastnostmi, ter z različnim številom robotov. Dobljene rezultate smo analizirali, ter opozorili na morebitne pomanjkljivosti posameznih hevristik.

Ključne besede: algoritem A*, hevristika, skladišče, zbiranje naročil, robot, planiranje.

Abstract

Planning of order picking is essential process in every warehouse. In this thesis, we developed a simple warehouse simulator, which allows us to do various searches on path finding for a certain amount of items for one or more robots, using the A* algorithm. Heuristic guidance of search is mainly based on heuristic evaluation. We have implemented five different heuristic estimates, which we tested experimentally on examples with different warehouse configurations and with different numbers of robots. We also analysed the results and pointed out the drawbacks of each heuristic.

Keywords: algorithm A*, heuristic, warehouse, order picking, robot, planning.

Poglavje 1

Uvod

Iskanje optimalnih poti s preiskovalnimi algoritmi se pogosto uporablja na mnogih področjih, še posebej to velja za področje umetne inteligence. Gre za iskanje poti, ki nas pripelje iz nekega začetnega stanja do ciljnega. Pogost primer na to temo je iskanje poti robota do določene ciljne točke. To je razmeroma enostaven problem, ki se v literaturi velikokrat pojavi kot primer pri razlagi ustreznih algoritmov.

Večina takšnih preiskovanj torej temelji na enem robotu, ki išče pot do točno določenega cilja. Tudi literatura je povečini omejena na take in podobno enostavne primere, veliko težje pa je najti informacije o kompleksnejših preiskovanjih, kjer se v skladišču lahko nahaja tudi več robotov z določeno kapaciteto, ki iščejo optimalno pot za pobiranje in odlaganje določene količine blaga. V tej nalogi smo zato poskusili raziskati ta, ne tako pogost primer iskanja poti.

Implementirali smo simulator, na katerem je mogoče poustvariti razmere v enostavnem skladišču. Na njem smo uporabili enega ali več robotov različnih kapacitet, s katerimi smo pobirali škatle dveh vrst in jih prenašali do odlagališča. Preiskovanja smo izvedli s pomočjo algoritma A*, ki je eden izmed najbolj uporabljenih algoritmov s tega področja.

V nalogi bomo najprej predstavili simulacijo skladišča. Razložili bomo pravila sveta v simulatorju, premike in akcije robotov, ter opisali način za

določanje lastnosti posameznih izvajanj v skladišču.

Nato bomo podrobno po korakih prikazali, kako algoritem A* deluje na omenjeni simulaciji, ter kako se iz posameznih premikov in možnih akcij robotov generirajo nova stanja.

Nadaljevali bomo z opisom treh optimističnih hevristik za enega robota, na katerih bomo izvedli tudi različne teste, ter dobljene rezultate analizirali.

Podobno bomo naredili tudi s hevristikami za več robotov, le da bomo v tem primeru uporabili samo dve, od tega eno optimistično. Tudi tu bomo izvedli različne teste in zapisali dobljene ugotovitve.

Poglavje 2

Simulacija robotovega sveta v skladišču

V okviru te naloge je bila razvita aplikacija, s katero je mogoče simulirati preprosto skladišče omejene velikosti. V skladišču se lahko nahaja eden ali več robotov, ki zbirajo in odlagajo dve vrsti škatel, modre in rdeče. Pobrane škatle je mogoče odlagati na enem ali več odlagališčih, odvisno od začetnih nastavitev. Obstaja tudi možnost, da se na površino postavi ovire. Smisel simulacije skladišča je testiranje različnih hevrističnih funkcij za enega ali več robotov, z uporabo algoritma A*, kjer je cilj, da roboti poberejo določeno število škatel in ga odnesejo na odlagališče. Podrobnejše delovanje aplikacije bomo predstavili v nadaljevanju.

2.1 Nastavitve za simuliranje različnih stanj

Preden aplikacijo zaženemo, je potrebno nastaviti nekaj začetnih parametrov. V program moramo vnesti okolje, ki ga želimo simulirati, v računalniku razumljivi obliki. Začnimo s predstavitvijo sveta v skladišču. Skladišče ima omejeno površino pravokotne oblike, sestavljeno iz kvadratov, ki je predstavljena z dvodimenzionalno tabelo velikosti m x n. Velikost tabele odraža velikost površine skladišča, vsak kvadrat na tej površini pa je predstavljen z

urejenim parom (x,y), kjer število x spada v interval (2.1), y pa v (2.2):

$$0 \leq x \leq m - 1, \tag{2.1}$$

$$0 \leq y \leq n - 1. \tag{2.2}$$

Na vsak kvadrat na ustvarjeni površini lahko položimo enega izmed 6 različnih elementov, ki so v programu predstavljeni s številkami od 0 do 5, kot je prikazano v tabeli (2.1).

`Predstavitev v programu`	`Opis elementa`	Grafičen prikaz
0	prazno polje	sivo polje
1	robot	zelen krog
2	modra škatla	moder kvadrat
3	rdeča škatla	rdeč kvadrat
4	odlagališče	rumeno polje
5	ovira	črno polje
6	robot na odlagališču	zelen krog na rumenem polju
7	robot in modra škatla	moder kvadrat na zelenem krogu
8	robot in rdeča škatla	rdeč kvadrat na zelenem krogu

Tabela 2.1: Opis programske in grafične predstavitve elementov

Obstajajo še naslednja pravilna stanja, prikazana v tabeli (2.1), označena s številkami od 6 do 8, ki se lahko pojavijo med delovanjem programa, praviloma pa naj jih ne bi vnašali v začetno stanje.

Omejitve pri vnosu elementov v prostor so naslednje: vnesti je potrebno vsaj eno ali več odlagališč, vsaj enega ali več robotov, poljubno število modrih in rdečih škatel, ter poljubno število ovir. Preostala polja morajo biti označena kot prazno polje. Program dopušča možnosti za raziskovanje veliko

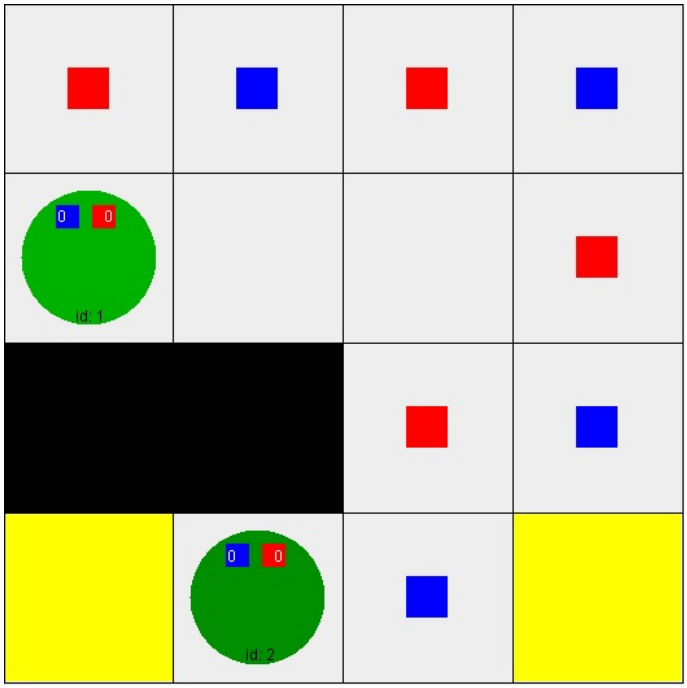
različnih scenarijev, vendar naj omenim, da se bomo v tej nalogi osredotočili zgolj na uporabo enega odlagališča, brez vnosa ovir.

Poleg začetnega stanja moramo nastaviti še lastnosti preiskovanja. Obstajajo 4 lastnosti: spremenljivka *heuristic*, ki se uporablja za izbiro vrste hevristične funkcije, po kateri se izračuna oceno za preostanek poti do ciljnega stanja za vsa vozlišča. Vrsto hevristike se izbere s številom od 1 naprej. Izbrano število predstavlja ustrezno hevristično funkcijo, ki je predstavljena pod enako zaporedno številko v poglavjih (4) in (6).

Druga vrsta nastavitve je *setCapacity*, kjer se izbere kapaciteta za vse robote v skladišču. S kapaciteto postavimo omejitev, koliko modrih in rdečih škatel skupaj lahko robot prenaša naenkrat. To pomeni, da je ustrezna nastavitev kapacitete katerokoli celo število, večje od 0.

Kot zadnje je potrebno določiti še ciljno število modrih in rdečih škatel. S tem povemo, koliko modrih in koliko rdečih škatel želimo, da roboti odložijo na odlagališču. Število modrih škatel se nastavi s $setBlueGoalNr$, rdečih pa s $setRedGoalNr$.

Primer enega izmed pravilnih nastavitev začetnega stanja z dvema robotoma, dvemi odlagališči, ter uporabo ovir, kjer uporabljamo hevristično funkcijo št. 1, roboti imajo kapaciteto 2, ter želimo pobrati 2 modri škatli in eno rdečo, je grafično prikazan na sliki (2.1).



Slika 2.1: Primer ene izmed pravilnih nastavitev začetnega stanja.

```
STATE ={ {3,2,3,2},
         {1,0,0,3},
         {5,5,3,2},
         {4,1,2,4} };

heuristic = 1;
setCapacity = 2;
setBlueGoalNr = 2;
setRedGoalNr = 1;
```

Poglavje 3

Opis delovanja algoritma A*

Ko je okolje, ki ga želimo preiskovati, vneseno v program, se po zagonu iz njega generirajo množice naslednikov, dokler se ne naleti na ciljno stanje. Naslednik je vsakršno stanje, ki izhaja iz trenutnega in na katerem se zgodi pravilen premik enega ali več robotov. Definicija pravilnih premikov za robote je opisana v nadaljevanju. Če se ciljnega stanja ne najde in je algoritem zaključil z delovanjem, pomeni, da rešitev za dani problem ne obstaja. Stanje je ciljno, če je v njem bilo odloženo toliko škatel ustreznih vrst, kolikor je bilo določeno v začetnih nastavitvah. Za preiskovanje prostora stanj skrbi algoritem A* s pomočjo hevristične funkcije. Poglejmo si, kako algoritem deluje.

Za izvajanje ustreznih operacij nad množico stanj algoritem potrebuje dva seznama. Poimenovali ju bomo $OPEN$ in $CLOSED$. Prvi seznam je namenjen hranjenju stanj, ki so potencialni kandidati za razvijanje. Na tem seznamu tudi pričakujemo, da bomo odkrili ciljno stanje. Stanja so razporejena od najmanjšega proti največjemu po vrednosti f (ang. f value), ki se za vsako stanje n izračuna po formuli (3.1), kjer je $g(n)$ cena prehojene poti v trenutnem stanju n, $h(n)$ pa hevristična ocena preostale poti iz trenutnega stanja n do ciljnega[3].

V procesu pregledovanja se velikokrat zgodi, da do nekega cilja vodi veliko cenovno enakovrednih poti. To lahko privede do tega, da algoritem zapravi

veliko časa s preverjanjem različic teh zelo podobnih poti, ki pa na optimalnost rešitve nimajo vpliva. Reševanja takega problema se lahko lotimo na več različnih načinov. V tej nalogi smo uporabili naslednjega: stanja, pri katerih je vrednost f izenačena, se razporedijo od najmanjše vrednosti proti največji po hevristični oceni (vrednost h). S tem dosežemo, da algoritem da prednost pregledovanju stanj, ki so bliže cilju[4].

$$f(n) = g(n) + h(n) \tag{3.1}$$

Stanje definiramo kot razvito, ko iz njega tvorimo vse mogoče naslednike. Seznam $CLOSED$ je namenjen hranjenju že razvitih vozlišč. Tak seznam je potreben, saj nam omogoča zavržbo vseh vozlišč, ki imajo enako stanje kot predhodno razvita vozlišča, hkrati pa imajo tudi večjo ali enako vrednost g. Takih stanj nočemo ponovno razvijati, saj se njihovi nasledniki že nahajajo na seznamu $OPEN$.

Ko sta seznama pripravljena, dodamo začetno stanje na seznam $OPEN$. To pomeni, da bomo iz tega stanja razvijali naslednike, dokler ne najdemo povezane vrste naslednikov, ki nas pripelje do rešitve. Naslednji del algoritma se ponavlja v zanki, dokler ne najdemo rešitve, oz. dokler ni seznam $OPEN$ prazen, kar pa bi pomenilo, da rešitve nismo našli.

Algoritem odvzame prvo stanje s seznama $OPEN$. Ker je seznam urejen po velikosti, ima to stanje v danem trenutku najmanjšo vrednost f, torej, to je stanje, od katerega si največ obetamo. Takoj zatem, ko stanje odvzamemo iz $OPEN$, ga, iz prej omenjenih razlogov, dodamo na seznam $CLOSED$.

Na tem mestu preverimo, če je stanje, ki smo ga pričeli obravnavati, mogoče že ciljno. Če je, zaključimo z zanko in vrnemo dobljeno stanje kot rešitev. V nasprotnem primeru ga razvijemo. Z razvitjem stanja dobimo vse njegove mogoče naslednike, ki so odvisni od možnosti premikanja robotov. Podroben opis razvoja stanj je opisan nadalje. Ko izpeljemo vse naslednike, izmed njih izločimo vse tiste, ki so že na seznamu $OPEN$ ali $CLOSED$, in imajo enako ali večjo vrednost g. Nato za vse preostale naslednike izračunamo hevristično oceno in jo shranimo kot lastnost posameznega

naslednika. Seznam $OPEN$ dopolnimo z omenjenimi nasledniki, in ga nato ponovno uredimo po velikosti. Postopek ponavljamo s ponovnim odvzemom prvega elementa s seznama $OPEN$[3].

3.1 Pravilnost premikov enega ali več robotov

Premiki robotov so pomemben del naloge, saj je od njih odvisna tvorba novih stanj. Napačen premik bi privedel do napačnega stanja in s tem bi lahko povzročil zmedo v celotnem delovanju programa. Zagotavljanje pravilnosti premikov za enega robota je dokaj enostavno. Premik robota se lahko zgodi v 4 smeri, to so gor, dol, levo ali desno. Kar je potrebno preveriti je samo to, da se robot ne premakne izven omejenega sveta. Če se robot nahaja v svetu velikosti m x n, se morata koordinati polja x in y, kamor se robot želi premakniti, nahajati znotraj intervalov (2.1) in (2.2), omenjenih v poglavju (2).

Bolj zapleteno je preveriti pravilnost premikov, če se v svetu nahaja več robotov. Seveda morajo še vedno vsi premiki izpolnjevati prej navedene zahteve za enega robota. Poleg tega, pa moramo biti pozorni, da ne pride do trkov med roboti. Preveriti moramo torej, da premiki med seboj niso izključujoči (ang. mutex, mutually exclusive). Premiki so medsebojno izključujoči, kadar velja vsaj ena izmed naslednjih trditev[1]:

(a) njuni predpogoji so nekonsistentni,

(b) njuni posledici sta nekonsistentni, ali

(c) predpogoj ene akcije je nekonsistenten s posledico druge, ali obratno.

Predpogoj za vsak premik robota je, da je polje, kamor se namerava premakniti, prazno. Če se želi robot r_1 premakniti na polje p, kjer se že nahaja nek drug robot r_2, predpogoj robota r_1(prazno polje p) ne bo konsistenten s predpogojem robota r_2, ki je njegovo nahajanje na polju p. Tak primer zadošča

prej omenjeni točki (a) in bi se ga med izvajanjem programa zavrglo. Primer, ki bi spadal pod točko (b) bi se zgodil, ko bi želela dva robota storiti premik na isto polje. Posledici obeh robotov bi bili, da se nahajata na istem polju. Takšno stanje ponovno ni pravilno, zato bi se tudi tak premik zavrgel. Za točko (c) lahko vzamemo enak primer predhodnemu, le da podamo drugačno razlago. Za premik robota na polje p moramo izpolnjevati predpogoj, da je polje p prazno. Če se namerava nek drug robot premakniti na isto polje p, bo njegova posledica nahajanje na tem polju, in zato predpogoj prve akcije na bo konsistenten s posledico druge.

3.2 Akcije robota

Vsak robot lahko z vsakim premikom opravlja različne akcije. Te so:

- robot pobere škatlo,
- robot odloži škatlo in
- robot zamenja škatlo ene vrste z drugo (modro z rdečo ali obratno).

V nalogi smo predpostavili, da se vse akcije zgodijo po premiku in so, izključujoč ceno premika, časovno brezplačne. To pomeni, da se z vsakim premikom robota v enem koraku lahko hkrati izvede tudi ena izmed omenjenih akcij.

Za omenjene akcije seveda veljajo logični predpogoji. Če želi robot pobrati škatlo, ne sme imeti zapolnjene kapacitete. Za odložitev škatle mora biti polje, na katerem se robot nahaja, prazno in hkrati mora robot škatlo, ki jo želi odložiti, prenašati. Za zamenjavo škatle morata podobno veljati dva predpogoja. Prvi je, da robot škatlo, ki jo želi zamenjati, nosi. Drugi pa, da se škatla, s katero robot želi izvesti zamenjavo, nahaja na robotovem polju.

3.3 Razvijanje vozlišč

Kot smo že omenili, je razplet razvitja vozlišč povsem odvisen od premikanja robotov. Program zato prične razvijati vozlišča z ugotavljanjem vseh mogočih kombinacij premikov za vse robote. Za vsakega preveri najprej prostorske omejitve, to je, da se robot ne premakne izven omejenega sveta. Ko pridobi pravilne premike za vsakega posameznega robota, te združi s premiki ostalih robotov. Združenje predstavlja vse kombinacije vsakega izmed premikov posameznega robota z vsakim izmed možnih premikov ostalih robotov, kjer vrstni red premikov nima teže. Pri združevanju je potrebno upoštevati še izpolnjevanje pogojev medsebojnega izključevanja(ang. interference), omenjenega v poglavju o robotovih premikih. Če katera izmed kombinacij premikov te pogoje izpolnjuje, jo zavržemo.

Algoritem zgoraj omenjene skupine premikov pospravi v seznam. Ta seznam vsebuje sezname pravilnih kombinacij premikov vseh robotov v prostoru. Iz vsakega seznama pravilnih premikov torej lahko tvorimo naslednika trenutnemu stanju, ki bo zagotovo pravilno stanje. Iz tega lahko sklepamo, da toliko, kolikor seznamov vsebuje omenjeni seznam, toliko je vseh različnih kombinacij pravilnih premikov. To pa ne pomeni, da bo tudi število naslednikov tolikšno. Z vsakim premikom namreč lahko v večini primerov tvorimo različna stanja. Če se robot premakne na prazno polje (oz. ostane na istem mestu), lahko iz takega premika tvorimo 3 različna stanja. To so premik na prazno polje, premik z odložitvijo modre škatle, ter premik z odložitvijo rdeče škatle. Za premike, kjer se škatlo odloži, se seveda predpostavlja, da jo robot v danem trenutku tudi prenaša. Če se na mestu, kamor se robot premakne, nahaja modra škatla, ravno tako lahko tvorimo troje različnih stanj: premik brez pobiranja škatle, premik s pobiranjem škatle, ali premik z menjavo škatle modre barve z rdečo. Podobno velja, če se premaknemo na polje z rdečo škatlo. Edini premik, iz katerega je mogoče tvoriti eno samo stanje je premik na odlagališče. Tam se v vsakem primeru odložijo vse škatle, ki jih nosi robot, in ki prispevajo k izpolnitvi cilja. Če robot nosi škatlo, ki za izpolnitev cilja ni potrebna, je ne odlaga.

Poglavje 4

Iskanje optimalne poti za enega robota

Preiskovalni algoritem je popoln, če garantirano vedno najde optimalno pot. Pogoj, da je algoritem A* popoln, najdemo v teoremu iz leta 1968 [2] in se nanaša na hevristično oceno h, uporabljeno v algoritmu. Torej, A* je popoln, če za vsako vozlišče n v prostoru stanj velja neenačba (4.1), kjer je h*(n) dejanska vrednost najcenejše poti od stanja n do cilja.

$$h(n) \leq h^*(n) \tag{4.1}$$

Če torej želimo s preiskovanjem vedno najti optimalno pot, moramo za vsako stanje v procesu preiskovanja podati optimistično hevristično oceno. To pomeni, da ocena poti, ki jo hevristika oceni, ni v nobenem primeru večja od dejanske najcenejše poti. V primeru iskanja optimalne rešitve z enim robotom se bomo osredotočili na tri optimistične hevristične ocene.

4.1 Hevristika 1

Če začnemo z najbolj trivialno hevristično oceno, lahko rečemo, da je ocena od vsakega vozlišča do cilja vedno enaka 0. Taka napoved je seveda optimistična. Ker je hevristična ocena enaka pri vseh vozliščih, bo A* pri preiskovanju upošteval samo ceno prehojene poti. Zapis formule za tak algoritem

najdemo v enačbi (4.2), kjer je h(n) = 0.

$$f(n) = g(n) + h(n) \tag{4.2}$$

4.2 Hevristika 2

Naslednji način za izračun hevristične ocene, uporabljene v tej nalogi, vsebuje nekaj več korakov. Za vsako škatlo v prostoru izračunamo manhattansko razdaljo do odlagališča. To je minimalna razdalja, ki jo potrebujemo od ene točke do druge, če se pomikamo po kvadratih v smereh gor, dol, levo ali desno. Vsak premik ocenimo z vrednostjo 1. Matematičen zapis za izračun omenjene razdalje je zapisan z enačbo (4.3), kjer sta a_x in a_y x in y koordinati točke a v prostoru. Enako velja za točko b.

$$Manhattan(a, b) = |a_x - b_x| + |a_y - b_y| \tag{4.3}$$

Nato te razdalje razporedimo po vrsti, od najmanjše do največje v dva seznama, za vsako vrsto blaga posebej. Iz teh dveh seznamov izračunamo vsoto, ki vsebuje le toliko najmanjših razdalj iz vsako vrste, kolikor jih je še trenutno potrebno pobrati. To pomeni, da od začetnega cilja odštejemo že odložene škatle in tudi tiste, ki jih robot v danem trenutku prenaša. Nato izračunano vsoto dobljenih razdalj delimo z začetno kapaciteto robota. Dobljeni količnik uporabimo kot končno oceno. Tak izračun je optimističen, saj smo vzeli le najkrajše razdalje, torej razdalje škatel najbližjih odlagališču, ter hkrati predpostavili, da bo robot vedno zapolnil kapaciteto.

Matematični zapis za hevristični del je zapisan pod enačbo (4.4), kjer B_i predstavlja i-ti element v urejenem seznamu manhattanskih razdalj modrih škatel do odlagališča, n_b začetno ciljno število modrih škatel, d_b trenutno število odloženih modrih škatel, ter c_b število modrih škatel, ki jih robot trenutno prenaša. Na podoben način so uporabljene oznake za rdeče škatle.

$$h(n) = \frac{\sum_{i=0}^{n_b - d_b - c_b} B_i + \sum_{i=0}^{n_r - d_r - c_r} R_i}{kapaciteta} \tag{4.4}$$

4.3 Hevristika 3

Zadnji način za izračun hevristične ocene pa je naslednji: podobno kot v 2. primeru, izračunamo manhattanske razdalje in jih razporedimo v dva seznama, za vsak tip škatel posebej. Tokrat sta seznama urejena po dveh kriterijih. Prvi izmed kriterijev je manhattanska razdalja med škatlo in odlagališčem, razvrščena od najmanjše proti največji. Drugi kriterij, ki se uporabi samo v primeru, če je prvi izenačen, pa je manhattanska razdalja od škatle do robota, tudi razvrščena od najmanjše proti največji. Uvedba drugega kriterija je pomembna, saj v nasprotnem primeru lahko hevristična ocena ne bi bila več optimistična.

Iz teh dveh seznamov nato sestavimo tretjega, ki vsebuje samo toliko elementov, kolikor škatel je še potrebno pobrati, tako modrih, kot rdečih. Tudi elementi v tretjem seznamu so razporejeni po enakih kriterijih kot prva dva. Nadalje izračunamo razdaljo od robota do najbližje izmed najbolj oddaljenih škatel iz dobljenega seznama. To pomeni, da v primeru, če je na tem seznamu na zadnjem mestu več razdalj z enako vrednostjo, vzamemo tisto, ki je najbližja robotu. Ta razdalja je že prvi seštevanec, ki ga štejemo v končni rezultat. Tukaj naj omenim, da v primeru, kadar je izračunana razdalja enaka 0, se ji dodeli vrednost 1. Takšna dodelitev je smiselna, saj bo robot za pobiranje škatle še vedno potreboval 1 korak (ostati bo moral na istem mestu), četudi se že nahaja na mestu škatle. Nato tej vrednosti dodamo omenjeno najbolj oddaljeno razdaljo škatle s seznama. Ta dva seštevanca smo obravnavali posebej, za ostale pa velja naslednji način: od zadnjega elementa na seznamu skočimo naprej za toliko elementov, kolikor je kapaciteta robota. Dvakratnik vrednosti na tem mestu prištejemo k skupni dosedanji vsoti. Ta postopek ponavljamo, dokler ne skočimo izven seznama.

Tudi ta hevristika je optimistična, saj se še vedno predpostavi, da robot prehodi najkrajšo pot do blaga, ter nato najkrajše poti do odlagališča, pri čemer na teh najkrajših poteh vedno zapolni kapaciteto. Krajša pot zato zagotovo ne more obstajati.

Izračun (4.5) je v veliko primerih že dober približek resnični najkrajši

poti, vendar pa postane neuporaben, kadar robotov cilj ni več pobiranje blaga, temveč odlaganje. To se zgodi vsakič, ko robot zapolni kapaciteto, oz. kadar že pobere ciljno kvoto škatel in se mora vrniti do odlagališča. Za take primere se za izračun ocene uporabi nekoliko spremenjena formula (4.6). Ta oceni manhattansko razdaljo od robota do odlagališča ter vsoto dvakratnikov preostalih škatel, ki jih je še potrebno pobrati, po prej omenjenem algoritmu. S tem dosežemo boljšo natančnost vmesnih ocen, saj bi v nasprotnem primeru robot odlagališče iskal v vse smeri. Naj omenim, da se za zapolnjeno kapaciteto ne šteje primer, ko je le-ta zapolnjena z vsaj eno škatlo, ki je ni potrebno odnesti do odlagališča.

$$h(n) = max(Manhattan(M_m, robot), 1) + max(M) + 2 \times \sum_{i=1}^{n/k} M_{(n-k\times i)}, \quad (4.5)$$

kjer M_m predstavlja element v seznamu M, ki ima največjo oddaljenost od odlagališča in je hkrati izmed najbolj oddaljenih elementov najbližji robotu.

$$h(n) = Manhattan(robot, odlagališče) + 2 \times \sum_{i=0}^{n/k} M_{(n-k\times i)}, \quad (4.6)$$

kjer je M urejen seznam z manhattanskimi razdaljami blaga, ki ga moramo pobrati, n je dolžina seznama, ter k kapaciteta robota.

Poglavje 5

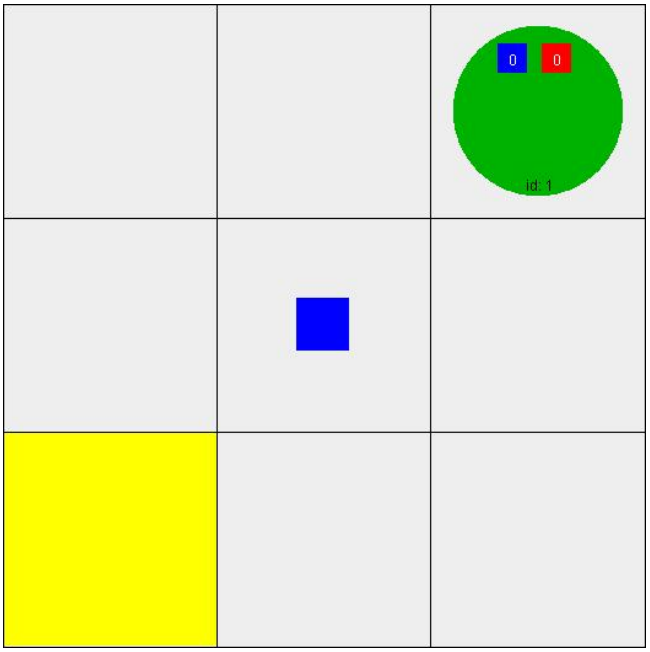
Eksperimentalna analiza optimističnih hevrističnih ocen za enega robota

Na omenjenih treh hevristikah, bomo izvedli poskuse na primerih z različnimi začetnimi stanji ter cilji. Opazovali bomo število vozlišč, ki jih algoritem razvije, ter za to porabljen čas. Za razvito vozlišče se šteje tisto, iz katerega se izpelje vse možne naslednike in potencialne kandidate doda v vrsto za preiskovanje. Naj omenim, da bo pri vseh rezultatih o številu razvitih vozlišč v naslednjem poglavju prišteto tudi ciljno stanje, ki omenjeni definiciji ne ustreza. Naredili bomo tudi primerjavo med hevristično oceno začetnega stanja(v nadaljevanju $h(start)$), ter dejanskim številom korakov optimalne rešitve.

5.1 Primer 1

Začnimo na preprostem primeru, velikosti 3x3. Naloga robota, ki ima kapaciteto nastavljeno na 1 je, da pobere modro škatlo, ter jo odloži na odlagališče. Primer je prikazan na sliki (5.1).

Rešitev po korakih, ki je enaka za vse tri hevristične funkcije, je naslednja:



Slika 5.1: Grafični prikaz sveta za primer 1.

premik navzdol, premik levo s pobiranjem, premik navzdol, premik levo z odlaganjem.

Že na enostavnem primeru je razviden prihranek z nepotrebno razvitimi vozlišči v primeru boljše hevristične ocene za posamezna stanja. Hevristika 1 enostavno razvija vozlišča do globine 4, kjer naleti na rešitev. Hevristika 2 že prihrani 11 razvitih vozlišč s tem, da imajo stanja, kjer je blago pobrano, prednost pred ostalimi. Zadnja hevristika pa je najbolj uspešna, ker imajo

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	21	0.11	0	4
2	10	0.08	2	4
3	5	0.08	4	4

Tabela 5.1: Rezultati dobljeni na primeru 1

Korak	Hevristika 1	Hevristika 2	Hevristika Št 3
0	0	2	4
1	0	2	3
2	0	0	2
3	0	0	1
4	0	0	0

Tabela 5.2: Prikaz, kako se hevristične ocene za vsako funkcijo spreminjajo po korakih.

najprej prednost stanja, ki vodijo do škatle, nato pa tista, bliže odlagališču.

Na tem mestu velja omeniti še to, da povečevanje kapacitete robota načeloma ne bi smelo vplivati na rezultate, vendar se pri hevristiki 2 zaradi neposredne odvisnosti ocene od kapacitete ta vseeno spreminja. Če kapaciteto povečamo na dva, bi zato ta hevristika razvila 5 stanj več, pri kapaciteti 3 bi jih bilo razvitih skupaj že 19, s povečanjem kapacitete na 4 pa bi bila enakovredna hevristiki 1, z rezultatom 21 razvitih vozlišč. To bi bilo mogoče popraviti s tem, da bi pri izračunu ocene vzeli za deljenje minimum med kapaciteto in vsoto končnega ciljnega blaga.

V tabeli (5.2) so po korakih razčlenjeni rezultati hevurističnih ocen vseh treh funkcij za rešitev primera 1. Pri hevristiki 2 in 3 si poglejmo, kako smo do takšnih rezultatov prišli. Vzemimo najprej hevristiko 2. V začetnem stanju je hevristična ocena 2. To je seštevek vseh manhattanskih razdalj modrih in rdečih škatel do odlagališča, deljen s kapaciteto, ki je 1. Ker rdečih škatel ni, modra pa je samo ena, in ta je od odlagališča oddaljena za 2 koraka, je tudi skupni seštevek enak 2. Ker je kapaciteta ena, po deljenju do spremembe rezultata ne pride. Ko se robot v naslednjem koraku premakne proti škatli, so vsi podatki za izračun ocene popolnoma enaki kot v predhodnem koraku. Do drugačnega izračuna pride, ko robot v drugem koraku škatlo pobere. Sedaj v svetu robota ni več škatel, zato imajo izračuni v korakih 2, 3 in 4 oceno 0.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	4058	1.56	0	26
2	3900	1.93	14	26
3	27	0.11	26	26

Tabela 5.3: Rezultati dobljeni na primeru 2.1

Hevristika 3 izračun izvede drugače. V začetnem stanju oceni razdaljo robota do ene izmed tistih škatel, ki jih namerava pobrati, in je najbolj oddaljena odlagališču. V primeru 1 je škatla samo ena, tako da se izračuna razdalja do te škatle. Ta razdalja šteje 2 koraka. Temu se prišteje oddaljenost omenjene škatle od odlagališča. Ta razdalja je tudi 2. Če bi bilo ciljno število škatel večje, bi izračun upošteval še dvakratnike do preostalih, odlagališču najbližjih škatel, ki bi jih morebiti še morali pobrati. Ker pa drugih škatel ni, je vsota že naša končna ocena, ki znese 4. V naslednjem koraku se robot približa škatli, zato se prvi del ocene zmanjša za 1, skupna ocena pa se spremeni na 3. Ko robot v drugem koraku škatlo pobere, se izračun formule osredotoči samo še na oddaljenost robota od odlagališča, zato dobimo oceno 2, ki se nato z vsakim korakom bliže odlagališču manjša za 1.

5.2 Primer 2

5.2.1 Izvedba 1

Za naslednji primer vzemimo isto površino, katero bomo popolnoma zapolnili s škatlami. Kapaciteta robota je še vedno 1, do odlagališča pa želimo odnesti vse škatle. Primer je prikazan na sliki (5.2).

Ta primer mogoče ni toliko zanimiv z vidika iskanja optimalne rešitve, saj robot lahko, z izjemo prve škatle, blago pobira v poljubnem vrstnem redu in bo končno število korakov na koncu še vedno enako, je pa zanimivo

pri tem opazovati količino razvitih vozlišč (5.3). Robotov plan za dosego cilja, ki ga poda npr. prva hevristika so naslednji premiki: dol s pobiranjem, dol, levo, levo z odlaganjem, gor, gor, desno s pobiranjem, dol, dol, levo z odlaganjem, gor, gor s pobiranjem, dol, dol z odlaganjem, gor, desno s pobiranjem, dol, levo z odlaganjem, gor s pobiranjem, dol z odlaganjem, desno, desno s pobiranjem, levo, levo z odlaganjem, desno s pobiranjem, levo z odlaganjem.

Hevristika 2 razvije skoraj toliko vozlišč, kot hevristika 1. To se zgodi zaradi dveh razlogov. Zelo pride do izraza neupoštevanje cene poti do blaga pri hevrističnem izračunu. Ker je kapaciteta robota 1, se odlaganje škatle zgodi sedemkrat, kar povzroči, da je dejanska cena poti do ciljnega stanja skoraj dvakrat večja od ocenjene. Slaba hevristična ocena pa ima za posledico veliko več preiskanih stanj. Drug razlog je ta, da hevristika ne obravnava posebej stanj, ko ima robot zapolnjeno kapaciteto, kar pa se v tem primeru pojavi zelo pogosto. V takih primerih se zato obnaša podobno, kot hevristika št. 1.

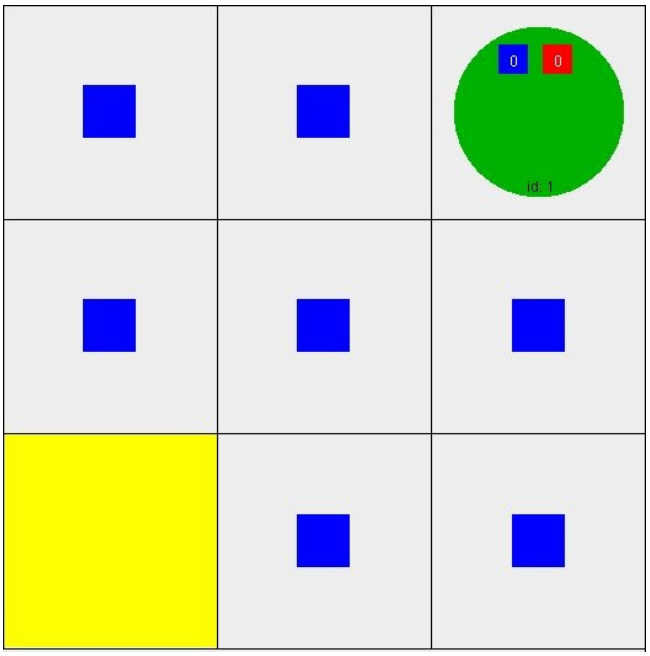
5.2.2 Izvedba 2

Vsi parametri iz primera 2.1 ostanejo enaki, razen velikost kapacitete, ki jo povečamo na 4. Rezultati so zbrani v tabeli (5.4).

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	3589	1.92	0	10
2	2001	1.09	3	10
3	389	0.31	8	10

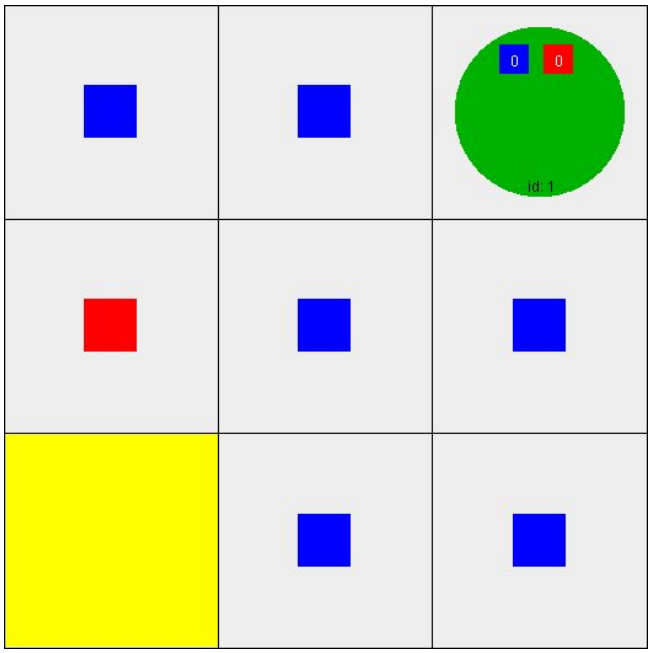
Tabela 5.4: Rezultati dobljeni na primeru 2.2

S povečanjem kapacitete pridemo do opažanj, da se hevristiki 1 število razvitih stanj zmanjša za približno osmino, hevristika 2 razvije približno po-



Slika 5.2: Grafični prikaz sveta za primer 2.

lovico manj stanj kot v predhodnem primeru, tretji hevristiki pa se število stanj poveča za več kot desetkrat. Povečanje kapacitete v tem primeru povzroči različen učinek. Ker se število korakov zmanjša iz 26 na 10 bi pričakovali zmanjšanje razvitih vozlišč. To se pri prvih dveh hevristikah tudi zgodi, pri tretji pa pride do povečanja. V predhodnem primeru je hevristična ocena tretje hevristike za stanje na vsakem koraku popolno pravilna, zato razvije minimalno število stanj. Ko povečamo kapaciteto na 4, pa hevristična funkcija v tem primeru izgubi natančnost. Eden izmed razlogov je to, da je manhattanska razdalja najbolj oddaljenega blaga manjša od kapacitete, izračun ocene pa predpostavi, da bo robot na poti do odlagališča vedno zapolnil kapaciteto. Posledica nenatančne ocene je povečanje nepotrebno razvitih stanj.



Slika 5.3: Grafični prikaz sveta za primer 3.

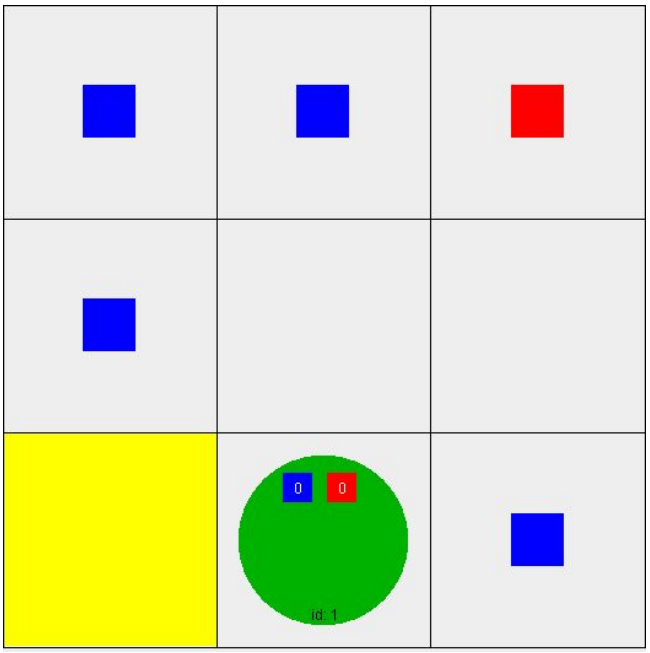
5.3 Primer 3

Želimo pobrati vse blago, robot ima kapaciteto 1, kot v primeru 2.1, vendar s to razliko, da smo zamenjali eno modro škatlo z rdečo, kot je prikazano na sliki (5.3). Rezultati so zbrani v tabeli (5.5).

Hevristika 3 je v primerjavi s primerom 2.1 obdržala enako število razvitih stanj, saj formula za izračun ocen, kljub rdeči škatli, še vedno ostane

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	20828	27.96	0	26
2	17025	32.31	14	26
3	27	0.1	26	26

Tabela 5.5: Rezultati dobljeni na primeru 3



Slika 5.4: Grafični prikaz sveta za primer 4.

popolnoma natančna. Ostali dve hevristiki pri izračunu nista tako natančni, zato sta že v primeru 2.1 razvili obe po nekaj tisoč stanj. Zaradi pojavitve rdeče škatle, se tukaj pojavijo možnosti za nova stanja, kjer se lahko modra škatla zamenja z rdečo. Ta menjava bi v večini primerov še vedno lahko pripeljala do optimalne rešitve, zato so ta stanja tudi pregledana. To je vzrok, da se pri prvih dveh hevristikah število razvitih stanj poveča za približno 5 krat.

5.4 Primer 4

5.4.1 Izvedba 1

Kapaciteto nastavimo na 1, končni cilj je do odlagališča prenesti 1 modro in 1 rdečo škatlo. Ta primer je zanimiv zato, ker mora robot za optimalno rešitev najprej pobrati rdečo škatlo, ter nato tisto modro, ki je najbližje odlagališču.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	1805	1.11	0	9
2	212	0.22	5	9
3	10	0.08	9	9

Tabela 5.6: Rezultati dobljeni na primeru 4.1

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	3527	3.72	0	9
2	1332	1.13	3	9
3	19	0.09	9	9

Tabela 5.7: Rezultati dobljeni na primeru 4.2

Torej za doseg optimalnega plana je možen samo 1 pravilen razplet. Stanje je prikazano na sliki (5.4), rezultati pa so zbrani v tabeli (5.6).

Primer optimalnega plana, ki ga izračuna hevristika 3, je zaporedje naslednjih premikov: gor, gor, desno s pobiranjem, dol, dol, levo, levo z odložitvijo, gor s pobiranjem, dol z odložitvijo.

Ta primer lepo pokaže, kako pomembna je čimbolj točna hevristična ocena. Z manj točno oceno se število nepotrebnih razvitih vozlišč zelo hitro povečuje.

5.4.2 Izvedba 2

Robot s kapaciteto 2 želi pobrati 2 modri in 1 rdečo škatlo. Takšne parametre smo izbrali zato, ker je optimalno število korakov še vedno enako, kot v predhodnem primeru (4.1), s to razliko, da je kapaciteta večja in se pobere 1 škatlo več. Zanima nas, koliko vpliva večja kapaciteta na število razvitih vozlišč, če je število korakov enako.

Iz dobljenih rezultatov (5.7) opazimo povečanje razvitih stanj pri vseh hevristikah. Večja kapaciteta in večje število ciljnega blaga praviloma ponujata več možnosti za dosego optimalne poti. Kjer izračun hevristike ni popolnoma natančen, se zato število raziskanih stanj hitro poveča. V prejšnjih primerih je hevristika 3 zaradi natančne ocene razvila le najmanjše možno število vozlišč, v tem primeru pa jih razvije dodatnih 9, kljub natančni začetni oceni. Razlog se skriva med delovanjem programa, kjer se začetna natančna ocena s posameznimi koraki pokvari, kot lahko vidimo v tabeli (5.8). Hevristična ocena 3. funkcije ne upošteva delno zapolnjene kapacitete, zato se s prvim premikom robota, ki je premik v desno s pobiranjem, hevristična ocena pokvari. Iz začetne 9, bi morali z enim korakom priti na 8, nov izračun pa poda rezultat 6. Ker se delno zapolnjena kapaciteta ne upošteva, izračun upošteva, da bo po pobrani najbolj oddaljeni rdeči škatli, na poti do odlagališča prostor še za pobiranje zadnje ciljne škatle modre barve. V tabeli vidimo, da sta ključna 1. in 5. korak. V prvem se ocena preveč zmanjša, zaradi že omenjenih razlogov, v 5. koraku pa se zapolni kapaciteta, zato se za izračun ocene uporabi spremenjena formula, ki je zopet natančna.

5.5 Primer 5

Vzemimo ploščo velikosti 5x5 ter na njej rekonstruirajmo primer 1, ki je bil izveden na plošči 3x3, kot je to prikazano na sliki (5.5). Rezultate najdemo v tabeli (5.9).

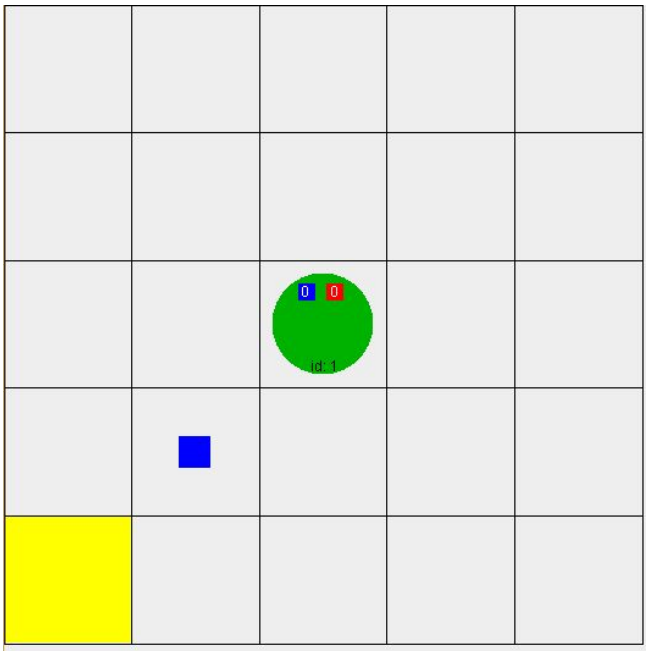
Ta primer nas privede do ugotovitve, da povečanje površine na 3. hevristiko sploh ne vpliva, kar pomeni, da ne preiskuje poti v nasprotno smer cilju. Ostalima hevristikama pa se število raziskanih vozlišč poveča, kar pomeni ravno nasprotno, torej, da iščeta tudi rešitev poti v nasprotno smer odlagališču. Za hevristiko 1 je to samoumevno, 2. hevristiki pa se to zgodi, ker njena hevristična funkcija ni informirana s ceno poti od robota do škatle, zato najprej preiskuje pot do škatle v vse smeri. Ko na škatlo naleti, imajo od te točke naprej stanja s pobrano škatlo prednost pred ostalimi, do izraza

Korak	Opis premika	Hevristična ocena do cilja
0	/	9
1	desno s pobiranjem	6
2	gor	5
3	gor z menjavo	4
4	levo	3
5	levo s pobiranjem	4
6	dol	3
7	dol z odlaganjem	2
8	gor s pobiranjem	1
9	dol z odlaganjem	0

Tabela 5.8: Podroben opis rešitve 3. hevristike po korakih

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	35	0.1	0	4
2	12	0.08	2	4
3	5	0.08	4	4

Tabela 5.9: Rezultati dobljeni na primeru 5



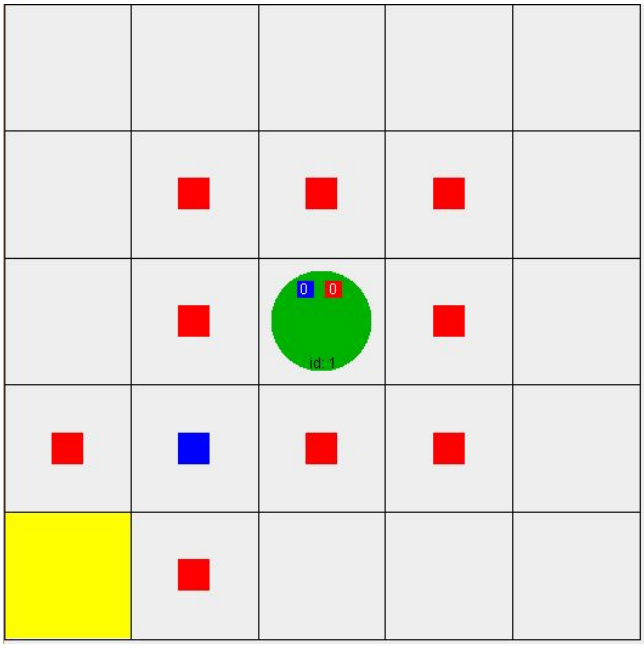
Slika 5.5: Grafični prikaz sveta za primer 5.

pa na tem mestu pride še druga pomanjkljivost, ki je, da ocena ni prilagojena za primer, ko je vse ciljno blago že pobrano. Torej, ko enkrat v tem primeru (Primer 5) robot pobere modro škatlo, se spremeni v hevristiko 1 (hevristična ocena je enaka 0), kar povzroči iskanje odlagališča v vse smeri.

5.6 Primer 6

Kaj se zgodi, če parametri ostanejo popolnoma enaki, s to razliko, da se v okolici robota nahaja blago, ki ga ni potrebno pobrati (5.6)? Odgovor najdemo v tabeli (5.10).

Opazimo povečanje razvitih vozlišč pri prvih dveh hevristikah. To je posledica večjega števila možnih stanj, ki ga povzročijo rdeče škatle. Za primer vzemimo prva dva koraka cenovno optimalne poti, to je pot do modre škatle. Za dosego te optimalne poti, lahko robot prvi korak naredi na 4 načine. Se premakne levo ali navzdol in v obeh primerih lahko ali pobere



Slika 5.6: Grafični prikaz sveta za primer 6.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	213	0.27	0	4
2	26	0.11	2	4
3	5	0.08	4	4

Tabela 5.10: Rezultati dobljeni na primeru 6

rdečo škatlo ali pa je ne. Če škatlo pobere, jo bo moral v naslednjem koraku zamenjati z modro. Vsi ti scenariji so kandidati za optimalno pot, medtem ko sta bila v primeru 5 v prvem koraku na voljo zgolj dva možna premika, levo ali navzdol. Hevtistika 3 še vedno razvije samo 5 vozlišč, kot v primeru 5. Zaradi natančnih ocen in lastnosti algoritma A*, ki da prednost razvoju vozlišč z manjšo hevristično oceno (ki jo imajo praviloma stanja, ki so bliže cilju), se nepotrebnemu razvoju stanj popolnoma izogne.

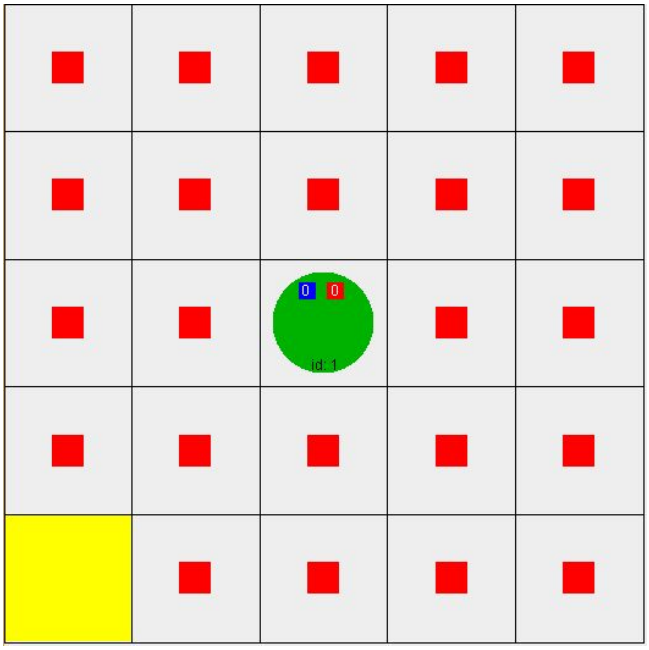
5.7 Primer 7

Na tem primeru bomo preizkusili, za kolikšno ciljno število škatel algoritem še deluje v praksi, če je njegova kapaciteta neomejena in je površina popolnoma zapolnjena z blagom (5.7). Za 11 pobranih škatel 3.hevristika razvije kar 10111 vozlišč in porabi že 137 sekund. Ostali dve hevristiki za reševanje porabita preveč časa. Po 10 min hevristiki 1 in 2 obe dosežeta zgolj globino 7, optimalen plan za 11 pobranih škatel pa ima globino 12. Tudi ocene 3. hevristike, četudi ta najde rešitev, niso tako natančne. To je posledica večje kapacitete robota od manhattanske razdalje najbolj oddaljene škatle, kar izvzame iz izračuna popolnoma vse škatle razen 1, najbolj oddaljene.

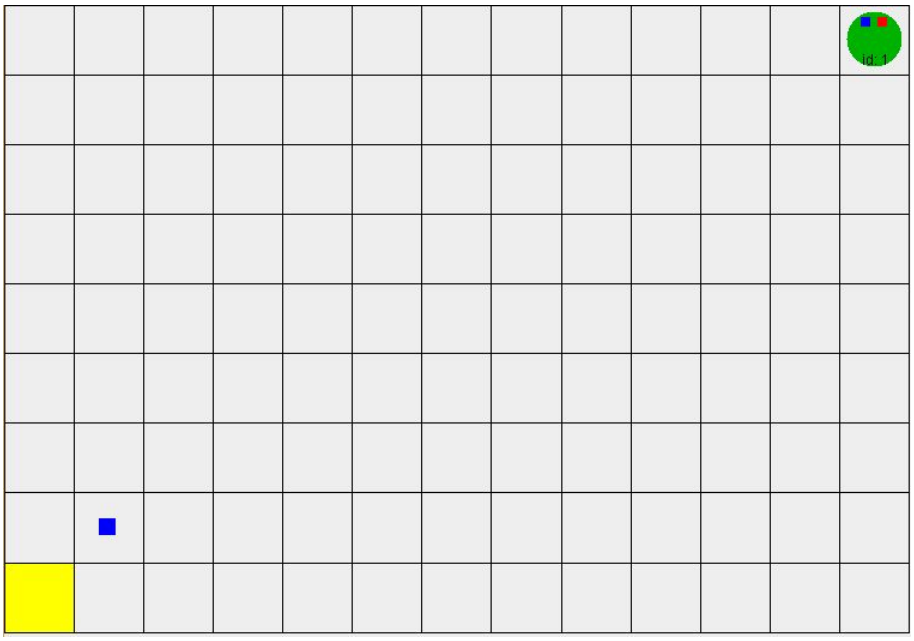
5.8 Primer 8

Površino povečamo na velikost 13x9 (5.8). Ohranili bomo parametre iz primera 1, vendar ne bomo izvedli njegove ponovitve kot v primeru 5, saj bi bili zaključki podobni. Iz tega razloga oddaljimo robot na konec, nasproten odlagališču. Rezultate najdemo v tabeli (5.11).

Opazimo lahko, da je hevristika 3 ponovno rešila problem optimalno. Ostali dve hevristiki sta dosegli med seboj podoben rezultat. Preiskali sta skoraj ves prostor, dokler nista naleteli na stanje, kjer je modra škatla pobrana. Temu je tako, ker se dotlej nobeni izmed teh dveh hevristik ne spreminja ocena h. Za prvo hevristiko je ocena vedno 0, za drugo pa 2. Povečanje



Slika 5.7: Grafični prikaz sveta za primer 7.



Slika 5.8: Grafični prikaz sveta za primer 8.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	129	0.13	0	20
2	118	0.14	2	20
3	21	0.1	20	20

Tabela 5.11: Rezultati dobljeni na primeru 8

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	9106	18.81	0	20
2	120	0.18	18	20
3	21	0.11	20	20

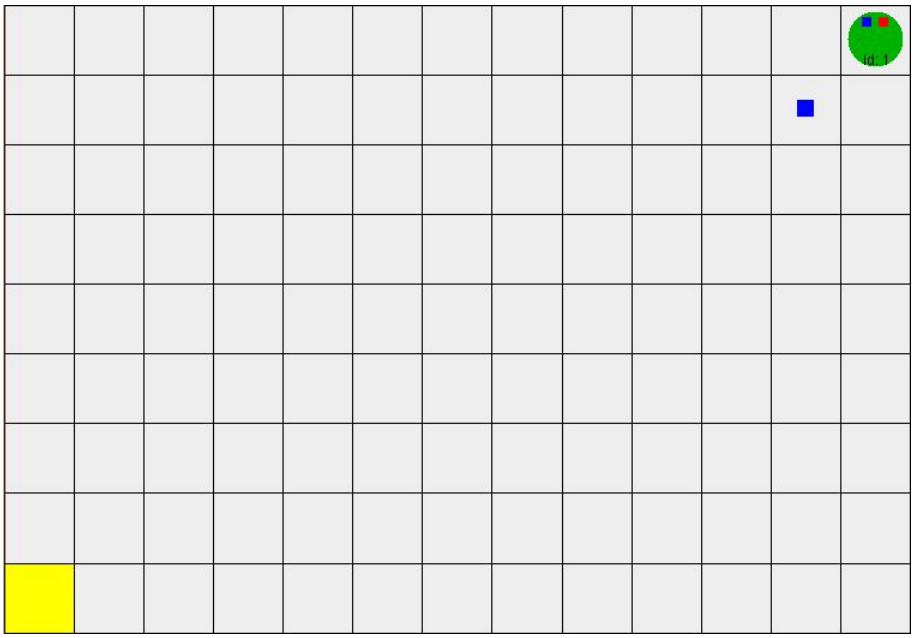
Tabela 5.12: Rezultati dobljeni na primeru 9

razvitih stanj prve hevristike pred drugo, pa se zgodi, ko se pobere modra škatla. Hevristika 2 bo po takem dogodku dala prednost naslednikom stanja s pobrano škatlo, medtem ko bo prva hevristika še naprej pregledovala vse možne poti. Omeniti velja še to, da hevristika 1 v tem primeru ni tako slaba, ker je začetna pozicija robota v kotu, nasprotnem odlagališču. To privede do posledice, da se robot zagotovo premika v pravo smer, ter hkrati preprečuje vračanje nazaj, saj za to poskrbi algoritem, ki izloči stanja, ki so že bila predhodno razvita.

5.9 Primer 9

Nadaljujmo z istimi parametri ter podobnim problemom, le da se blago nahaja bliže robotu in ne odlagališču (5.9). Rezultate najdemo v tabeli (5.12).

Opazimo veliko povečanje razvitih vozlišč pri prvi hevristiki. Ker je blago na začetku v robotovi bližini, lahko algoritem z omenjeno hevristično oceno,



Slika 5.9: Grafični prikaz sveta za primer 9.

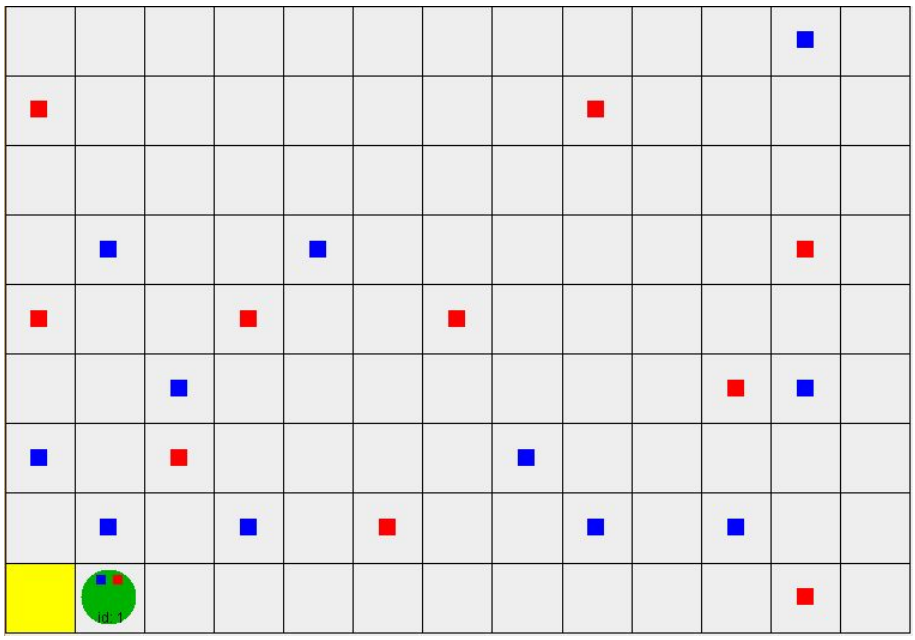
ki je za vsa stanja enaka, ustvarja vse mogoče premike, kjer odlaga in nazaj pobira modro škatlo, do globine 20. Na drugi strani hevristika 3 razvije enako število vozlišč kot v predhodnem primeru, saj nepotrebnih odlaganj ne pregleduje.

5.10 Primer 10

Osredotočimo se še na bolj splošen, naključen primer, kjer imamo veliko različnega blaga na veliki površini, od katerega moramo pobrati le nekaj škatel (5.10).

5.10.1 Izvedba 1

Začnimo s kapaciteto 1, ker je robotov cilj pobrati 1 modro in 1 rdečo škatlo. rezultati so v tabeli (5.13).



Slika 5.10: Grafični prikaz sveta za primer 10.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	/	>20 min	0	do globine 9
2	2931	27.82	6	11
3	12	0.11	11	11

Tabela 5.13: Rezultati dobljeni na primeru 10.1

Globina	1	2	3	4	5	6	7	8	9
Razvita stanja	2	6	18	54	169	572	1987	6712	21936

Tabela 5.14: Razvoj stanj po globini pri hevristiki 1 za primer 10.1

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	3443	55.06	0	7
2	493	2.34	3	7
3	8	0.1	7	7

Tabela 5.15: Rezultati dobljeni na primeru 10.2

Primer prikazuje, kako za prvo hevristiko že na videz popolnoma enostaven problem postane prezahteven, medtem ko ga dobra hevristika lahko reši v trenutku. Razlogi, da je hevristika 1 tako neučinkovita so: veliko število korakov do ciljnega stanja (optimalen plan traja 11 korakov), velikost sveta, ter veliko število modrih in rdečih škatel. Da torej hevristika 1 najde rešitev, mora preveriti vse mogoče poti do globine 11. Z vsemi škatlami na poti se možnost različnih stanj s takim številom korakov zelo poveča. V tabeli (5.14) lahko vidimo, kako se pri hevristiki 1 z globino povečuje število razvitih stanj
.

5.10.2 Izvedba 2

Kapaciteto povečamo na 2, cilj ostane enak, to je odložiti 1 modro in 1 rdečo škatlo. Rezultati so v tabeli (5.15).

Povečanje kapacitete, s katero se poveča tudi število možnih stanj, ima za posledico zmanjšanje števila korakov za 4. To je dovolj, da se zmanjša tudi končno število razvitih vozlišč.

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	2199	16.97	0	7
2	111	0.22	4	7
3	8	0.09	7	7

Tabela 5.16: Rezultati dobljeni na primeru 10.3

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	3306	50.67	0	7
2	945	6.9	2	7
3	8	0.1	7	7

Tabela 5.17: Rezultati dobljeni na primeru 10.4

5.10.3 Izvedba 3

Kapaciteta je nastavljena na 1, cilj je odložiti 1 rdečo škatlo (5.16).

Optimalna rešitev tega primera je enake dolžine predhodnemu. Kljub enakemu številu korakov, se število razvitih vozlišč pri prvi in drugi hevristiki dokaj zmanjša zaradi manjše kapacitete. Z manjšo kapaciteto v splošnem obstaja manjša možnost za ustvarjanje različnih stanj. Hevristika 3 pričakovano še vedno razvije minimalno število vozlišč.

5.10.4 Izvedba 4

Cilj ostane enak kot v izvedbi 3, kapaciteta se poveča na 2 (5.17).

Na tem primeru je še enostavneje opaziti, da povečanje kapacitete povzroči večje število razvitih stanj, saj so vsi parametri enaki predhodnemu primeru, le kapaciteta je za 1 večja, kar pa na optimalen plan rešitve nima vpliva. Torej večja kapaciteta omogoča večjo možnost različnih stanj. Spet iz opažanj lahko izvzamemo hevristiko 3, na katero takšno povečanj kapacitete

Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov
1	518	1.67	0	5
2	50	0.17	2	5
3	10	0.1	3	5

Tabela 5.18: Rezultati dobljeni na primeru 10.5

nima vpliva.

5.10.5 Izvedba 5

Robot s kapaciteto 2 želi odložiti 2 modri škatli (5.18).

Do sedaj smo prišli do zaključka, da večja kapaciteta, ob enakem številu korakov, poveča število razvitih stanj. S tem primerom se pokaže, da pozitivni učinek, ki ga ima povečanje kapacitete - zmanjšanje števila korakov, praviloma privede do manjšega števila razvitih stanj. Še posebno je to vidno pri slabših hevristikah, kot je hevristika 1. V tem primeru je potrebno še obrazložiti, da se je hevristiki 3 izjemoma število razvitih korakov povečalo, saj njena ocena za dotični primer ni popolnoma pravilna v vseh korakih izvajanja, kot se je to dogajalo v predhodnih primerih.

5.11 Analiza hevristike 2 in njena izboljšava

Med primeri smo večkrat omenili, da ima hevristika 2 nekoliko slabše rezultate, ker v primeru, ko robot nosi vse blago in je njegova naloga samo še sprehod do odlagališča, hevristična ocena vsako stanje oceni z 0. Poglejmo, kaj se zgodi, če tej hevristiki dodamo izboljšavo tudi za take primere. To storimo tako, da v primeru, ko robot prenaša eno ali več škatel, ki jih je potrebno odložiti na odlagališču, se v seštevek, preden se ga deli s kapaciteto robota, vključi še manhattansko razdaljo robota do odlagališča.

Primer	Hevristika 2	Izboljšana hevr. 2
1	10	8
2.1	3900	3693
2.2	2001	1662
3	17025	14821
4.1	212	59
4.2	1332	735
5	12	11
6	26	16
8	118	116
9	120	24
10.1	2931	448
10.2	493	104
10.3	111	24
10.4	945	305
10.5	58	28
10.6	50	28

Tabela 5.19: Primerjava hevrističnih ocen hevristike 2 in njene izboljšane verzije

Iz rezultatov na tabeli (5.19) opazimo, da se je prav na vseh primerih izboljšana verzija izkazala z manjšim številom razvitih vozlišč. Za takšno ugotovitev obstaja tudi teoretična podlaga. Če je algoritem A_2^* bolj informiran od A_1^*, potem bo ob zaključku njunih preiskovanj na kateremkoli grafu, na katerem obstaja pot od začetnega do ciljnega stanja, vsako vozlišče, ki bo razvito pri A_2^*, razvito tudi pri A_1^*[3]. To pomeni, da manj informirana hevristika razvije vsaj toliko ali večje število vozlišč, kot bolj informirana.

5.12 Analiza hevristike 3

Opazili smo, da je hevristika 3 uspešna, zato bomo na zadnjem primeru (5.10) poskusili, kako zahtevne probleme lahko še rešuje.

S kapaciteto 1 se je takšna hevristika do sedaj znašla na popolnoma vseh primerih, saj je vedno razvila minimalno število vozlišč. Podobno se zgodi, če kot cilj nastavimo 11 modrih in 10 rdečih škatel. Rešitev najde v 0.68s, in razvije le 366 vozlišč, kar je tudi minimalno število, saj optimalna pot znaša 365 korakov. Iz tega lahko sklepamo, da hevristika 3 s kapaciteto 1 pravzaprav nima omejitev pri reševanju problemov, saj število razvitih vozlišč narašča linearno s številom korakov rešitve.

S povečanjem kapacitete na 2 se podere optimalnost izračunov ocen, kar privede do enormnega povečanja razvitih vozlišč. Ocena optimalne rešitve do cilja bi v takem primeru znašala 197 korakov. Po petih minutah algoritem doseže zgolj globino 35 in razvije že 12474 vozlišč.

Podobno se zgodi, če kapaciteto nastavimo na neomejeno. V tem primeru se po petih minutah doseže globino 25 in razvije 9520 vozlišč.

Razlogi za slabo učinkovitost hevristike 3 z večjo kapaciteto se nanašajo na neupoštevanje delno zapolnjene kapacitete, kot smo že obrazložili v primeru 4.2.

Poglavje 6

Iskanje optimalne poti za množico robotov

Preiskovanje stanj z več roboti je veliko kompleksnejši problem. Kot smo že omenili, je generiranje naslednikov odvisno od možnih premikov robotov. Če imamo pri enem robotu maksimalno število naslednikov 12 (4 premiki pomnoženi s 3 akcijami), jih imamo pri n robotih 12^n. V tabeli (6.1) je prikazano, kolikšno je maksimalno število generiranih naslednikov pri vsakem razvitem vozlišču, glede na število robotov.

Problem, ki ga povzroči veliko število generiranih naslednikov, nastane predvsem pri tistem delu algoritma A^*, kjer se preveri, če so generirani nasledniki že v seznamu $OPEN$. Tukaj bi lahko teoretično pri treh robotih naleteli po petih razvitih vozliščih na preverjanje 1728 naslednikov s 8640 stanji s seznama $OPEN$, kar nanese slabih 15 milijonov primerjav za razvijanje zgolj enega vozlišča. Toliko primerjav(z vsakim korakom bi jih bilo še več) bi bilo potrebno izvesti tudi pri vsakem naslednjem razvijanju. V primerih z enim robotom smo videli, da je lahko, preden pridemo do rešitve, razvitih tudi nekaj tisoč vozlišč. Zaradi takšnih težav, bo v primeru več robotov, vsako prihranjeno razvito vozlišče bistvenega pomena.

Omenjeno težavo je mogoče reševati na način, da seznam $OPEN$ hranimo kot kompleksnejšo strukturo, kjer je časovna zahtevnost iskanja enakih

Število robotov	Maksimalno število naslednikov
1	12
2	144
3	1728
4	20736
5	248832

Tabela 6.1: Naraščanje števila naslednikov s številom robotov

elementov nižja. Druga možnost, ki je na voljo, je izpustitev preverjanja naslednikov s stanji seznama $OPEN$. Če to storimo, bo seznam $OPEN$ med izvajanjem algoritma lahko postal daljši, saj bo vseboval še kakšno podvojeno stanje z večjo vrednostjo g. Z dobro hevristiko pa bi se bilo mogoče razvijanju takšnih stanj najverjetneje tudi izogniti. V primeru, da bi takšno stanje vseeno razvili, bi njegove naslednike, ki zagotovo ne vodijo v optimalno rešitev, ponovno dodali na seznam $OPEN$. Takšna dodajanja pomenijo izgubo časa, vendar če s tem prihranimo več časa drugje, se takšna sprememba v izvajanju vsekakor izplača. Pri izvajanju hevristike za več robotov bomo zato takšno rešitev uporabili, saj nam bo prihranila veliko časa.

6.1 Hevristika 4

Hevristično oceno za več robotov bomo izračunali s pomočjo treh seznamov, ki hranijo manhattanske razdalje, podobno kot smo to storili pri hevristiki 3. To pomeni, da bomo najprej enega napolnili z manhattanskimi razdaljami modrih škatel do odlagališča, ter nato še drugega z razdaljami rdečih škatel. Seznama bomo uredili po razdaljah od najmanjše proti največji. V primeru izenačenosti, bomo za drug kriterij vzeli manjšo manhattansko razdaljo do kateregakoli robota v skladišču. Škatla, ki ima torej do najbližjega robota manjšo razdaljo, se bo uvrstila nižje na seznam. Iz tako razvrščenih seznamov vzamemo toliko spodnjih elementov iz vsakega seznama, kolikor jih je še

potrebno odložiti pri posamezni vrsti, in jih vnesemo v tretji seznam (v nadaljevanju $s3$). Pri tem upoštevamo tudi škatle, ki jih roboti že nosijo (na $s3$ vnesemo toliko škatel manj).

Na tem mestu uvedemo nov seznam h, kjer bomo hranili hevristične ocene za vsakega robota, ki bodo predstavljale število predvidenih korakov posameznega robota do skupnega ciljnega stanja. Ideja tega seznama je, da ugotovimo, kateri robot bo prenašal blago največ korakov. To število, z nekaj popravkov, opisanih v nadaljevanju, pa bomo lahko že podali kot končno hevristično oceno stanja.

Sledi preverjanje zapolnjenosti kapacitete robotov v skladišču. Če ima robot prostor za vsaj še eno škatlo, ga dodamo na nov seznam robotov, ki ga bomo poimenovali r, razen v primeru, ko robot ne prenaša nobene škatle, ostali roboti pa imajo že naložene vse ciljne škatle. V slednjem primeru je robot zaključil z delom in se ga izvzame iz izračuna. Če ima robot kapaciteto polno, dodamo njegovo manhattansko razdaljo do odlagališča na seznam h, na seznam r pa ga ne vključimo.

Vstopimo v zanko, ki se ponavlja, dokler seznam robotov r ni prazen. Tukaj najprej preverimo, če seznam $s3$ vsebuje še kakšen element. V prvi izvedbi zanke ta seznam ne bo nikoli prazen, vendar je ta preverba pomembna v naslednjih izvajanjih. Če je torej seznam $s3$ prazen, preverimo za vse preostale robote s seznama r, če nosijo kakšno škatlo. Če jo, dodamo njihovo manhattansko razdaljo do odlagališča na seznam h. Roboti, ki ne nosijo nobene škatle, pri izračunu niso več potrebni. V tem primeru tukaj z zanko končamo.

Če seznam $s3$ ni prazen, pa naredimo naslednje. Najprej najdemo robota, ki je najmanj oddaljen od najbolj oddaljene škatle od odlagališča s seznama $s3$. Nato seštejemo to razdaljo robota do omenjene škatle, ter oddaljenost škatle od odlagališča in jo vnesemo v seznam h. Preverimo, za koliko škatel ima robot še prostora, in toliko elementov izbrišemo s seznama $s3$, vendar samo tiste, ki se nahajajo v "kvadratu"med najbolj oddaljeno škatlo in odlagališčem. Škatla se nahaja v kvadratu, če lahko do nje dostopimo na poti

od najbolj oddaljene škatle do odlagališča brez dodatnih korakov. Prednost pri izbrisu imajo škatle z zadnjega konca seznama. Uvedba preverjanja s tako imenovanim kvadratom je potrebna, saj se predpostavi, da bo robot, potem, ko bo pobral najbolj oddaljeno škatlo, na poti do odlagališča zapolnil kapaciteto. Vendar če se škatla nahaja izven poti, je robot ne bo mogel pobrati brez dodatnih korakov. Če robot lahko zapolni kapaciteto s škatlami v kvadratu, potem v njegov izračun ocene števila korakov ne vključujemo ničesar več. Če pa robot kapacitete ne more zapolniti, pa se pojavi problem, saj je izredno težko določiti, ali je bolje, da v takem primeru robot vseeno odide do odlagališča z nezapolnjeno kapaciteto in bodo ostale škatle pobrali drugi roboti, ali je mogoče bolje, da zavije s poti in zapolni kapaciteto. Pri hevristiki 4 je uporabljena rešitev, kjer se v takih primerih predpostavi, da bo robot na poti do odlagališča zapolnil kapaciteto, vendar se vsaki škatli do zapolnitve kapacitete, ki se nahaja zunaj kvadrata, doda kazen dveh korakov. To je minimalna kazen, če robot krene izven poti za 1 korak. Za vsako dodeljeno kazen izbrišemo 1 element iz zadnjega mesta seznama $s3$. Nazadnje izbrišemo še robota s seznama r in ponovimo zanko, kjer bomo spet iskali najbolj oddaljeno škatlo s seznama $s3$ in robota, ki ji je najbližje.

Ko se zanka zaključi, dobljen seznam h uredimo po velikosti od najmanjšega proti največjemu. Na tem seznamu imamo praviloma toliko vrednosti, kolikor je robotov v skladišču. Vrednosti v tem trenutku predstavljajo ocenjeno število korakov posameznega robota do odlagališča. Te vrednosti moramo še ustrezno prilagoditi, saj v trenutku, ko se eden izmed robotov nahaja na odlagališču, lahko že sklepamo, da bo naslednji do odlagališča prišel najhitreje v dveh korakih. Počakati mora namreč 1 korak, da se robot z odlagališča umakne. Zato seznam, takoj po velikostni ureditvi, popravimo na minimalni razmik dveh korakov. To pomeni, da za vsak element na seznamu velja, da je njegov levi sosed za vsaj 2 manjši, desni pa za vsaj 2 večji. S tem dobimo bolj točno število korakov, ki jih posamezen robot potrebuje do prve odložitve blaga.

Nato se sprehodimo po seznamu h, začenši s prvim elementom. Če se-

znam $s3$ še ni prazen, prvemu elementu s seznama h prištejemo dvakratnik zadnjega elementa s $s3$. S seznama $s3$ nato na enak način s pravilom kvadrata in kazni dveh korakov odstranimo toliko elementov, kolikor ima robot kapacitete. S postopkom nadaljujemo pri naslednjem elementu na seznamu h in tako naprej, dokler seznam $s3$ ni prazen.

Seznam h z novimi vrednostmi ponovno uredimo in ga popravimo na minimalni razmak dveh korakov. Na zadnjem mestu seznama h se ob zaključku nahaja število korakov robota, ki bo najdlje prenašal škatle do cilja. To število vzamemo kot končno oceno izračuna hevristike 4.

Kot smo omenili v poglavju (4), mora biti hevristika optimistična, če želimo zagotovo najti optimalno rešitev. V primeru hevristike 4, takega pogoja nismo izpolnili. Hevristika bo v veliko primerih optimistična, vendar se moramo zavedati, da obstaja možnost, da se to ne bo zgodilo. Zalomi se pri delu, kjer se robotu pripiše kazen dveh korakov, ker je škatla, ki naj bi jo na poti do odlagališča pobral, izven njegove poti. V primeru, da lahko takšno škatlo pobere nek drug robot, ki pa s temi dodatnimi koraki še vedno ne preseže števila, ki je za dva manjše od števila korakov bolj oddaljenega robota (robota, ki je dobil kazen dveh korakov), bo torej pripis dveh dodatnih korakov pokvaril optimističnost hevristične ocene. Kazen dveh korakov pa smo kljub temu v izračun vključili, saj ta v večini primerov veliko pripomore k natančnejši oceni in posledično manjšemu številu razvitih vozlišč. To pa je prihranek, ki je pri velikem številu robotov zelo pomemben.

Matematična formula za omenjen izračun ne bi bila pregledna, zato bomo hevristiko 4 predstavili s psevdo kodo v dveh zaporednih delih pod oznako algoritem (1) in (2).

6.2 Hevristika 5

Kot smo ugotovili, hevristika 4 ni garantirano optimistična. Ker si želimo v testiranje vključiti tudi popolnoma optimistično hevristiko, ki nam bo po-

Algorithm 1: Psevdo koda za izračun hevristike 4, del 1/2

```
seznam s1 napolnimo z razdaljami modrih škatel
seznam s1 uredimo po velikosti
seznam s2 napolnimo z razdaljami modrih škatel
seznam s2 uredimo po velikosti
while število modrih škatel v seznamu s3 ni enako preostalemu ciljnemu številu modrih škatel do
    seznam s3 napolnimo z najmanjšimi elementi s1
end
while dokler število rdečih škatel v seznamu s3 ni enako preostalemu ciljnemu številu rdečih
škatel do
    seznam s3 napolnimo z najmanjšimi elementi s2
end
s3 uredimo po velikosti
forall the roboti do
    if robot ne nosi ničesar, medtem ko ostali roboti že prenašajo vse ciljne škatle then
        nadaljuj z izvajanjem zanke
    end
    if ima robot še prostor za prenašanje škatel then
        dodaj ga na seznam robotov r
    else
        dodaj njegovo manhattansko razdaljo do odlagališča na seznam h
    end
end
```

kazatelj za število optimalnih korakov do cilja, bomo hevristiko 4 preuredili v optimistično. To je zelo enostavno storiti, le del kode, ki izbriše škatle po pravilu kvadrata in nato ostalim škatlam izven kvadrata dodeli kazen dveh korakov, bomo nadomestili s kodo, ki enostavno briše elemente s seznama $s3$ z zadnjega mesta brez kakršnekoli kazni.

Algorithm 2: Psevdo koda za izračun hevristike 4, del 2/2

```
while seznam robotov r ni prazen do
    if seznam s3 je prazen then
        if robot prenaša blago then
            dodaj Manhattan(robot, odlagališče)na seznam h
            robot izbriši s seznama r
            nadaljuj izvajanje zanke od začetka
        end
    end
    najdi robota, ki ima najmanjšo razdaljo do najbolj oddaljene škatle na seznamu s3
    seštevek ← razdalja robot do škatle + škatla do odlagališča
    while robot zapolni kapaciteto oz. kvadrat ni prazen do
        s seznama s3 po pravilu "kvadrata"odstrani element
    end
    if robot ni zapolnil kapacitete then
        while robot zapolni kapaciteto do
            s seznama s3 odstrani zadnji element
            seštevek ← seštevek + 2
        end
    end
    seštevek dodaj v seznam h
    odstrani robot s seznama r
end
seznam h razvrsti po velikosti
seznam h popravi na 2 razmika
while seznam s3 ni prazen do
    prvemu/naslednjemu elementu s seznama h povečaj vrednost za dvakratnik zadnjega
    elementa seznama s3
    while robot zapolni kapaciteto oz. kvadrat ni prazen do
        s seznama s3 po pravilu kvadrata odstrani element
    end
    if robot ni zapolnil kapacitete then
        while robot zapolni kapaciteto do
            s seznama s3 odstrani zadnji element
            seštevek ← seštevek + 2
        end
    end
end
seznam h razvrsti po velikosti
seznam h popravi na 2 razmika
vrni zadnji element s seznama h kot rešitev
```

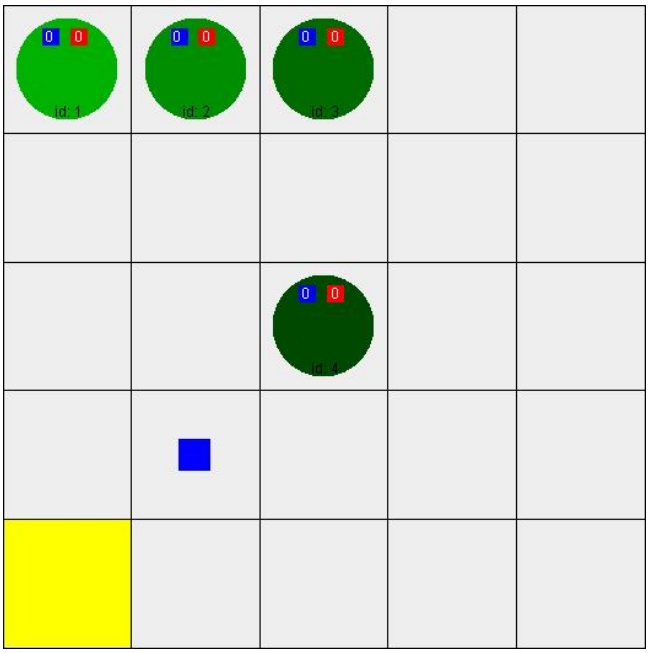
Poglavje 7

Eksperimentalna analiza hevrističnih ocen za množico robotov

Pri analizi hevrističnih ocen za več robotov se bomo oprli na primere iz poglavja (5). Na enostavnejših primerih bomo poskusili, kako se obnaša hevristika 1. Na zahtevnejših primerih pa bomo preizkusili hevristiko 4 in 5. Opazovali bomo število razvitih vozlišč, čas, hevristično oceno začetnega stanja, število izvedenih korakov ter velikost seznama $OPEN$ ob koncu izvajanja. Seznama $OPEN$ pri izvajanju testov za enega robota nismo opazovali, saj tam ni dosegal tolikšnih velikosti in zato njegova številčnost ni bila tako pomembna.

7.1 Primer 1

Za prvi primer bomo rekonstruirali primer (5.5), s to razliko, da bomo na površino postopoma postavljali večje število robotov, do števila 4, kot je prikazano na sliki (7.1). Vsak dodan robot ne bo vplival na rešitev, zanimalo pa nas bo, kako se bodo pri tem povečevale ostale lastnosti, ki jih bomo zbrali v tabeli (7.1). Primere bomo zagnali s hevristiko 1 in 4. Ker bomo v tem



Slika 7.1: Grafični prikaz sveta s 4 roboti za primer 1

primeru uporabljali tudi hevristiko 1, ki pregleduje vse mogoče poti, bomo v tem primeru preverjali, če se nasledniki že nahajajo v seznamu $OPEN$ in take tudi izločili. V nasprotnem primeru se slaba hevristika, kot je to hevristika 1, že lahko izvaja veliko časa, saj najde mnogo različnih poti do istega stanja, kar pomeni mnogokratno preverjanje in dodajanje enakih naslednikov.

Iz rezultatov je opaziti, da je hevristika 4 popolnoma imuna na dodajanje robotov, ki so brez dela. Nalogo opravi zelo uspešno, saj razvije v vseh primerih optimalno število vozlišč. Edina posledica dodajanja robotov je povečanje seznama $OPEN$, kar se zgodi zaradi večjega števila možnih naslednikov.

Nasprotno hevristika 1 z vsakim dodanim robotom že na tako enostavnem primeru razvija veliko večje število vozlišč. Taka hevristika bo na zapletenejših primerih neuporabna, zato poskusov na njej ne bomo več izvajali.

Število robotov	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama $OPEN$
1	1	35	0.08	0	4	29
	4	5	0.1	4	4	15
2	1	268	0.36	0	4	467
	4	5	0.09	4	4	55
3	1	1820	16.67	0	4	4779
	4	5	0.12	4	4	115
4	1	9883	2846.86	0	4	35915
	4	5	0.15	4	4	231

Tabela 7.1: Rezultati dobljeni na primeru 1

7.2 Primer 2

Primer 2 bomo izvedli na površini z enako postavitvijo škatel kot v primeru (5.10). Poskuse bomo izvedli s hevristiko 4 in 5. Ker pri posameznih izvedbah pričakujemo večje število korakov, ki bodo vodili do rešitve, s tem pa tudi večje število elementov na seznamu $OPEN$, bomo pri naslednjih poskusih umaknili preverjanje naslednikov s tem seznamom. Opustitev takega preverjanja ima za posledico večje število razvitih vozlišč in večjo velikost seznama $OPEN$.

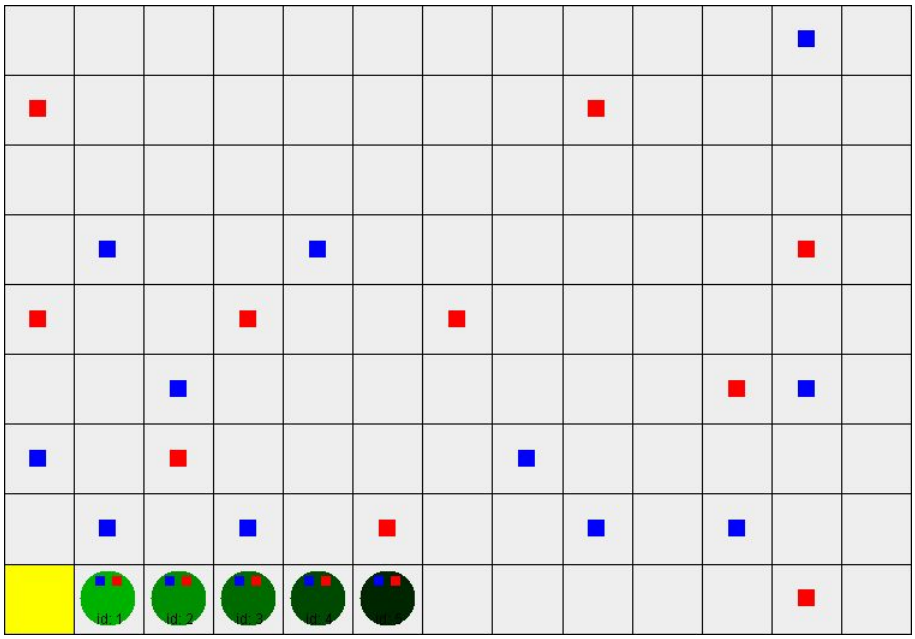
Primere bomo izvedli s številom robotov od 1 do 5. Robote se bo vključevalo v svet po vrsti od leve proti desni, kot je prikazano na sliki (7.2). Na vseh naslednjih primerih bo naš cilj z roboti prenesti do odlagališča štiri modre škatle in dve rdeči.

7.2.1 1 robot

Četudi opazujemo hevristiki za več robotov, bomo začeli samo z enim, saj hevristiki delujeta tudi v takih primerih. Pri robotovi kapaciteti 1 vrneta

Kapaciteta robota	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama *OPEN*
1	4	42	0.19	41	41	165
	5	42	0.15	41	41	165
2	4	28	0.13	25	25	151
	5	1676	2.9	21	25	6308
3	4	24	0.14	21	21	149
	5	6900	40.03	17	21	30354
4	4	28	0.13	19	19	212
	5	7073	49.1	13	19	33666
5	4	39	0.17	19	19	296
	5	9361	122.39	13	17	48796
6	4	38	0.15	13	13	293
	5	5422	27.01	9	13	28375

Tabela 7.2: Rezultati z enim robotom dobljeni na primeru 2



Slika 7.2: Grafični prikaz sveta s 5 roboti za primer 2

enake rezultate, saj se sprememba, ki smo jo naredili pri hevristiki 5, nikoli ne izvede. Če je torej kapaciteta nastavljena na 1, se pri hevristiki 4 in 5 izvede popolnoma identična koda. Ta ugotovitev bo veljala pri vseh nadaljnjih primerih s takšno kapaciteto, zato je ne bomo več ponovno omenjali. S povečanjem kapacitete na 2, 3, 4, 5 in 6 pa je opazna velika razlika med razvitimi stanji. Ker hevristika 4 v vseh primerih točno izračuna hevristično oceno, je temu ustrezno število razvitih vozlišč majhno. Ravno obratno se zgodi pri hevristiki 5, ki nikoli ne izračuna točne ocene.

Pri iskanju optimalne rešitve je hevristika 4 uspešna povsod, razen v primeru kapacitete 5. Tukaj najde rešitev, ki je za dva koraka daljša od optimalne. Vidimo lahko, da je tako dolga rešitev predvidena že z oceno začetnega stanja. Poglejmo, kako pride do takšnega izračuna. Če vzamemo, da je odlagališče na koordinatah (0,0), potem se najbolj oddaljena škatla, ki jo moramo še pobrati, nahaja na (2,3), od robota oddaljena 4 korake, od odlagališča pa 5. Hevristika 4 izbriše vse škatle v kvadratu, to so: (2,3), (2,2),

Kapaciteta robotov	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama *OPEN*
1	4	23	0.23	21	21	820
	5	23	0.2	21	21	820
2	4	23	0.24	15	15	1138
	5	4218	87.3	11	14	192800
3	4	133	0.88	11	11	10010
	5	7813	470.4	9	11	452174
4	4	29	0.27	10	11	1705
	5	9917	$>$ 10 min	8	do h=5	443239
5	4	77	0.57	11	11	5691
	5	9548	$>$ 10 min	8	do h=5	403024
6	4	23	0.26	12	12	1312
	5	7748	$>$ 10 min	8	do h=5	316943

Tabela 7.3: Rezultati z dvema robotoma dobljeni na primeru 2

(0,2) in (1,1). Do zapolnitve kapacitete preostane še ena škatla, ki se jo pobere s kaznijo dveh korakov, to je (0,4). Do sedaj imamo v izračunu 11 korakov, s katerimi poberemo prvih 5 škatel. Zadnja škatla ki ostane je (1,3), oddaljena 4 korake od odlagališča, zato se k vsoti prišteje še 8, kar skupaj znese 19. Hevristika je pravzaprav popolnoma točno ocenila število korakov do ciljnega stanja, vendar je bil njen plan napačen. Pravilen plan bi bil pobrati najprej vse bolj oddaljene škatle, za konec pa bi pustili eno, ki je najbliže odlagališču. S tem za ceno dveh izgubljenih korakov pri prvem odlaganju, prihranimo 4 na drugem. Eden izmed pravilnih planov bi bilo pobiranje škatel v naslednjem vrstnem redu: (3,1),(2,2),(2,3),(0,4),(0,2), odložitev, (1,1).

7.2.2 2 robota

Pri kapaciteti 2 se ponovno zgodi, da hevristika 4 ne najde optimalne rešitve, temveč je ta za korak daljša. Tudi ocena začetnega stanja je za korak daljša od optimalne. Nenatančna ocena se ponovno pojavi zaradi pravila kvadrata. Najbolj oddaljena škatla v planu pobiranja se nahaja na (2,3), 3 korake stran od najbližjega robota in 5 od odlagališča. Hevristika 4 pri izračunu izbriše do zapolnitve kapacitete vse škatle v kvadratu, to sta (2,3) in (2,2). Do sedaj imamo v izračunu 8 korakov, s katerimi odložimo 2 škatli. Sledi izračun za drugega robota, ki je od najbolj oddaljene škatle (3,1) oddaljen 3 korake, škatla od odlagališča pa 4. S pravilom kvadrata se izbriše škatli (3,1) in (1,1). Skupaj izračun za drugega robota znese 7 korakov, kar pomeni, da moramo za doseg razlike dveh korakov, prvemu robotu rezultat povečati za 1, torej na 9. Ostali sta še škatli (0,4) in (0,2). Pobiranje teh škatel pade v plan drugega robota z manjšim rezultatom (7), saj bo tak robot najprej odložil blago, zato bo lahko pričel prej pobirati preostale škatle. Torej številu 7 prištejemo dvakratnik števila štiri. Po izbrisu škatel s pravilom kvadrata se izračun zaključi. Na takšen način smo dobili rezultat 15.

Hitrejši plan bi bil, če bi vsak robot pobral najprej bolj oddaljene škatle. Prvi robot, ki se nahaja na (1,0), bi najprej pobral in odložil škatli na (2,3) in (0,4), za to bi potreboval 11 korakov. Drug robot bi enako storil s škatlami (3,1) in (2,2), za kar bi potreboval 8 korakov. Po osmih korakih, bi drugi robot pobral še škatli na (1,1) in (0,2), za kar bi potreboval dodatnih 6 korakov, kar bi skupaj zneslo 14.

Pri hevristiki 5 opazimo, da se z večanjem kapacitete precej poveča tudi čas in število razvitih vozlišč. Kot smo opazili pri hevristikah za enega robota, povečanje kapacitete vodi v večje število razvitih stanj, razen če s tem ne zmanjšamo dovolj števila korakov končne rešitve. V teh primerih zmanjšanje korakov očitno ni dovolj za časovno hitrejšo rešitev. To postane zelo očitno pri kapaciteti 4, 5 in 6, kjer hevristika 5 ne najde več rešitve v doglednem času (do 10 minut).

Ker je skupno ciljno število škatel 6, pomeni, da pri dveh robotih s ka-

Kapaciteta robotov	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama *OPEN*
1	4	32	0.53	15	15	4949
	5	32	0.48	15	15	4949
2	4	24	0.6	10	10	7404
	5	3574	$>$ 10 min	8	do h=4	867929
3	4	39	0.89	10	10	14949
4	4	19	0.5	9	10	5003
5	4	20	0.6	11	10	7149
6	4	19	0.52	12	10	6888

Tabela 7.4: Rezultati s tremi roboti dobljeni na primeru 2

paciteto 3 dosežemo, da je potrebno samo eno odlaganje za vsakega robota. Na tem mestu je v splošnem izkoriščenost robotov največja. Pri kapaciteti 4 bi se sicer optimalna rešitev lahko zmanjšala še za en korak, ker se robotova plana lepo poklopita, če robot, ki se nahaja na (2,0) pobere 4 ((2,2),(2,3),(0,4),(0,2)) in robot z mesta (1,0) 2 škatli((1,1,)(3,1)), vendar hevristika 4 takšne optimalne rešitve ne najde.

Podobno se zgodi pri kapacitetah 5 in 6 kjer dolžina rešitve znaša namesto 10 korakov, v prvem primeru 11, v drugem pa 12. Razlog za to je način izračuna hevristične ocene, ki se izvaja po robotih. Robotu, ki je najbližji najbolj oddaljeni škatli, bo v plan všteto toliko škatel, da bo imel zapolnjeno kapaciteto. Pri kapaciteti 6 so to že vse ciljne škatle. To povzroči, da se iz izračuna izključi drug robot, ki pa bi seveda lahko pripomogel k izboljšanju rešitve.

7.2.3 3 roboti

S tremi roboti v svetu vidimo, da hevristika 5 že pri kapaciteti 2 ne najde več rešitve v primernem času, saj razvije preveč stanj. Od te točke naprej

Kapaciteta robotov	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama *OPEN*
1	4	46	1.17	15	14	16443
2	4	45	1.4	10	10	27047
3	4	79	5.02	10	10	96684
4	4	51	3.46	9	10	80092
5	4	59	2.94	11	10	62850
6	4	56	2.87	12	10	59500

Tabela 7.5: Rezultati s štirimi roboti dobljeni na primeru 2

zato za to hevristiko ne bomo več izvajali testov. Hevristika 4 nasprotno v vseh primerih rešitev najde prej kot v sekundi. Vse rešitve, ki jih najde s tremi roboti so tudi optimalne. Optimalnost rešitve se v tem primeru zgodi, ker so plani posameznih robotov med seboj dobro usklajeni.

Pri kapaciteti 2 robot (2,0) pobira škatli (2,3) in (2,2), robot (3,0) škatli (3,1) in (1,1), ter zadnji robot (1,0) škatli (0,4) in (0,2). Vsi plani ustrezajo pravilu kvadrata, hkrati pa si roboti z njimi zapolnijo kapaciteto. Vsak robot se tudi odpravi po dodeljeno najbolj oddaljeno škatlo s seznama $s3$, ki je njemu najbližja, zato o optimalnosti rešitve ni dvoma.

Pri vseh naslednjih povečanjih kapacitete se zgodi, da robot (2,0) pobira škatli (2,3) in (0,4). Za to potrebuje 10 korakov (optimalno število rešitve). Če torej ostala dva robota v osmih koraki uspeta odložiti preostale škatle, je dobljena rešitev optimalna. To se v vseh primerih tudi zgodi.

7.2.4 4 roboti

S štirimi roboti v svetu, kjer je ciljnih škatel zgolj 6, postane že tesno s prostorom, saj so si roboti lahko velikokrat napoti. Tudi hevrističen izračun je slabši, saj se, npr. pri kapaciteti 1, gledajo izračuni najbližjih robotov do najbolj oddaljenih škatel. Ker je robotov toliko, škatel, ki so robotom blizu

zmanjka, in npr. robot z mesta (4,0) dobi v izračun pobiranje škatle na (0,4). Tak načrt ni dober, saj je ta robot od nje najbolj oddaljen in bi bilo bolje, da bi se po to škatlo odpravil kakšen drug robot.

Druga stvar, ki se pojavi je, da hevristika deluje bolje za robote bliže odlagališča, saj se predpostavi, da bodo roboti z izjemo začetne postavitve, iz odlagališča tudi prihajali. Ker pa je robotov toliko, je npr. robot (4,0) oddaljen od odlagališča za kar 4 korake. V tem primeru bi bilo za robota bolj ugodno pobiranje škatle (5,1), saj je njegova začetna pozicija samo 2 koraka stran. Vendar te škatle robot nima na vidiku, saj se ne uvrsti v seznam najbližjih odlagališču (seznam $s3$). Končno število korakov se kljub temu na koncu zmanjša, saj roboti na poti do najbolj oddaljenih škatel naletijo na stanja, kjer lahko poberejo druge škatle, iz nekaterih takšnih stanj pa izhajajo ugodnejše hevristične ocene, zato ta dobijo prednost pri nadaljnjem razvijanju.

Pri kapaciteti 2 za 4 robote že pomeni, da teoretično ne bo potrebno nobenemu robotu dvakrat hoditi do odlagališča. Ker je škatel 6, to pomeni tudi, da bo četrti robot že izvzet iz izračuna. Pri kapaciteti 3 bosta v začetni izračun vključena le še 2 robota.

7.2.5 5 robotov

Pri petih robotih na površini se dogaja podobno, kot pri štirih. Že pri kapaciteti 2 so v izračun hevristične ocene vključeni le trije od petih robotov. Ta ocena kljub temu ni tako slaba, saj bi, če si predstavljamo, da bi pri odlaganju sodelovali vsi roboti, zaradi potrebnega razmaka dveh korakov pri čakanju na odlagališče, najhitrejša rešitev vsebovala vsaj 10 premikov. V tabeli pa opazimo, da hevristika 4 najde tudi rešitve dolžine 9. To pomeni, da je lahko v nekaterih primerih celo bolje, če pri odlaganju ne sodelujejo vsi roboti.

Razvitih vozlišč, preden se najde ciljno stanje, ni veliko, opazimo pa precejšnje povečanje seznama $OPEN$, ki ima v vseh primerih preko sto tisoč elementov. To je posledica velikega števila naslednikov pri vsakem razvijanju

Kapaciteta robotov	Vrsta Hevristike	Razvita stanja	Porabljen čas (s)	h(start)	Dejansko št korakov	velikost seznama $OPEN$
1	4	129	5.83	15	13	102576
2	4	244	21.26	10	10	286950
3	4	114	8.89	10	9	138699
4	4	94	37.96	9	9	554657
5	4	167	7.52	11	9	111373
6	4	167	7.36	12	9	111373

Tabela 7.6: Rezultati s petimi roboti dobljeni na primeru 2

vozlišča, omenjena že v prejšnjih poglavjih v tabeli (6.1). Preden se naslednike uvrsti na seznam $OPEN$, jim moramo izračunati hevristično oceno, zato se pri tako velikem številu naslednikov večino časa pri iskanju rešitve porabi ravno pri tem.

Poglavje 8

Sklepne ugotovitve

Skozi nalogo smo ugotovili, da je pri preiskovalnih algoritmih, kot je to A*, zelo pomembna natančnost hevrističnega izračuna. Že majhne spremembe v oceni lahko povzročijo, da algoritem v preiskovanju razvije nekajkrat več oz. manj stanj.

Pri hevristiki 3 smo ugotovili, da bi jo lahko izboljšali za primere, kjer je kapaciteta večja od manhattanske razdalje najbolj oddaljene škatle do odlagališča. V takih primerih se zgodi, da se iz izračuna ocene odvzame preveč škatel. Če je omenjena manhattanska razdalja m, kapaciteta robota pa k, se iz izračuna izbriše $k - m$ preveč škatel.

Drug del izboljšave se nanaša na delno zapolnjeno kapaciteto. Hevristika 3 namreč na poti pobrane škatle upošteva zgolj pri zmanjšanju ciljnega števila, ne upošteva pa zmanjšanja robotove kapacitete, temveč vedno predpostavi, da je prazen. Taka napaka se pozna le do prvega odlaganja, saj bo po odlaganju robot v resnici postal prazen, vendar je to lahko že razlog, da hevristika 3 pri zahtevnejših primerih s kapaciteto večjo od ena, ne najde več rešitve v doglednem času.

Ugotovili smo tudi, da se na mestu, kjer iz enega stanja izhaja velika množica naslednikov, v našem primeru se je to zgodilo s povečevanjem števila robotov, pojavijo novi problemi. Slabost algoritma A* je, da mora hraniti celotno vrsto vseh zgeneriranih naslednikov do danega trenutka. Če je teh ve-

liko, se lahko pojavijo tudi problemi v fizični velikosti spomina. V našem primeru smo naleteli na problem velikega števila primerjav stanj. Tudi izračun hevristike za tako veliko množico stanj prej ali slej postane časovno opazen.

Hevristika 4 prav tako ponuja nekaj možnosti za izboljšave. Ena izmed njih je dodelitev natančnejše kazni pri škatlah, ki se nahajajo izven kvadrata. Omenili smo, da trenutna kazen dveh korakov pokvari optimističnost. Če bi bilo mogoče kazen izračunati popolnoma natančno, bi hkrati izboljšali točnost hevristične ocene in zagotovili njeno optimističnost. Vendar je takšno oceno, ki bi to z gotovostjo omogočala, zelo težko izpeljati.

Druga možnost izboljšave se pojavi pri večjih kapacitetah. Omenili smo, da se v primerih z več roboti, ki imajo veliko kapaciteto pogosto zgodi, da se nekaj robotov iz izračuna izključi. To bi lahko popravili s tem, da bi namesto dodeljevanja škatel posameznemu robotu, dodelili robotom posamezne škatle. Vendar tudi ta izboljšava ni tako enostavna, saj smo v nekaterih primerih opazili, da je rešitev krajša, če pri njej ne sodelujejo vsi roboti, saj bi se v nasprotnem primeru morali med seboj predolgo čakati na odlagališču.

Literatura

[1] I. Bratko, *Prolog Programming for Atificial Intelligence, Fourth Edition*. Addison-Wesley, 2012, str. 417–421.

[2] P. E. Hart, N. J. Nilsson, B. Raphael *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Trans. Syst. Science and Cybernetics, 1968

[3] N. J. Nilsson, *Atificial Intelligence: a new synthesis*. Morgan Kaufmann Publishers, 1998, str. 139–162.

[4] A*'s Use of the Heuristic (2014) dostopno na: http://theory.stanford.edu/ amitp/GameProgramming/Heuristics.html