Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Jernej Janež

Iskanje in izvajanje paralelnih planov pri variantah igre 8 kvadratov

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: akad. prof. dr. Ivan Bratko

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Klasično igro 8 kvadratov, ki se igra na pravokotni mreži velikosti 3x3, lahko obravnavamo kot stiliziran model avtomatiziranega skladišča, v katerem deluje 8 mobilnih robotov. Ta model skladišča lahko posplošimo na mrežo velikosti NxN ter poljubno število robotov do N*N - 1, ko se lahko hkrati premika tudi po več robotov. V nalogi razvijte hevristične algoritme za iskanje paralelnih planov za množico robotov na taki mreži. S poskusi ocenite uspešnost raznih hevristik odvisno od velikosti mreže in števila robotov.

Zahvaljujem se akad. prof. dr. Ivanu Bratku za napotke in ideje pri izdelavi diplomske naloge. Prav tako bi se rad zahvalil svojim staršem za podporo in vzpodbudo. Nazadnje pa še hvala bratoma, sestri in punci Špeli, ki so mi zmeraj stali ob strani in bili v izjemno pomoč.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
AVG	average	povprečje ocen
MAX	maximum	največja ocena
SUM	summation	vsota ocen

Povzetek

Naslov: Iskanje in izvajanje paralelnih planov pri variantah igre 8 kvadratov

Avtor: Jernej Janež

V tej nalogi smo si zadali problem iskanja pararelnih planov pri variantah igre 8 kvadratov. Uporabili smo algoritem A* in z uporabo različnih hevrističnih funkcij ugotovili, katere pripeljejo do boljše rešitve in katere do slabše. Primerjali smo štiri hevristične funkcije in glede na potrebe predlagali, katera je najbolj primerna za uporabo. Nekatere potrebujejo več časa in s tem pridejo do bolj kvalitetne rešitve, nekatere pa najdejo rešitev mnogo hitreje, a so zaradi tega rešitve slabše kvalitete.

Ključne besede: iskanje, pararelno izvajanje, A* algoritem, hevristične funkcije.

Abstract

Title: Finding and executing parallel plans for variants of the 8-puzzle problem

Author: Jernej Janež

In this thesis we deal with the problem of finding parallel plans for variants of the 8-puzzle problem. We used the A* algorithm and experimented with different heuristic functions trying to guide the search. We compared four different heuristic functions and depending on the environment proposed the most suitable one. Some of them need more time to find a better solution and some are faster but may miss good quality solutions.

Keywords: search, parallel execution, A* algorithm, heuristic functions.

Poglavje 1

Uvod

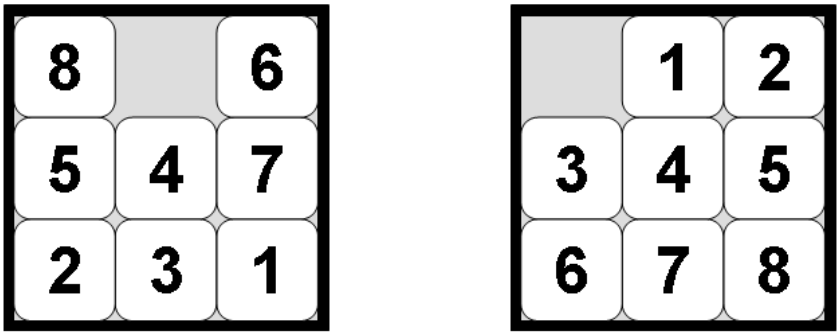
1.1 Motivacija

Iskanje rešitve pri igri 8 kvadratov je lahko zabavno; s poskušanjem in ugibanjem premikov ploščic lahko včasih hitro najdemo rešitev, a včasih je lahko problem preveč zahteven in rešitve preprosto ne vidimo. Zato si lahko problem resneje zastavimo in začnemo iskati optimalne poti, katere potrebujejo minimalno število potez, da pridemo iz začetnega do končnega stanja. Pri tem si pomagamo z iskalnimi algoritmi, ki nam pomagajo najti optimalne poti. V diplomski nalogi smo uporabljali algoritem A*, za katerega velja, da je eden izmed najbolj razširjenih iskalnih algoritmov ter uporabljali različne hevristične funkcije, katere lahko predstavimo kot strategije, ki uporabljajo ohlapno uporabne informacije za bolj usmerjeno iskanje optimalnih poti.

Igra 8 kvadratov je dober model za merjenje delovanja hevurističnih iskalnih algoritmov [2, 6, 7, 8] in učnih metod [5]. Obstajajo tudi večje variante; npr. igra 15 kvadratov, ki je tudi rešljiva [4].

1.2 Igra 8 kvadratov

Cilj igre 8 kvadratov je, da preuredimo osnovno postavitev osmih kvadratnih ploščic na 3 x 3 polju, v specifično določeno končno stanje z uspešnim premikanjem ploščic v prazno polje. Primer osnovne in končne postavitve je na sliki 1.1. Čeprav na prvi pogled izgleda dokaj enostavno najti rešitev tega problema, nas zanimajo tiste rešitve, za katere porabimo najmanj potez. Te rešitve imenujemo optimalne poti, ki potrebujejo najmanj možnih premikov iz začetnega do končnega stanja.

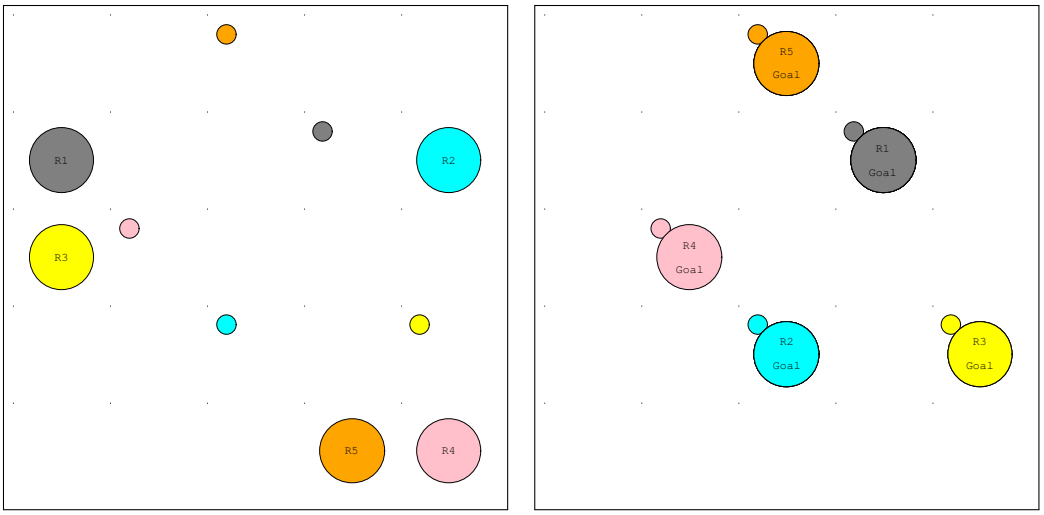


Slika 1.1: Začetna in končna postavitev ploščic.

1.3 Cilji diplomske naloge

V tej nalogi bomo igro 8 kvadratov razširili tako, da bomo ploščice na polju zamenjali z roboti, polje povečali na velikost 5 x 5, tako da bo na voljo 25 različnih stanj, na katerih so lahko roboti oz. njihovi cilji. Prav tako bomo zmanjšali število ploščic oz. robotov, saj želimo izvajati paralelno premikanje, za kar mora biti dovolj prostora, da se roboti lahko sočasno premikajo. Primer nastavitve s petimi roboti na 5 x 5 mreži je na sliki 1.2, na kateri so roboti označeni kot večji krogi, vsak v svoji barvi in zaporednim številom, manjši krogi, postavljeni v levi zgornji del polja, predstavljajo njihove končne lokacije, vsak z barvo tistega robota, ki mora priti do te lokacije

ter mreža z označenimi polji s črnimi pikami v kotih. Če bi bilo preveč robotov naenkrat na polju, bi bilo paralelno iskanje nepotrebno, saj bi se roboti med seboj čakali in premikali zaporedno.



(a) Začetna postavitev. (b) Končna postavitev.

Slika 1.2: Nastavitev na 5 x 5 mreži s petimi roboti

S pomočjo algoritma A* in različnih hevrističnih ocen bomo izvedli iskanje poti, katere cilj je priti v čim manj korakih iz postavitve (a) v postavitev (b), ugotavljali bomo optimalnost in učinkovitost ter na podlagi rezultatov primerjali, katere rešitve so bolj uspešne in katere manj. Simulator, ki ga bomo podrobneje predstavili v 2. poglavju je na voljo na spletnem mestu: `https://github.com/jernejjanez/multi-robot-pathfinder`.

Poglavje 2

Simulacija okolja

V diplomski nalogi smo razvili simulator dostopen na spletnem naslovu: `https://github.com/jernejjanez/multi-robot-pathfinder`, s katerim lahko prikažemo okolje, robote ter njihova končna stanja. V našem primeru je to mreža velikosti N x N, na kateri se lahko nahaja več robotov, predstavljenih kot krogi, vsak v svoji barvi in z oznako Rn, pri čemer je n številka robota.

Pred začetkom so roboti postavljeni na svoja začetna stanja ter čakajo na začetek izvajanja programa. Izvajanje začnemo s pritiskom na gumb miške, ta nato poteka, dokler vsi roboti ne prispejo do svojih končnih stanj. Končna stanja so označena z manjšimi krogi, ki so postavljeni v levi zgornji kot ciljnih stanj robotov, vsak v barvi tistega robota, katerega cilj je to stanje. Roboti se premikajo zvezno in paralelno. Ko prispejo do cilja, se jim pod oznako doda še zastavica *Goal*, s katero robot javi, da je prispel na cilj. Po končani simulaciji, s prikazanimi roboti na svojih končnih stanjih, se izvajanje zaključi.

2.1 Zahteve pred izvajanjem programa

Pred začetkom izvajanja programa moramo najprej podati velikost mreže. V tej diplomski nalogi bomo uporabljali mrežo velikosti 5 x 5, saj se pri taki velikosti roboti lahko neodvisno premikajo ter s tem dobro prikažejo paralelnost premikov. Mreža je v našem programu predstavljena kot dvodimenzionalna tabela velikosti N x N, vsak kvadrat v mreži pa s koordinatama (y, x), za kateri velja:

$$\begin{aligned} 0 \leq y, x < N \\ y, x \in \mathbb{N} \end{aligned} \tag{2.1}$$

Zatem vnesemo število robotov v spremenljivko R, za katero velja:

$$0 < R < N^2 \tag{2.2}$$

Začetna in končna stanja robotov so podana v seznamih. Če ta niso na voljo, jih generiramo naključno in za vsa stanja ponovno velja pravilo, da ima vsak robot podani koordinati (y, x), za kateri velja (2.1).

Nato izberemo hevristično funkcijo s katero bomo računali ceno poti do ciljnih stanj. Na voljo imamo številke od 1 do 4, pri čemer velja, da s številko 1 izberemo hevristično funkcijo AVG (ang. average), s številko 2 izberemo MAX (ang. maximum), s številko 3 SUM (ang. summation) in s številko 4 weighted SUM (ang. weighted summation). Več o posameznih hevristikah bomo govorili v 4. poglavju, kjer bomo vsako bolj podrobno predstavili.

Če smo izbrali hevristično funkcijo 4 oz. weighted SUM, moramo po izboru funkcije podati še utež, s katero bomo računali hevristiko SUM. Utež je predstavljena s spremenljivko w, za katero velja:

$$\begin{aligned} 0 < w \leq 1 \\ w \in \mathbb{R} \end{aligned} \tag{2.3}$$

2.2 Primer pravilno generiranega začetnega stanja

Poglejmo si primer pravilno generiranih začetnih stanj za 5 robotov, postavljenih na 5 x 5 mreži.

Začetna in končna stanja so podana kot seznami (y, x) koordinat, prikazani v tabeli 2.1. Da so začetna stanja legalna, morajo biti vsi roboti na različnih pozicijah, saj ni dovoljeno, da je več robotov istočasno na istem polju. Če sta (y_n, x_n) začetni koordinati robota n in (y_m, x_m) začetni koordinati robota m, potem velja:

$$n \neq m \Rightarrow y_n \neq y_m \vee x_n \neq x_m \tag{2.4}$$

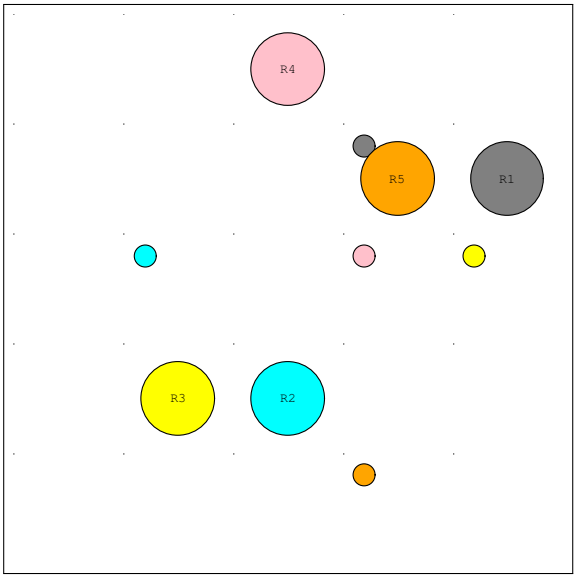
Enačba (2.4) velja tudi za končna in vsa druga stanja, kjer ima ponovno lahko samo en robot določeno končno stanje v dani nastavitvi.

Kot prej omenjeno, so roboti okrogle oblike in različnih barv postavljeni na svoje začetne lokacije. Končna stanja so pa označena kot manjši krogi v levem zgornjem kotu polja, z barvo tistega robota, katerega cilj je to polje.

	R1	R2	R3	R4	R5
start =	[[1, 4],	[3, 2],	[3, 1],	[0, 2],	[1, 3]]
end =	[[1, 3],	[2, 1],	[2, 4],	[2, 3],	[4, 3]]

Tabela 2.1: Primer začetnih in končnih lokacij robotov.

Mreža z roboti na svojih začetnih lokacijah ter označenimi končnimi lokacijami je na sliki 2.1. Opazimo, da so roboti lahko postavljeni na končne lokacije drugih robotov, kot je na sliki robot $R5$ na končni lokaciji robota $R1$. To je seveda dovoljeno, saj se bo robot $R5$ premaknil in tako naredil prostor za robota $R1$. Še več, roboti bodo s paralelnim premikanjem verjetno večkrat prečkali ciljne lokacije ostalih robotov. Vse to je odvisno od začetne in končne razporeditve ter poti vsakega robota.



Slika 2.1: Primer začetne postavitve robotov.

2.3 Primer izvajanja

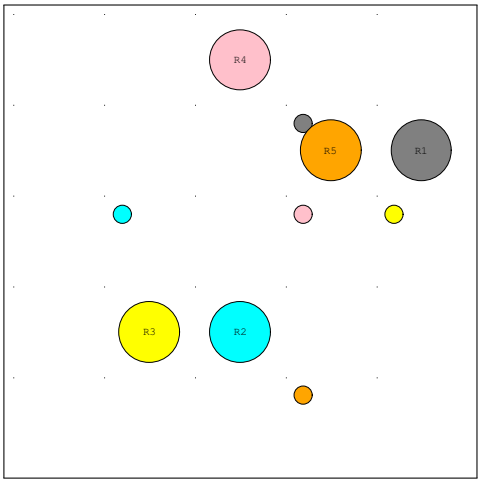
Če nadaljujemo s primerom iz slike 2.1, bomo iz danega začetnega stanja postopoma prišli do končnega.

Če pogledamo v tabelo 2.1, lahko za vsakega robota izračunamo minimalno število potez, ki jih potrebuje od svoje začetne do končne lokacije. Če sta (x_n, y_n) koordinati robota n, ter $(\overline{x}_n, \overline{y}_n)$ koordinati končne lokacije robota n, potem je minimalno število premikov enako manhattanski razdalji:

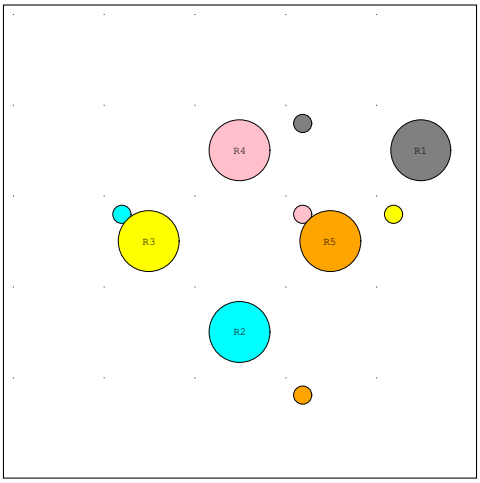
$$|x_n - \overline{x}_n| + |y_n - \overline{y}_n| \tag{2.5}$$

Po enačbi (2.5) ugotovimo, da robot $R1$ potrebuje en premik, robot $R2$ dva, robot $R3$ štiri, robot $R4$ tri in robot $R5$ tri premike. Iz teh izračunov bi potem morali roboti potrebovati največ štiri premike za rešitev celotnega problema. A kot je razvidno iz slike 2.2, je bilo za rešitev problema potrebno pet premikov. Iz tega lahko ugotovimo, da izračunane vrednosti za vsakega robota po enačbi (2.5) veljajo, če bi bil vsak robot na svoji mreži brez ostalih robotov. Ker pa so vsi skupaj na eni, so si roboti med seboj v napoto in se morajo pametno izogniti drug drugemu, bodisi s premikom ali brez (ostati na istem polju).

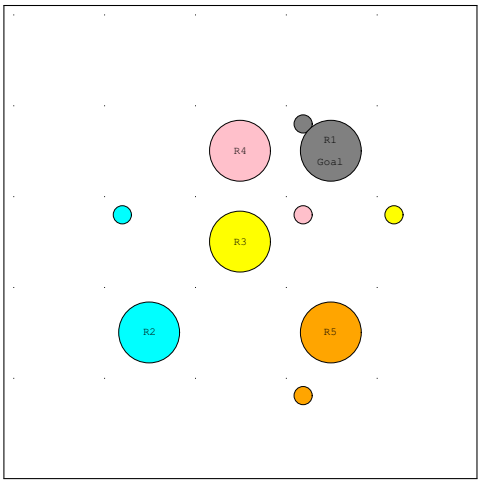
Če dobro pogledamo sliko 2.2, so roboti dejansko potrebovali izračunano število premikov, vendar moramo prišteti še ne premike, ko so roboti v nekem koraku počakali na istem polju. Tako lahko opazimo, da je robot $R4$ potreboval največ premikov kljub temu, da se je robot dejansko samo trikrat premaknil, a je v stanjih (c) in (d) ostal na istem polju in posledično imel skupaj pet premikov.



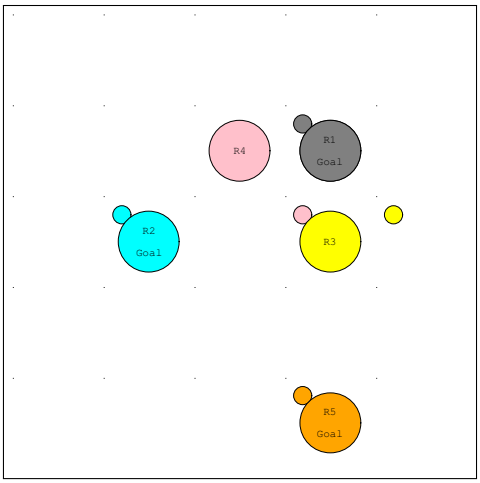
(a) Začetna postavitev.



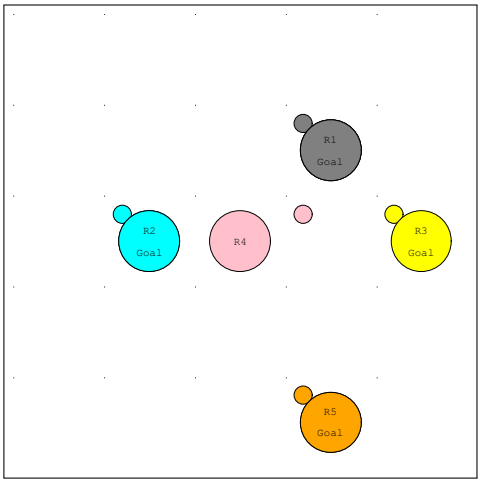
(b) Prvi premik.



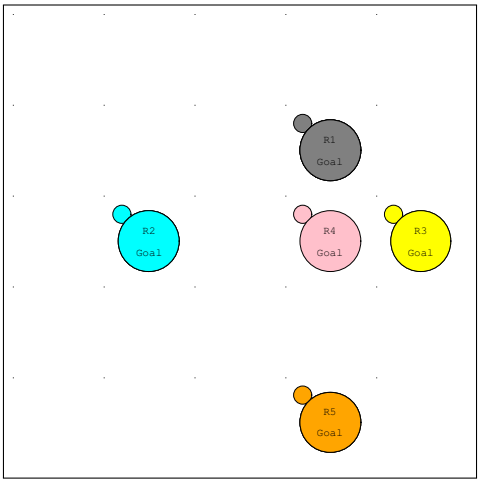
(c) Drugi premik.



(d) Tretji premik.



(e) Četrti premik.



(f) Peti premik in končna postavitev.

Slika 2.2: Primer izvajanja na 5 x 5 mreži s petimi roboti.

Poglavje 3

Algoritem A*

Algoritem A* je informirani iskalni algoritem oz. „best-first“ iskalnik, kar pomeni da išče rešitve med vsemi mogočimi potmi od začetnega do končnega stanja in se odloča za tiste, ki so najcenejše (najkrajši čas, najkrajša pot, itd.).

A* se mora z vsako iteracijo glavne zanke odločiti, katero delno pot bo razširil v eno ali več daljših poti. To stori s pomočjo hevristične funkcije, katera izračuna ceno preostale poti do cilja. Specifično, A* izbere tisto pot ki ima najmanjšo $f(n)$ vrednost po enačbi:

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

pri kateri je n trenutno stanje dosedanje poti, $g(n)$ je cena poti od začetnega stanje do n in $h(n)$ hevristična funkcija, ki izračuna oceno cene najcenejše poti od n do ciljnega stanja. Hevristična funkcija je specifična glede na problem. V tej diplomski nalogi smo raziskovali 4 različne hevristične funkcije in jih med seboj primerjali. Zadostni pogoj, da algoritem najde dejansko najkrajšo pot je ta, da je hevristična funkcija popolna, kar pomeni, da nikoli ne preceni dejanske cene poti do ciljnega stanja, kar prikažemo v naslednjem izreku o popolnosti A* [3].

Izrek 3.1 *A* je popoln, če za vsako vozlišče n v prostoru stanj velja*

$$h(n) \leq h^*(n). \tag{3.2}$$

Pri čemer je $h^*(n)$ dejanska cena najcenejše poti od n do cilja.

Če hevristika h zadošča pogoju

$$h \leq h^*, \tag{3.3}$$

potem je h „optimistična“ ocena [1].

Tipično so algoritmi A* implementirani na takšen način, da uporabljajo prioritetno vrsto, s katero izvajajo ponavljajoče izbiranje najcenejše poti za nadaljnje razvijanje stanj. To vrsto bomo poimenovali *cand_paths* (ang. candidate paths). Stanje z izračunano najmanjšo vrednostjo $f(n)$ izbrišemo iz vrste, f in g vrednosti njegovih naslednikov so pravilno spremenjene in ti nasledniki so nato dani v vrsto. Algoritem poteka, dokler ne pridemo do ciljnega stanja ali dokler ni vrsta prazna.

Algoritem A* lahko implementiramo bolj učinkovito, če vsa obiskana stanja shranjujemo v *visited* vrsti, s katero primerjamo vsa nova obiskana stanja ter na podlagi izračunane cene ugotovimo ali so primerna za vključitev v vrsto *cand_paths* ali ne.

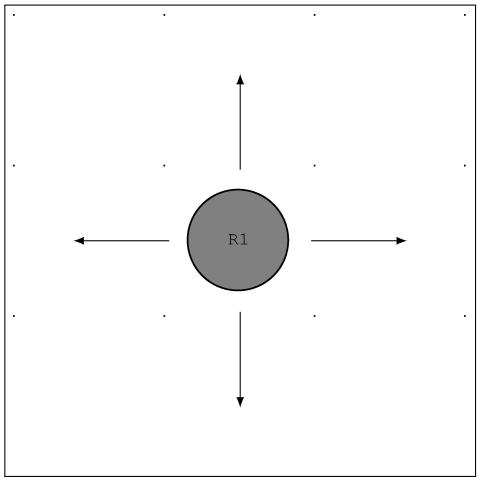
3.1 Planiranje poti

Ko smo generirali pravilna začetna in končna stanja robotov, lahko pričnemo s planiranjem legalnih potez. Kot je prikazano na sliki 3.1, je legalna poteza za robota premik gor, dol, levo ali desno, če na sosednjem polju ni ovire (ostali roboti, stena). Roboti se ne morejo premikati diagonalno, zaporedno en robot takoj za drugim ter v enem koraku zamenjati mesta med seboj.

Z določitvijo pogojev za premike robotov lahko pričnemo z generiranjem vseh možnih potez, ki jih bo nato A* sprejel in razvrstil po vrednosti ocenitvene funkcije f v spremenljivko *cand_paths* oz. jih bo zavrgel zaradi že najdene cenejše poti v spremenljivki *visited*.

Pri generiranju potez moramo biti pozorni na poseben primer:

- če se robot $R1$ premika iz polja a na polje b, pri čemer je polje b prazno, se ostali roboti v istem časovnem intervalu ne smejo premakniti ne na polje a ne na polje b, saj bi zaradi paralelnega izvajanja lahko prišlo do trka.



Slika 3.1: Možni premiki robota.

Ko najdemo vse mogoče poteze za vse robote, jih podamo algoritmu A*, ki bo izbral najbolje ocenjeno še nerazvito stanje (tj. „odprto stanje“) in nato začel nov krog generiranja potez ter izbire zopet najcenejših poti. A* se zaključi, ko so vsi roboti na svojih ciljnih stanjih oz. če v vrsti *cand_paths* ni več elementov, kar pomeni, da algoritem ni našel rešitve.

Poglavje 4

Iskanje optimalnih planov

Kot smo že omenili v 3. poglavju, si algoritem A* pomaga s hevrističnimi funkcijami. Naša naloga je bila ugotoviti, katere bodo dobro usmerjale program, da bo našel optimalno oz. skoraj optimalno pot in katere bodo zaradi netočne hevristične ocene računale nepotrebne poti in s tem povečevale kompleksnost oz. nižale kvaliteto rešitve, da je pot daljša. Raziskovali smo štiri različne hevristične ocene:

- AVG (ang. average),
- MAX (ang. maximum),
- SUM (ang. summation),
- utežena SUM (ang. weighted summation).

Vsako bomo bolj podrobno pogledali v podpoglavjih, jo primerjali glede na ostale in na podlagi rezultatov poskušali ugotoviti, katera je najbolj učinkovita. Pri vseh smo za računanje razdalje do cilja uporabljali manhattansko razdaljo (ang. Manhattan distance), ki je definirana kot vsota razdalj med točkama po x in y oseh. Enačbo (2.5) za manhattansko razdaljo smo omenili že v 2. poglavju.

4.1 Hevristična funkcija AVG

Najprej smo obravnavali AVG. Če sta $x_i(s)$ in $y_i(s)$ koordinati robota i v stanju s, $\overline{x}_i$ in $\overline{y}_i$ ciljni koordinati, ter n število robotov, potem je AVG enaka:

$$h(s) = \frac{\sum_{i=1}^{n}(|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|)}{n} \tag{4.1}$$

Pričakujemo lahko, da bo $h(s)$ bistveno nižji od dejanske cene poti do cilja. Posledica tega je razvijanje veliko stanj, večja kompleksnost, ter optimalna rešitev. Ocena je optimistična, kar pomeni, da nikoli ne preceni ceno poti do cilja – cena poti do cilja ni nikoli večja od cene najcenejše možne poti od trenutnega stanja do cilja [9]. Glavne lastnosti hevristične funkcije AVG so:

- za rešitev je potrebno razviti veliko stanj,
- z večanjem števila robotov kompleksnost narašča zelo hitro,
- rešitev je optimalna.

4.2 Hevristična funkcija MAX

Kot naslednjo poglejmo hevristično funkcijo MAX (ang. maximum). Zanjo velja, da če sta $x_i(s)$ in $y_i(s)$ koordinati robota i v stanju s, $\overline{x}_i$ in $\overline{y}_i$ ciljni koordinati, ter n število robotov, potem je MAX enaka:

$$h(s) = \max_{1 \leq i \leq n}(|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|) \tag{4.2}$$

Iz enačbe (4.2) opazimo, da zopet izračunamo Manhattansko razdaljo pri vseh robotih, vendar tokrat vzamemo le največjo izmed vseh. S tem zagotovimo, da je ocena še vedno optimistična, saj z izborom robota z najdaljšo potjo upoštevamo pravilo, da mora biti ocena poti enaka ali manjša najcenejši dejanski poti do cilja [3]. Glavne lastnosti funkcije MAX so:

- za rešitev je potrebno razviti veliko stanj, a manj kot pri AVG,

- z večanjem števila robotov kompleksnost narašča hitro, a počasneje kot pri AVG,
- rešitev je optimalna.

4.3 Hevristična funkcija SUM

Tretja hevristična funkcija, ki smo obravnavali, je SUM (ang. summation). Zanjo velja, če sta $x_i(s)$ in $y_i(s)$ koordinati robota i v stanju s, $\overline{x}_i$ in $\overline{y}_i$ ciljni koordinati ter n število robotov, potem je hevristika SUM enaka:

$$h(s) = \sum_{i=1}^{n}(|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|) \tag{4.3}$$

Pri tej hevristični funkciji pa seštejemo manhattanske razdalje vseh robotov. Enačba (4.3) je podobna enačbi za funkcijo AVG (4.1), razlika je v tem, da dobljeni seštevek ne delimo s številom robotov, kar privede do večje vrednosti. Zaradi tega ne moremo več trditi, da je ta ocena optimistična, saj je večinoma večja od najcenejše dejanske poti do cilja. Zato je tudi pot, ki jo najdemo, večkrat ne optimalna. Kot bomo videli v naslednjem poglavju, funkcija SUM do določenega števila robotov sovpada z rešitvami optimističnih hevrističnih funkcij AVG in MAX, nato ne več. Glavne lastnosti za SUM so:

- za rešitev je potrebno razviti manj stanj kot pri AVG in MAX,
- z večanjem števila robotov kompleksnost narašča hitro, a veliko počasneje kot pri AVG in MAX,
- rešitev je pri našem testiranju do določenega števila robotov enaka AVG in MAX, kar pomeni da je optimalna, nato ne več.

4.4 Hevristična funkcija utežena SUM

Zadnja obravnavana hevristična funkcija je utežena SUM. Enačba je podobna enačbi (4.3) za funkcijo SUM, z dodatno spremenljivko oz. utežjo. Če sta $x_i(s)$ in $y_i(s)$ koordinati robota i v stanju s, $\overline{x}_i$ in $\overline{y}_i$ ciljni koordinati, n število robotov ter w utež, ki je realno število z vrednostjo med 0 in vključno 1, potem je utežena SUM enaka:

$$h(s) = (\sum_{i=1}^{n}(|x_i(s) - \overline{x}_i| + |y_i(s) - \overline{y}_i|)) \times w \tag{4.4}$$

$$0 < w \leq 1, w \in \mathbb{R}$$

Kot smo že omenili, je enačba enaka funkciji SUM, le da jo množimo še z utežjo. Namen uteži je najti vrednost, s katero se bo delovanje programa izboljšalo. Učinkovitost delovanja je tako že v majhnih spremembah, saj že povečanje oz. zmanjšanje uteži za 0.1 spremeni algoritem, da ta privede do dobre oz. slabše rešitve. V naslednjem poglavju si bomo podrobno pogledali, kako večje oz. manjše vrednosti uteži vplivajo na delovanje in katera vrednost je najbolj primerna za uporabo. Glavne lastnosti utežene SUM so:

- število razvitih stanj je odvisno od uteži, manjša ko je utež, več stanj razvije program,
- večanje števila robotov privede do večje kompleksnosti, pri manjših utežeh se kompleksnost povečuje hitreje, pri večjih pa počasneje,
- rešitev je zopet odvisna od uteži, manjša ko je utež boljša je rešitev, vendar je kompleksnost nato problem,
- želimo najti utež, pri kateri je razmerje med kompleksnostjo in učinkovitostjo najboljše.

Poglavje 5

Primerjava delovanja algoritma pri različnih hevrističnih funkcijah

Predstavljene hevristične ocene smo med seboj primerjali in glavna podatka, ki sta nas zanimala, sta bila:

- povprečno število razvitih stanj ali p.š.r.s.,
- čas izvajanja plana, ki smo ga merili s povprečno dolžino rešitvene poti ali p.d.r.p.

Hevristične funkcije smo primerjali na takšen način, da smo na mreži velikosti 5 x 5, generirali n število robotov, pri čemer smo začeli z enim, za katerega paralelnost ne pride v poštev in nato dodajali robote, dokler ni bila kompleksnost že prevelika. Pri vsakem številu robotov smo za vsako hevristično funkcijo naredili 100 ponovitev, pri čemer smo začetna in končna stanja za vsako ponovitev generirali naključno. Po končanem izvajanju smo izračunali povprečno število razvitih stanj ter povprečno dolžino rešitvene poti in ju nato med seboj primerjali. V tabelah smo primere s preveliko kompleksnostjo označili z oznako *t.c.* (ang. too complex).

Poskusni primeri so bili za posamezno število robotov generirani naključno in nato uporabljeni pri vseh štirih hevrističnih funkcijah. Verjetnostna porazdelitev dolžin optimalnih planov je bila tako naključna. Razporeditev testnih primerov v razmerju s številom robotov in dolžinah optimalne rešitve je prikazana v tabeli 5.1.

Dolžina opt. rešitve / Število robotov	1	2	3	4	5	6	7	8	9
1	12	23	25	14	14	7	4	1	0
2	1	14	23	23	18	14	6	1	0
3	0	5	13	22	30	22	7	1	0
4	0	1	3	21	26	37	11	1	0
5	0	0	5	17	35	33	9	1	0
6	0	0	1	12	29	28	27	2	1

Tabela 5.1: Tabela s številom testnih primerov glede na dolžino optimalne rešitve ter številom robotov.

5.1 AVG

Iz tabele 5.2 lahko potrdimo trditev, da je hevristična funkcija AVG popolna, saj zmeraj najde najkrajšo možno pot, vendar za to potrebuje veliko časa, na kar kaže število razvitih stanj. Ta naraščajo hitro in kot je vidno iz tabele, potrebuje pri štirih robotih razviti približno 60000 stanj, pri petih pa že več kot 800000. Za 6 robotov je program spominsko že preveč zahteven, a če ocenimo, bi za 6 robotov potreboval razviti približno 15 miljonov stanj.

število robotov	1	2	3	4	5	6
povprečno število razvitih stanj	9	157	3485	61143	804764	t.c.
povprečna dolžina rešitvene poti	3.37	4.14	4.76	5.32	5.27	

Tabela 5.2: Tabela z rezultati za AVG.

5.2 MAX

Za funkcijo MAX lahko glede na tabelo 5.3 tudi potrdimo, da je popolna, vendar v primerjavi z AVG razlikuje v tem, da pri iskanju razvije veliko manj stanj, pri čemer pa še zmeraj ohrani optimalnost. Razlog za to je boljša ocena, ki je višja, a še zmeraj manjša oz. enaka ceni najcenejše poti do končnega stanja, kar pomeni, da je še zmeraj optimistična. Posledično je iskanje hitrejše. Funkcija lahko obravnava več robotov, v našem primeru enega več, nato postane zopet spominsko preveč zahtevna. Čas računanja je odvisen od primera. Če je primer prezahteven, se izvajanje ustavi po približno dveh urah, ker obravnavanje vedno večjega števila poti zahteva vedno več spomina na računalniku in ga na koncu zmanjka.

AVG in MAX bomo uporabili za oceno ostalih funkcij ter tako ugotovili, katere so bolj učinkovite in katere manj, pri iskanju optimalnih poti.

število robotov	1	2	3	4	5	6	7
povprečno število razvitih stanj	9	62	481	4378	24589	122594	t.c.
povprečna dolžina rešitvene poti	3.37	4.14	4.76	5.32	5.27	5.78	

Tabela 5.3: Tabela z rezultati za MAX.

5.3 SUM

Za tretjo obravnavano hevristično funkcijo SUM lahko iz tabele 5.4 razberemo, da ni popolna. V primerjavi z AVG iz tabele 5.2, ter MAX iz tabele 5.3 opazimo, da se z večanjem števila robotov vedno bolj razlikuje od optimalne rešitve. Iz tega lahko zaključimo, da naraščanje števila robotov privede do slabše ocene, saj zaradi načina izračuna po enačbi (4.3) hevristični algoritem vedno bolj preceni ceno najcenejše poti do ciljnega stanja.

Glede časa iskanja oz. števila razvitih stanj je pa izjemno hitrejša oz. razvije veliko manj stanj kot prej omenjeni AVG in MAX. Iskanje je hitro, najdena rešitev pa slabše kvalitete.

št. r.	1	2	3	4	5	6	7	8	9	10
p.š.r.s.	9	57	286	1208	4466	14417	47152	147645	325365	t.c.
p.d.r.p.	3.37	4.18	4.94	5.76	6.27	7.38	8.61	10.45	12.15	

Tabela 5.4: Tabela z rezultati za SUM.

5.4 Utežena SUM

Zadnjo hevristično funkcijo uteženo SUM smo, kot je predstavljeno v tabeli 5.5, razdelili na devet uteži od 0.1 do 0.9, z 0.1 intervali. Kot pričakovano je funkcija z manjšo utežjo privedla do optimalne oz. skoraj optimalne rešitve, pri čemer je bila kompleksnost posledično večja, nasprotno pa je bilo pri večjih utežeh, s katerimi je program našel suboptimalne rešitve, vendar je bila kompleksnost veliko manjša. Zaradi tega smo želeli najti zlato sredino, za katero bi veljalo, da program še zmeraj najde optimalno oz. skoraj optimalno rešitev, kompleksnost bi zmanjšali in tako lahko obravnavali več robotov.

Kot se da razbrati iz tabele 5.5 je najboljše razmerje pri uteži z vrednostjo 0.5, ki je privedla do optimalne rešitve oz. se je z večanjem števila robotov počasi poslabševala. Rešitev je blizu optimalne, kompleksnost pa ne narašča

tako hitro in posledično lahko obravnavamo tudi več robotov. Za končno primerjavo bomo zato uporabili funkcijo z utežjo 0.5.

št. r.	1	2	3	4	5	6	7	8	9	utež
p.š.r.s.	14	353	8233	183345	t.c.	t.c.	t.c.	t.c.	t.c.	0.1
p.d.r.p.	3.37	4.14	4.76	5.32						
p.š.r.s.	14	311	5648	93258	875942	t.c.	t.c.	t.c.	t.c.	0.2
p.d.r.p.	3.37	4.14	4.76	5.32	5.27					
p.š.r.s.	14	262	4057	41352	387543	759487	t.c.	t.c.	t.c.	0.3
p.d.r.p.	3.37	4.14	4.76	5.32	5.33	5.89				
p.š.r.s.	13	205	2428	13482	62399	317829	392123	t.c.	t.c.	0.4
p.d.r.p.	3.37	4.14	4.76	5.33	5.39	6.13	7.04			
p.š.r.s.	12	157	1293	4380	17052	50483	158239	549283	1174632	0.5
p.d.r.p.	3.37	4.14	4.77	5.39	5.57	6.32	7.23	8.54	10.15	
p.š.r.s.	12	108	691	2179	8766	29604	80724	358273	986231	0.6
p.d.r.p.	3.37	4.14	4.79	5.41	5.80	6.55	7.45	8.76	10.34	
p.š.r.s.	12	81	430	1592	5887	20829	62161	286423	795488	0.7
p.d.r.p.	3.37	4.14	4.84	5.45	5.93	6.79	7.73	8.97	10.58	
p.š.r.s.	12	67	327	1298	5009	16469	53150	195553	623856	0.8
p.d.r.p.	3.37	4.15	4.86	5.48	6.00	6.91	7.91	9.27	10.76	
p.š.r.s.	12	65	310	1272	4789	15587	49659	171923	515468	0.9
p.d.r.p.	3.37	4.15	4.88	5.53	6.09	7.00	8.00	9.47	11.01	

Tabela 5.5: Tabela z rezultati za uteženo SUM.

5.5 Grafična primerjava

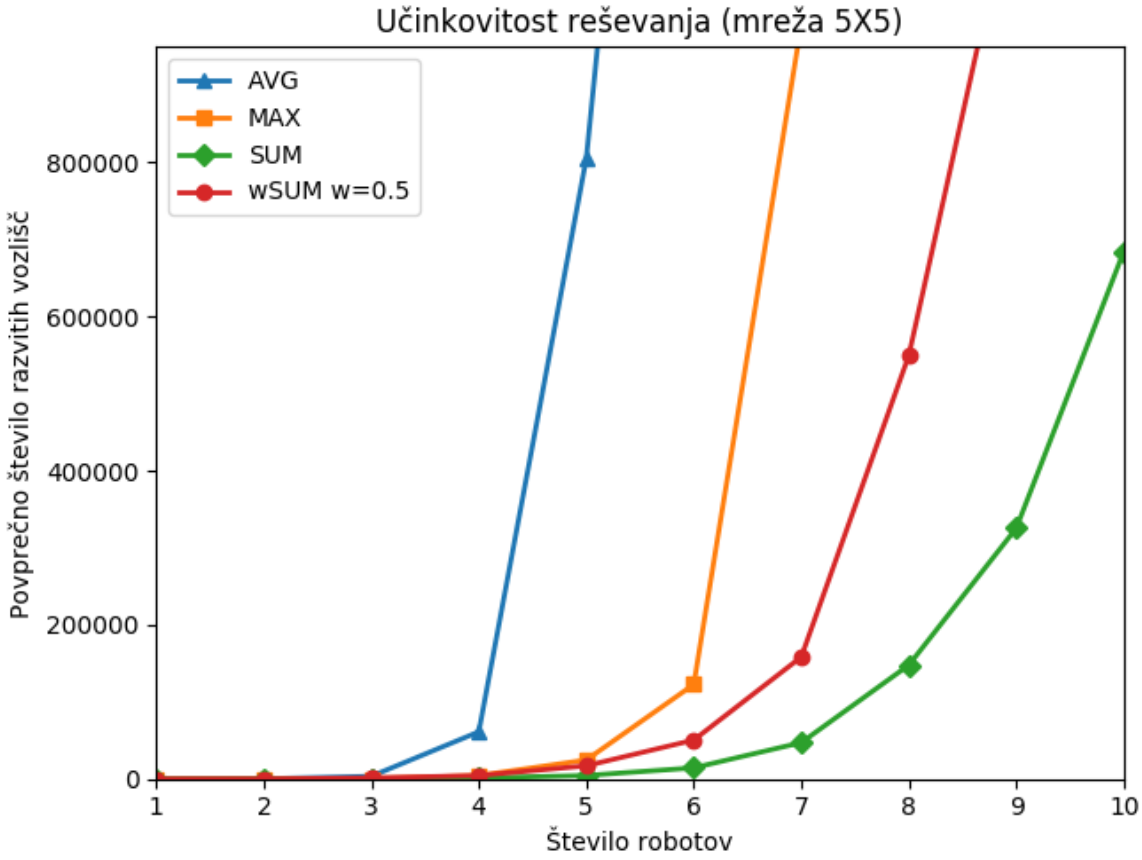
Vse ugotovitve za AVG, MAX, SUM in uteženo SUM ali u.SUM z utežjo $w = 0.5$, smo zbrali v tabeli 5.6.

št. r.	1	2	3	4	5	6	7	8	9	h.f.
p.š.r.s.	9	157	3485	61143	804764	t.c.	t.c.	t.c.	t.c.	AVG
p.d.r.p.	3.37	4.14	4.76	5.32	5.27					
p.š.r.s.	9	62	481	4378	24589	122594	t.c.	t.c.	t.c.	MAX
p.d.r.p.	3.37	4.14	4.76	5.32	5.27	5.78				
p.š.r.s.	9	57	286	1208	4466	14417	47152	147645	325365	SUM
p.d.r.p.	3.37	4.18	4.94	5.76	6.27	7.38	8.61	10.45	12.15	
p.š.r.s.	12	157	1293	4380	17052	50483	158239	549283	1174632	u.SUM
p.d.r.p.	3.37	4.14	4.77	5.39	5.57	6.32	7.23	8.54	10.15	

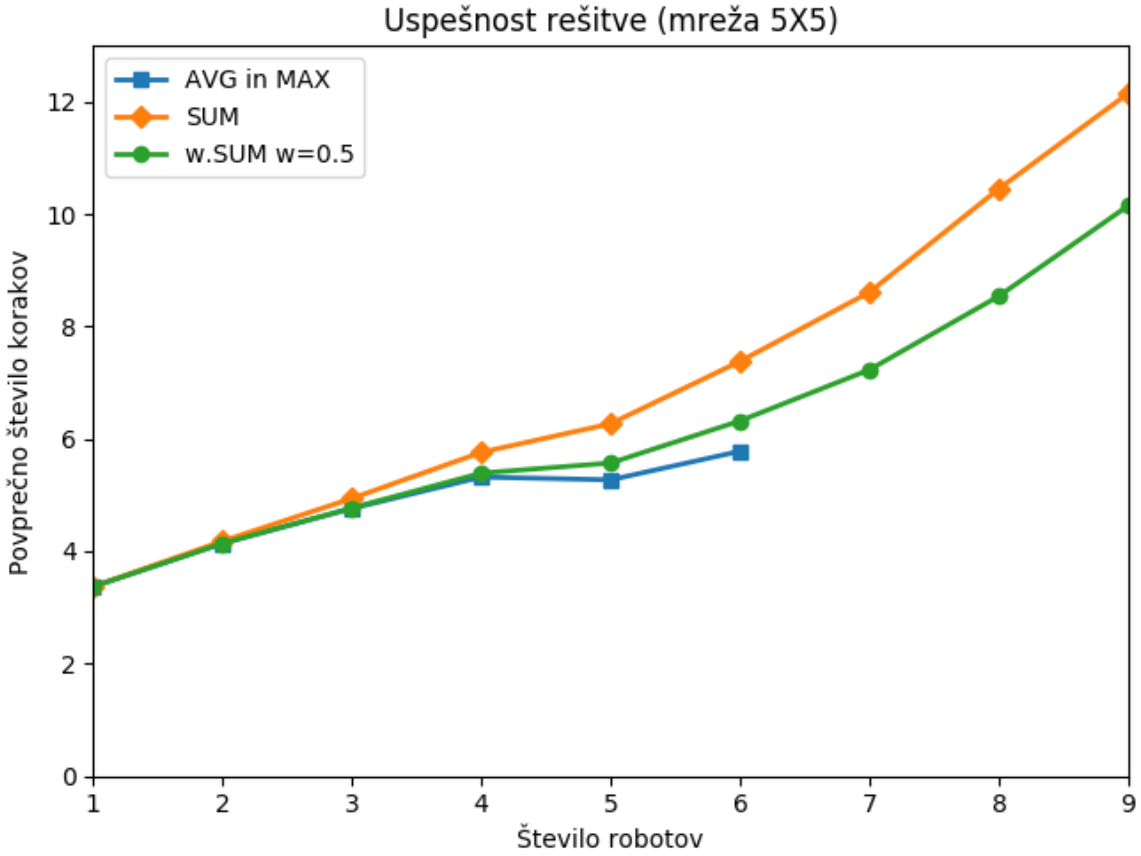
Tabela 5.6: Tabela z rezultati vseh hevrističnih funkcij.

Na teh podatkih smo naredili grafična prikaza za učinkovitost in uspešnost reševanja, ki sta prikazana na slikah 5.1 in 5.2. Kot je razvidno iz prve slike, pri vseh funkcijah število razvitih stanj z večanjem števila robotov hitro narašča, pri čemer SUM in utežena SUM naraščata najpočasneje, MAX in AVG pa veliko hitreje. To potrdi našo hipotezo, da MAX in AVG potrebujeta veliko časa oz. morata razviti veliko stanj, da najdeta rešitev. Pri naslednji sliki 5.2 pa lahko opazimo, da je uspešnost funkcij do štirih robotov približno enaka z izjemo funkcije SUM, ki je nekoliko slabša. S povečevanjem števila robotov, se nato razlike med funkcijami še povečujejo.

Če vzamemo v razmislek obe sliki, ima najboljše razmerje uspešnosti in učinkovitosti utežena SUM z utežjo 0.5. Za izračun poti potrebuje manj časa kot bolj zahtevni AVG in MAX, pot, katero najde, pa je v povprečju skoraj enako dobra. Ko pa je število robotov preveliko za AVG in MAX, utežena SUM še zmeraj najde poti, ki so blizu optimalne.



Slika 5.1: Učinkovitost reševanja.



Slika 5.2: Uspešnost rešitve.

Poglavje 6

Sklepne ugotovitve

Cilj diplomske naloge je bil ugotoviti, kateri hevristični algoritmi so dobri za paralelno izvajanje pri variantah igre 8 kvadratov. Ugotovili smo, da če iščemo algoritem, ki bo imel najboljše razmerje med učinkovitostjo in uspešnostjo, je to A* z uporabo hevristike weighted SUM z utežjo 0.5. Ta nam bo našel skoraj optimalno rešitev, pri čemer bomo veliko pridobili na času planiranja.

Seveda pa je vse odvisno od dane situacije. Če potrebujemo hitro najti pot in nam ni tako pomembno, če je optimalna ali ne, bomo vzeli algoritem A* s hevristično funkcijo SUM. Tako iskanje bi potrebovali npr. pri računalniških igricah, kjer nimamo časa iskati optimalne poti, ampak želimo najti čim boljšo pot v danem času. Drugi primer je pa obraten, ko imamo veliko časa in nas zanimajo zares samo optimalne poti. Takrat bi uporabili A* s hevristično funkcijo MAX, katera nam bo po daljšem računanju, a krajšem kot pri AVG našla optimalno pot.

Literatura

[1] Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Pearson Education, 2001.

[2] John Gaschnig. Performance measurement and analysis of certain search algorithms. Technical report, DTIC Document, 1979.

[3] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[4] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[5] John E. Laird, Allen Newell, and Paul S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[6] Nils J. Nilsson. Principles of Artificial Intelligence. 1980.

[7] Judea Pearl. Heuristics: Intelligent Search Strategies for Computer Problem Solving. 1984.

[8] Stuart Russell. Efficient memory-bounded search methods. *ECAI-1992, Vienna, Austria*, pages 1–5, 1992.

[9] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.