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

Jure Medvešek

Optimizacija prostorsko varčnega preiskovanja grafov z zaokroževanjem hevrističnih ocen

diplomsko delo
univerzitetni študij računalništva in matematike

Mentor: akad. prof. Ivan Bratko

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

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

Št. naloge: 00035/2012
Datum: 02.04.2012

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

Kandidat: **JURE MEDVEŠEK**

Naslov: **OPTIMIZACIJA PROSTORSKO VARČNEGA PREISKOVANJA GRAFOV Z ZAOKROŽEVANJEM HEVRISTIČNIH OCEN**

OPTIMISATION OF SPACE-SAVING GRAPH SEARCH WITH APPROXIMATE HEURISTIC VALUES

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Prostorsko varčni preiskovalni algoritmi imajo v primeru razpršenih ocenitvenih funkcij visoko časovno zahtevnost. Zahtevnost lahko zmanjšamo z zaokroževanjem ocen, vendar gre to navadno na račun optimalnosti rešitev. Naloga je preučiti vpliv zaokroževanja na časovno učinkovitost in odstopanje od optimalnih rešitev pri algoritmih IDA* in RBFS ter izvedbo teh algoritmov tako, da je še vedno zagotovljena optimalnost rešitev.

Mentor:

akad. prof. dr. Ivan Bratko

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan Fakultete za računalništvo in informatiko:

prof. dr. Nikolaj Zimic

Dekan Fakultete za matematiko in fiziko:

akad. prof. dr. Franc Forstnarič

UNIVERZA V LJUBLJANI • Fakulteta za matematiko in fiziko • 6

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Jure Medvešek, z vpisno številko **63050197**, sem avtor diplomskega dela z naslovom:

Optimizacija prostorsko varčnega preiskovanja grafov z zaokroževanjem hevrističnih ocen

S svojim podpisom zagotavljam, da:

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

V Ljubljani, dne 5. julija 2012 Podpis avtorja:

Zahvaljujem se mentorju Ivanu Bratku in članu laboratorija Aleksandru Sadikovem, ki sta z razpravami pripomogla priti do dobljenih rezultatov.

Nalogo posvečam staršem in bratu, ki so mi stali ob strani v času študija.

Kazalo

Povzetek

Iskanje optimalne poti v prostoru stanj je v praksi navadno težek problem. Namen te diplomske naloge je najti prostorsko učinkovit algoritem, pri katerem je čas iskanja optimalne rešitve računsko sprejemljiv tudi pri velikih problemskih prostorih.

Algoritem A^* deluje razmeroma hitro, vendar ima eksponencialno prostorsko zahtevnost. Algoritmi z linearno prostorsko zahtevnostjo glede na dolžino optimalne poti imajo časovno zahtevnost, ki je lahko tudi kvadratna glede na A^*, kjer v nobenem primeru ne izločamo duplikatov.

Za poddružino grafov s takimi lastnostmi bomo podali optimizacijo algoritma IDA^* s pomočjo granulacije f ocene, ki najde optimalno rešitev s časovno zahtevnostjo $O(N \log N)$, kjer je N število vozlišč, ki jih razvije algoritem A^*. Prostorska zahtevnost algoritma je linearno odvisna od najdaljše preiskane poti.

Enaka ideja deluje tudi na algoritmu $RBFS$, saj z enako transformacijo $RBFS$ deluje enako hitro ali celo hitreje kot IDA^*.

Ključne besede

preiskovanje grafov, optimizacija, granulacija

Abstract

Finding an optimal path in a state space is often very difficult in practise. The goal of this diploma thesis is to find a space efficient algorithm which finds an optimal solution within an acceptable time frame, even in large problem spaces.

Algorithm A^* is relatively fast, but it has exponential space complexity. Algorithms with linear space complexity can have quadratic time complexity, in terms of A^*'s time complexity. In both cases we do not check for duplicates achieved by different paths.

For this type of graphs we propose an optimization of the IDA^* algorithm, where we granulate the heuristic function f. The new algorithm finds an optimal solution with time complexity $O(N \log N)$, where N is the number of nodes expanded by algorithm A^*. Space complexity is linear in the length of the longest expanded path.

The same idea can be applied to algorithm $RBFS$ where $RBFS$ can be even slightly faster than IDA^*.

Keywords

search, optimization, granulation

Poglavje 1

Uvod

Cilj hevrističnih preiskovalnih algoritmov je najti najcenejšo rešitev v usmerjenem grafu G. Rešitev je pot med začetnim vozliščem s in enim izmed končnih vozlišč. Pri iskanju take poti si mnogi algoritmi pomagajo z oceno $f(a) = g(a) + h(a)$, kjer je $g(a)$ cena trenutno najcenejše znane poti od začetnega vozlišča s do vozlišča a. $h(a) \geq 0$ je hevristična ocena cene poti od a do cilja, $h^*(a)$ je dejanska cena te poti.

1.1 Zgodovina

Najbolj poznan algoritem za iskanje najcenejše poti je A^* [14], saj zagotavlja, da bo vrnil optimalno rešitev ob asimptotično najkrajšem času izvajanja, a ima preveliko prostorsko zahtevnost za uporabo na težjih problemih.

Predlaganih je bilo že mnogo algoritmov, ki bi zapolnili vrzel med algoritmom A^* in algoritmi z linearno prostorsko zahtevnostjo.

Našo idejo smo razvili iz ideje v [2], kjer uvedejo preprosto granulacijo na algoritmu preiskovanja v globino. Pri preiskovanju s pomočjo hevristične ocene je naša ideja najbližja algoritmu $RIDA^*$ [1], kjer ob vsaki iteraciji mejo premaknejo na vrednost, ki jo ocenijo s števila razvitih vozlišč v prejšnjih iteracijah. Ob analizi se izkaže, da ob omejitvi napake oba algoritma delujeta enako. Trdimo, da je sprememba f ocene manjši in bolj splošen poseg v

poljuben preiskovalni algoritem.

Ostali predlagani algoritmi [12], [10] temeljijo na preiskovanju v naprej in običajno porabijo ves razpoložljiv pomnilnik.

Poglavje 2

Analiza časovne zahtevnosti algoritmov

2.1 Predpostavke

Zaradi poenostavitve analize algoritmov predpostavimo, da je hevristična ocena h optimistična, in da vsi algoritmi razvijejo vozlišča na optimalni poti kot zadnje izmed tistih z enako f oceno (drugi del pri unikatnih f ocenah vedno drži).

Algoritem A^* ob taki hevristični funkciji najde optimalno pot ob asimptotično najmanjšem številu razvitih vozlišč. Pravzaprav razvije samo vsa vozlišča s hevristično oceno manjšo ali enako f^* (cena optimalne rešitve v grafu).

Če je ocena h optimistična, a hkrati f ni monotona, se lahko zgodi, da A^* izpusti katero izmed vozlišč, ki ima oceno manjšo od optimalne, a obstaja konstrukcija [8], ki vsako popolno funkcijo f trivialno preslika v monotono (ocena sina je maksimum med njegovim f-jem in f-jem očeta). Pri taki preslikavi vsa vozlišča z oceno $f < f^*$, ki jih algoritem A^* ni razvil, dobijo vrednost $f' \geq f^*$. V primeru popolnosti f ni škode za optimalnost.

Preverjanje duplikatov bomo opustili, saj je v [5] dokazano, da ne moremo preverjati duplikatov, če si ne zapomnimo vseh dosedanjih stanj. Zaradi te

lastnosti bi ob preverjanju duplikatov izgubili linearno porabo pomnilnika.

2.2 Časovna zahtevnost algoritma IDA^*

Izrek 2.1 *Algoritem IDA^* v najslabšem primeru razvije*

$$X_m(N) = mN - \frac{m(m-1)}{2} \tag{2.1}$$

vozlišč, kjer je N število vozlišč, ki jih razvije algoritem A^* in m število iteracij algoritma IDA^* [1].

Dokaz. Ob danih omejitvah algoritem IDA^* razvije natanko tista vozlišča kot A^*, le da tu mnogo vozlišč raziščemo več kot enkrat.

Definiramo naslednje povezave:

- N je število vozlišč, kjer velja $f(a) \leq f^*(a)$
- t_i je število razvitih vozlišč v iteraciji i
- m je število iteracij (število različnih ocen, kjer velja $f \leq f^*$)
- po definiciji v vsaki iteraciji razvijemo vsaj eno dodatno vozlišče $t_{i-1} \leq t_i - 1$
- v zadnji iteraciji algoritma razvijemo $t_m = N$ vozlišč

Iz zgornjih zvez izpeljemo $t_{m-i} \leq t_m - i = N - i$ in vstavimo v enačbo, ki sešteje vsa razvita vozlišča:

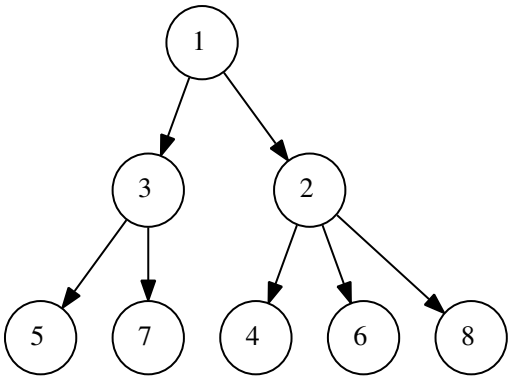
$$X_m(N) = \sum_{i=1}^{m} t_i \leq \sum_{i=1}^{m} (N - i + 1) = mN - \frac{m(m-1)}{2}$$

□

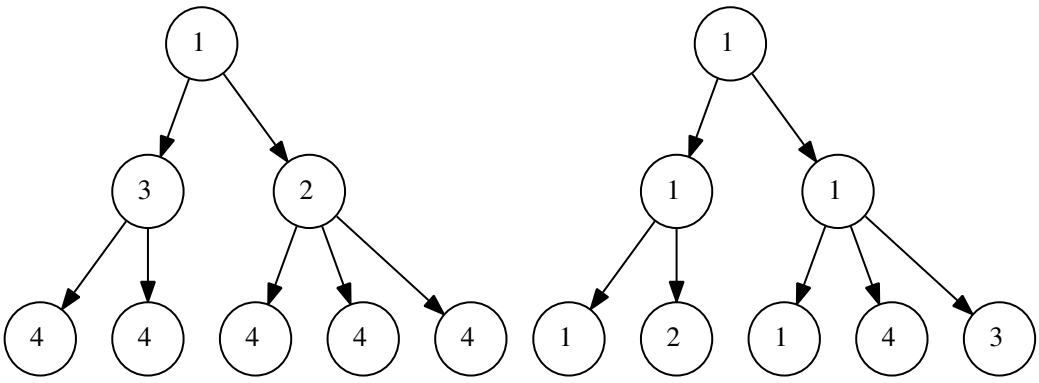
[1] Podoben nastavek $X_m = \sum_{i=1}^{m} \sum_{j=1}^{i} x_{new}(j)$, kjer število $x_{new}(j)$ predstavlja število na novo razvitih vozlišč v j-ti iteraciji, predlagajo že v [6]. Hkrati tam ne podajo splošne zgornje meje v odvisnosti od števila iteracij m.

V primeru, ko je funkcija f monotona in so vse vrednosti f unikatne, velja $m = N$, od koder dobimo zgornjo mejo za časovno zahtevnost algoritma IDA^*, ki je $O(N^2)$. Poleg tega lahko opazimo tudi, da v primeru, ko število iteracij m uspemo navzgor omejiti s konstanto c, časovna zahtevnost algoritma pade na $O(mN) \leq O(cN) = O(N)$, ki je asimptotično enaka časovni zahtevnosti algoritma A^*.

Iz zadnje ugotovitve sledi, da v primeru, ko učinkovito zmanjšamo m, povečamo učinkovitost algoritma IDA^*.



Slika 2.1: Primer grafa, kjer velja $m = N$



Slika 2.2: Število razvitih vozlišč, ki jih razvije algoritem IDA^*, s parametroma N in m ni popolnoma določeno. Levo je najugodnejši primer s 14 razvitimi vozlišči, desno najbolj neugoden s 26 razvitimi vozlišči.

Poglavje 3

Algoritem

3.1 Osnovna ideja

Število iteracij algoritma IDA^* zmanjšamo tako, da s pomočjo granulacije med seboj združimo sosednje vrednosti f. Paziti moramo, da je funkcija združevanja monotono naraščajoča funkcija, ki slika v množico števil.

Definiramo naslednje povezave, kjer so vse enolično določene z relacijo (3.1), ki je tudi edina, ki jo uporabljamo med izvajanjem algoritma.

$$\lfloor x \rfloor_r = t;\ \forall x, y : x < y \Rightarrow \lfloor x \rfloor_r \leq \lfloor y \rfloor_r \tag{3.1}$$

$$f_{min(r)}(x) = min\left\{x'; \lfloor x \rfloor_r = \lfloor x' \rfloor_r\right\} \tag{3.2}$$

$$f_{max(r)}(x) = max\left\{x'; \lfloor x \rfloor_r = \lfloor x' \rfloor_r\right\} \tag{3.3}$$

$$\lfloor x \rfloor_r^{-1} = \left[f_{min(r)}(x), f_{max(r)}(x)\right) \tag{3.4}$$

$$|x|_r = f_{max(r)}(x) - f_{min(r)}(x) \tag{3.5}$$

$$M_r(x) = |\{t; \lfloor y \rfloor_r = t \wedge t \leq \lfloor x \rfloor_r\}| \tag{3.6}$$

Relacija (3.1) predstavlja združevanje vrednosti, (3.4) pa interval, kjer ima funkcija (3.1) enake vrednosti. Relacija (3.6) predstavlja število intervalov v transformiranem prostoru, kjer je poljubna vrednost manjša ali enaka x.

Preprosta implementacija take transformacije je kar razširjena definicija zaokroževanja navzdol, kjer z parametrom r označimo ločljivost:

$$\lfloor x \rfloor_r = (x \div r) * r \tag{3.7}$$

Nekaj primerov:

$$\lfloor 23 \rfloor_{15} = (23 \div 15) * 15 = 1 * 15 = 15$$

$$\lfloor 29 \rfloor_{15} = (29 \div 15) * 15 = 1 * 15 = 15$$

$$\lfloor 32 \rfloor_{15} = (32 \div 15) * 15 = 2 * 15 = 30$$

$$\lfloor 37 \rfloor_{15}^{-1} = [30, 45)$$

$$|37|_{15} = 45 - 30 = 15$$

S pomočjo $\lfloor x \rfloor_r$ definiramo transformacijo hevristične funkcije f:

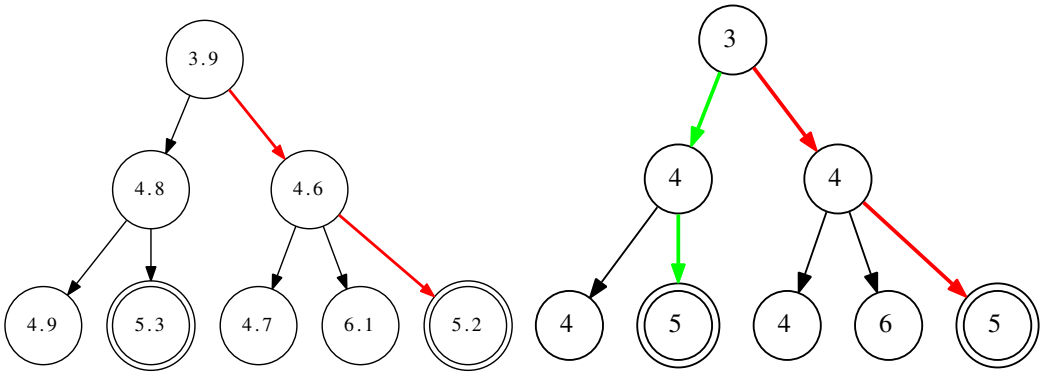
$$f_r(a) = \lfloor f(a) \rfloor_r \tag{3.8}$$

3.2 Implementacija transformacije

Transformacijo lahko implementiramo v poljubnem algoritmu, ki uporablja hevristično funkcijo f, kjer le-to nadomestimo s f_r. Učinek na preiskovanje si najlažje predstavljamo kot transformacijo preiskovalnega prostora, ki ima vsa vozlišča in povezave enake, le da ima spremenjene f ocene vozlišč.

Zaradi enake strukture grafa je preslikava rešitve iz transformiranega v originalen prostor preprosta:

- najdena pot v originalnem in transformiranem grafu je enaka
- cena rešitve v originalnem grafu je enaka $g = \sum_{e \in pot} c(e)$, kjer funkcija c predstavlja ceno povezave e



Slika 3.1: Levo graf z oceno f, desno z granulirano f oceno f_r in $r = 1$. Odebeljene povezave predstavljajo poti do najcenejših rešitev v grafu.

3.2.1 Definicija oznak

Pri dokazovanju prostorske in časovne zahtevnosti algoritma se bomo opirali predvsem na lastnosti grafa pred in po transformaciji. Zaradi tega si definiramo nekaj oznak. Optimalno ceno rešitve označimo z f^*, kjer a predstavlja poljubno vozlišče v grafu.

$$f^* = min\{f(a); isGoal(a)\}$$

Najdena rešitev v transformiranem grafu je zaradi izreka o popolnosti in monotonosti transformacije $\lfloor x \rfloor_r$ tudi najcenejša v transformiranem grafu. Ceno te rešitve v originalnem grafu bomo označili z f^+.

$$f^+ = min\{f(a); isGoal(a) \wedge f_r(a) = min\{f_r(a); isGoal(a)\}\}$$

Naj bo f_d katerakoli vrednost iz množice cen ciljnih vozišč.

$$f_d \in \{f(a); isGoal(a)\}$$

Prostor pred transformacijo razdelimo na tri disjunktne podmnožice, kjer množica A predstavlja vozlišča, ki jih razvije algoritem A^*, množica B vsebuje vozlišča, ki jih dodatno razvijemo zaradi granulacije. V množici C so vsa preostala vozlišča v grafu.

$$A = \{a; f(a) \leq f^*\}$$

$$B = \left\{a; f^* < f(a) \leq f_{max(r)}(f^*)\right\}$$

$$C = \left\{a; f(a) > f_{max(r)}(f^*)\right\}$$

Kadar je velikost množice B končna, bomo zapisali:

$$|B| = i|A|$$

3.3 Velikost napake

Zaradi zaokroževanja vrednosti f lahko v transformiranem grafu najdemo pot, ki ni najkrajša v originalnem grafu, a lahko napako algoritma navzgor omejimo še pred začetkom preiskovanja.

Izrek 3.1 *Za poljuben graf z monotono hevristično funkcijo f in transformacijo f_r velja $\lfloor f^+ \rfloor_r = \lfloor f^* \rfloor_r$.*

Dokaz. *Optimalna rešitev je najkrajša izmed vseh, zato za vse dopustne rešitve velja $f^* \leq f_d$, ker je transformacija $\lfloor x \rfloor_r$ monotono naraščajoča funkcija, sledi*

$$\lfloor f^* \rfloor_r \leq \lfloor f_d \rfloor_r \tag{3.9}$$

Zaradi izreka o popolnosti za najdeno rešitev v transformiranem grafu za vse dopustne rešitve velja:

$$\left\lfloor f^+ \right\rfloor_r \leq \lfloor f_d \rfloor_r \tag{3.10}$$

Sedaj v enačbi (3.9) *nadomestimo f_d z f^+, v enačbi* (3.10) *pa z f^*, in dobimo.*

$$\lfloor f^* \rfloor_r \leq \left\lfloor f^+ \right\rfloor_r \wedge \left\lfloor f^+ \right\rfloor_r \leq \lfloor f^* \rfloor_r \Rightarrow \lfloor f^* \rfloor_r = \left\lfloor f^+ \right\rfloor_r$$

□

Izrek 3.2 *Zgornja meja za napako algoritma je $\epsilon = |f^*|_r$, kjer je f^* cena optimalne rešitve v originalnem grafu.*

Dokaz.

$$\lfloor f^* \rfloor_r = \left\lfloor f^+ \right\rfloor_r \Rightarrow \epsilon = f^+ - f^* \leq \left| f^+ \right|_r = |f^*|_r$$

□

V primeru, ko je funkcija f_r razširjeno zaokroževanje navzdol, je funkcija ϵ konstantna in velja $\epsilon = r$.

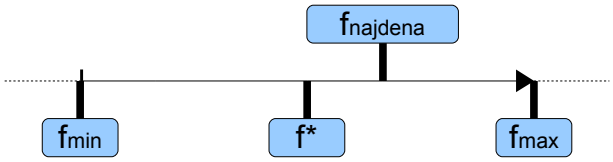
Izrek 3.3 *Kadar poznamo rešitev v transformiranem grafu, lahko napako še natančneje omejimo na* $\epsilon^* \leq f^+ - f_{min(r)}(f^+)$.

Dokaz.

$$f_{min(r)}(f^+) = f_{min(r)}(f^*) \leq f^*$$

$$\epsilon^* = f^+ - f^* \leq f^+ - f_{min(r)}(f^+)$$

□



Slika 3.2: Sosledje vrednosti v splošnem položaju

3.4 Časovna zahtevnost

Poleg funkcije X_m definirajmo še funkciji Y_m in Z_m, ki nam pokažeta, da kadar velja $m \approx N$, v vsakem primeru razvijemo $O(N^2)$ vozlišč.

Enačba za splošni primer, kjer predpostavimo, da v vsaki iteraciji na novo razvijemo enako število razvitih vozlišč:

$$Y_m(N) = \sum_{i=1}^{m} t_i = \sum_{i=1}^{m} \frac{iN}{m} = \frac{N}{m} \sum_{i=1}^{m} i = \frac{N}{m} * \frac{m(m+1)}{2} = \frac{m+1}{2} N$$

Enačba za najugodnejši primer, kjer se večina iteracij zgodi, ko imamo majhen horizont:

$$Z_m(N) = N + \sum_{i=1}^{m-1} t_i = N + \sum_{i=1}^{m-1} i = N + \frac{m(m-1)}{2}$$

Izrek 3.4 *Kadar so vse vrednosti funkcije f v grafu unikatne, razvijemo $O(N^2)$ vozlišč.*

Dokaz. *Če so vse vrednosti funkcije f v grafu unikatne, velja $m = N$.*

$$X_N(N) = Y_N(N) = Z_N(N) = \frac{N(N+1)}{2} = O(N^2)$$

□

Izrek 3.5 *Ob hevristični funkciji f_r ima algoritem IDA^* kvečemu $M_r(f^*)$ iteracij.*

Dokaz. *Po izreku 3.1 velja $\lfloor f^+ \rfloor_r = \lfloor f^* \rfloor_r$. Po definiciji je število $M_r(x)$ odvisno le od $\lfloor x \rfloor_r$, torej je*

$$M_r(f^+) = M_r(f^*)$$

□

Izrek 3.6 *Obstaja družina grafov z $m \approx N$, kjer z algoritmom IDA^* in granulacijo ob omejeni napaki najdemo rešitev v času $O(cN)$, in velja $c \ll N$.*

Dokaz. *Število iteracij v transformiranem grafu lahko omejimo navzdol z $m' \leq M_r(f^*)$. Po definiciji algoritma IDA^* smo v zadnji iteraciji razvili vsa vozlišča v množici B in vsaj enega iz množice A. S pomočjo funkcije X_m navzgor omejimo časovno zahtevnost algoritma po granulaciji:*

$$T(N, m') = X_{m'}(|A|) + |B| = O(m'N) + O(|B|) \tag{3.11}$$

V primeru, ko je $|B|$ končno, to vrednost omejimo z $|B| \leq i|A|$, kjer vrednosti i ponavadi ne poznamo vnaprej, a se jo ob poznavanju problemskega prostora da oceniti.

$$T(N, m') = X_{m'}(|A|) + |B| = O(m'N) + i|B| = O((m' + i)N) \tag{3.12}$$

$$m' + i \leq M_r(f^*) + i \leq c$$

□

Iz meritev se je v splošnem pokazala povezava $|x|_r \leq \overline{c} \Rightarrow i \approx b$, kjer $\overline{c}$ predstavlja povprečno ceno povezave v grafu. Povprečno vejanje je označeno z b. Pri takšni granulaciji dobimo $m' + i = d + b$, ki je v logaritemskem razmerju s številom raziskanih vozlišč $O(b^d)$.

Deljenje bolj na drobno se nam ne obrestuje, saj se poveča število razredov in s tem vrednost funkcije X_m. Ob bolj grobi granulaciji i zraste preko vsake meje, saj ocenjujemo, da v splošnem velja $|B| \approx b^{\frac{r}{\overline{c}}}$.

Izrek 3.7 *Obstajajo grafi, kjer brez predhodnega poznavanja f^* ne moremo definirati funkcije f_r tako, da bi bila množica $|B|$ končna.*

Dokaz. *Dokaza se lotimo s konstrukcijo grafa G, ki vsebuje podgraf G_1 s cenami, ki sledijo neskončnemu navzgor omejenemu zaporedju. Primer takega zaporedja je $f = 1 + \frac{d}{d+1}$. Hkrati moramo paziti, da cene vozlišč v G_1 omejimo z $f^* < f(v) < f_{max(r)}(f^*)$.* □

Takih grafov je v realnosti malo. V prid pa nam govori tudi povprečna dolžina povezav v podgrafu G_1. Ta limitira proti 0 in nam narekuje konstrukcijo transformacije s popolnoma fino granulacijo, in zato velja $f_r = f$. V splošnem velja, da se noben izmed algoritmov z dinamičnim premikom zgornje meje v takem primeru ne konča (tudi $RIDA*$ [1]).

3.5 Analiza algoritma pri znani granulaciji in optimalni rešitvi

Dolžino optimalne poti v grafu lahko izrazimo kot $f^* = f_{min(r)}(f^*) + \delta$.

Glede na vrednost δ, ločimo 2 robna primera:

- $\delta \approx 0$: tu je interval, kjer se nahajajo vozlišča iz množice B zelo velik, kar lahko privede do nepotrebnega razvijanja velikega števila vozlišč. Ob tem zelo malo izvemo o točnosti rešitve ($\epsilon \leq |f^*|_r - \delta \approx |f^*|_r$).
- $\delta \approx |f^*|_r$: tu je množica B z veliko verjetnostjo zelo majhna, kar zmanjša število po nepotrebnem razvitih vozlišč. Hkrati algoritem vrne rešitev, ki ima približno enako ceno kot optimalna ($\epsilon \leq |f^*|_r - \delta \approx 0$).

Poglavje 4

Ponovna uporaba predhodne rešitve pri računanju točnejše ocene

Naš algoritem lahko dopolnimo tako, da s pomočjo predhodnih približkov najdemo rešitev, ki je po ceni bližja optimalni.

Pomagamo si s funkcijo $f_{min(r)}(x)$, kjer kot x vstavimo ceno rešitve, ki jo je algoritem IDA^* našel pri trenutni granulaciji. Po izreku 3.1 sklepamo, da se cena optimalne rešitve nahaja na intervalu $\left[f_{min(r)}(x), x\right]$. Trivialna uporaba te lastnosti je nastavitev meje, preden zaženemo preiskovalni algoritem:

- $RBFS$: $root.F = f_{min(r)}(x)$
- IDA^*: $threshold = f_{min(r)}(x)$

Ta podatek obema preiskovalnima algoritmoma omogoči, da pride do regeneriranja vozlišč šele, ko imajo vsa vozlišča na trenutnem horizontu višjo oceno kot $f_{min(r)}(x)$.

Na praktičnih primerih se izkaže, da podana uporaba predhodne rešitve ne zadošča, saj se največ vozlišč v originalnem grafu regenerira ravno zaradi vozlišč, ki so na intervalu $\left[f_{min(r)}(x), x\right]$. Premik spodnje meje na $f_{min(r)}(x)$ na razvijanje le-teh ne vpliva.

4.1 Ponovna uporaba ob želeni omejeni napaki

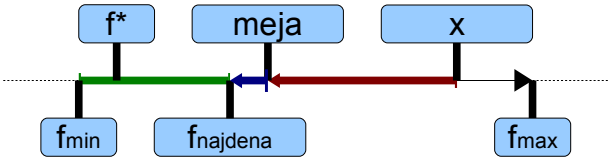
Uporaba podatka $f_{min(r)}(x)$ je uspešnejša, če se iskanja optimalne rešitve lotimo z bisekcijo, kjer v vsaki njeni iteraciji razpolovimo interval, na katerem se lahko nahaja cena optimalne rešitve. Za potrebo bisekcije definiramo novo funkcijo granulacije f_b, ki jo uporabimo s poljubnim preiskovalnim algoritmom (algoritem pri taki ocenitveni funkciji deluje enako kot preiskovanje v globino, kjer le-to vnaprej omejimo).

$$f_b(threshold, f) = \begin{cases} 0; & f \leq threshold \\ \infty; & f > threshold \end{cases} \tag{4.1}$$

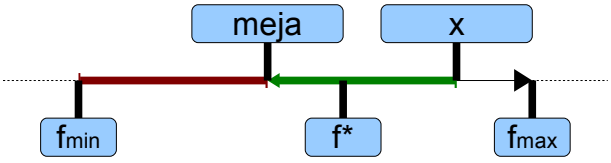
Kot mejo uporabimo $threshold = \frac{f_{min(r)}(x)+x}{2}$, in glede na novo dobljeno rešitev, ločimo dva primera. V primeru, ko algoritem najde rešitev, velja

$$f^* \in \left[f_{min(r)}(x), f^*\right] \wedge f^* \leq threshold \tag{4.2}$$

drugače $f^* \in (threshold, x]$.



Slika 4.1: Graf, kjer algoritem z granulacijo f_b najde rešitev. Vrednost $f_{najdena}$ predstavlja ceno rešitve, ki je bila najdena s pomočjo bisekcije.



Slika 4.2: Graf, kjer algoritem z granulacijo f_b ne najde rešitve

V obeh primerih razpolovimo območje, kjer se nahaja cena optimalne rešitve. Hkrati v primeru (4.2) v splošnem interval še natančneje ocenimo.

Izrek 4.1 *Rešitev z omejitvijo napake* $\frac{|f^*|_r}{u}$ *izračunamo v času*

$$T = O((m' + i \log_2 u)N)$$

kjer velja $|B| \leq i|A|$ *in* $m' \leq M_r(f^*)$.

Dokaz. *Algoritem* IDA^* *z granulacijo* f_b *vedno izvede eno iteracijo, katere čas je navzgor omejen z*

$$T({IDA^*}_b) = |A \cup B| = O((1 + i)N)$$

Ker na vsakem koraku razpolovimo interval, na katerem se nahaja cena optimalne rešitve, za skupni čas izvajanja velja [1]*:*

$$T = T({IDA^*}_r) + T({IDA^*}_b) \log_2 u$$

$$T = O((m' + i)N) + O((1 + i)N) \log_2 u = O((m' + i \log_2 u)N)$$

□

4.2 Ponovna uporaba ob želeni optimalni rešitvi

Opazimo lahko, da kadar vrednost f^* poznamo, lahko z algoritmom ${IDA^*}_b$ ali $RBFS$ dobimo optimalno pot z eno iteracijo algoritma. In sicer z linearno porabo pomnilnika in časovno zahtevnostjo $O(A^*)$.

Definiramo 3-fazni algoritem $GIDA^*$.

1. algoritem IDA^* poženemo z ocenitveno funkcijo f_r in dobimo rešitev s ceno x

2. preiščemo vsa vozlišča z oceno $f \leq x$ in si zapomnimo najmanjšo vrednost f (f^*), kjer je bilo najdeno ciljno vozlišče

[1] Točna ocena časovne zahtevnosti je $T_1 = O((m' + i + \log_2 u + i \log_2 u)N)$, ki je pri $u > 2$ navzgor omejena z $2 \times T$. Zaradi tega lahko majhne člene pri O notaciji izpustimo.

3. uporabimo algoritem $IDA^*{}_b$ z ocenitveno funkcijo f_b in $threshold = f^*$.

Algoritem, ki ga potrebujemo v drugi fazi, najlaže definiramo, če preuredimo algoritem IDA^*. Kot začetno mejo nastavimo $threshold = x$ in jo spreminjamo le, če je vozlišče končno. Mejo ob vsakem najdenem končnem vozlišču v nastavimo na $threshold = \min(threshold, f(v))$, saj iščemo najcenejšo rešitev v grafu. Rezultat, ki ga algoritem vrne, je $threshold = f^*$ [2].

Izrek 4.2 *Algoritem $GIDA^*$ vrne optimalno rešitev.*

Dokaz. *V prvi fazi zaradi izreka 3.1 velja, da se cena optimalne rešitve nahaja na intervalu $\lfloor x \rfloor_r^{-1}$.*

Na drugem koraku preverimo vsa vozlišča na tem intervalu, s tem da vozlišča z $f \geq x$ izpustimo, saj smo v prvi iteraciji dokazali, da obstaja pot, ki je vsaj tako dobra. Optimalne poti si ne zapomnimo, saj bi nam to vzelo preveč časa ali prostora ($T(alg) * S(alg) \geq O(N) = O(b^d)$). Zapomnimo si le dolžino minimalne poti f^*.

Algoritem $IDA^*{}_b$ z $thresold = f^*$ najde pot z ceno f^*, ki je optimalna, saj smo v drugi fazi dokazali, da cenejša rešitev ne obstaja. Hkrati ne razvije nobenega vozlišča, ki ima ceno višjo od f^*. □

Izrek 4.3 *Algoritem $GIDA^*$ za izvajanje potrebuje $O(bd)$ prostora.*

Dokaz. *V vseh treh fazah uporabimo prirejen algoritem IDA^*, kjer si ničesar dodatno ne zapomnimo. Med posameznimi fazami si zapomnimo vrednosti x in f^*, ki zasedeta konstanten prostor. Od tod sledi $S(GIDA^*) = S(IDA^*) = O(bd)$.* □

Izrek 4.4 *Časovna zahtevnost algoritma $GIDA^*$ je $O(N(m'+2i))$, kjer velja $|B| \leq i|A|$ in $m' \leq M_r(f^*)$.*

Dokaz.

$$T = X_{m'} + 2|A \cup B| = O(N(m' + 2i))$$

□

[2] Namesto druge in tretje faze lahko uporabimo tudi algoritem DFS^* [13].

Slaba lastnost tega algoritma je, da mora pri vsaki razporeditvi vozlišč v B, razviti vsa vozlišča v množici $A \cup B$, kar pri nobenemu od ostalih algoritmov ni potrebno (v splošnem primeru je dovolj razviti vsa vozlišča v množici A).

Podani algoritem pri ugodni izbiri f_r, ki navzgor omeji število $m'+2i \leq c$, asimptotično deluje enako hitro kot A^*. Hkrati najde optimalno rešitev pri linearni porabi pomnilnika.

Izrek 4.5 *Algoritem $GIDA^*$ ima na grafih, kjer pri granulaciji f_r velja $|x|_r = \overline{c} \Rightarrow i \approx b$, časovno zahtevnost $O(N \log(N))$.*

Dokaz. *Ker je $|x|_r$ enaka ceni povprečne poti, velja $M_r = f^*/|x|_r = d$, kjer je d število povezav na optimalni poti. Hkrati velja $O(N) = O(b^d)$.*

$$T = O(N(m' + 2i)) = O(N(d + 2b)) = O(N \log(N))$$

□

Izrek 4.5 nam pove, da je pričakovani čas izvajanja primerljiv s časom izvajanja algoritma A^*. Pri algoritmu A^* moramo upoštevati, da imamo množico odprtih vozlišč implementirano kot prioritetno vrsto, ki ima časovno zahtevnost vstavljanja in jemanja elementov $O(\log(N))$. N je število elementov v prioritetni vrsti.

Poglavje 5

Klasifikacija problemskih prostorov

Problemske prostore klasificiramo glede na vrednost m, saj smo v izreku 2.1 pokazali, da ta vrednost ključno vpliva na število ponovno razvitih vozlišč.

Klasifikacijo definiramo na sledeči način:

$$\mathbf{R} = \begin{cases} R_{log}; & m \leq O(\log N) \\ R_{id}; & m > O(\log N) \wedge \exists f_r : (M_r + i = O(\log N)) \\ R_0; & ostali \end{cases} \tag{5.1}$$

Nad problemskimi prostori v razredu R_{log} sta algoritma, ki minimizirata vse tri kriterije (ceno, čas in prostorsko zahtevnost), kar IDA^* in $RBFS$. Oba vrneta optimalno rešitev, porabita linearno količino prostora in imata časovno zahtevnost $O(N \log N)$, ki je primerljiva z časovno zahtevnostjo algoritma A^* [7].

Za razred problemov r_{id} smo definirali algoritem $GIDA^*$, ki je po vseh treh merah primerljiv z algoritmoma IDA^* in $RBFS$ pri izvajanju algoritmov tipa r_{log}. Problem, ki ostaja odprt, je nepoznavanje množice problemskih prostorov, ki spadajo v ta razred. Za mnoge problemske prostore je dovolj, da funkcijo f_r definiramo kot razširjeno zaokroževanje navzdol. Pri ostalih je problem, ki nastopi ta, da ne vemo, ali ugodna funkcija f_r ne obstaja, ali je le nismo našli. Ena izmed možnosti je, da na problemu

algoritem/mera	`Časovna zahtevnost`	`Prostorska zahtevnost`
A^*	$O(N)$	$O(N)$
IDA^*	$O(N^2)$	$O(\log N)$
$RBFS$	$O(N^2)$	$O(\log N)$
$GIDA^*$	$O(N \log N)$	$O(\log N)$

Tabela 5.1: Učinkovitost algoritmov pri reševanju problemov tipa R_{id} z $m = O(N)$

poženemo algoritem $RIDA^*$ [1], ki funkcijo f_r konstruira med samim izvajanjem algoritma. Če je število razvitih vozlišč s tem algoritmom večje od reda $O(N \log N)$, lahko sklepamo, da granulacija f_r za probleme takega tipa ne obstaja.

Problemom v razredu R_0 se v tej nalogi nismo posvetili. V literaturi nismo našli algoritmov, ki bi ob linearni porabi pomnilnika in omejeni napaki zagotavljali časovno zahtevnost manjšo od $O(N^2)$. V praksi se dobro obnašajo algoritmi, ki obtežijo hevristično oceno h, a ne zagotavljajo, da bodo rešitev našli hitreje kot tisti brez obtežitve. Hkrati za rešitev nikoli ne moremo trditi, da je optimalna - čeprav velikost napake ob določitvi uteži navzgor omejimo.

Poglavje 6

Praktični primeri

6.1 Igra 15

Igra je definirana kot kvadrat velikosti 4×4 na katerem imamo 15 kvadratnih ploščic velikosti 1×1 in eno prazno polje. Ploščice lahko zapeljemo v vodoravni ali navpični smeri na sosednje polje, če je le-to prazno. Cilj je razporediti ploščice v pravi vrstni red.

Cena vsakega premika je enaka 1. Hevristično oceno razdalje od trenutnega do ciljnega stanja definiramo kot seštevek razdalj ploščic od njihove trenutne do ciljne pozicije v prvi normi. Pri taki definiciji imajo vse razporeditve ceno najcenejše poti manjšo ali enako 80.

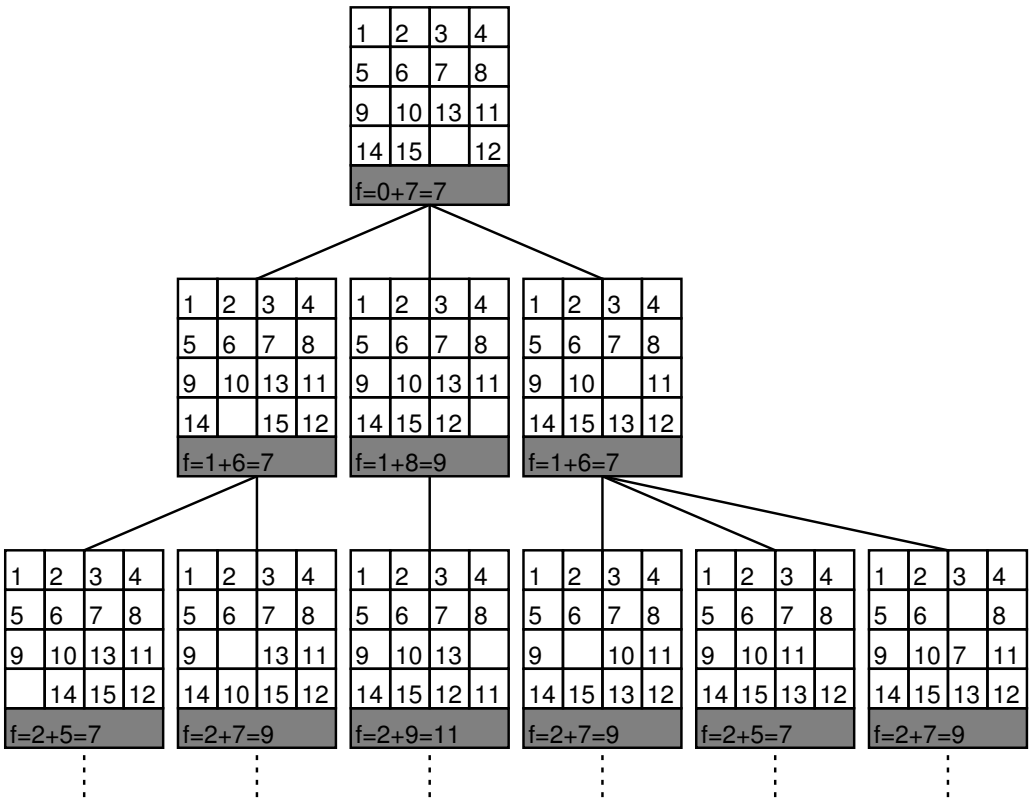
Izpostavimo nekaj lastnosti funkcij $g(a)$, $h(a)$, $f(a)$, kjer trenutna pozicija a ni ciljna.

$$g(a) \in \mathbb{N}_0$$

$$h(a) \in \mathbb{N}$$

$$0 < f^* = g + h \leq 80$$

Ker je seštevanje notranja operacija v množici naravnih števil, velja $f(a) \in \mathbb{N}$, kar navzgor omeji število iteracij, ki jih bo potreboval algoritem IDA^*. V najbolj neugodnem primeru funkcija f zavzame kar vso zalogo vrednosti $(1, 2, ..., 80)$, katere moč je 80 (pravzaprav se izkaže, da je korak vedno 2). Ker velja $m = O(\log N)$, spada igra 15 v razred R_{log}.



Slika 6.1: Primer reševanja igre 15

Funkcije f ne poskušamo nadalje granulirati, saj če vrednosti f^* vnaprej ne poznamo, v večini primerov čas iskanja povečamo. Hkrati algoritem tudi v najboljšem primeru, kjer uporabimo funkcijo f_b z mejo $threshold = f^*$, pospešimo največ za konstanten faktor $f^* \leq c \leq 40$.

Rezultati v tabeli 6.1 kažejo na to, da je razlika v časovni zahtevnosti med algoritmoma IDA^* in A^* še precej manjša. Časovno zahtevnost algoritma A^* ocenimo z algoritmom IDA^* in granulacijo f_b, kjer velja $b = f^*$.

$$c \approx \frac{T(IDA^*)}{T(A^*)} = 2^{24,70-23,47} = 2^{1,23} = 2,35$$

Tako velik razkorak med napovedano zgornjo mejo in izmerjenimi podatki si lahko razložimo s tem, da algoritem IDA^* v iteraciji i razvije $O(b_e^i)$ vozlišč.

r/mera	$\log_2(razvitih)$	$\bar{\epsilon}$	$max(\epsilon)$
/	24,70	0	0
3	24,50	0,58	2
4	24,43	1,00	2
5	24,22	1,46	4
6	24,06	1,66	4
7	24,20	2,44	6
8	24,00	2,60	6
9	24,05	3,60	8
f^*	23,47	0	0

Tabela 6.1: Rezultati pri granulaciji f_r (ϵ predstavlja napako)

Večino vozlišč zato razvijemo natanko enkrat (število razvitih vozlišč najlepše opiše funkcija Z_m).

6.2 Iskanje najkrajše poti med dvema točkama

Kot množico testnih primerov smo naključno konstruirali množico 1000 točk na mreži $[0, 1000)x[0, 1000)$, ki smo jih povezali s pomočjo Delaunayove triangulacije. Za vsak tako konstruiran graf smo iskali najkrajšo pot od začetne točke do 200 naključnih točk v grafu.

Razdaljo med poljubnima povezanima točkama a in b smo definirali kot $c = \sqrt{(b.x - a.x)^2 + (b.y - a.y)^2}$. Z enako formulo smo definirali tudi hevristično oceno trenutnega stanja.

$$g(a) \in \mathbb{R}$$

$$h(a) \in \mathbb{R}$$

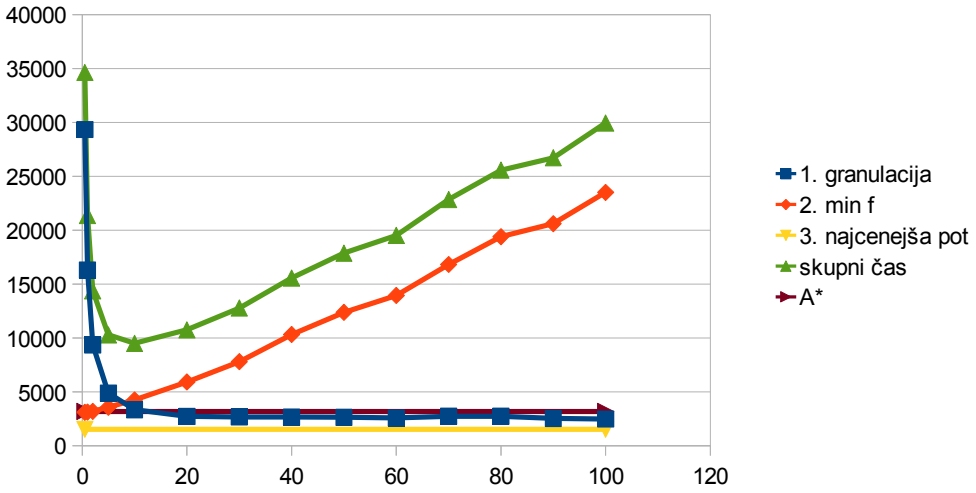
$$f(a) \in \mathbb{R}$$

V testnih primerih se je izkazalo, da velja $m \approx N$,in zato algoritma IDA^* in $RBFS$ potrebujeta $O(N^2)$ časa. Zaradi evklidskih razdalj in Delaunayeve

triangulacije lahko še pred začetkom izvajanja algoritma zelo natančno ocenimo ceno optimalne rešitve. Za ceno optimalne poti med točko a in končnim vozliščem velja $h(a) \leq f^*(a) \leq 1{,}5h(a)$.

Povprečna dolžina povezave med dvema točkama v grafu pri podanih parametrih prostora je $\overline{c} \approx 35$.

Kot funkcijo granulacije smo uporabili funkcijo razširjenega zaokroževanja navzdol $\lfloor x \rfloor_r$. Za vrednost r smo postopoma vzeli vsak element iz množice $\{0{,}5, 1, 2, 5, 10, 20, ..., 100\}$.



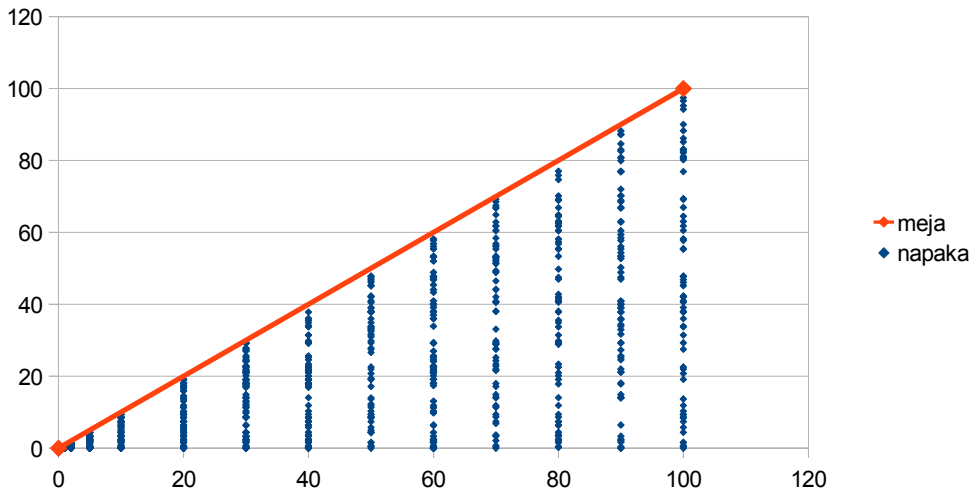
Slika 6.2: Na vodoravni osi so prikazane zaokrožitve v prvi fazi, na navpični osi pa število povprečno razvitih vozlišč. Kot primerjava sta algoritma IDA^* in $RBFS$ razvila 2^{21} in 2^{19} vozlišč.

Iz izmerjenih podatkov je razvidno, da vsaka od naštetih granulacij prevede ta problem do te mere, da pade v razred R_{id}. Hkrati opazimo, da se algoritem obnaša skladno z izrekom 4.5, ki za velikosti razreda predlaga $r = \overline{c}$. Ocenimo lahko tudi časovno zahtevnost algoritma v splošnem primeru, kjer je N število vseh poti, ki so krajše ali enake f^*.

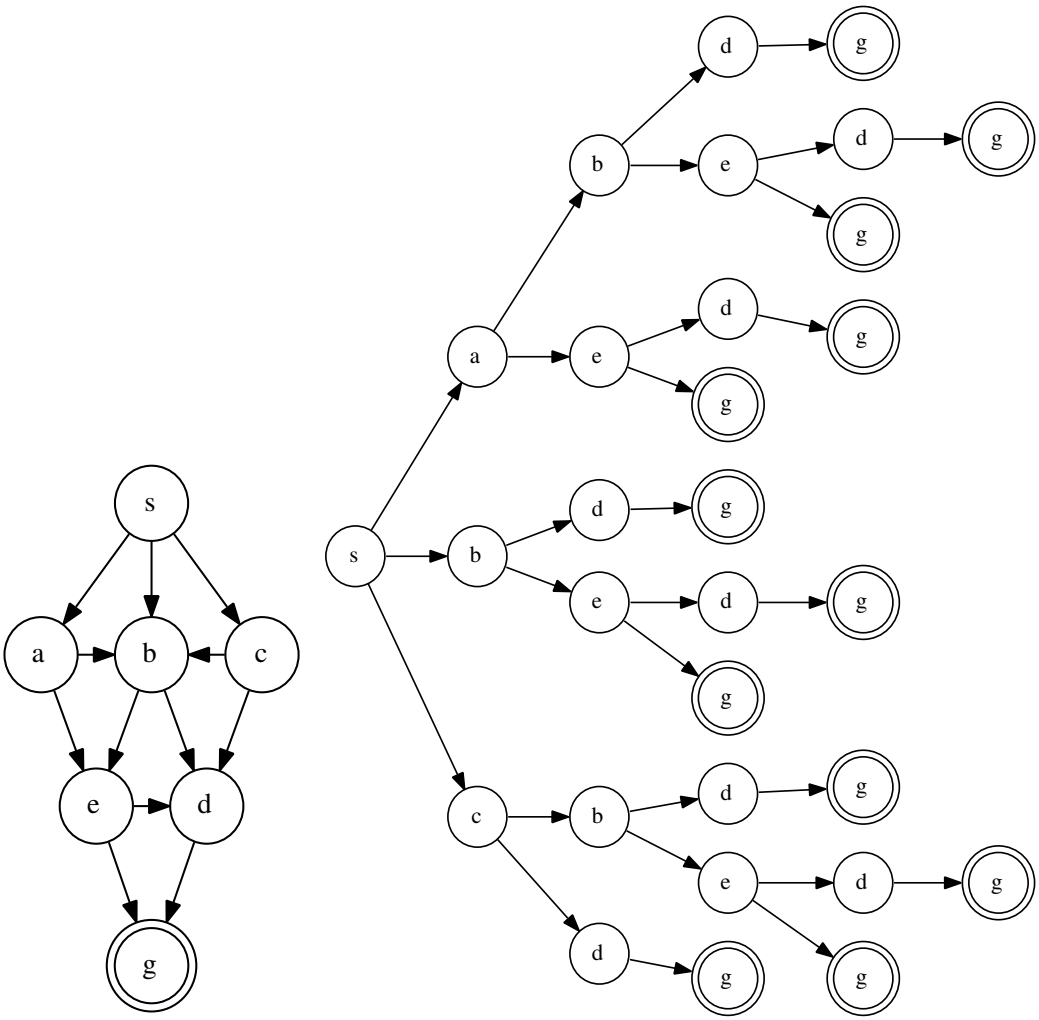
$$T = O(GIDA^*) = O(N(d + 2b)) = O(N(\frac{f^*}{\overline{c}} + 12))$$

V našem primeru velja $N \approx 2^{12} \approx 4000$ in $T = O(N(\frac{2100}{35} + 12)) = O(60N)$. S tem pokažemo, da v primeru ne preverjanja duplikatov stanj pri algoritmu A^*, algoritem $GIDA^*$ pri testnih primerih razvije kvečemu 60-krat več vozlišč, a pri tem porabi zanemarljivo malo pomnilnika.

Po naši oceni bi se morala granulacija podobno dobro odrezati tudi na realnih primerih iskanja najkrajše poti, saj ima cestno omrežje precej podobne lastnosti. Pazljivi pa moramo biti na to, da je na takem grafu zelo veliko transpozicij, ki jih moramo pri analizi algoritma v realni situaciji upoštevati (v praksi je tu algoritem A^* še vedno najboljša izbira).



Slika 6.3: Graf, ki potrjuje pravilnost izreka 3.2, ki omejuje največjo možno napako. Na grafu je prikazanih 1000 naključnih meritev, kjer je na vodoravni osi zaokroževanje, na navpični osi pa oddaljenost od najcenejše rešitve. Meja predstavlja zgornjo vrednost napake, ki jo izrek še dopušča.



Slika 6.4: Levo graf cestnega omrežja, desno vse možne poti do ciljnega vozlišča

Poglavje 7

Sklepne ugotovitve

Pokazali smo, da algoritem $GIDA^*$ deluje v razredu problemov R_{id} hitreje kot $RBFS$ in IDA^* pri enaki porabi pomnilnika.

Predlaganih je bilo že mnogo algoritmov ($RIDA^*$ [1], DFS^* [13], $IDA^* - CR$ [3]), ki delujejo na enaki osnovi (združevanje iteracij, ki privede do zmanjšanega števila ponovno razvitih vozlišč), vendar nobeden od njih ne reši tega z granulacijo f ocene. Ocenjujemo, da je naša rešitev enostavnejša in splošnejša, saj jo lahko uporabimo na poljubnem algoritmu, ki ima definirano hevristično oceno z $f(g, h)$.

Algoritem smo razvili neodvisno in brez poznavanja algoritma $RIDA^*$[1], a po podrobnejši analizi se je izkazalo, da se algoritem [1] v večini primerov obnaša bolje od našega.

7.1 Primerjava z algoritmom $RIDA^*$

Algoritem $RIDA^*$ izračuna naslednjo mejo s pomočjo regresije števila predhodno razvitih vozlišč. Od algoritma IDA^* se razlikuje samo v tem, da si zapomni število razvitih vozlišč in mejo po vsaki iteraciji. Ker je število iteracij ponavadi linearno odvisno od cene poti, to zasede zanemarljivo malo pomnilnika. Hkrati se tudi regresija izvede malokrat in algoritem zanjo potrebuje malo časa.

Trdimo, da je regresija ena izmed funkcij f_r (3.8), ki jo lahko uporabimo kot transformacijo grafa. Hkrati se zavedamo, da smo kot f_r preizkusili samo razširjeno zaokroževanje navzdol, saj se je na testnih primerih odrezalo dovolj dobro.

Delovanje algoritma $RIDA^*$ lahko natančno analiziramo s pomočjo naših izpeljav. Omejimo lahko napako v prvi fazi in skupno časovno zahtevnost, saj vse potrebne vrednosti (m, $|A \cup B|$, $f_{min(r)}$, ...) trivialno dobimo brez velikih sprememb algoritma.

7.2 Prednosti našega algoritma

Idejo granulacije lahko uporabimo na poljubnem algoritmu, ki ima definirano hevristično oceno f, česar pri $RIDA^*$ in IDA^*_{CR} ne moremo trditi. V praksi se izkaže, da algoritem IDA^* pri enakem številu razvitih vozlišč deluje najhitreje (hitreje kot A^*, $RBFS$, ...), kar izniči prednost prenosljivosti.

Kadar v našem algoritmu za funkcijo f_r uporabimo razširjeno zaokroževanje navzdol, v naprej poznamo vse možne robne vrednosti razredov. Ker je število takih razredov navzgor omejeno, lahko poljuben problem zelo dobro paraleliziramo. Katerikoli problem se po transformaciji obnaša podobno kot igra 15, za katero je definiranih že mnogo učinkovitih vzporednih algoritmov.

Literatura

[1] Benjamin W. Wah , Yi Shang, "Comparison and Evaluation of a Class of IDA* Algorithms", *Int'l Journal of Tools with Artificial Intelligence*, World Scientific, št. 4, zv. 3, str. 493–523, 1995

[2] M.E. Stickel and W.M. Tyson, "An analysis of consecutively bounded depth-first search with applications in automated deduction", Procs. 9th Intl. Joint Conf. on AI, str. 1073–75, 1985

[3] U. K. Sarkar, P. P. Chakrabarti, S. Ghose, S. C. DeSakar, "Reducing reexpansions in iterative-deepening search by controlling cutoff bounds", *Artificial Intelligence*, št. 50, zv. 2, str. 207–221, 1991

[4] E. Burns, W. Ruml, "Iterative-Deepening Search with On-line Tree Size Prediction", *Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION-12)*, 2012.

[5] A. Mahanti, S. Ghosh, D.S. Nau, A.K. Pal and L. Kanal, "Performance of IDA* on trees and graphs", 10th Nat. Conf. on Art. Int., AAAI-92, San Jose, CA, str. 53–544, 1992

[6] A. Mahanti, S. Ghosh, D. S. Nau, A. K. Pal, L. N. Kanal "On the asymptotic performance of IDA*", *Annals of Mathematics and Artificial Intelligence*, št. 20, zv. 1–4, str. 161–193, 1997

[7] Richard E. Korf, "Linear-space best-first search", *Artificial Intelligence*, št. 62, zv. 1, str. 41–78, 1993

[8] Richard E. Korf, "Depth-first iterative-deepening: An optimal admissible tree search", *Artificial Intelligence*, št. 27, zv. 1, str. 97–109, 1985

[9] I. Bratko, "Prolog Programming for Artificial Intelligence" 4th Edition, Pearson, 2011

[10] B. W. Wah, "MIDA*: An IDA* search with dynamic control", Univ. of Illinois, Champaign, Tech. Rep. UILU-ENG-91-2216, CRHC-91-9, 1991

[11] Z. Zhang, Nathan R. Sturtevant, R. Holte, J. Schaeffer, A. Felner, "A* Search with Inconsistent Heuristics", *IJCAI*, str. 634–639, 2009

[12] R. Stern, T. Kulberis, A. Felner, R. Holte, "Using Lookaheads with Optimal Best-First Search", *AAAI*, 2010

[13] V. N. Rao, V. Kumar, R. E. Korf, "Depth-first vs. best-first search", *Proceedings AAAI-91*, Anaheim, str. 434-440, 1991

[14] P. E. Hart, N. J. Nilsson, B. Raphael, "A formal basis for the heuristic determination of minimum cost paths", *IEEE Trans. Syst. Cybern*, št. 4, zv. 2, str. 100–107, 1968