SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

**ZAVRŠNI RAD br. 5390**

# Optimizacija pronalaženja najkraćeg puta uz dinamičko okruženje

*Pavao Jerebić*

Zagreb, Lipanj, 2018.

ZAHVALA

*Veliku zahvalu zaslužuje moja obitelj koja me bez suzdržavanja podržavala i poticala duž cijelog mog obrazovanja. Htio bi se posebno zahvaliti mojim roditeljima koji su bili uz mene kroz sve izazove koje sam prošao.*

*Zahvaljujem se mentoru prof. dr. sc. Domagoju Jakobovi*ću *na razumijevanju, savjetovanju i pomoći, kako pri odabiru teme tako i tijekom same razrade ovog završnog rada.*

# Sadržaj

# 1. Uvod

U užurbanom svijetu današnjice, vrijeme je vrlo vrijedan resurs koji je potrebno pažljivo trošiti. Ljudi su raznim izumima pokušavali olakšati svoj život skraćujući vrijeme utrošeno na razne poslove, od izuma osnovnih alata i kotača pa sve do automobila i Interneta. Ubrzavanjem i automatizacijom osnovnih ljudskih radnji veliki se dio svakodnevnih poslova olakšao. Prijenos ljudi, dobara i informacije jedna je od temeljnih takvih radnji. Promatrajući otkrića tijekom povijesti, brojna su bila usmjerena upravo prema ovom problemu. Poboljšanjem transporta, poput boljih cesta, brodova, vlakova i automobila, pojavili su se novi problemi kao što je bolje iskorištavanje trenutne infrastrukture. Često dolazi do situacija gdje nije iskorištena puna prohodnost takvih sustava. Gužve i zatvaranje nekih dijelova prometne mreže može poprilično efikasne načine prijevoza podosta usporiti.

Pretpostavka u ovom radu jest kako bi se većina takvih situacija mogla lako izbjeći boljim iskorištavanjem sustava i objekata koji se kreću u toj mreži. Kada bi se vozila dinamički preusmjeravala ovisno o trenutnom stanju mreže s prometnijih dijelova na one manje prometne, cjelokupno opterećenje na takvoj mreži bilo bi manje. Kod složenijih sustava, kao što su cestovne prometne mreže, postoje i dodatna svojstva, poput upravljanja prometa znakovima. To donosi dodatne komplikacije pri oblikovanju modela, pa takvi slučajevi nisu uzeti u obzir. Budući da je glavni fokus pronalaženje najkraćeg puta uz dinamičko okruženje, za opis modela su korištena samo vozila, njihove putanje i koncentracija vozila na nekoj prometnici. Kada se mreže promatraju na takav način, može se na više različitih načina pronaći rješenja.

U prvom dijelu rada nalaze se opisi deterministčkih algoritama i njihov razvoj. Predstavljeni su osnovni algoritmi pretrage grafova te njihove nadogradnje sve do konačnog algoritma koji se koristio kasnije. Drugi dio sadrži drukčiji pristup problemu. Opisuju se nedeterministički načini pronalaska rješenja. Budući da se ovaj problem može prikazati kao optimizacijski algoritam, gdje se traži minimum vremena koje je vozilu potrebno da stigne od početne do završne točke, možemo koristiti različite meta-heuristike. Predložene heuristike su neki od prirodom inspiriranih algoritama optimizacije. U poglavlju 3.1, Simulacija okruženja, opisana je mehanika simulatora

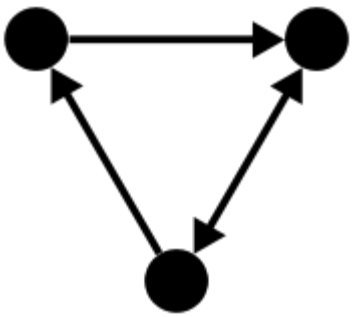
modela stvarnog sustava i u poglavlju 3.2 dan je opis predloženog algoritma koji koristi dijelove iz oba načina pretraživanja u svrhu postizanja boljih rezultata. U zaključku je dan sažetak rezultata, prednosti i nedostatci ovakvog načina rješavanja problema i moguće primjene.

# 2. Algoritmi pretraživanja

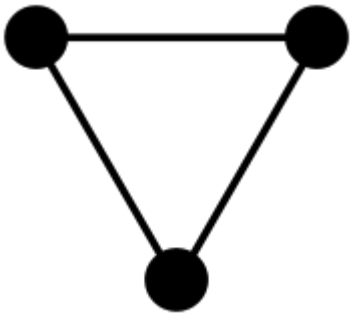
## 2.1 Grafovi

Graf je struktura koja je opisana međusobno povezanih objektima. U matematičkom smislu objekti se preslikavaju u čvorove i svaka veza između para čvorova naziva se brid. Čvorovi se također nazivaju i vrhovima. Bridovi mogu biti usmjereni (Slika 2.1 Usmjereni graf) i neusmjereni (Slika 2.2 Neusmjereni graf). Usmjerenost se određuje smjerom povezanosti, na primjer. Objekt A je povezan s objektom B, ali ne postoji povratna veza. Kod neusmjerenih bridova postoji veza u oba smjera. Također bridovi mogu imati pridodane vrijednosti koje se zovu težine. Težine mogu predstavljati cijene, kapacitete ili duljine, ovisno o problemu koji se obrađuje. Grafovi čiji bridovi imaju težine nazivaju se težinski grafovi. Ukoliko težina nije definirana, ona se po dogovoru postavlja na 1, pa se takvi grafovi nazivaju bestežinski grafovi. Grafovi se u matematici označavaju velikim slovom G, a čvorovi (*eng. Vertices*) i bridovi (*eng. Edges*) redom V, i E. Graf se opisuje kao uređeni par kao u izrazu (2.1).

$$G = (V, E) \tag{2.1}$$



*Slika 2.1 Usmjereni graf*



*Slika 2.2 Neusmjereni graf*

Broj bridova koji ulaze i izlaze iz čvora definira se kao stupanj čvora. Ako je početak i kraj brida isti čvor tada se taj brid naziva petlja i broji se dva puta. Zbog povezanosti čvorova po grafu se može prelaziti iz vrha u vrh, odnosno po njemu se može šetati. Šetnja u grafu je konačan slijed bridova u kojem su svaka dva uzastopna brida ili susjedna ili jednaka [1]. Šetnja u kojoj su svi bridovi različiti zove se staza, a ako su i svi vrhovi isti onda se radi o putu. Ako su početni i završni čvor u stazi ili putu isti tada se radi o zatvorenoj stazi ili putu. Zatvorena staza u kojoj u svi unutarnji vrhovi međusobno različiti se naziva ciklus. Ako postoji šetnja koja obuhvaća sve čvorove onda se radi o povezanom grafu. Poseban tip grafova je stablo. Stablo je graf koji ima *n* čvorova i *n-1* bridova. U stablu ne postoje ciklusi i petlje. Kod stabala se odnos između čvorova opisuje kao roditelj – dijete. Čvor koji nema roditelja se naziva korijenski čvor, a čvorovi bez djece su listovi.

## 2.2 Algoritmi pretraživanja grafova

Pretraživanje grafova, ili šetnja grafom, je proces posjećivanja svakog čvora u grafu. Za razliku od stabala, u grafovima je moguće da će neki čvor biti posjećen više puta zato što sve dok ne prijeđemo u taj čvor ne znamo jesmo li ga već posjetili. Što su grafovi veći to se češće događa ponovno posjećivanje nekog čvora. Ovaj problem se rješava pamćenjem posjećenih čvorova. Moguće je pridodati boju određenim čvorovima ili za svaki čvor pamtiti stanje posjećenosti. Te vrijednosti se ispituju i ažuriraju svakim posjećivanjem nekog čvora. Kod stabla nemamo ovakve probleme jer se pože pretpostaviti da su svi roditeljski čvorovi  već posjećeni. Posjećivanjem čvorova se stvara stablo pretraživanja. Pomoću tog stabla vidimo sam redoslijed

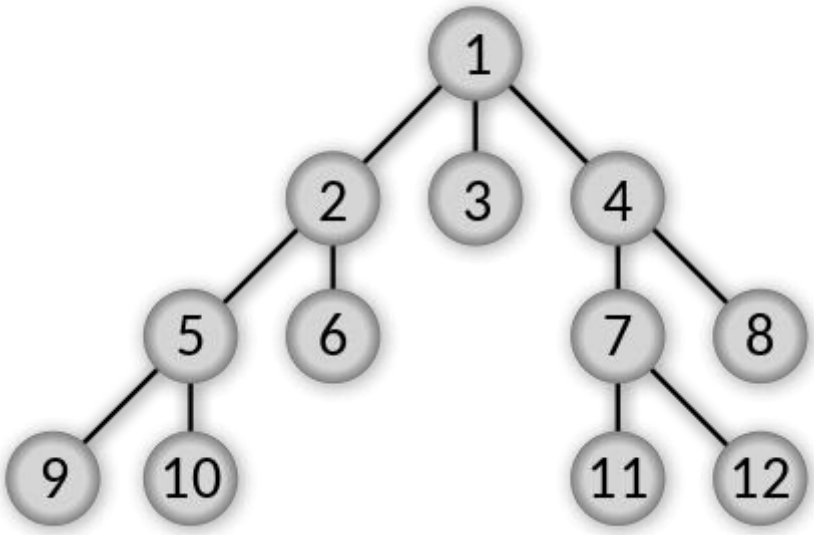
obilaska grafa gdje je početni čvor korijen stabla, a djeca svakog čvora su oni čvorovi koje smo posjetili iz tog čvora. Algoritmi pretraživanja imaju svojstva potpunosti i optimalnosti. Potpunost nekog algoritma označava da taj algoritam može u svim slučajevima pronaći rješenje a optimalnost znači da je algoritam pronašao rješenje s najmanjom cijenom. Algoritmi pretraživanja grafova se temelje redoslijedom posjećivanja čvorova pa tako imamo pretraživanje u širinu i pretraživanje u dubinu.

### 2.2.1 Pretraživanje u širinu

Pretraživanje u širinu (*eng. Breadth first search*) je algoritam za prolazak ili traženje po grafu. Počinje od korijenskog čvora i istražuje prvo sve susjedne čvorove pa tek onda one na sljedećoj razini. Ovaj algoritam su prvi izmislili Konrad Zuse i Michael Burke 1945. godine ali ga nisu objavili sve do 1972. [2] Za to vrijeme je Američki znanstvenik, Edward Moore, došao na istu ideju i 1959. predstavio svoj algoritam za rješavanje najkraćeg puta u labirintu. [3] Pretraživanje u širinu se obavlja dodajući u red svaku razinu stabla svaki put kada se pronađe korijen nekog pod stabla. Algoritam je opisan u Algoritam 2.1 Pretraživanje u širinu. Vremenska i prostorna složenost ovog algoritma je jednaka $O(b^d)$ gdje je b faktor grananja, a d je dubina na kojoj se nalazi rješenje. Redoslijed posjećivanja čvorova se vidi na Slika 2.3 Redoslijed posjećivanja BFS-a.

```
Funkcija BFS(G, v):
    Q := prazan red
    dodaj v u Q
    označi v
    dok Q nije prazan:
        t := dohvati prvi iz Q
        ako je t ciljni čvor:
            vrati t
        za svaki čvor o koji je susjedan čvoru v:
            ako o nije označen:
                označi o
                dodaj o na kraj reda Q
    vrati grešku
```

*Algoritam 2.1 Pretraživanje u širinu*

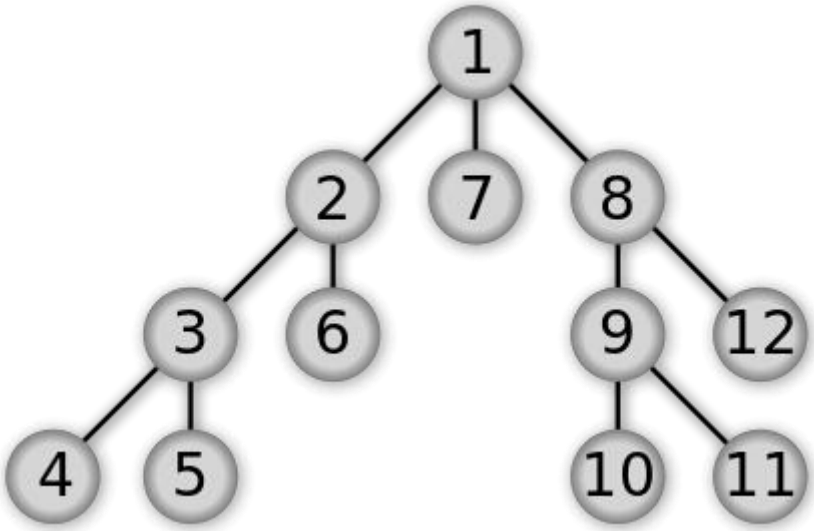



*Slika 2.3 Redoslijed posjećivanja BFS-a*

### 2.2.2 Pretraživanje u dubinu

Pretraživanje u dubinu (*eng. depth first search*) je još jedan od algoritama pretraživanja grafa. Počinje od korijena i istražuje što dublje može po nekoj grani prije nego li krene vraćati se unatrag (*eng. Backtracking*). Ovaj algoritam je predstavljen već u 19. stoljeću. Francuski matematičar Charles Pierre Trémaux ga je

koristio kao strategiju rješavanja labirinta. [4] Pretraživanje u dubinu se obavlja dodajući redom djecu trenutnog čvora na vrh stoga svaki put kada se pronađe korijen nekog pod stabla. Kod beskonačnih stabala moguće je da pretraživanje u dubinu ne pronađe rješenje. Zbog toga ovaj algoritam nije potpun. Također zbog pretraživanja grane po granu (Slika 2.4 Redoslijed posjećivanja DFS-a) može se dogoditi da algoritam pronađe ciljni čvor na većoj dubini u stablu pretraživanja od dubine do koje bi došli da smo pronašli stvarni najkraći put. Ako se algoritam zaustavi u tom trenutku onda on nije pronašao nabolje rješenje i u tom slučaju ovaj algoritam nije optimalan. Prostorna složenost je $O(bm)$ gdje je b faktor grananja, a m maksimalna dubina stabla, a vremenska složenost je $O(b^m)$. Ovaj algoritam je prikazan u Algoritam 2.2 Pretraživanje u dubinu. Problemi nepotpunosti i pronalaska optimalnog rješenja se mogu riješiti dodavanjem maksimalne dubine koja se iterativno povećava od 1 pa do dubine *d* na kojem je rješenje. Takva verzija pretraživanja u dubinu se zove iterativno pretraživanje u dubinu. Vremenska složenost je onda $O(b^d)$, a prostorna je $O(bd)$.




*Slika 2.4 Redoslijed posjećivanja DFS-a*

```
Funkcija DFS(G, v):
    S := prazan stog
    dodaj v na S
    označi v
    dok S nije prazan:
        t := dohvati prvi sa S
        ako je t ciljni čvor:
            vrati t
        za svaki čvor o koji je susjedan čvoru v:
            ako o nije označen:
                označi o
                dodaj o na vrh stoga S
    vrati grešku
```

*Algoritam 2.2 Pretraživanje u dubinu*

## 2.3 Traženje najkraćeg puta

### 2.3.1 Dijkstrin algoritam

Dijkstrin algoritam je jedan od algoritama traženja najkraćeg puta između čvorova u grafu. Taj algoritam je nizozemski znanstvenik Edsger W. Dijsktra prvi razradio 1956. i objavio tri godine kasnije. [5] Originalna verzija ovog algoritma je tražila najkraći put između dva čvora, ali češća verzija pronalazi najkraće puteve od početnog čvora do svih ostalih u grafu. Jedan od primjera korištenja je graf gdje su čvorovi gradovi a bridovi predstavljaju udaljenost između dva grada. Dijsktrinim algoritmom možemo izračunati najkraću udaljenost svih gradova od početnog grada. Zbog njegovih svojstva ovaj algoritam ima široku primjenu u protokolima za mrežno usmjeravanje. U originalu ne koristi podatkovnu strukturu prioritetni red pa je asimptotska vremenska složenost $O(V^2)$, gdje je V broj vrhova u grafu. Implementacija koja se temelji na prioritetnom redu ima tu složenost $O(E + VlogV)$. Ova implementacija Dijsktrinog algoritma je, po asimptotskoj složenosti, najbrži algoritam pronalaska najkraćeg puta za jedan izvor u grafu s usmjerenim i nenegativnim težinama bridova. Pseudokod osnovne verzije ovog algoritma je dan u Algoritam 2.3. Budući da neki grafovi mogu

biti nepoznate veličine, ne možemo sve čvorove dodati u početni skup. Ali ovo ne predstavlja nikakav problem. Može se postaviti samo jedan čvor, onaj početni, u početni skup i dodavati ih po pronalaženju. U literaturi o umjetnoj inteligenciji se ova metoda zove pretraga s jednolikom cijenom (*eng. Uniform-cost search,* UCS) i njen pseudokod se vidi u Algoritam 2.4. Vremenska asimptotska složenost ovog algoritma je $O(b^{1+\lfloor C^*/\epsilon \rfloor}$ gdje je C* duljina najkraćeg puta do završnog čvora, najmanja cijena nekog brida je $\epsilon$, a b je prosječan broj susjeda nekog vrha.

```
Funkcija Dijsktra(Graf, izvor):
    za svaki vrh v iz Graf:
        udaljenost[v] := beskonačno
        prethodni[v] := ne definirano
    udaljenost[izvor] := 0
    Q := skup svih vrhova iz Graf
    dok Q nije prazan:
        u := vrh iz Q s najmanjom vrijednosti udaljenost[u]
        za svaki susjed v od u:
            alt := udaljenost[u] + udaljenost_između(u, v)
            ako je alt < udaljenost[v]:
                udaljenost[v] := alt
                prethodni[v] := u
    vrati udaljenost[], prethodni[]
```

*Algoritam 2.3 Dijkstrin algoritam*

```
Funkcija UCS(Graf, izvor, cilj):
    trenutni_vrh := izvor
    cijena := 0
    otvoreni := prioritetni red koji sadrži samo trenutni_vrh
    zatvoreni := prazni skup
    dok otvoreni nije prazan:
        trenutni_vrh := prvi vrh iz otvoreni
        ako je trenutni_vrh cilj:
            vrati rješenje
        za svaki susjed v od trenutni_vrh:
            ako v nije u otvoreni:
                dodaj vrh v u otvoreni
                dodaj vrh v u zatvoreni
    vrati grešku
```

*Algoritam 2.4 Pretraživanje s jednolikom cijenom*

### 2.3.2 A* ALGORITAM

A* (*eng. A star)* je jako popularan algoritam koji se često koristi u pronalaženju najkraće putanje i šetnjama po grafu. Razlog zbog kojeg je popularan je zbog svoje brzine i točnosti. Ovaj algoritam je nastao 1968. godine na Stanford Reasearch Institute kao poboljšanje Dijsktrinog algoritma. Istraživač umjetne inteligencije Nils Nilsson je 1968. pokušavao poboljšati planiranje puta svog robota, Shakey the Robot. Taj robot je mogao prolaziti kroz prostoriju sa zamkama. Nillson je prvu verziju poboljšanog Dijstrinog algoritma nazvao A1. Bertram Raphael je predložio neke izmjene nakon koje su značajno poboljšale taj algoritam. Novu verziju su nazvali A2. Nakon toga je Peter E. Hart iznio argumente, nakon manjih izmjena A2 algoritma, koji su tvrdili da je to najbolji mogući algoritma za traženje najkraćih puteva. Ovi znanstvenici su zajedničkim radom došli do dokaza koji je potvrdio da je njihov algoritam optimalan pod točno određenim uvjetima. Značajan dio poboljšanja je zbog boljeg određivanja redoslijeda promatranja različitih puteva. A star je algoritam koji funkcionira na principu najbolji prvi. Promatraju se svi mogući putevi gdje se redoslijed određuje tako da se promatra onaj za koji nam se čini da bi imao najmanju

cijenu. Počevši od početnog čvora na grafu stvara se stablo putanja koja se proširuju korak po korak. Određivanje koji će se sljedeći put proširiti je određeno funkcijom $f(n) = g(n) + h(n)$ gdje je g(n) cijena puta do čvora n, a h(n) heuristika kojom procjenjujemo cijenu od čvora n do cilja. Koju ćemo heuristiku koristiti ovisi o konkretnom problemu koji rješavamo. Da bi algoritam bio optimalan heuristika mora biti optimistična. Optimistična heuristika je ona heuristika koja nikad ne precjenjuje stvarnu cijenu od zadanog čvora pa do najbližeg ciljnog čvora.

```
Funkcija A*(Graf, izvor, cilj):
    trenutni_vrh := izvor
    cijena := 0
    otvoreni := prioritetni red koji sadrži samo trenutni_vrh
    zatvoreni := prazni skup
    dok otvoreni nije prazan:
        trenutni_vrh := prvi vrh iz otvoreni
        ako je trenutni_vrh cilj:
            vrati rješenje
        za svaki susjed v od trenutni_vrh:
            ako v nije u otvoreni:
                dodaj vrh v u otvoreni po prioritetu f(n)
                dodaj vrh v u zatvoreni
    vrati grešku
```

## 2.4 Prirodom inspirirani algoritmi

Zbog velike vremenske složenosti, deterministički se algoritmi rijetko koriste za rješavanje problema s većim brojem čvorova u grafu. U tim slučajevima potrebno je posegnuti za drukčijim načinom rješavanja. Ako problem traženja najkraćeg puta modeliramo kao funkciju koja prima put a vraća ukupnu udaljenost tog puta onda možemo tu funkciju minimizirati. Prirodom inspirirani algoritmi se često koriste za rješavanje optimizacijskih problema. Optimizacijski problemu su definirani pomoću funkcije dobrote / kazne koju je potrebno maksimizirati / minimizirati i ograničenja koja moraju biti zadovoljena za sva prihvatljiva rješenja. Funkcija koju optimiziramo prima vektor odlučujućih varijabli koje također mogu biti ograničene na nekom

prostoru. Svi algoritmi se bave pretraživanjem prostora mogućih rješenja. Kreću od jednog ili više početnih rješenja i nakon toga, pomoću trenutnih ili nekakvog njihovog utjecaja, se generiraju nova rješenja. Dobri algoritmi najčešće imaju dvije faze: grubu i finu pretragu. Gruba pretraga je prva faza u kojoj algoritam nasumično bira rješenja kako bi pronašao podprostor koji bi mogao sadržavati neko dobro rješenje. Nakon što se takvo rješenje pronađe kreće druga faza. Fina pretraga služi da bi se istražila okolica pronađenog podprostora da bi se pronašao globalni optimum. Više o različitim algoritmima i detaljnije o svim fazama pretrage se može pročitati u raznoj literaturi. [6]

### 2.4.1 Genetski algoritam

Inspiracija za genetski algoritam došla je iz Darwinove teorije o postanku vrsta. U njoj imamo nekoliko postavki koje nam govore kako će u svakoj populaciji postojati borba za preživljavanje. Ako potomaka ima više nego što je potrebno, a količina hrane je ograničena, neće sve jedinke preživjeti. Zbog toga će bolje i jače jedinke imati šansu preživjeti i dobiti priliku stvarati potomke. Takvi potomci su u velikoj mjeri slični svojim genetski materijalom svojim roditeljima, ali nisu identični jer postoji odstupanje uzrokovano mutacijom. Genetski algoritam se služi ovim principima da bi pronašao optimalno rješenje. U početku se bira populacija jedinki. Svaka jedinka je potencijalno rješenje zadanog problema. Izračunavanjem funkcije koju optimiziramo u točki koju ta jedinka predstavlja možemo dobiti dobrotu (*eng. fitness*). Ako rješavamo problem maksimizacije onda nam dobrota mora biti što veća. Operatorom selekcije biramo jedinke iz populacije koji će postati roditelji. Roditelji pomoću operatora križanja stvaraju svoje potomke nad kojima opet djeluje operator mutacije koji dodatno mijenja njihov genetski materijal. Na kraju operatorom zamjene djeca se dodaju u populaciju i tako završava jedan ciklus rada algoritma.

### 2.4.2 Mravlji algoritmi

Promatranjem mrava uočeno je kako oni, iako su jako jednostavna bića, zahvaljujući socijalnim interakcijama postižu iznimne rezultate. Zajedničkim radom mogu pronaći izvor hrane i prateći jedan drugog u nizu mogu donijeti tu hranu natrag do mravinjaka. Mravi su sposobni uvijek pronaći najkraći put između hrane i mravinjaka.

Zanimljivo je da prilikom kretanja ne koriste osjet vida, već je njihovo kretanje određeno interakcijom s ostalim mravima. Na putu od mravinjaka do hrane, svaki mrav ostavlja kemijski trag nazvan feromon. Mravi imaju razvijen osjet feromona i pomoću njega se orijentiraju. Ovisno o jačini feromonskog traga, mrav će donijeti odluku kojim će se smjerom kretati. Denebourg i njegovi kolege [7] su postavili eksperiment u kojem su ispitivali kretnje mrava kroz dvokrake mostove. Prvi eksperiment je uključivao dva mosta koja su bila jednako duga. Mravi su na početku nasumično birali kojim će mostom doći do hrane. Što je više vrijeme odmicalo to su mravi sve više birali isti most. Ponavljajući pokus uvijek bi na kraju, s jednakom vjerojatnošću, odabrali jedan ili drugi most kojim bi se svi mravi kretali. U drugom testu je jedan most bio kraći. Na početku su mravi nasumično birali mostove, ali bi na kraju uvijek svi odabirali onaj kraći. Iz ovih pokusa se dalo zaključiti da mravi iza sebe ostavljaju feromonski trag. Ako jedan mrav slučajno krene nekim mostom, iza sebe će ostaviti feromon. Zatim će zbog tog traga drugi mravi više skretati na taj most, a na drugom mostu će feromonski trag sve više isparavati dok svi mravi krenu prelaziti preko istog mosta. Promatrajući ponašanje mrava na makroskopskoj razini vidimo da oni pronalaze najkraći put od mravinjaka do hrane. Međutim, to ponašanje je određeno na mikroskopskoj razini. Mravi jednostavnom socijalnom interakcijom, bez ikakvih informacija o širem svijetu, uspijevaju riješiti složeniji problem. Algoritam jednostavnog mravljeg algoritma je prikazan u Algoritam 2.5 Jednostavni mravlji algoritam

```
Dok nije kraj:
    ponovi za svakog mrava:
        stvori rješenje
        ocijeni rješenje
    odaberi podskup mrava
    ponovi za odabrane mrave:
        ažuriraj feromonske tragove
    ispari feromonske tragove
```

*Algoritam 2.5 Jednostavni mravlji algoritam*

# 3. Pristup zadatku i modeli

## 3.1 Simulacija okruženja

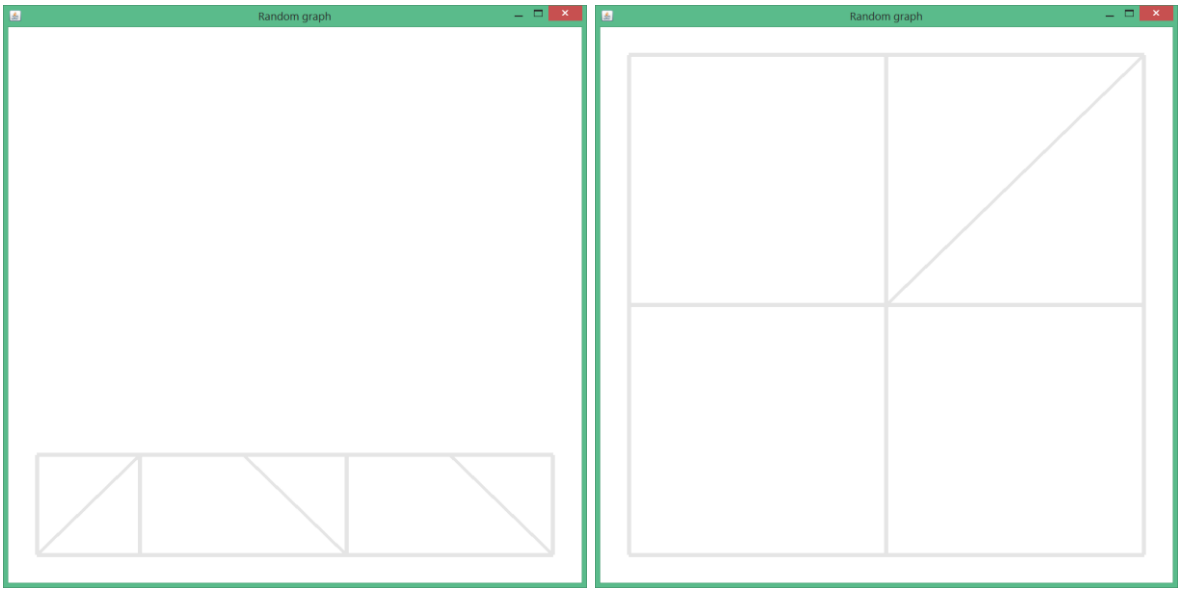
Za potrebe je izračuna bilo potrebno definirati opis svijeta u kojem bi se odvijalo gibanje zadanih objekata. Za objekte su odabrana vozila koja se gibaju kroz predefiniranu prometnu mrežu. Tu mrežu predstavlja neusmjereni težinski graf gdje su križanja čvorovi a prometnice između njih bridovi. Bridovi imaju svojstvo kapaciteta. Kapacitet je postotak prohodnosti određene prometnice, odnosno brida. Što je ta vrijednost manja to je prometnica manje prohodna. Vrijednost je definirana u skupu realnih brojeva od 0,001 do 1,0. Kapacitet brida mijenja se ovisno o količini vozila koji bi se trebali gibati preko tog brida. Kod dodavanja novog vozila kapacitet se postavlja na vrijednost manju za 10% od stare, to jest $kapacitet = maksimum(kapacitet * 0.9, 0.001)$. Nakon što vozilo prijeđe taj brid kapacitet postaje $\text{kapacitet} = \text{minimum}(\text{kapacitet} / 0{,}9\,, 1{,}0)$. Bridovi imaju i informaciju o težini, ali radi pojednostavljivanja sustava u svim primjerima su susjedni vrhovi međusobno udaljeni za 1, pa su i sami bridovi postavljeni na 1. U ovom radu su korišteni nasumično generirani grafovi. Parametri pri stvaranju prometne mreže su dimenzije pravokutnika nad kojim bi se postavile prometnice i križanja. Za cjelobrojnu mrežu definiranu zadanim dimenzijama bira se  hoće li postojati vrh na zadanim koordinatama. Zatim se uzimaju svi mogući susjedi i onda se nasumično određuje hoće li odabrani vrh biti povezan sa svojim susjedom. Vjerojatnosti odabrane za generiranje su redom 90% i 30%. U svrhe testiranja stvorena su tri grafa koja možemo vidjeti na slikama Slika 3.1 Testovi Test0 i Test1, Slika 3.2 Test2.

Simulatoru kretanja vozila kroz prometnu mrežu predaje se graf i moguće je dodavati nova vozila te niz bridova koji čine put od početne do završne točke. Simuliranje kretanja je izvedeno preko vremenskih otkucaja (*eng. Tick*) u kojima vozilo pokušava prijeći preko jednog brida. Udaljenost koju vozilo može prijeći u jednom otkucaju je jednako težini brida pomnoženo s trenutnim kapacitetom tog brida, Ukoliko vozilo ne može prijeći preko brida njemu se pridodaje vrijednost *delta* koja označava koliki je put po tom bridu prešlo to vozilo. Kada zbroj *delte* i udaljenost koju vozilo može

prijeći u tom otkucaju postane veća od težine tog brida vozilo prelazi iz početnog čvora u sljedeći. Pseudokod funkcije koraka u simulatoru je definirana u Algoritam 3.1 Korak simulatora

```
funkcija korak():
    prijeđeni_bridovi = ø
    prošla_pozicija = {}
    za vozilo v iz svih vozila:
        ako prošla_pozicija od v nije jednaka trenutna pozicija od v:
            smanji kapacitet sljedećeg brida na putu vozila v
    za vozilo v iz svih vozila:
        ako vozilo v nema sljedeći brid:
            preskoči
        dodaj 1 na ukupno vrijeme vozila v
        preljev := delta + težina brida * kapacitet brida – težina brida
        ako je preljev veći od 0:
            dodaj sljedeći brid u prijeđeni_bridovi
            ažuriraj lokaciju vozila v
            ukloni sljedeći brid iz puta vozila v
            ažuriraj deltu vozila v
        inače:
            delta := delta + težina brida * kapacitet brida
    za svaki brid b iz prijeđeni_bridovi:
        povećaj kapacitet brida b
```

*Algoritam 3.1 Korak simulatora*




*Slika 3.1 Testovi Test0 i Test1*


*Slika 3.2 Test2*

## 3.2 Korišteni algoritmi optimizacije

Problem koji se rješava u ovom radu je problem pronalaska najkraćeg puta za više vozila koja istovremeno putuju po zajedničkom grafu. Potrebno je za određeni početni i završni čvor, uz trenutno stanje grafa, pronaći najkraći put između njih. Metrika udaljenosti vremensko je trajanje prelaska puta. Fokus problema jest na pronalaženju puta koji izbjegava prometnice koje imaju malu prohodnost zbog velike količine vozila koji žele proći uzimajući u obzir da zaobilazni put nije previše dug. Zbog oblika problema odabran je mravlji algoritam, točnije algoritam *Ant System*. Na tragu radova [8] i [9] neki su dijelovi osnovnog algoritma izmijenjeni. Postavljanje početne količine feromona određeno je prema putu koji se dobije A star pretragom za početni i ciljni čvor. Po tom putu se postavljaju veće početne razine feromona. Ovakav je način postavljanja feromona odabran da bi mravi brže i lakše pronalazili puteve. Usput se javlja problem pristranosti (*eng. bias*) prema putu dobivenom A star algoritmom. Ovisno o tome koliko želimo da putevi budu sličniji najkraćem putu toliko možemo povećavati ili smanjivati taj utjecaj. Na ostalim čvorovima razina feromona je postavljena na predodređenu vrijednost. Odabrana početna vrijednost iznosi 1,0. Prilikom odlučivanja koji će se sljedeći čvor posjetiti korišteno je slučajno proporcionalno pravilo (*eng. Random proportional rule*) koje uključuje jakost feromonskog traga i vrijednost heurističke funkcije. Heuristička funkcija je broj bridova pronađen A star algoritmom od sljedećeg čvora do cilja. Formula za izračun vjerojatnosti je određena sljedećim izrazom:

$$p_{ij} = \begin{cases} \frac{\tau_j^{\alpha} \cdot \eta_j^{\beta}}{\sum_{l \epsilon N_i} \tau_l^{\alpha} \cdot \eta_l^{\beta}}, & \text{ako je } j \epsilon N_i \\ 0, \text{inače} \end{cases} \tag{3.1}$$

Fermononski trag na čvoru $j$ predstavljen je s $\tau_j$. Skup $N_i$ označava skup svih indeksa svih čvorova u koje je moguće prijeći iz čvora $i$. Parametri $\alpha$ i $\beta$ postavljeni su redom na 1,2 i 1,0. Postavljen je i faktor isparavanja feromona i on iznosi 0,1%. U svakoj se iteraciji simulira šetnja jednog mrava što se vidi iz Algoritam 3.2 Ant System. Kod ažuriranja feromona prvo se ispari dio svih feromona i onda se, ako je mrav stigao do cilja, dodaje dodatan feromonski trag na sve čvorove na putu od

početnog do ciljnog čvora. Zbog potrebe izvođenja simulacije prilikom šetnje svakog mrava broj iteracija je postavljen na 1000.

```
Funkcija pronađi_put(simulator, izvor, cilj):
    postavi početne razine feromona
    pojačaj razine na putu dobivenom A* pretragom od izvora do cilja
    najbolji_put := ø
    ponovi BROJ_ITERACIJA:
        put := pronađi_put_za_mrava(simulator, izvor, cilj)
        ažuriraj_feromone(put)
        ako je put bolji od najboljeg:
            najbolji_put := put
        inače:
    vrati najbolji_put
```

*Algoritam 3.2 Ant System - v1*

U prvoj verziji algoritma optimizacije, početna količina feromona koja bi se tako postavila bila je konstantna. U drugoj verziji primijenjen je drukčiji način raspoređivanja početnih feromonskih tragova. Sama količina feromona linearno ovisi o duljini cijelog puta i udaljenosti od početnog čvora. Što je čvor bliže početnom čvoru to je količina feromona koja je ostavljena pri inicijalizaciji manja. To načelo je odabrano da bi se potaklo istraživanje u ranijim fazama i lakše pronalaženje cilja kasnije. U drugoj verziji mravljeg algoritma također je dodano ponovno postavljanje feromona ako za određeni prag broja iteracija nije pronašlo bolje rješenje. Te promjene su vidljive u Algoritam 3.3 Ant System - v2.

```
Funkcija pronađi_put(simulator, izvor, cilj):
    postavi početne razine feromona
    pojačaj razine na putu dobivenom A* pretragom od izvora do cilja
    brojač_istog_rješenja := 0
    najbolji_put := ø
    ponovi BROJ_ITERACIJA:
        put := pronađi_put_za_mrava(simulator, izvor, cilj)
        ažuriraj_feromone(put)
        ako rješenje nije bolje:
            brojač_istog_rješenja++
        inače:
            brojač_istog_rješenja := 0
            najbolji_put := put
        ako je brojač_istog_rješenja >= PRAG_RESETIRANJA:
            ponovno postavi razine feromona
    vrati najbolji_put
```

*Algoritam 3.3 Ant System - v2*

Zbog uspoređivanja rezultata korištena je još jedna verzija konačnog algoritma. U toj verziji nije korišten put koji je dobiven A star algoritmom, već su sve vrijednosti iste.

```
Funkcija pronađi_put(simulator, izvor, cilj):
    postavi početne razine feromona
    brojač_istog_rješenja := 0
    najbolji_put := ø
    ponovi BROJ_ITERACIJA:
        put := pronađi_put_za_mrava(simulator, izvor, cilj)
        ažuriraj_feromone(put)
        ako rješenje nije bolje:
            brojač_istog_rješenja++
        inače:
            brojač_istog_rješenja := 0
            najbolji_put := put
        ako je brojač_istog_rješenja >= PRAG_RESETIRANJA:
            ponovno postavi razine feromona
    vrati najbolji_put
```

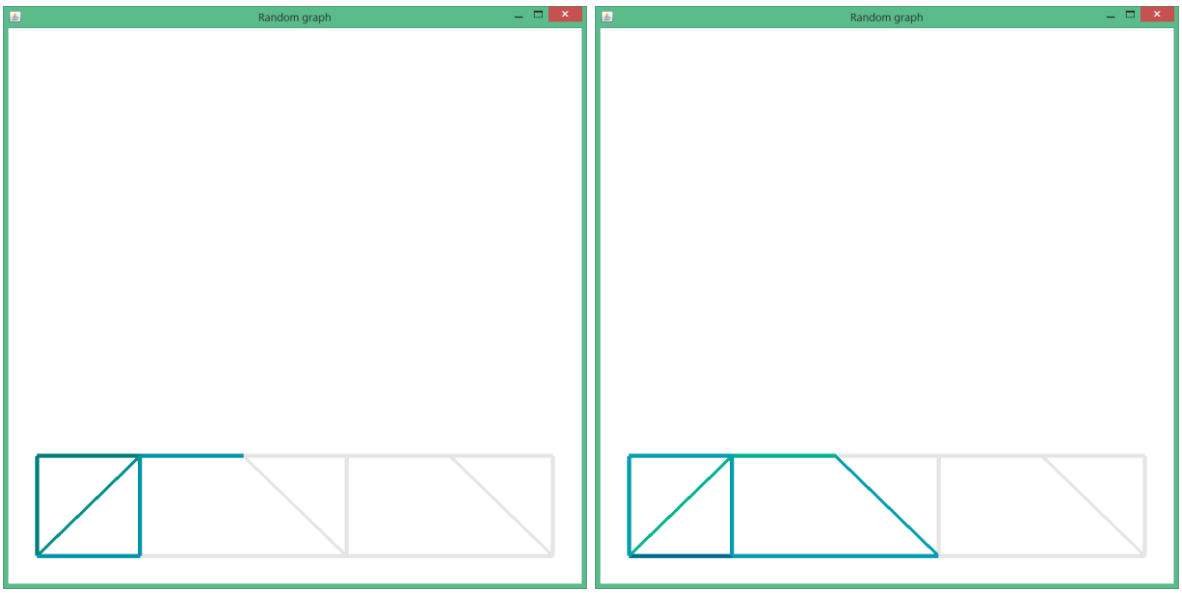
*Algoritam 3.4 Ant System - noastar*

# 4. Rezultati

Algoritam je testiran na tri nasumično složena grafa, navedena u prošlim poglavljima. Pripremljena su četiri testa. Testovi su provedeni na način da je pripremljen određen broj upita za najkraćim putem te su oni po redu izračunati i ubačeni u sustav. Za svako novo vozilo se izračunavao najkraći put do cilja samo jedan put, prilikom dobivanja zahtjeva. Na taj način je simulirano odabiranje putanje korištenjem navigacijskog sustava pritom izbjegavajući potencijalne zastoje u prometu. U svim slučajevima je za izračunavanje ovog podatka bilo potrebno manje od jedne sekunde. Prilikom razvijanja rješenja kao osnova se koristilo pohlepno rješenje u kojem se uvijek bira put dobiven A star heuristikom zanemarujući ostala vozila i njihove putanje.

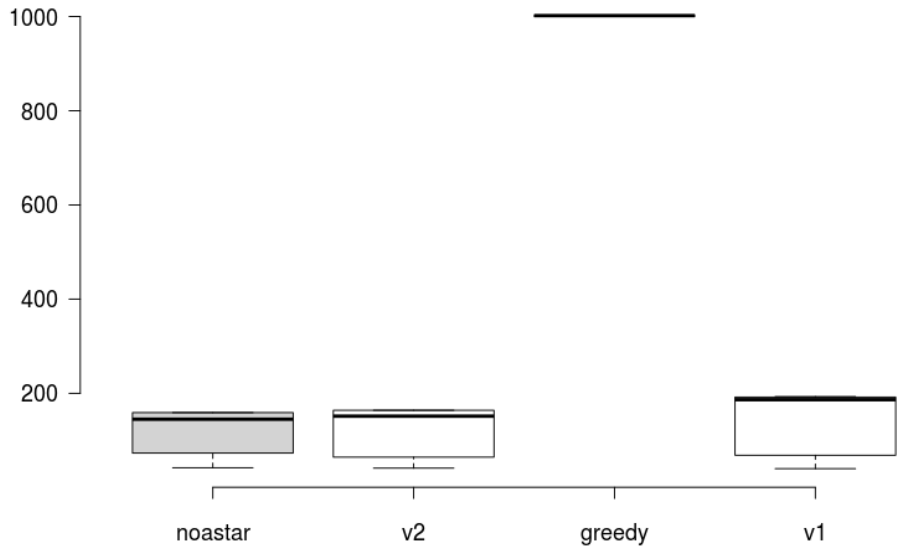
## 4.1 Test zaobilaženja duljim putevima

U prvom testu se traži put za 100 vozila koji počinju u (0,0) i trebaju doći do cilja u (2,1). Ovaj test ispituje ponašanje algoritma kad su putevi koji zaobilaze zagušene bridove puno dulji od najkraćeg puta.

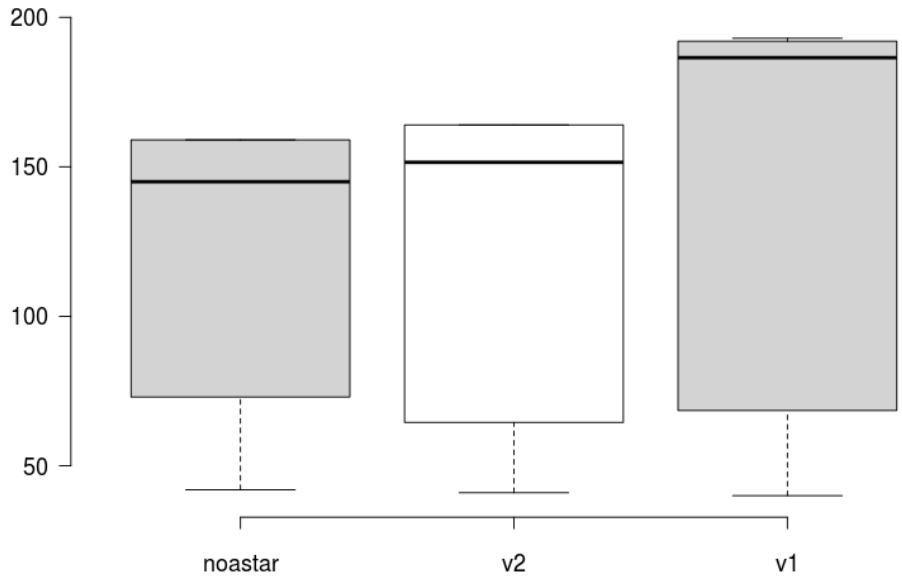



*Slika 4.1 Test0 - generiranje putanja*

Za prva vozila se odabiru putevi koji su najkraći uz minimalne pomake od najkraćeg puta. Tek nakon što se odredi putanja za nekoliko vozila se počinju predlagati putevi za koje treba proći dvostruko veću udaljenost kao što se vidi na Slika 4.1 Test0 - generiranje putanja. Nakon toga se jednoliko raspoređuju vozila po dostupnim putevima kako bi olakšali prohodnost cjelokupne prometne mreže. Na slikama su prikazan kutijasti dijagrami (*eng. Box plot*). Algoritmi koji su se koristili su napisani na osi *x,* a na osi *y* su brojevi koraka u simulatoru potrebnih za prelazak vozila. Na Slika 4.2 Test0 možemo vidjeti koliko je pohlepno rješenje lošije od ostalih rješenja. Zbog toga što sva vozila prolaze preko dva ista brida razlika između rješenja je poprilična. Slika 4.3 Test0 – varijante mravljeg algoritma predstavlja različite verzije mravljeg algoritma koje su se koristile na prvom testu. Verzija *v1* predstavlja prvu inačicu mravljeg algoritma danu u Algoritam 3.2 Ant System - v1, *v2* predstavlja njeno poboljšanje, Algoritam 3.3 Ant System - v2, a *noastar* predstavlja rješenje kod kojeg su početne razine feromona sve jednake opisano u Algoritam 3.4 Ant System - noastar. Jasno se vidi napredak iz prve verzije do druge. Utjecaj pristranosti se smanjio pa je rješenje dobiveno drugom verzijom mravljeg algoritma približno jednako onom bez postavljanja pristranosti.



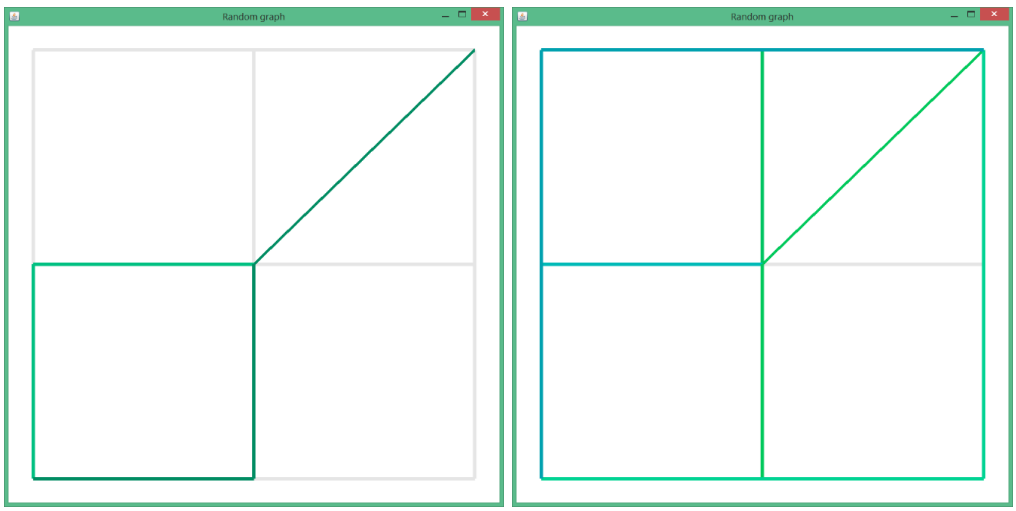

*Slika 4.2 Test0*




*Slika 4.3 Test0 – varijante mravljeg algoritma*

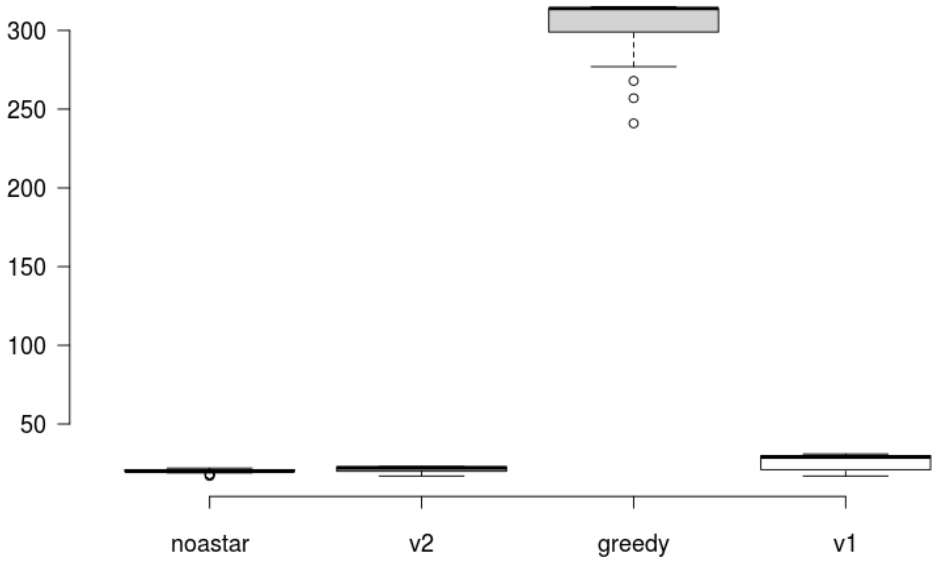
## 4.2 Test alterniranja sličnih puteva

U drugom testu se ispituju mogućnosti traženja različitih puteva. Traži se put za 50 vozila od donjeg lijevog do gornjeg desnog kuta. Za prva vozila nije potrebno previše tražiti različite puteve pa se mijenjaju odabrani putevi između dva najkraća. Kasnije se bira između različitih puteva da bi se mreža iskoristila u potpunosti.

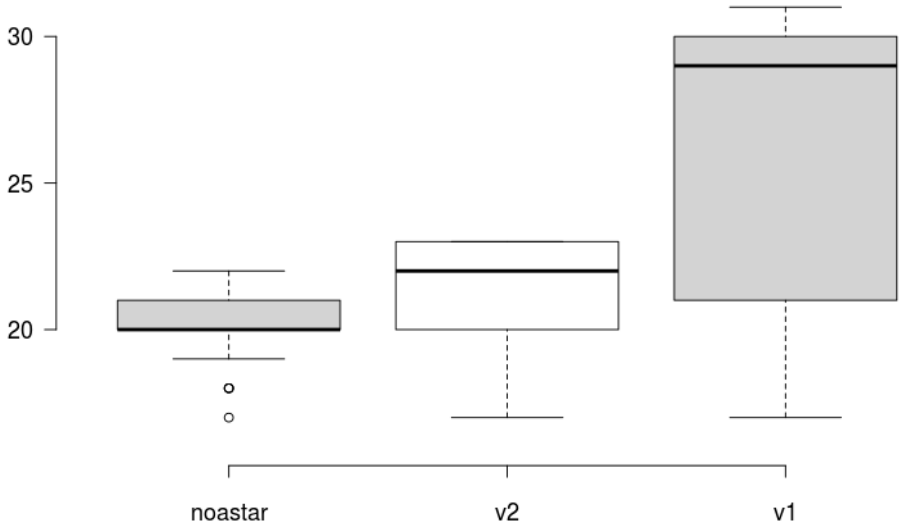


*Slika 4.4 Test1 - generiranje putanja*

Zbog pristranosti prema najkraćem putu ne pronalaze se najbolja rješenja i ne koriste se svi mogući bridovi. Zbog toga se vidi razlika između verzije *v2* i *noastar*. Na slikama Slika 4.5 Test1 i Slika 4.6 Test1 - varijante mravljeg algoritma vidimo da ta razlika nije toliko značajna, ali ipak postoji.



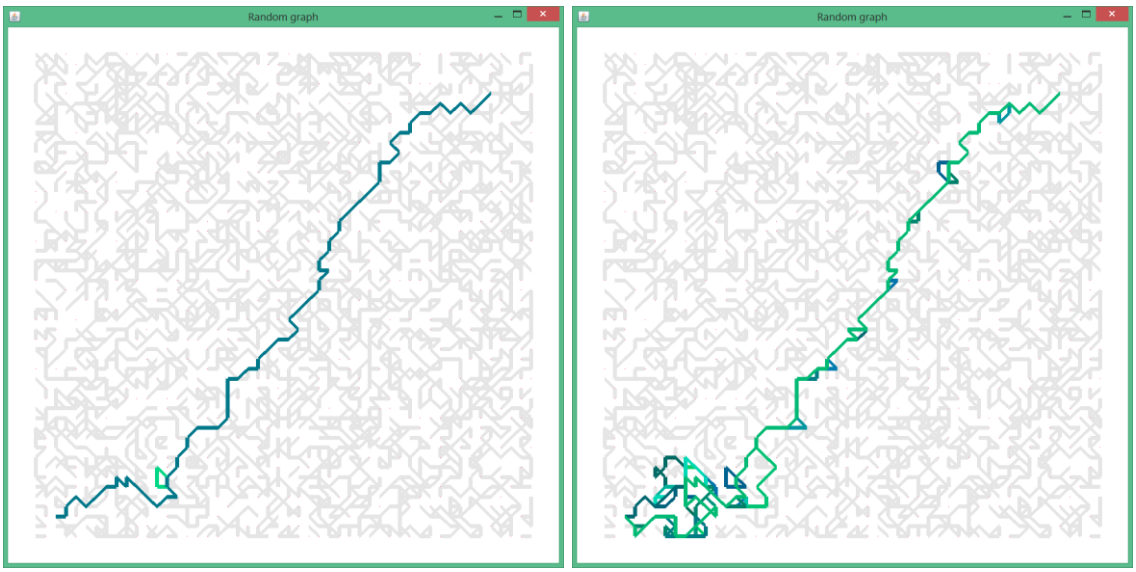

*Slika 4.5 Test1*




*Slika 4.6 Test1 - varijante mravljeg algoritma*

## 4.3 Test na većoj mreži

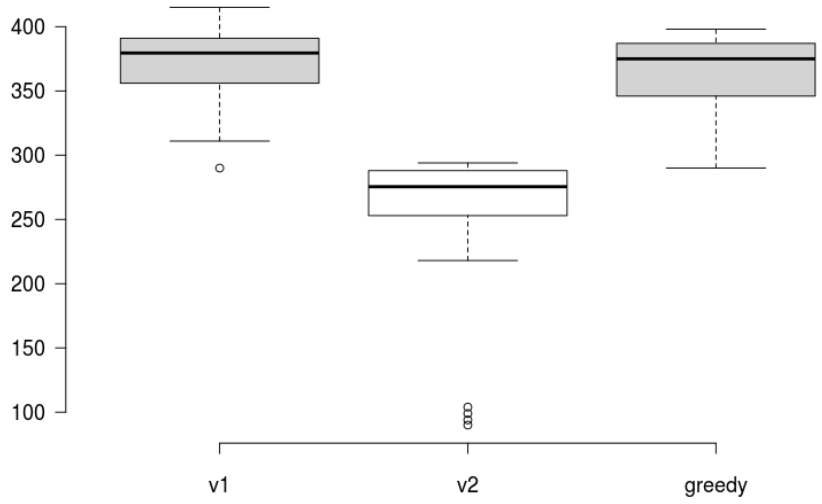
U trećem testu promatramo veći graf. Ovaj graf sadrži preko dvije tisuće čvorova. Zbog njegove veličine, izazov je pronaći rješenje u takvom grafu. Traži se put od donjeg lijevog kuta do gornjeg desnog za 50 vozila. Budući da je jedan od ciljeva brzo izvršavanje potrebno je bilo pronaći dovoljno mali broj iteracija mravljeg algoritma kojim bi dobili dobro rješenje. Baš zbog ovakvih slučajeva je uključeno postavljanje pristranosti prema najkraćem rješenju. S brojem iteracija koji je odabran, mravlji algoritam u većini slučajeva nije mogao pronaći put od početne do ciljne točke bez dodavanja pristranosti putu dobivenom *A star* algoritmom. Zbog toga se na Slika 4.8 Test2 i ne prikazuje *noastar* verzija. Slika 4.7 Test2 - generiranje putanja predstavlja traženje putanja na ovom grafu. Lijeva slika predstavlja putanje za prvih par vozila, a desna konačne putanje za svih 50 vozila.




*Slika 4.7 Test2 - generiranje putanja*

Kao i kod ostalih testova u početku su rješenja slična najkraćem putu. Ali u ovom slučaju se dobro vidi da je pristranost takvom rješenju slabija što je čvor bliže početku. Zbog takvog ponašanja se stvara efekt zatvarača. Vozila se u početku gibaju različitim putevima i tek se na kraju spajaju u jednu putanju. U prometu je ovo dobro poznati efekt gdje se više traka kasnije na prometnici spaja u jednu i tako

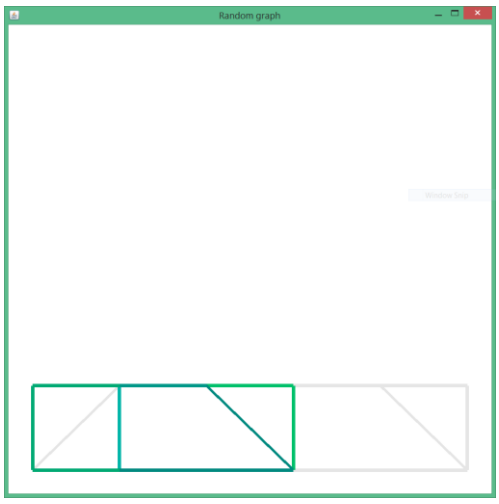
omogućava veći tok prometa jer se obje trake podjednako koriste. Zbog svih ovih poboljšanja se vidi popriličan napredak od prvih verzija mravljeg algoritma.




*Slika 4.8 Test2*

## 4.4 Test prekida u mreži

Jedan od testova je bilo i prekidanje veze između dva čvora i ponovno izračunavanje putanja za slučaj prikazan na Slika 4.9 Test3. Nakon odabranog broja koraka u simulatoru uklanja se brid između donjeg lijevog i njemu dijagonalo desnog čvora. Brzo se izračunavaju nove putanje i izbjegava se taj brid.




*Slika 4.9 Test3*

## 4.5 Pregled testova

U prvoj verziji *Ant System* algoritma korišteno je samo postavljanje početne vrijednosti feromonskih tragova prema rezultatu A star algoritma i slučajno proporcionalno pravilo. Takav pristup je davao bolja rješenja od pohlepnog pristupa, ali je teže pronalazio različite puteve pa bi sva vozila bila usmjerena na one puteve koji bi bili dobiveni *A star* algoritmom. Zbog toga se nije iskorištavala ukupna prohodnost mreže. Medijan je uvijek bio jako blizu trećeg kvartila, što znači da je većina vozila putovala sličnim i vremenski duljim putem, a manji broj se gibao manje zagušenim dijelovima prometne mreže. Dodavanjem ponovnog postavljanja feromona nakon određenog broja iteracija bez boljeg rješenja i promjenom rasporeda dodavanja feromona kod inicijalizacije vremena koja su trebala vozilima da stignu do cilja su se smanjila. Čak je i medijan potrebnog broja koraka u simulatoru bio puno bliže sredini razdiobe. Također je i razmak između drugog i trećeg kvartila bio manji. Sve to znači da su vozila stizala brže i za otprilike jednaku količinu vremena što također bio jedan od početnih ciljeva.

# 5. Zaključak

Problem određivanja najkraćeg puta u dinamičkim okruženjima, kao što je cestovni promet, je jako složen, ali rješiv. Prvi problem bilo je postaviti simulator koji bi vjerodostojno opisivao takvo okruženje. Zbog kompleksnosti stvarnog svijeta, bilo je potrebno minimizirati broj čimbenika. Simulator je na kraju sadržavao samo težinski neusmjereni graf gdje su težine bile kapaciteti prometnica, a čvorovi su bili križanja. Simulator je osim toga imao i informaciju o vozilima koja su se kretala tom mrežom. Takva je reprezentacija bila sasvim dovoljna za opis stvarnog svijeta u ovom radu.

Kod jako velikih sustava, s mnogo čvorova, bridova i vozila, javljaju se dodatni problemi poput brzine izračunavanja. Zbog toga je jako bitno odabrati prikladan algoritam. Budući da se kao primjer u ovom radu koristila prometna mreža, bilo je potrebno odabrati algoritam koji je u stanju brzo izračunavati putove kojima bi se vozila kretala. Zato su odabrani mravlji algoritmi i A* kao izvor heuristika. Dodatnim popravljanjem algoritma i korištenjem već poznatih pravila za ubrzavanje prometnog toka poboljšala su se rješenja. Na prva dva ispitna slučaja rezultati su bili očekivani. Algoritam je dobro radio, ali se ipak vidio utjecaj pristranosti koji je postavljen prilikom inicijalizacije feromona. Rješenja su manje odstupala od najkraćeg puta za razliku od verzije bez pristranosti. Poboljšavanjem algoritma i boljim početnim postavljanjem se uspjelo postići rješenje koje je približno jednako po rezultatima bez pristranosti.

Treći test sastojao se od značajno veće prometne mreže. To je predstavljalo veći izazov jer je mnogo teže bilo pronaći puteve u takvom sustavu. Prva verzija je teško pronalazila puteve koji su različiti od onog najkraćeg po udaljenosti, a verzija bez pristranosti uopće nije davala ikakve rezultate. Tek se u drugoj iteraciji mravljeg sustava vidio napredak. Ta rješenja su bila iznenađujuće bolja od prethodnih. Algoritam je pronalazio različite puteve i raspoređivao vozila po njima na način da su svi puno brže stizali do odredišta uz male razlike u trajanju samog puta. Kad se svi ovi rezultati uzmu u obzir, iako se utjecaj pristranosti može smanjiti, još uvijek malo utječe na rezultat. Taj je utjecaj vidljiv na manjim grafovima dok je na većim grafovima zanemariv i ubrzava dolazak do rješenja.

# 6. Literatura


[1] Uvod u teoriju grafova, Anamari Nakić, Mario-Osvin Pavčević, Element, 2015.

[2] Zuse, Konrad (1972), Der Plankalkül

[3] Moore, Edward F. (1959). "The shortest path through a maze". Proceedings of the International Symposium on the Theory of Switching. Harvard University Press. pp. 285–292.

[4] Charles Pierre Trémaux (1859–1882) École polytechnique of Paris (X:1876), French engineer of the telegraph in Public conference

[5] Numerische Mathematlk l, 269 - 27 I (l 959), Edsger W: Dijsktra, 1959.

[6] Marko Čupić: “Prirodom inspirirani optimizacijski algoritmi”, Sveučilište u Zagrebu, prosinac 2013.

[7] J. L. Denebourg, S. Aron, S. Goss, and J. M. Pasteels. The self-organizing exploratory pattern of the argentine ant. Journal of Insect Behaviour, 3:159168, 1990.

[8] M. Eghbali and M. A. Sharbafi, "Multi agent routing to multi targets via ant colony," 2010 The 2nd International Conference on Computer and Automation Engineering (ICCAE), Singapore, 2010, pp. 587-591. doi: 10.1109/ICCAE.2010.5451346

[9] I. Sabbani, M. Youssfi and O. Bouattane, "A multi-agent based on ant colony model for urban traffic management," 2016 5th International Conference on Multimedia Computing and Systems (ICMCS), Marrakech, 2016, pp. 793-798. doi: 10.1109/ICMCS.2016.7905551

# 7. Sažetak

Tema ovog rada je “Optimizacija pronalaženja najkraćeg puta uz dinamičko okruženje“. Problem je predstavljen u svijetu prometnih mreža i vozila koja se koja se njima gibaju. Prometna mreža je predstavljena grafom, a gužve su predstavljene kapacitetom kao težinama na bridovima. Pristup koji je odabran se sastoji od dviju komponenti. Prva komponenta je A* algoritam kojim se traži najkraći put u smislu prostorne udaljenosti između početne i krajnje točke. Drugi dio rješenja čini mravlji algoritam kojim to rješenje optimiziramo. Staza koja se dobije A* algoritmom služi za postavljanje početnog feromonskog traga gdje je trag jači što je taj čvor bliže cilju. Time je postignut efekt zatvarača koji se koristi i u stvarnim prometnim mrežama. Dobiveni rezultati su bili očekivani. Rješenja dobivena mravljim algoritmima su bila bolja od onih dobivenih samo A* algoritmom. Na manjim grafovima je dolazilo do utjecaja pristranosti najkraćem rješenju, ali je kod većih grafova bilo potrebno da bi se brzo pronašao neki put. Daljnjim iteracijama mravljeg algoritma, uspio se smanjiti taj utjecaj do zanemarive razine. Takvi rezultati su se na kraju pokazali dobrima i vidjelo se znatno poboljšanje od pohlepnog rješenja koje uzima u obzir samo najkraću udaljenost.

**Ključne riječi:** Pronalaženje puta; A*; evolucijsko računarstvo; inteligencija rojeva; optimizacija mravljim kolonijama; mravlji sustav; optimizacije prometa

# 8. Abstract

Subject of this thesis is „Path finding optimization in a dynamic environment“. The problem is presented in a traffic network environment with vehicles as object which are moving on said network. Traffic network is modeled with a graph and traffic jams are represented by capacities as weights on the edges of the graph. The chosen approach can be described with two components. The first component is A* algorithm that is used to find the shortest path, considering only space distance between the starting and the ending point. The second part is an ant algorithm which further optimizes given path. The path that is calculated with A* is used to set the initial pheromone trace where the closer the vertex is to the goal the higher the pheromone value will be. This achieved so called zipper effect, where two or more lanes merge at a later point, improving traffic flow. Results achieved were expected. Solutions calculated by ant algorithm were better then those calculated by only A* algorithm. There was some bias towards the shortest path on smaller graph which caused somewhat worse solutions, but on much bigger graphs that same bias helped to speed up ant algorithm search time. With further improving the optimization algorithm, negative effect on smaller graph was reduced to dismissably small levels. Such results have been shown to be satisfactory in the end and there was a significant improvement on the greedy algorithm where solutions were based only on the shortest space distance.

**Key words:** Path finding; A* algorithm; Evolutionary computing; Swarm intelligence; Ant colony optimizations; Ant system; Traffic optimizations