Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Simon Ivanšek

Vizualizacija optimizacije prometa z genetskimi algoritmi

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: prof. dr. Marko Robnik Šikonja

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Simon Ivanšek, z vpisno številko **63080273**, sem avtor diplomskega dela z naslovom:

Vizualizacija optimizacije prometa z genetskimi algoritmi

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom prof. dr. Marka Robnik Šikonje
- so elektronska oblika diplomskega dela, naslov, povzetek, ter ključne besede identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 15. avgusta 2012 Podpis avtorja:

Zahvalil bi se mentorju prof. dr. Marku Robnik Šikonji za koristne nasvete in vso izkazano podporo med izdelavo diplomskega dela.

Zahvalil bi se tudi direktorju podjetja Xlab d.o.o., Gregorju Pipanu, ki mi je kljub moji navzočnosti v podjetju, omogočil nemoteno in kakovostno okolje za izdelavo diplomskega dela.

Kazalo

Povzetek

Cilj diplomske naloge je priprava učnega pripomočka za poučevanje genetskih algoritmov. Implementirali smo ga kot spletno stran, genetske algoritme pa uporabili za optimizacijo prometa. Optimizacijo prometa smo zastavili kot optimizacijo časa rdečih in zelenih luči na semaforjih tako, da vozila v cestnem omrežju prevozijo obravnavani segment cestnega omrežja kar najhitreje. Cestno omrežje smo definirali kot graf, za iskanje najkrajše poti v grafu smo uporabili algoritem Dijkstra. Končne nastavitve časov rdečih in zelenih luči za vsa križišča v cestnem omrežju smo vizualizirali z animacijo prometa in analizo uspešnosti optimizacije.

Abstract

We describe a learning tool for genetic algorithms. We implemented it as a web site where genetic algorithms solve a traffic optimization problem. The traffic optimization problem is defined as a timing of red and green traffic lights in a given road segment. The road network is presented with a graph and the shortest paths are found with Dijkstra algorithm. Final configurations of red and green lights for all intersections in the road segment are visualized with an animation and graphical analysis of optimization.

Poglavje 1

Uvod

S hitrim razvojem ekonomije in porastom prebivalstva so prometni zastoji postali eden resnejših problemov v večjih mestih. V preteklosti so take probleme reševali z dodajanjem pasov in novih povezav v že obstoječa omrežja. Z razvojem tehnologije pa so tudi semaforji postali bolj inteligentni in se prilagajajo trenutnemu prometu. Ponavadi so križišča in neusklajeni časi luči na semaforjih glavni vzrok za zastoje na cestah zato je pomembno, kako določimo čase rdečih in zelenih luči. [2]

Z dodajanjem križišč in semaforjev narašča zahtevnost optimizacije prometa. Klasične deterministične postopke je težko uporabiti na problemih z visoko zahtevnostjo, saj pogosto niso časovno sprejemljivi. Genetski algoritmi, s posnemanjem procesa evolucije, delujejo časovno bolj sprejemljivo, vendar ne zagotavljajo optimalnosti.

Z genetskimi algoritmi želimo optimizirati čase rdečih in zelenih luči na semaforjih tako, da vsa vozila pridejo od začetka do cilja čimhitreje in po najkrajši poti. Rešitve vizualiziramo z animacijo premikanja vozil v cestnem omrežju s semaforji in križišči. Cestno omrežje vnaprej definiramo, število vozil pa je parameter. Animacija poteka toliko časa, da vsa vozila prispejo na cilj. Animacija služi kot orodje za lažjo razlago delovanja genetskih algoritmov na realnem problemu.

1.1 Napoved vsebine po poglavjih

V drugem poglavju opišemo delovanje genetskih algoritmov, razložimo delovanje različnih genetskih operatorjev in opišemo različne metode izbire osebkov za reprodukcijo.

V tretjem poglavju opišemo problem urejanja prometa, razložimo kako smo pristopili k problemu in kako smo reševanje problema implementirali z genetskimi algoritmi.

V četrtem poglavju opišemo tehnologije uporabljene pri implementaciji genetskih algoritmov in animaciji.

V petem poglavju predstavimo in razložimo rezultate genetskih algoritmov.

V zadnjem poglavju predstavimo glavne ugotovitve in ideje za izboljšave.

Poglavje 2

Genetski algoritmi

Genetski algoritmi omogočajo enostaven in skoraj generičen pristop k reševanju kompleksnih optimizacijskih problemov. Kljub enostavnosti genetski algoritem potrebuje smiselno določene parametre, da lahko najde dobre rešitve. Izbira neustreznih parametrov vodi v daljši čas izvajanja programa in slabše rezultate [4].

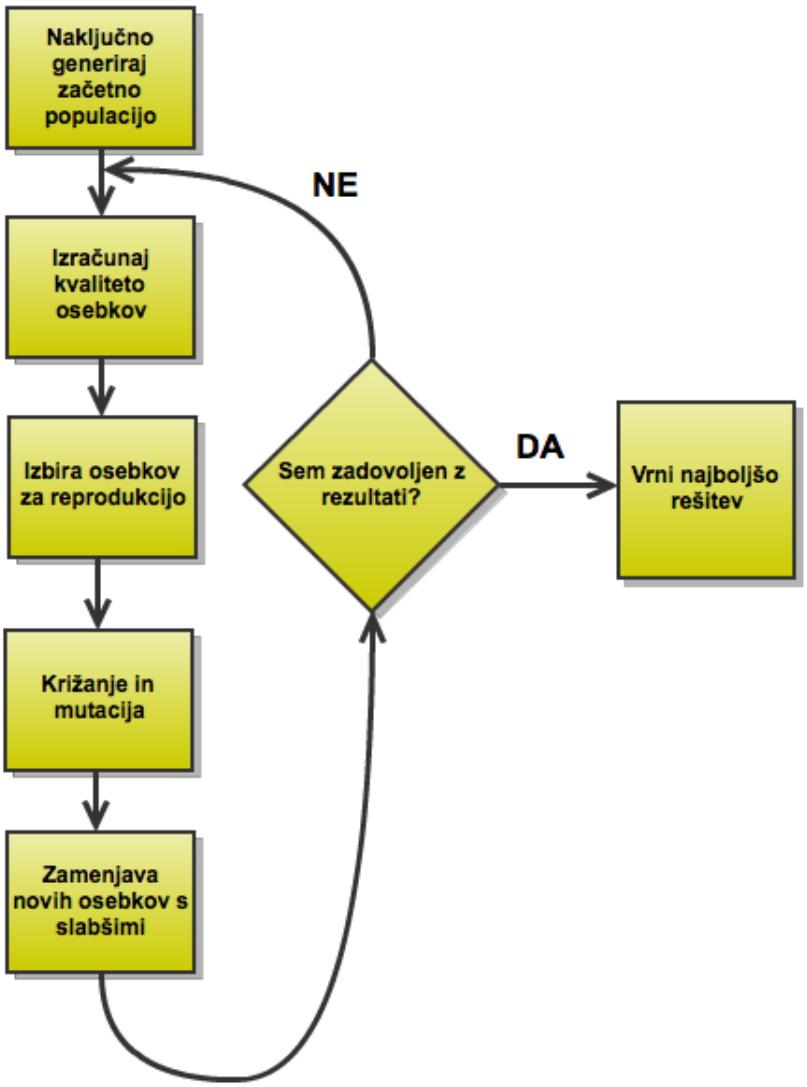
V nadaljevanju opišemo predstavitev značilno za genetske algoritme, genetske operatorje križanja in mutacije, funkcijo kvalitete in nekaj postopkov za izbiro osebkov za reprodukcijo. Delovanje genetskega algoritma prikazuje slika 2.1.

2.1 Predstavitev osebkov

Za delo z genetskimi algoritmi moramo problem predstaviti v taki obliki, da lahko rešitvi določimo kvaliteto in nad njo izvajamo genetske operatorje. Predstavitvi ene rešitve pravimo **osebek**.

2.1.1 Biološko ozadje

Vsak organizem vsebuje množico pravil in načrt, ki opisuje, kako je organizem zgrajen. Ta pravila so zakodirana v *genih* organizma, ki so povezani v dolg niz, ki se imenuje *kromosom*. Vsak gen predstavlja eno lastnost organizma,



Slika 2.1: Prikaz delovanja genetskega algoritma.

npr. barva oči ali barva las in še veliko drugih lastnosti [6]. Osebke lahko predstavimo na več načinov:

- z biti,
- vektorji
- nizi,
- drevesi, itd.

Osebek mora biti predstavljen tako, da mu je mogoče določiti kvaliteto, saj glede na izračunano kakovost, izberemo najboljše med osebki in iz njih ustvarimo nove osebke.

Za naš problem je osebek predstavljen kot zaporedje parov časov rdečih in zelenih luči na semaforjih. Primer za pet križišč bi zgledal takole:

(2, 3), (6, 8), (1, 5), (5, 8), (1, 7)

En par števil ustreza enemu križišču, časi pa so v sekundah. Par (2, 3) pomeni, da rdeča luč gori 2 sekundi, zelena pa 3 sekunde.

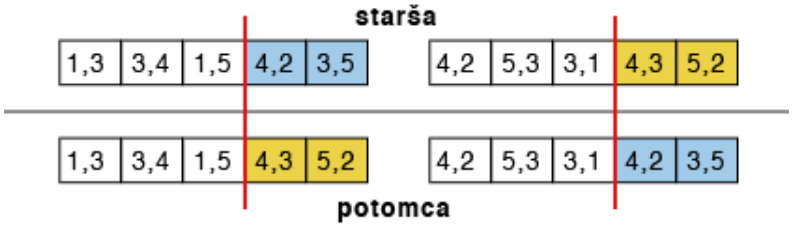
2.2 Začetna populacija

Populacija je v terminologiji genetskih algoritmov množica osebkov. Začetna populacija je lahko generirana naključno ali stohastično, nato pa se z genetskimi operatorji spreminja.

Večja populacija izboljšuje delovanje genetskih algoritmov, vpliva pa tudi na število generacij, potrebnih, da kvaliteta rešitev konvergira. Velikost populacije je odvisna od vrste problema, ponavadi vsebuje več sto ali tisoč osebkov.

2.3 Funkcija kvalitete

Funkcija kvalitete (ang. *fitness function*) je funkcija, ki objektivno oceni, kako blizu cilja je neka rešitev.



Slika 2.2: Mesto križanja je označeno z rdečo črto. Prvi del (bela barva) genetskega zapisa se ohrani, drugi del (modra in rumena) pa se izmenjata.

Razlog, da genetski algoritmi niso trivialen način reševanja problemov, je predvsem v energiji vloženi v oblikovanje delujoče funkcije kvalitete. Če je funkcija kvalitete oblikovana napačno, bo algoritem konvergiral k napačnim rešitvam, ali pa sploh ne bo konvergiral.

Znaki da je funkcija kvalitete slabo oblikovana so:

- čas računanja kvalitete rešitve je velik,
- natančen model za izračun kvalitete manjka.

2.4 Križanje

Križanje (ang. *crossover*) je najpomembnejša tehnika izmenjave genetskega materiala med osebki. Uporablja se eno, dvo in večmestno križanje. Enomestno križanje poteka tako, da naključno izberemo točko križanja in preko te točke izmenjamo genetski material, kar ilustrira slika 2.2.

Križanje naj bi generiralo boljše osebke, saj naj bi se dobri deli genetskega zapisa ohranili in prenesli na potomce [1].

2.5 Mutacija

Mutacija je postopek, ko na nekem naključno izbranem mestu spremenimo genetski zapis izbranega osebka. Brez mutacij bi križanje lahko izmenje-

valo samo genetski material vsebovan v začetni slučajno generirani generaciji osebkov. Mutacija doda potencialno novo informacijo. Verjetnost mutacije kljub njeni koristnosti ne sme biti prevelika, saj se sicer pokvari preveč koristnih genov v osebkih in postane evolucijsko računanje podobno naključnemu preiskovanju [1].

2.6 Evolucijski model

Evolucijski model imenujemo izbiro osebkov za razmnoževanje. Osebke želimo izbrati tako, da se dobri osebki, ki vsebujejo kakovosten genetski material, prenesejo v naslednjo generacijo. Zato je smiselno, da dobri osebki tvorijo več potomcev in da se morda celo nespremenjeni prenesejo v naslednjo generacijo. Slednje imenujemo elitelizem. Nekaj uveljavljenih evolucijskih modelov je predstavljenih v nadaljevanju [1].

2.6.1 Proporcionalna izbira

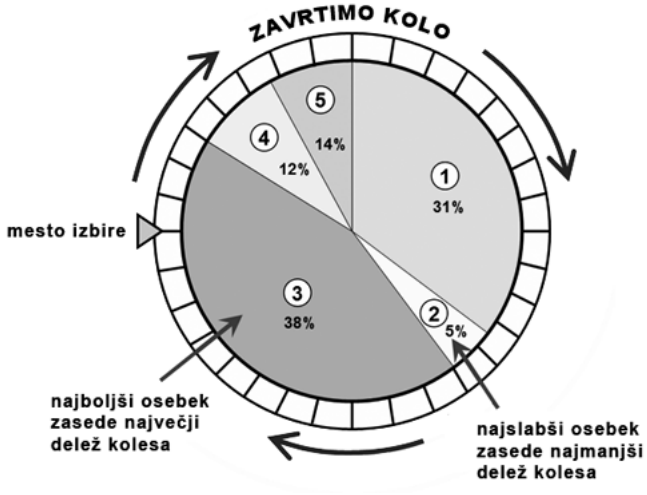
Vsakemu osebku je dodeljena verjetnost izbire glede na njegovo kvaliteto. Naslednja enačba prikazuje, kako je dodeljena verjetnost izbire p_i:

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2.1}$$

Osebki so nato izbrani za reprodukcijo s pomočjo metode ruletnega kolesa. Delež kolesa, ki ga osebek zaseda, je sorazmeren njegovi uspešnosti. Izbiro z metodo ruletnega kolesa na populaciji velikosti 5 osebkov ilustrira slika 2.3.

Slabost takšnega načina izbire je, da lahko dobri osebki zelo hitro prevladajo, že po nekaj generacijah, ker vsakič zasedajo večji delež ruletnega kolesa.

Druga slabost izbire z ruletnim kolesom je, da **ne deluje** za probleme, kjer želimo kvaliteto osebkov minimizirati. Težavo odpravimo tako, da problem minimizacije kvalitete prevedemo na problem maksimizacije kvalitete z ustreznimi transformacijami.



Slika 2.3: Izbira osebkov za reprodukcijo z metodo ruletnega kolesa. Osebek zaseda delež kolesa proporcionalen njegovi kvaliteti [11].

2.6.2 Enoturnirska izbira

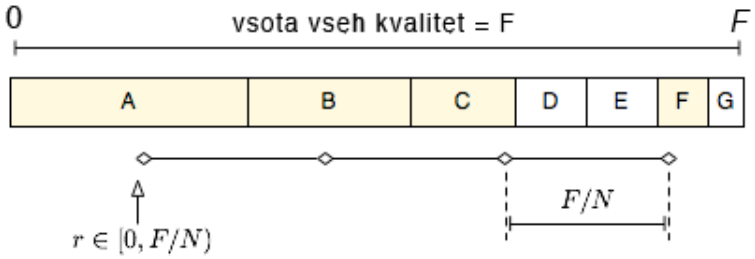
Gre za priljubljen in preprost način hkratne izbire osebkov za razmnoževanje in istočasno nadomeščanja osebkov. Poteka takole:

1. naključno razbij populacijo na majhne skupine velikosti g,
2. dva najboljša osebka iz vsake skupine se križata in njuna potomca nadomestita najslabša osebka v skupini.

Prednost enoturnirske izbire je, da preživi v naslednjo generacijo (g - 2) najboljših osebkov in tako ne izgubljamo prehitro dobrih osebkov pa tudi maksimalna kvaliteta populacije ne pada. Po drugi strani strategija zagotavlja, da še tako dober osebek nima več kot dveh potomcev, in se tako ohranja raznolikost populacije [1].

2.6.3 Stohastično univerzalno vzorčenje

Slabost do sedaj opisanih metod izbire je velika varianca glede na funkcijo kvalitete. Pri nekaterih evolucijskih modelih se lahko zgodi, da najboljši



Slika 2.4: Ilustracija stohastičnega univerzalnega vzorčenja. Osebke razvrstimo na interval $[0, 1]$ proporcionalno njihovi kvaliteti. Izbiramo N osebkov, zato začetno točko r naključno izberemo na intervalu $[0, \frac{1}{N}]$. Osebke izberemo na mestih $r + \frac{i}{N}, i \in [0, 1, \ldots, N-1]$ [1].

osebek nima nobenega potomca. Manjšo varianco zagotavlja stohastično univerzalno vzorčenje, ki ga ilustrira slika 2.4.

Na podlagi kvalitete osebka f_i tvorimo verjetnostno distribucijo kot pri proporcionalni izbiri $p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}$. Osebke naključno razvrstimo na številski trak na interval med 0 in 1 in vsak dobi dolžino traku sorazmerno p_i. Določimo število osebkov N, ki jih želimo generirati in naključno izberemo število z intervala $[0, \frac{1}{N}]$. Za razmnoževanje izberemo osebke, ki pokrivajo točke $r + iN, i \in [0, 1, \ldots, N-1]$ [1].

2.7 Elitizem

Elitizem je strategija, ki določen del najboljših osebkov prenese direktno v naslednjo generacijo. Dobra stran te strategije je, da imamo v vsaki generaciji na voljo najboljše osebke in jih ni potrebno posebej hraniti, poleg tega pa zaradi variance pri izbiri ne izgubljamo dobrega genetskega materiala. Slabost je v mogoči prezgodnji konvergenci, če elita prevlada [1].

Poglavje 3

Problem optimizacije prometa

V tem poglavju predstavimo raziskave na področju optimizacije prometa, opišemo naš pristop k reševanju problema in ga formaliziramo. Predstavimo algoritem za iskanje najkrajše poti v grafu in uporabo genetskih algoritmov na problemu optimizacije prometa.

3.1 Ozadje

Promet na cestah se povečuje s hitrejšim tempom kot porast prebivalstva. Problem urejanja prometa je prisoten v vseh večjih mestih v vseh državah. Večanje prometa povzroča prometne zastoje, zato se vozniki in potniki srečujejo s problemi, kot so nepotrebno zapravljanje časa, stresom, onesnaževanjem, povečano nevarnostjo nesreč in ostalimi resnimi problemi. Obstajajo različni pristopi kako reševati take probleme. Ena od možnosti je uporaba obstoječih cest s pravilnim rokovanjem semaforjev.

Današnje raziskave na področju optimizacije semaforjev lahko razdelimo na dva dela. Po eni strani je bilo veliko truda vloženega v razvoj visoko prilagodljivih semaforjev, ki so sposobni reagirati na spremembe v prometu takoj, v realnem času. Po drugi strani, pa se ob delitvi časa na faze (npr. jutranja, popoldanska, nočna), zdi uporaba prednastavljenih semaforjev dokaj razumljiva.

Drugo razlikovanje bi lahko naredili med hevrističnimi pristopi k optimizaciji in čistimi matematičnimi pristopi in njihovimi osnovnimi modeli. Hevristični pristopi temeljijo na genetskih algoritmih, mehki logiki in nevronskih mrežah. Za njih je značilno, da ponavadi dajejo dobre in hitre rešitve, ampak ne zagotavljajo kvalitete dane rešitve v primerjavi z optimalno.

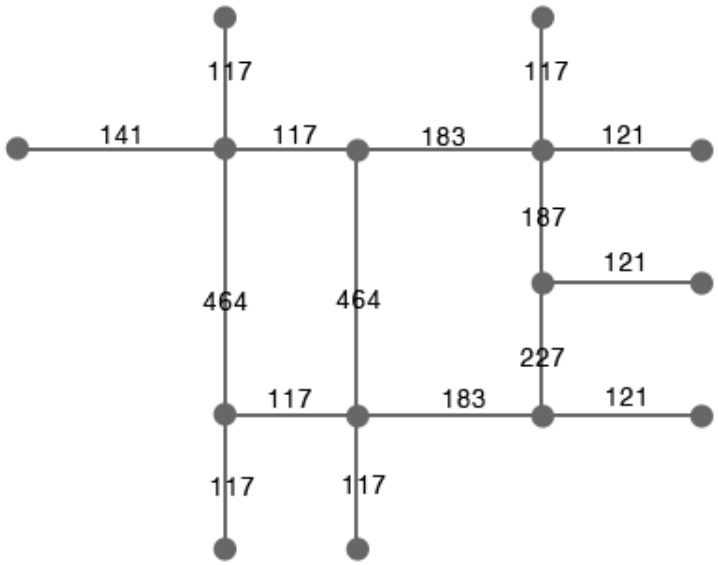
Matematični pristopi pa ponavadi ne morejo pokriti vseh zahtev in parametrov resničnega cestnega omrežja, ampak ponujajo dokazano dobre rešitve. Take rešitve so lahko včasih dobre začetne rešitve za hevristične pristope.

3.2 Formalizacija problema

Postavimo se v vlogo voznika, ki želi priti od doma v službo čimhitreje in po najkrajši poti skozi središče mesta, kjer je veliko semaforiziranih križišč. Imamo še 19 prijateljev, nekateri želijo priti od doma do trgovine, ki je na drugem koncu mesta, drugim pa se mudi v šolo, a vsi želijo priti na svoj cilj čimhitreje in po najkrajši poti. Potovalni čas vsakega vozila bo najkrajši takrat, ko bo najmanj časa čakal na rdečih lučeh in v koloni ter bo šel po najkrajši poti [1].

Cestno omrežje je predstavljeno z grafom $G = (V, E)$, kjer je V množica vozlišč in E množica uteženih povezav med vozlišči. Vozlišča ustrezajo križiščem, povezave pa cestam med križišči. Uteži na povezavah ustrezajo razdalji (v metrih), ki jih mora vozilo prepotovati, da doseže naslednje vozlišče. Dodatno zahtevamo, da je vozlišče križišče, če je njegova stopnja $>$ 2, sicer je vozlišče bodisi začetno bodisi končno. Vozilo lahko začne in konča svojo pot le v končnih oziroma začetnih vozliščih. Slika 3.1 v obliki grafa prikazuje možno postavitev cestnega omrežja.

[1] Za iskanje najkrajše poti v grafu na sliki 3.1 smo uporabili algoritem Dijkstra, opisan v razdelku 3.3.



Slika 3.1: Primer grafa s 15 vozlišči in uteženimi povezavami med njimi.

3.3 Iskanje najkrajše poti v grafih

Algoritem Dijkstra je zasnoval nizozemski računalniški inženir Edsger Dijkstra. Algoritem poišče najkrajšo pot v usmerjenem grafu z nenegativnimi uteženimi povezavami. Za podano začetno vozlišče v grafu algoritem poišče najcenejšo pot med začetnim vozliščem in vsemi ostalimi vozlišči. Lahko ga uporabimo tudi za iskanje najcenejše poti med začetnim vozliščem in končnim vozliščem. Primer take uporabe je naš graf na sliki 3.1.

3.4 Genetski algoritem

V poglavju 3 smo na splošno opisali delovanje genetskih algoritmov. V tem razdelku pa opišemo, kako smo uporabili genetske algoritme na našem problemu.

V nadaljevanju so naštete privzete nastavitve genetskih algoritmov in nastavitve animacije. Te nastavitve bomo uporabljali tudi v nadaljevanju,

ko opisujemo, kako smo implementirali operacije genetskih algoritmov in pri predstavitvi rezultatov.

Nastavitve genetskih algoritmov:

- št. generacij = 100,
- velikost populacije = 100,
- delež križanja = 0.70,
- delež mutacije = 0.05.

Nastavitve animacije:

- FPS = 50,
- hitrost vozila = 50 $\frac{km}{h}$,
- hitrost animacije = 20.

Hitrost animacije lahko spreminjamo na intervalu $[1, 20]$ s korakom 1.

Parameter *FPS* (ang. *frames per second*) in *hitrost vozila* sta vnaprej definirana in ju uporabnik ne more spreminjati, vendar ju navajamo, saj sta pomembna za nadaljevanje.

3.4.1 Funkcija kvalitete

V razdelku 2.1 smo osebek predstavili kot zaporedje parov časov rdečih in zelenih luči na semaforjih, optimizacijski problem pa smo formalizirali na primeru 20 vozil.

Naš cilj je za vseh 20 vozil poiskati optimalno nastavitev časov rdečih in zelenih luči na semaforjih, tako da bodo vsa vozila prišla na cilj čimprej. To pomeni, da moramo izračunati ali izmeriti čas potovanja **vseh** vozil.

Kvaliteto rešitve ocenimo s simulacijo [2], pri čemer smo parameter *hitrost animacije* nastavili na vrednost 20 zato, da je simulacija hitrejša. Simulacija

[2]Simulacijo smo izvedli enako kot animacijo, le da ničesar ne izrišemo.

simulira premikanje vseh vozil od začetnega vozlišča do končnega, pri tem pa merimo čas dokler vsa vozila ne pridejo na cilj. Zaradi tehničnih razlogov merimo število korakov potrebnih za to, da vsa vozila prispejo na cilj. Število korakov pretvorimo v *čas potovanja* z naslednjo formulo:

$$\textit{čas potovanja} = \frac{\textit{prepotovana dolžina vseh vozil}}{\textit{hitrost vozila } (\frac{m}{s})} \tag{3.1}$$

$$\textit{prepotovana dolžina vseh vozil} = \textit{število iteracij} * \textit{velikost koraka} \tag{3.2}$$

$$\textit{velikost koraka} = \frac{\textit{hitrost vozila} * \textit{hitrost animacije}}{FPS} \tag{3.3}$$

Enačba 3.3 za *velikost koraka* služi za pretvorbo realne hitrosti vozila ($\frac{km}{h}$) v enoto potrebno za animacijo, oz. simulacijo ($\frac{px}{iteracijo}$).

3.4.2 Proporcionalna izbira

Slabost izbire z ruletnim kolesom je, da ne deluje za minimizacijske probleme in moramo zato minimizacijski problem prevesti na maksimizacijski. V našem primeru, kjer manjša vrednost pomeni boljšo kvaliteto, smo to naredili tako, da pred klicem izbire z ruletnim kolesom, kvaliteto pomnožimo s faktorjem -1. Na ta način majhne in velike vrednosti zamenjajo vrstni red, medsebojna razmerja med elementi pa se ohranijo.

Po koncu izbire z ruletnim kolesom kvalitete ponovno pomnožimo s faktorjem -1, da dobimo nazaj prvotne vrednosti. Algoritem 2 prikazuje psevdo kodo za izbiro osebkov z metodo ruletnega kolesa.

Po pretvorbi in pred izvajanjem izbire z ruletnim kolesom kvalitete še normaliziramo in izračunamo kumulativne vrednosti normaliziranih kvalitet. Kvalitete normaliziramo na interval med $[0, 1]$ tako, da za vsak osebek izračunamo verjetnost proporcionalno glede na kumulativno oceno kvalitete vseh osebkov (enačba 2.1). Kumulativne vrednosti normaliziranih kvalitet pa izračunamo tako, da prištejemo k vsaki kvaliteti še kvalitete vseh prejšnih osebkov, kar prikazuje algoritem 1.

```
for vse osebke iz populacije do
    verjetnost = vsota vseh verjetnosti + (kvaliteta osebka / vsota
    kvalitet)
    vsota vseh verjetnosti + = verjetnost
end
```

Algoritem 1: Izračun kumulativne normalizirane ocene kvalitete osebka.

```
repeat
    for ponovi dvakrat do
        število = naključno število med 0 in 1
        for vse osebke iz populacije do
            if verjetnost osebka >= število then
                si bil izbran
            end
        end
    end
    križaj starša in boljšega od potomcev dodaj v novo populacijo
until nova populacija ni dovolj velika;
```

Algoritem 2: Izbira osebkov z metodo ruletnega kolesa.

Elitizem

V metodo proporcionalne izbire smo vključili elitizem, t.j. direkten prenos najboljših osebkov v novo populacijo. Delež elitizma spreminjamo s spreminjanjem deleža križanja in deleža mutacije:

$$delež\ elitizma = 1 - (delež\ križanja + delež\ mutacije) \tag{3.4}$$

Z nastavitvami iz razdelka 3.4 se v novo populacijo direktno prenese 25 osebkov.

Mutacija in križanje

Križanje izvajamo, kot je opisano v razdelku 2.4, mutacijo pa izvajamo nad potomci. Delež mutacije smo nastavili na 0.05, zato petim (100*0.05 = 5) naključno izbranim potomcem naključno spremenimo genetski zapis in jih damo v novo populacijo.

3.4.3 Enoturnirska izbira

Simulacijo turnirjev smo implementirali tako, da smo velikost turnirja nastavili na 4 elemente. To pomeni, da moramo za populacijo velikosti 100 osebkov, odigrati 25 turnirjev. Osebki so za vsak turnir izbrani naključno in brez vračanja (en osebek **ne** more nastopati v več turnirjih). Vsak turnir nato uredimo glede na oceno kvalitete osebkov in vzamemo dva najboljša ter ju križamo. Dobimo dva potomca, ki ju zamenjamo s preostalima osebkoma in shranimo v *množico potomcev*. Nad to množico izvedemo mutacijo tako, da petim osebkom naključno spremenimo genetski zapis.

3.4.4 Stohastično univerzalno vzorčenje

Stohastično univerzalno vzorčenje smo prepisali v Javascript po izvorni kodi, ki smo jo našli na internetu [7]. Njegovo psevdo kodo prikazuje algoritem 3.

```
velikost izbire = velikost populacije * delež križanja
začetni odmik = naključno število med 0 in 1
vsota = 0
index = 0
for vsak osebek iz populacije do
    vsota += (kvaliteta osebka / vsoto vseh kvalitet) * velikost izbire
    while vsota > začetni odmik + index do
        osebek je bil izbran
        index++
    end
end
```

Algoritem 3: Psevdo koda za stohastično univerzalno vzorčenje. Vsak izbran osebek shranimo v *monžico izbranih osebkov.*

Vsak osebek, ki je bil izbran za reprodukcijo s stohastičnim univerzalnim vzorčenjem shranimo v *množico izbranih osebkov.* Iz te množice nato, z vračanjem (en osebek lahko večkrat izberemo), izberemo dva starša, ki s križanjem tvorita dva nova potomca. Boljšega izmed potomcev shranimo v novo populacijo. Ta postopek ponavljamo dokler nismo napolnili nove populacije.

Elitizem in mutacijo izvedemo na enak način kot pri proporcionalni izbiri, saj 25 najboljših osebkov prenesemo direktno v novo populacijo, mutacijo pa izvedemo nad petimi naključno izbranimi potomci in jih shranimo v novo populacijo.

3.5 Vizualizacija

Vizualizacijo smo zasnovali kot spletno stran, saj je namen diplomskega dela priprava učnega pripomočka za poučevanje genetskih algoritmov. Spletna stran je povsod dosegljiva, pri programih pa so različne omejitve glede plaftorme in namestitve.

Celotna animacija prometa je napisana v tehnologijah HTML5 in Java-

script, prav tako pa tudi genetski algoritem, zato uporabnik potrebuje le enega od naprednejših spletnih brskalnikov, npr. Google Chrome, Mozilla Firefox, Safari ali Opera.

Vizualizacija omogoča različne nastavitve parametrov genetskega algoritma in problema. Za genetski algoritem lahko nastavljamo:

- način izbire osebkov (enoturnirska izbira, proporcionalna izbira in stohastično univerzalno vzorčenje)
- število generacij,
- velikost populacije,
- delež križanja in
- delež mutacije.

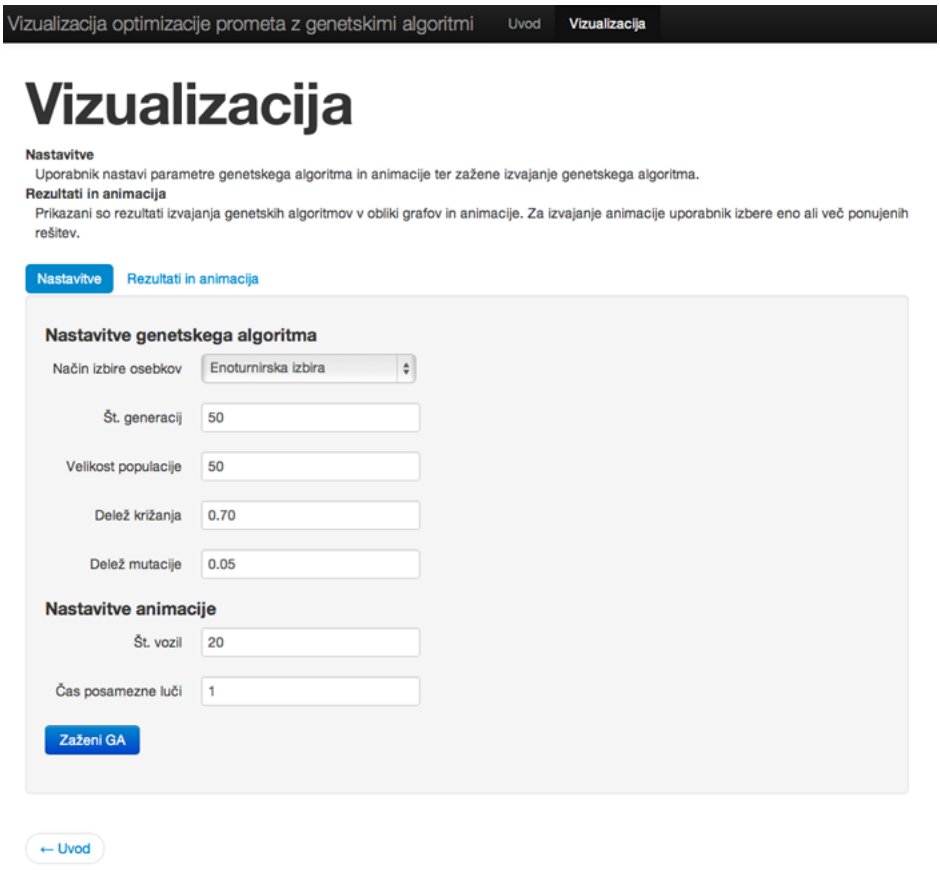
Za optimizacijski problem pa lahko nastavljamo:

- hitrost animacije,
- število vozil,
- maksimalen čas, ko je prižgana posamezna luč.

Od nastavitev parametrov je odvisna hitrost izvajanja genetskega algoritma, zato mora uporabnik previdno nastavljati te parametre.

Večji je maksimalen čas posamezne luči, večji je prostor možnih rešitev, zato je potrebno temu ustrezno povečati velikost populacije. S tem omogočimo algoritmu, da bo preiskal dovolj velik prostor možnih rešitev, vendar pa moramo zato povečati število generacij, saj bo algoritem potreboval dlje časa, da najde najboljšo rešitev. Na hitrost izvajanja genetskega algoritma vpliva tudi število vozil.

Vizualizacija je na voljo na GitHubu [12], kjer si lahko uporabnik ogleda izvorno kodo, ali pa jo pretoči k sebi in sodeluje pri njenem razvoju.

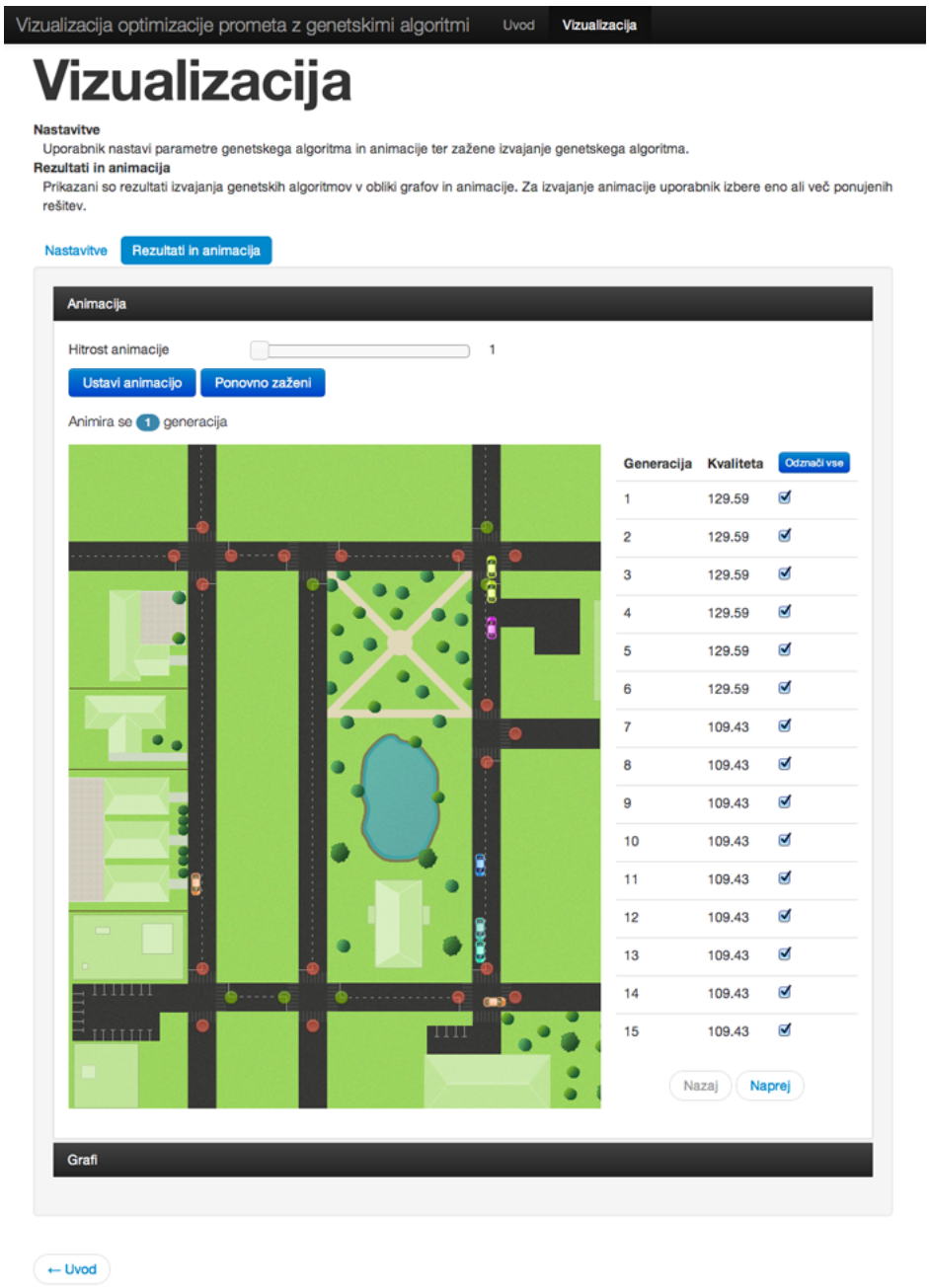


Slika 3.2: Nastavitve za genetski algoritem in animacijo.

3.5.1 Spletna stran

Spletna stran na začetku uporabniku ponudi kratek pregled genetskih algoritmov in razloži, kako so genetski algoritmi uporabljeni na problemu optimizacije prometa. Uporabnik lahko nastavi parametre genetskih algoritmov in animacije. S pritiskom na gumb "*Zaženi GA*" zažene genetski algoritem, ki potrebuje nekaj časa da se izvede. Slika 3.2 prikazuje stran z nastavitvami.

Po koncu izvajanja so uporabniku ponujeni rezultati v obliki grafov in animacije. Uporabnik izbere rešitve, ki jih želi animirati. Rešitve so urejene od najslabše do najboljše. Animacijo lahko ustavlja in ponovno zažene. Slika 3.3 prikazuje izbiro rešitev, kontrolne gumbe in izvajanje animacije.



Slika 3.3: Animacija rezultatov.

3.5.2 Animacija

Animacija se izvaja v t.i. zanki igre (ang. *game loop*). Naloga zanke je izrisovanje in posodabljanje parametrov objektov, ki jih izrisuje. Zanka igre se mora izvajati dovolj hitro, da je animacija tekoča. Človesko oko lahko zazna med 10 in 12 različnih slik na sekundo, kljub temu pa se običajno izris dela 50-krat na sekundo.

Pri animaciji smo izris 50-krat na sekundo dosegli tako, da ga izvajamo na 20 ms. V Javascriptu obstaja metoda `setInterval(funkcija(), cas)`, ki na vsakih `cas` milisekund pokliče metodo `funkcija()`. V našem primeru je `funkcija()` metoda, ki poskrbi za izris in posodabljanje parametrov objekta, `cas` pa je nastavljen na 20 ms.

Pri animaciji se izrisujejo vozila in semaforji. Za poenostavitev problema in animacije smo vsem vozilom definirali isto hitrost premikanja in isti čas prihajanja. Vsa vozila vstopijo v animacijo ob času 0.

Poglavje 4

Tehnologije

V tem poglavju opišemo tehnologije uporabljene pri razvoju vizualizacije, animacije in genetskega algoritma ter predstavimo orodje za t.i. družabno programiranje (ang. *social coding*) GitHub.

4.1 HTML

HTML (*HyperText Markup Language*) je glavni označevalni jezik za prikazovanje spletnih strani in drugih vsebin v spletnem brskalniku. HTML je napisan v obliki elementov, sestavljenih iz značk (ang. *tags*) zaprtih med znaka < in >. Značke najbolj pogosto nastopajo v obliki parov, npr. <h1> in </h1>. Obstajajo tudi prazne značke, ki nastopajo samostojno, npr. <img>. Prvo značko v paru imenujemo *začetna značka*, drugo pa *končna značka*. Med te značke lahko oblikovalci spletnih strani dodajo besedilo, druge značke, komentarje in druge vsebine.

Privzete značilnosti vsakega HTML elementa so definirane v spletnem brskalniku, zato nekateri naprednejši spletni brskalniki vsebujejo značke, ki delujejo samo v določenem brskalniku v ostalih pa ne.

Namen spletnega brskalnika je prebrati HTML dokument in ga pretvoriti v vidne in slišne elemente spletne strani. Spletni brskalnik uporablja HTML značke za oblikovanje vsebine spletne strani in ne prikazuje HTML značk.

HTML predstavlja temelj spletnih strani in omogoča uporabo slik in drugih objektov, vključenih v spletno stran, lahko pa se uporablja tudi za grajenje spletnih obrazcev. Omogoča oblikovanje strukturiranih dokumentov z označevanjem semantike besedila, npr. naslovi, odstavki, seznami, povezave, citati in drugi elementi. Omogoča tudi vključevanje datotek drugih jezikov, ki vplivajo na prikaz spletne strani [8].

4.2 HTML5

HTML5 je naslednja generacija HTML standarda, ki nadomešča starejše standarde, kot so HTML 4.01, XHTML 1.0 in 1.1. Standard prinaša novosti potrebne za moderne spletne aplikacije. Standard vključuje in definira mnoge lastnosti, ki so jih spletni razvijalci uporabljali že leta, a niso bile vključene v katerega izmed standardov. HTML5 je prvi poskus formalno dokumentirati "de facto" standarde, ki jih spletni brskalniki podpirajo že leta. Kot njegovi predhodniki je tudi HTML5 zasnovan platformno neodvisno.

HTML5 lahko uporabljamo in pregledujemo na poljubnem operacijskem sistemu in mobilni napravi. Uporabniki potrebujejo le moderen spletni brskalnik, le ti pa so brezplačno na razpolago za praktično vse operacijske sisteme, npr. Google Chrome, Mozilla Firefox, Opera in Safari. Vsi našteti brskalniki podpirajo veliko HTML5 lastnosti, ne podpirajo pa vsi vseh. Odlično podporo standardu HTML5 imajo tudi mobilni spletni brskalniki za mobilne naprave.

HTML5 je zasnovan za čim boljšo združljivost z že obstoječimi spletnimi brskalniki. Nove funkcionalnosti so zgrajene na starih in omogočajo združljivost za starejše brskalnike. Obstoj posameznih funkcionalnost HTML5 lahko ugotovite že z nekaj vrsticami Javascripta [3].

HTML5 ne prinaša samo novih značk in novega standarda. Skupaj z razvojem naprednih spletnih brskalnikov omogoča tudi lastnosti, ki so jih do sedaj imele le namizne aplikacije. Naštejmo nekaj glavnih lastnosti, ki jih danes omogočajo spletni brskalniki [9]:

- internet v brezpovezavnem načinu (ang. *offline web*),
- hranjenje podatkov v lokalni bazi brskalnika,
- povezljivost - klepetanje v realnem času, hitrejše igre, itd.,
- dostop do datotečnega sistema,
- semantika - internacionalizacija, novi atributi za pomensko označevanje vsebine, itd.,
- avdio/video,
- 3d grafika,
- predstavitev - 2D in 3D transformacije omogočajo izdelavo bogatih uporabniških vmesnikov,
- zmogljivost - spletne aplikacije se lahko primerjajo z zmogljivostjo namiznih aplikacij.

Te lastnosti in druge, skupaj s standardom HTML5 definirajo splet kot platformo (ang. *web platform*).

4.3 CSS

CSS je kratica za Cascading Style Sheet, kar bi v prevodu pomenilo predlogo, ki določa izgled. Uporablja se za opisovanje izgleda dokumentov napisanih v označevalnih jezikih. Najbolj pogosto se uporablja za opisovanje izgleda spletnih strani napisanih v HTML-ju in XHTML-ju, vendar lahko jezik uporabimo tudi v dokumentih XML, SVG in XUL. CSS specifikacije ureja World Wide Web Consortium (W3C).

CSS je prvenstveno namenjen ločevanju vsebine od izgleda dokumenta. Taka ločitev lahko izboljša dostopnost vsebine, ponuja večjo fleksibilnost in nadzor nad specifikacjiami predstavitve, omogoča deljenje istega stila čez več strani ter zmanjša kompleksnost in ponavljanje stila v strukturiranih

body{
font-family: Arial;
font-size: 11px;
color: red;
}

Slika 4.1: CSS stil, ki nastavi barvo, velikost in tip pisave HTML znački <body>.

dokumentih. CSS omogoča, da je ista stran predstavljena z več različnimi stili za različne zahteve izrisovanja, npr. izris na zaslon, izris za printanje, itd.. Lahko se uporabi za prikazovanje spletne strani na različnih resolucijah zaslona ali napravah.

CSS ima določeno prioritetno shemo, da ugotovi, kateri stil uporabiti če enemu elementu ustreza več pravil. V tako imenovani *kaskadi* so prioritete ali uteži izračunane in dodeljene pravilom tako, da so rezultati ujemanja pravil predvidljivi. To omogoča oblikovalcem spletnih strani, da natančno določijo obliko vsebine strani. Primer preprostega CSS stila za HTML značko <body> je prikazan na sliki 4.1.

4.4 Javascript

Javascript je programski jezik spletnega brskalnika. Bil je razvit v samo desetih dneh pri podjetju Netscape z namenom, da uporabnikom ponudijo lahko in prenosno različico Jave, ki bi izboljšala uporabniško izkušnjo s spletno stranjo.

Javascript je skriptni jezik, dinamičen in šibko tipiziran ter podpira več različnih vzorcev programiranja, npr. objektno orientirano, funkcionalno, itd..

Največ se jezik uporablja za izvajanje na strani klienta, implementiran kot del spletnega brskalnika za obogatitev uporabniških vmesnikov in dinamičnih spletnih strani.

Z razvojem vse hitrejših navideznih strojev in programskih ogrodij za ta jezik, narašča tudi popularnost uporabe za strežniške spletne aplikacije. Že leta 1994, kmalu po izdaji Javascript jezika, je isto podjetje izdalo še različico na voljo za strežnike. Eden popularnejših okolij za razvoj strežniških spletnih aplikacij je orodje Node.js.

Javascript uporablja sintakso podobno programskemu jeziku C. Veliko imen in pravil poimenovanja spremenljivk ter funkcij je vzetih iz programskega jezika Java, vseeno pa sta tadva jezika nepovezana in imata popolnoma različno semantiko. Ključni principi pri oblikovanju so bili vzeti iz Self in Scheme programskega jezika.

4.5 GitHub

GitHub je spletna storitev za gostovanje programske opreme, ki uporablja za nadzor različic programske opreme Git kontrolni sistem. Git je distribuiran kontrolni sistem za nadzor različic programske opreme in sistem za upravljanje z izvorno kodo.

GitHub uporabnikom ponuja plačljive pakete za privatne projekte in zastonjski paket za odprtokodne projekte. V juliju 2011 je podjetje dobilo investicijo v vrednosti 100 miljonov ameriških dolarjev od znane investicijske hiše Andreessen Horowitz.

GitHub ponuja funkcionalnosti socialnega omrežja, kot so: viri (ang. *feeds*), spremljevalci (ang. *followers*) in graf omrežja za prikaz, koliko razvijalcev trenutno sodeluje pri razvoju projekta [10].

Poglavje 5

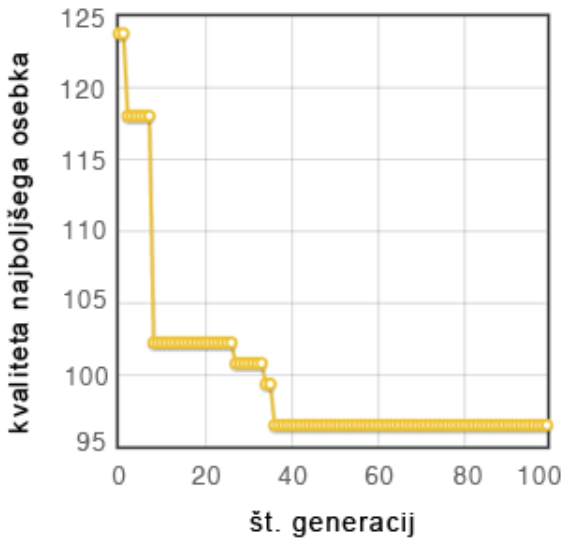
Rezultati

V tem poglavju predstavimo rezultate izvajanja genetskih algoritmov za vsako metodo izbire posebej in na koncu podamo ugotovitve. Vse metode izbire smo zagnali z nastavitvami iz razdelka 3.4:

- št. generacij = 100,
- velikost populacije = 100,
- delež križanja = 0.70,
- delež mutacije = 0.05.

5.1 Proporcionalna izbira

Analizo uspešnosti optimizacije za proporcionalno izbiro prikazuje slika 5.1. Opazimo, da se prvih nekaj generacij kvaliteta hitro izboljšuje (s 125 pade na 102.50) potem pa se izboljšuje počasneje. Včasih se celo nekaj generacij ne izboljša, npr. od 40 do 100 generacije. Brez mutacije so rešitve počasneje konvergirale in k slabšim vrednostim.



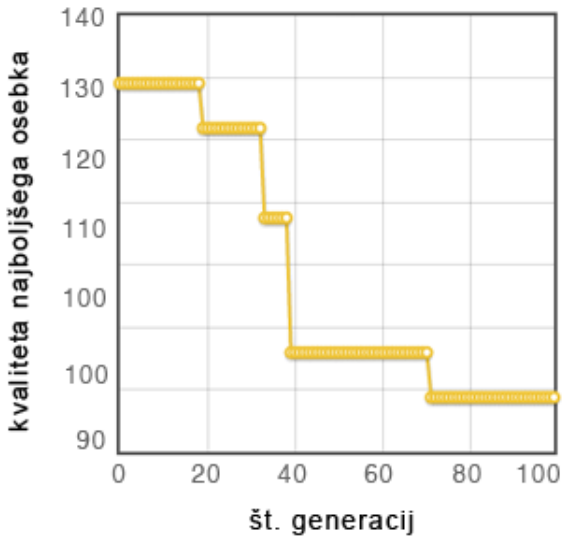
Slika 5.1: Rezultati izvajanja genetskega algoritma z nastavitvami iz razdelka 3.4 in proporcionalno izbiro.

5.2 Enoturnirska izbira

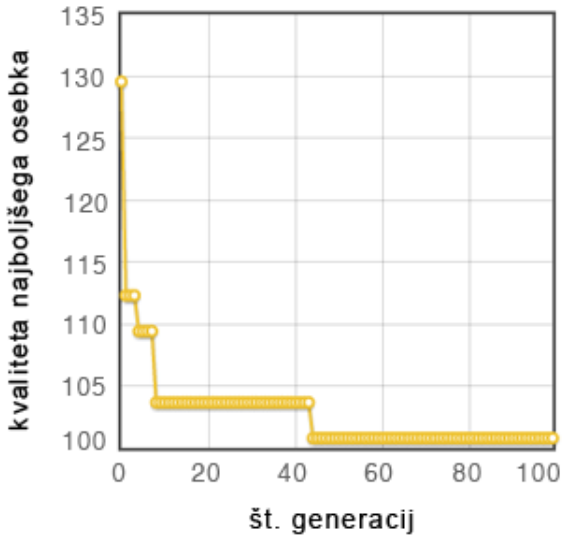
Analiza uspešnosti optimizacije za enoturnirsko izbiro je prikazana na sliki 5.2. Opazimo, da se kvaliteta rešitev izboljšuje stopničasto. To pomeni, da se nekaj generacij ne izboljša, nato se izboljša, potem spet nekaj generacij ne, se izboljša, itd. Brez mutacije so rešitve hitreje konvergirale in k slabšim vrednostim, saj brez mutacije nismo dobili nove informacije, ki jo mutacija doda.

5.3 Stohastično univerzalno vzorčenje

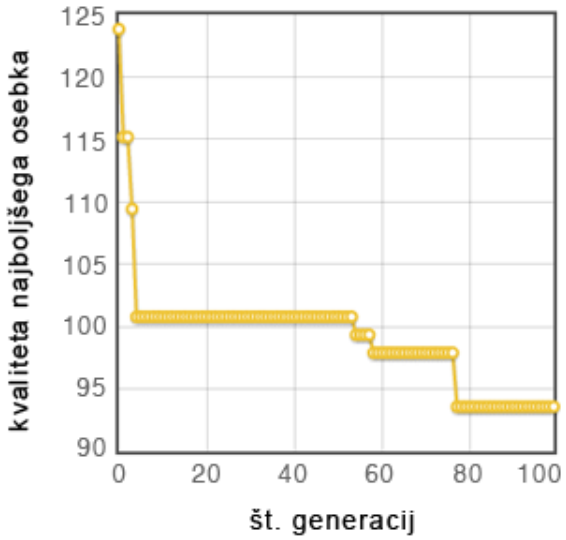
Analizo uspešnosti optimizacije s stohastičnim univerzalnim vzorčenjem prikazuje slika 5.3, na kateri lahko opazimo, da kvaliteta zelo hitro konvergira. Z večanjem mutacije je kvaliteta konvergirala počasneje, vendar k boljšim rešitvam. Slika 5.4 prikazuje izvajanje genetskega algoritma z deležem križanja 0.60 in deležem mutacije 0.10.



Slika 5.2: Rezultati izvajanja genetskega algoritma z nastavitvami iz razdelka 3.4 in enoturnirsko izbiro.



Slika 5.3: Rezultati izvajanja genetskega algoritma z nastavitvami iz razdelka 3.4 in metodo stohastičnega univerzalnega vzorčenja.



Slika 5.4: Rezultati izvajanja genetskega algoritma z nastavitvami iz razdelka 3.4 in metodo stohastičnega univerzalnega vzorčenja z deležem križanja 0.60 in deležem mutacije 0.10.

5.4 Ugotovitve

Primerjava kvalitet rešitev vseh treh metod izbire pokaže, da najslabše rezultate daje metoda enoturnirske izbire. Preostali dve metodi dajeta boljše rezultate in potrebujeta manjše število generacij za konvergenco.

Rezultati izvajanja genetskega algoritma so pokazali, da je kvaliteta rešitev precej odvisna od začetne populacije. Časovna zahtevnost vseh metod izbire je precejšna, za najpočasnejšo se izkaže metoda stohastičnega univerzalnega vzorčenja, za najhitrejšo pa metoda enoturnirske izbire.

Poglavje 6

Zaključek

V diplomski nalogi predstavimo delovanje genetskih algoritmov in problem usmerjanja prometa. Predstavimo naš pristop k reševanju problema in ga formaliziramo. Predstavimo implementacijo genetskih operatorjev in tehnologije uporabljene pri tem. Z analizo uspešnosti optimizacije predstavimo rezultate izvajanja genetskih algoritmov z različnimi nastavitvami in predstavimo ugotovitve.

Možne izboljšave

Genetske algoritme je zaradi posnemanja evolucije enostavno paralelizirati. Izboljšava, ki bi močno izboljšala hitrost izvajanja genetskega algoritma, je paralelizacija računanja kvalitete ene rešitve. To bi lahko naredili tako, da bi imeli en nadrejeni proces, ki bi hranil populacijo in izvajal genetske operatorje, osebke pa bi poslal podrejenim procesom. Edina naloga podrejenih procesov bi bila računanje kvalitete ene rešitve [13].

Za doseganje boljših rešitev bi lahko na mestih, kjer se kvaliteta najboljšega osebka nekaj generacij ne izboljšuje, povečali mutacijo. Tako bi dobili nove osebke s potencialno novo informacijo.

Literatura

[1] Igor kononenko, Marko Robnik Šikonja, *Inteligentni sistemi*, Založba FE in FRI, pogl. 12, 2010.

[2] Jiajia He, Zaien Hou, "Ant colony algorithm for traffic signal timing optimization", *Advances in Engineering Software*, št. 43, zv. 1, str. 14-18, 2012.

[3] Ciril Bohak, "HTML5", str. 3, 2010. Dostopno na: http://atlas.fri.uni-lj.si/html5-primeri/HTML5-Script.pdf

[4] (avgust 2012) Siamak Sarmady, *An Investigation on Genetic Algorithm Parameters*. Dostopno na: http://sarmady.com/siamak/papers/genetic-algorithm.pdf

[5] (avgust 2012) Genetic algorithm from Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Genetic_algorithm

[6] (julij 2012) Genetic Algorithms in Plain English. Dostopno na: http://www.ai-junkie.com/ga/intro/gat1.html

[7] (julij 2012) Izvorna koda stohastičnega univerzalnega vzorčenja v Javi. Dostopno na: https://github.com/dwdyer/watchmaker/blob/master/framework/src/java/main/org/uncommons/watchmaker/framework/selection/StochasticUniversalSampling.java

[8] (avgust 2012) HTML from Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Html

[9] (avgust 2012) HTML5 Rocks. Dostopno na: http://www.html5rocks.com/

[10] (avgust 2012) GitHub from Wikipedia, the free encyclopedia. Dostopno na: http://en.wikipedia.org/wiki/Github

[11] (avgust 2012) Roulette wheel selection. Dostopno na: http://www.edc.ncl.ac.uk/highlight/rhjanuary2007g02.php/

[12] (avgust 2012) Izvorna koda aplikacije. Dostopno na: https://github.com/Ivansek/thesis

[13] (avgust 2012) Erick Cantu-Paz, *A Survey of Parallel Genetic Algorithms*. Dostopno na: http://tracer.uc3m.es/tws/cEA/documents/cant98.pdf