Prirodom inspirirani algoritmi za planiranje kretanja robota

Korade, Dinko

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Dinko Korade

ZAGREB, 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Prirodom inspirirani algoritmi za planiranje kretanja robota

Voditelj rada:
Doc. dr. sc. Petar Ćurković

Dinko Korade

ZAGREB, 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

ZAVRŠNI ZADATAK

Student: **DINKO KORADE** Mat. br.:0035199393

Naslov rada na hrvatskom jeziku: **Prirodom inspirirani algoritmi za planiranje kretanja robota**

Naslov rada na engleskom jeziku: **Bio-inspired algorithms for robot motion planning**

Opis zadatka:

Problem planiranja kretanja robota u radnom prostoru koji sadrži prepreke težak je optimizacijski problem čak i u svojoj najjednostavnojoj formi. Kod primjene egzaktinih algoritama javlja se problem dugog vremena potrebnog za pronalazak rješenja, ili nemogućnost nalaska rješenja za složenije situacije.

U novije vrijeme heurističke metode optimiranja (evolucijski algoritmi, *swarm* algoritmi i sl.) temeljene na prirodom inspiriranim algoritmima (*bio-inspired algorithms*) pokazuju dobre rezultate u pronalaženju rješenja problema ovakvog tipa što je posljedica njihove robusnosti, stohastičke prirode, i rada s populacijom rješenja u slučaju evolucijskih algoritama.

U radu je potrebno napraviti literaturni pregled korištenja prirodom inspiriranih algoritama za rješavanje problema kretanja robota. Napraviti usporedbu karakteristika algoritama i njihovog uspjeha u rješavanju navedenog problema.

Dalje, potrebno je za pojednostavljeni model mobilnog robota pronaći optimirane trajektorije kretanja od unaprijed zadane početne do zadane konačne točke koristeći prirodom inspirirane algoritme.

Potrebno je napraviti grafički prikaz pronađenih trajektorija i usporediti više vrsta kodiranja trajektorije robota, kao i više različitih algoritama. Kritički se osvrnuti na dobivene rezultate, argumentirati najpogodniji oblik kodiranja trajektorije, kao i najpogodniji algoritam za rješavanje zadanog problema.

Zadatak zadan:
30. studenog 2017.

Zadatak zadao:

Doc. dr. sc. Petar Ćurković

Rok predaje rada:
1. rok: 23. veljače 2018.
2. rok (izvanredni): 28. lipnja 2018.
3. rok: 21. rujna 2018.

Predviđeni datumi obrane:
1. rok: 26.2. - 2.3. 2018.
2. rok (izvanredni): 2.7. 2018.
3. rok: 24.9. - 28.9. 2018.

Predsjednik Povjerenstva:

Izv. prof. dr. sc. Branko Bauer

IZJAVA

Izjavljujem da sam ovaj rad izradio samostalno služeći se stečenim znanjem i navedenom literaturom. Pri izradi ovoga rada korištena je i stručna pomoć mentora.

ZAHVALA

Zahvalio bih se svome mentoru dr. sc. Petru Ćurkoviću na stručnosti, dostupnosti i literaturi koju mi je omogućio za izradu ovoga rada.
Zahvalio bih se također svojoj obitelji na potpori koju mi je pružila tijekom studija.

Sadržaj

Popis slika

1. Uvod

Genetski algoritmi su algoritmi koji služe za pronalaženje rješenja različitih problema. Baziraju se na principima prirodne selekcije i gena, odnosno imitiranju prirodnih evolucijskih procesa. Sva se živa bića prilagođavaju različitim životnim uvjetima koje im okolina nameće. Tako u prirodi preživljavaju one jedinke koje su se najbolje prilagodile određenim promjenama. Slična se analogija odvija i u genetskim algoritmima, gdje fitness funkcija određuje najbolju jedinku za daljnju reprodukciju. Fitness funkcija specifična je za svaki problem, a njezin je zadatak da iz populacije jedinki prikaže onu koja sadrži najbolje rješenje. U genetskim algoritmima svaka jedinka prikazuje jedno rješenje problema i pronalazak optimalnog rješenja često nije najjednostavnije. Zato se iz svake populacije pomoću fitness funkcije biraju nabolje jedinke koje postanu roditelj i manipulacijom njihovog gena dobivaju se potomci koji čine novu populaciju. Taj se proces ponavlja sve dok se ne postigne optimalno rješenje.
Razvoj genetskih algoritama počelo 1960-ih i 70-ih radom Johna Hollanda i njegovih suradnika na sveučilištu Michigan. Njihova su istraživanja utemeljena na Darwinovoj teoriji prirodne selekcije. Holland je prvi koji je koristio križanja, rekombinacije i mutacije u primjeni umjetne inteligencije. To su osnovni alati koji pomažu pri određivanju optimalnih rješenja. Nakon Hollandovih istraživanja, genetski su se algoritmi nastavili razvijati i svoju su primjenu našli u širokom području problema kao na primjer problemu trgovačkog putnika, financijama ili u inženjerstvu.
Postoje određene razlike između genetskih algoritama i tradicionalnih načina optimiranja. Genetski algoritmi rade s čitavom populacijom rješenja, odnosno mogu vršiti istraživanja problema u različitim smjerovima zato što svaka jedinka predstavlja rješenje za sebe. Iz tog su razloga genetski algoritmi sposobni rješavati kompleksne probleme. Najveći je problem genteskih algoritama kreiranje fitness funkcije. Njezini parametri moraju biti pažljivo odabrani kako bi algoritam mogao konvergirati u optimalno rješenje. Ako fitness funkcija nije dobro odabrana, moguća su kriva rješenja. Osim fitness funkcije, na konvergenciju može utjecati i odabrana veličina populacije, kao i koeficijenti križanja i mutacije.

2. Teorija evolucije

Evolucija je naziv za proces u kojemu dolazi do promjene nasljednih osobina živih bića. Te se promjene ne odvijaju odmah, nego je potreban veliki broj generacija kako bi se one primijetile. Charles Darwin sredinom je 19. stoljeća postavio znanstvenu Teoriju evolucije. Ona ne objašnjava kako je život nastao, ali objašnjava svu raznolikost živog svijeta. Pomoću nje shvaćamo kako su se razni životni oblici prilagodili i najekstremnijim uvjetima života na Zemlji. U svakoj populaciji različite osobine variraju među jedinkama. Zbog tih razlika imamo različite stope preživljavanja i reprodukcije. Osobine se prenose s generacije na generaciju i zato u populacijama nalazimo više potomaka onih roditelja koji su bili prilagođeniji uvjetima življenja.
Prema dosadašnjim znanstvenim spoznajama i opažanjima, sva živa bića posjeduju nasljednu tvar u obliku gena. Te su gene nasljedili od svojih roditelja i prosljeđuju ih svojim potomcima. Među potomcima postoje genetske varijacije koje nastaju ili zbog mutacija DNK ili zbog rekombinacije postojećih gena tijekom mejoze. Zbog tih promjena postoje i razlike između roditelja i potomaka. Ako su promjene korisne, potomci imaju veće šanse za preživljavanje i očuvanje vrste. To će dovesti do toga da svaka sljedeća generacija posjeduje te korisne promjene. No, promjene ne moraju uvijek biti korisne. Može se dogoditi da promjena nije dobra i da vodi određenu vrstu prema izumiranju.
Genotip je genska osnovica organizma, odnosno svi geni koji čine organizam. Genotip je svojstvo koje organizam ima kao potencijal, što znači da na njegovo oblikovanje utječe okolina u kojoj se nalazi. Fenotip organizma čine sve njegove vidiljive karakteristike i osobine. On je rezultat genetskog koda (genotipa) i interakcije organizma s okolinom. Tako na primjer netko može imati svijetlu kožu, ali čestim izlaganjem suncu koža postane tamnija. Sve te osobine prosljeđuju se iz generacije u generaciju putem DNK molekule koja nosi kodiranu genetsku informaciju. DNK se sastoji od četiri tipa baze. Redoslijed baza određuje genetsku informaciju. DNK zajedno s proteinima čini kromosome. Svaka jedinka sadrži $2n$ broja kromosoma, što znači da svaki roditelj daje n broja kromosoma svom potomku. Ta se pojava odvija prilikom sinteze spolnih stanica staničnom diobom koja se zove mejoza. Za vrijeme mejoze dolazi do razdvajanja i udvostručavanja kromosoma. Zatim slijedi križanje i nastanak novih kromosoma, odnosno nove nasljedne tvari. Osim križanja moguće su i mutacije. Mutacije predstavljaju mogućnost slučajne promjene gena. Trajne su i nasljedne ako se dogode u spolnoj stanici. Mogu biti korisne i olakšati prilagodbu organizma nekom okolišu, ali mogu biti i loše ako se odnose na nasljedne bolesti i slično.

3. Genetski algoritam

U prethodnom je poglavlju ukratko objašnjena Teorija evolucije iz razloga što je ona osnova Genetskih algoritama. Iako Genetski algortimi ne pripadaju prirodnim znanostima, oni koriste principe evolucije u optimiranju različitih problema. Istraživanja su pokazala da se najbolja rješenja dobiju korištenjem binarnog zapisa iako to nije moguće u svakom problemu. Rješavanje problema počinje generiranjem početne populacije. Jedinke u populaciji na početku mogu biti iste, ali se preporučuje da sve budu različite zbog raznolikosti u potrazi za rješenjem. Nakon generiranja početne populacije, potrebno je pomoću fitness funckije odabrati najbolje jedinke kao roditelje koje će se koristiti za reprodukciju. Nakon odabira roditelja slijedi križanje i mutacija, te dobivanje nove populacije. U ovom će poglavlju biti objašnjen svaki postupak zasebno.

3.1. Generiranje populacije

Prvi korak u rješavanju problema je generiranje populacije. Najlakši način je korištenjem binarnog zapisa. Binarni zapis nije uvijek moguće koristiti za sve vrste problema, ali se pokazao kao najučinkovitiji. Početna se populacija generira slučajnim odabirom, ali postoji i uniformna populacija (sve su jedinke iste na početku). Uniformna populacija nije učinkovita, te se ne preporučuje.

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}. \tag{3.1}$$

Izraz 3.1 prikazuje jednu slučajno generiranu populaciju. Iz te se populacije odabiru najbolje jedinke koje će biti roditelji i iz njih ide reprodukcija nove populacije. Radi lakšeg shvaćanja odabira roditelja, primjerom će se objasniti postupak.

No.	Jedinka	Fitness	Relativni fitness
1.	0 1 1 1 1	20	0.2
2.	1 0 0 1 1	30	0.3
3.	1 1 1 0 0	25	0.25
4.	1 0 1 0 1	25	0.25
Suma		100	1.0

Slika 1. Tablica odabira roditelja

U tablici vidimo da je populacija podijeljena na jedinke. Svaka jedinka ima svoju vrijednost fitnessa i svoju relativnu vrijednost fitnessa u usporedbi s drugim jedinkama. O fitness funkciji ćemo detaljnije govoriti kasnije u potpoglavlju 3.2. Fitness funkcija je neka funkcija f, a relativnu funkciju dobijemo kada zbroj fitnessa svih jedinki zbrojimo u sumu i s tom sumom podijelimo fitness svake jedinke pojedinačno. Iz tablice vidimo kako jedinka pod rednim brojem 2 ima najbolji fitness.

$$Suma = \sum_{i}^{n} f_i \qquad Frel_i = \frac{f_i}{Suma} \tag{3.2}$$

3.2. Fitness funkcija ili Funkcija dobrote

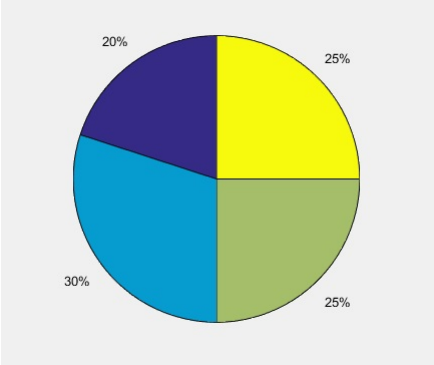
Fitness funkcija ili funkcija dobrote ocjenjuje kvalitetu (dobrotu) jedinke u populaciji. Ona je ključni segment genetskih algoritama zato što se preko nje odabiru jedinke koje će se koristiti u reprodukciji. Ako fitness funkcija nije dobro određena, program će konvergirati u krivo rješenje ili neće uopće konvergirati, te treba biti oprezan s njezinim definiranjem. Fitness funkcija nema definiran oblik nego se konstruira ovisno u vrsti problema, tako da njezinim korištenjem dođemo do optimalnog rješenja.

3.3. Selekcija jedinki

Nakon generiranja populacije i određivanja fitnessa svake jedinke, ostalo nam je izabrati jedinke koje će poslužiti kao roditelji za novu populaciju. Logično bi bilo uzeti jedinke s najboljim fitnessom, ali onda bi došlo do prebrze konvergencije i točnost rješenja bi bila upitna. Da bi se omogućila genetska raznolikost, trebalo je razviti sistem biranja roditelja u kojem će i lošije jedinke imati priliku postati roditeljima jer i one imaju neka dobra svojstva. Zato su razvijeni sistemi koji veću mogućnost za prolazak među roditelje daju boljim jedinkama, ali isto tako slabijim jedinkama omogućuju mogućnost za preživljavanje.

3.3.1. Jednostavna selekcija ili Ruletno pravilo

Kod Ruletnog pravila vjerojatnost izbora jedinke ovisi o iznosu dobrote za tu jedniku. Dobrota jedinke određuje se preko fitness funkcije te se iz jednadžbi (3.2) određuje relativni fitness za svaku jedinku.



Slika 2. Ruletno pravilo

Da bi lakše objasnili ovo pravilo, na slici 2. prikazani su podaci dobiveni u tablici na slici 1. Prvo se generira slučajan broj između 0 i 1 koji će odrediti koja jedinka ide u roditeljsku populaciju. Jedinka u čijem se intervalu nalazi generirani broj izabrana je za križanje. Na primjer, ako je generirani broj 0.4, to znači da jedinka koja zauzima **zeleni** dio kruga prolazi dalje zato što je taj broj u njezinom intervalu na krugu. Ovaj se postupak ponavlja sve dok se ne generira populacija roditelja koja je jednake veličine kao i početna populacija.

3.3.2. Turnirska selekcija

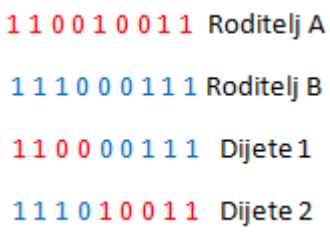
Turnirska selekcija za generiranje nove populacija odabire nekoliko jedinki iz stare populacije i među njima na temelju fitness funkcije odabire najbolju. Postupak se provodi sve dok se ne kompletira nova populacija roditelja koja će sudjelovati u križanju i mutaciji. Ova vrsta selekcije također omogućava preživljavanje slabijim jedinkama.

3.4. Križanje

Križanje je genetski operator koji imitra križanje iz prirode. U njemu sudjeluju dvije jedinke koje su prošle selekciju i iz njih nastaju jedna ili dvije nove jedinke. S obzirom na to da je nova jedinka nastala od roditelja s dobrim svojstvima, svojstva bi nove jedinke trebala isto biti dobra, ako ne i bolja od roditelja.

3.4.1. Križanje s jednom točkom prekida

Križanje s jednom točkom prekida najjednostavniji je način križanja. Nasumično se izabire jedan broj koji označava točku loma. Prvi se kromosom lomi u izabranoj točki i spaja se s drugim dijelom drugog kromosoma koji se lomi u istoj točki. Drugi dio prvog kromosoma se analogno spaja s prvim dijelom drugog kromosoma.



Slika 3. Križanje s jednom točkom prekida

3.4.2. Križanje s više točaka prekida

Križanje s više točaka prekida odvija se na sličan način kao i križanje s jednom točkom prekida. Razlika je u tome što se u ovome križanju kromosom lomi na više točaka, a ne u samo jednoj.

Slika 4. Križanje s više točaka prekida

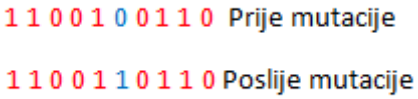
3.4.3. Uniformno križanje

Kod uniformnog se križanja dijelovi kromosoma ne spajaju međusobno nego se za svaki gen pojedinačno gleda hoće li doći od roditelja A ili od roditelja B. U uniformnom križanju mogućnost za dobivanjem gena od pojedinog roditelja je jednaka (50%), dok se križanje kod kojega je taj omjer drugačiji naziva P-uniformno križanje

Slika 5. Uniformno križanje

3.5. Mutacija

Nakon križanja dolazimo do mutacija. Mutacija djeluje na jedan kromosom i izmjenjuje gene u njemu. Ona može unjeti novo svojstvo ili obnoviti izgubljeno svojstvo. Njezina vrijednost nije velika jer kad bi mutirali svi geni u kromosomu, problem bi se sveo na slučajno generirane brojeva.



Slika 6. Primjer mutacije

Na slici 6. prikazan je primjer jednostavne mutacije. To znači da se mijenja samo jedan gen u kromosomu. Moguće su mutacije gdje se mijenja više gena u kromosomu ili čak određeni niz gena. Mutacije služe kako bi se izbjegao lokalni maksimum ili minimum u pretraživanju rješenja. Kada bi cijela populacija zapela u jednom od lokalnih ekstrema, onda bi mutacija bila jedina mogućnost za bolje rješenje.

3.6. Elitizam

Postoji mogućnost da se nakon više iteracija zbog posljedica selekcije i mutacija najbolje rješenje izgubi. To dovodi do povećanja vremena potrage optimalnog rješenja ili se može dogoditi da rješenje ne konvergira. Iz toga razloga najbolje rješenje treba zaštiti, a taj se postupak zove elitizam. Elitizam čuva najbolju jedinku te je uvijek stavlja kao dio sljedeće generacije. Jedinka elitizma se mijenja samo u slučaju da je pronađena nova još bolja jedinka. Na taj način čuvamo rješenje od nepogodnih utjecaja selekcije i mutacija, te program uvijek teži optimalnom rješenju. Nedostatak elitizma je u tome što zauzima dodatan prostor u programu, ali uz današnju tehnologiju ne usporava značajno izvođenje samog programa.

3.7. Parametri algoritma

Svaki algoritam sadrži parametre bez kojih ne može davati točna rješenja. Parametri u genetskom algoritmu su: veličina populacije, broj generacija (iteracija), koeficijent križanja i koeficijent mutacije. Glavni je problem što nikada ne znamo koje su optimalne vrijednosti tih parametara. Za različite vrijednosti dobivamo različite rezultate, a to se još očituje i u brzini konvergencije programa, kao i u točnosti samog rješenja. Njih odabiremo na temelju iskustva i provedenih eksperimenata, te koristimo one za koje dobivamo najbolja rješenja.

4. Pseudokod

Pseudokod, premda nalikuje računalnom programu, zapravo nije napisan u programskom jeziku koji bi se mogao upotrijebiti na računalu. Pseudokod je skup kratkih izraza koji opisuju pojedine dijelove algoritma. U ovom će poglavlju pomoću pseudokoda biti objašnjen program za kretanje robota, dok će njegov puni oblik u programskom jeziku MATLAB biti prikazan u prilogu ovog seminara.

begin
Funkcija za koju se traži rješenje f(x), x=(x_1,....,x_n)^T
Prikazati rješenje u obliku kromosoma(binarni zapis)
Definiranje fitnessa F
Generirati početnu populaciju
Odrediti koeficijente križanja (pc) and mutacije(pm)
while (t<Maksimalni broj generacija)
Generirat novo rješenje pomoću križanja i mutacija
if pc>rand, Križanje; **end if**
if pm>rand, Mutacija; **end if**
Prihvatiti novo rješenje ako se njegov fitness povećao
Izabrati najbolju jedinku za novu generaciju(elitizam)
end while
Dešifriranje rezultata i vizualna obrada podataka
end

Slika 7. Pseudokod

Na slici 7. prikazan je pseudokod genetskog algoritma. U njemu su sadržani svi dijelovi koje kod mora sadržavati kako bi se mogla postići rješenja određenog problema. Sljedeći je korak objasniti pseudokod na primjeru kretanja robota kroz labirint.

4.1. Inicijalizacija programa

Prva stvar u programu je definicija problema koji se rješava. U pseudokodu na slici 7. prvi je korak definiranje funkcije za koju se traži rješenje. Kod kretanja robota ta funkcija znači definiranje labirinta iz kojeg robot mora izaći. Također, definira se početna pozicija robota, kao i pozicija izlaza iz labirinta. Nakon što je problem zadan, potencijalna se rješenja zapisuju u obliku kromosoma. U pseudokodu je navedeno da se za definiranje rješenja najbolje poslužiti binarnim zapisom (*binary strings*), ali moguće su i modifikacije tog zapisa. Sljedeći je korak definiranje fitness funkcije F pomoću koje se traži optimalno rješenje. To je jedan od najzahtjevniji zadataka samog programa jer ako finess funkcija nije dobra, program neće dobro raditi. Nakon određivanja fitnessa, generira se početna populacije rješenja i unose se koeficijenti križanja i mutacije.

4.2. Tijek programa

Nakon zadavanja svih potrebnih parametara programa prelazimo na traženje najboljeg rješenja. U pseudokodu je zapisana **while** petlja, ali može se koristiti i neka druga, samo ju treba prilagoditi problemu. U **while** petlji odvija se dio programa koji traži optimalno rješenje, a prvo se generiraju nova rješenja pomoću križanja i mutacija. Najbolje se rješenje uvijek čuva pomoću elitizma, a sva ostala idu dalje u novu populaciju, te se postupak križanja i mutacija ponavlja sve dok nije zadovoljen određeni uvjet.

4.3. Rezultati i analiza

Nakon određenog broja iteracija dobiveni se rezultati dekodiraju iz binarnog zapisa i vrši se zapis rezultata, njihov grafički prikaz i daljnja analiza.

5. Program

U 4. je poglavlju pomoću pseudokoda opisan tijek programa koji koristi genetski algoritam. To je samo jedan od primjera kako program može izgledati. Naravno, moguće su razne varijacije i svaki programer svoj kod prilagodi sebi. Prije analize rezultata, u ovom će poglavlju ukratko biti opisani najbitniji dijelovi programa rađenog u MATLAB-u kako bi čitatelj imao bolji osjećaj i uvid u rezultate i analizu. U prilogu na kraju rada bit će napisan kompletan program s komentarima koji služe kao objašnjenja.

5.1. Problem labirinta

Problem labirinta ili *Maze solving algorithm* jedan je od osnovnih problema umjetne inteligencije. Na prvi se pogled čini da je problem jednostavan, ali to nije baš tako. Za rješavanje ovog problema stručnjaci su razvili različite metode i algoritme kao: *Shortest path algorithm, Dead-end filling, Wall follower* i druge. Problem se sastoji od labirinta koji može biti zadan nasumično ili generiran prema vlasitim željama. Labirint se sastoji od zidova i prepreka, a početna točka kretanja i izlaz iz labirinta su definirani. Cilj problema je pomoću genetskih algoritama naučiti robota da pronađe put od početne točke do izlaza iz labirinta koristeći optimalni broj koraka. Kako bi se problem još malo zakomplicirao, robot mora izbjegavati sudare sa zidovima i preprekama tako da svaki korak koji radi bude korak bliže cilju.

5.2. Definiranje labirinta

Kao što je opisano u pseudokodu, prvi zadatak u programu je definirati sve parametre problema koji rješavamo.

```
D=[9 9 9 9 9 9 9 9 9;
   9 9 0 0 0 9 9 9 9;
   9 0 0 9 0 9 9 9 9;
   9 0 9 9 0 0 9 9 9;
   9 0 0 9 9 0 9 9 9;
   9 0 9 9 0 0 9 9 9;
   9 0 9 9 0 9 9 9 9;
   9 0 9 9 0 9 9 9 9;
   9 9 9 9 9 9 9 9 9;]; %Labirint
```

Slika 8. Labirint

Slika 8. prikazuje labirint iz kojeg robot treba naći izlaz. S obzirom na to da program radi pomoću matričnog zapisa, na taj je način prikazan i labirint. Broj 9 simbolizira prepreke i zidove koje robot mora izbjegavati, dok broj 0 znači slobodan prolaz. Osim labirinta, potrebno je još zadati i početnu poziciju robota kao i izlaz iz labirinta.

$$\begin{bmatrix} 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \\ 9 & 9 & 0 & 0 & 0 & 9 & 9 & 9 & 9 \\ 9 & 0 & 0 & 9 & 0 & 9 & 9 & 9 & 9 \\ 9 & 0 & 9 & 9 & 0 & 0 & 9 & 9 & 9 \\ 9 & 0 & 0 & 9 & 9 & 0 & 9 & 9 & 9 \\ 9 & 0 & 9 & 9 & 0 & 0 & 9 & 9 & 9 \\ 9 & 0 & 9 & 9 & 0 & 9 & 9 & 9 & 9 \\ 9 & 0 & 9 & 9 & 7 & 9 & 9 & 9 & 9 \\ 9 & 8 & 9 & 9 & 9 & 9 & 9 & 9 & 9 \end{bmatrix}$$

Slika 9. Konačni labirint

Na slici 9. broj 7 označava početnu poziciju robota, a broj 8 označava izlaz iz labirinta. S tim podacima definiran je zadatak koji se rješava.

5.3. Početne varijable

Nakon što je definiran zadatak, potrebno je definirati u kojem će obliku biti zapisana potencijalna rješenja. Iako se u genetskim algoritmima preporučuje binarni zapis rješenja, odlučili smo malo pojednostaviti zapis i koristiti brojeve 1, 2, 3 i 4. Broj 1 označava gibanje prema gore, broj 2 prema dolje, broj 3 gibanje u desno i broj 4 gibanje u lijevo.

```
B=input('Broj kromosoma: ');
A=input('Dužina kromosoma: ');
L=input('Broj iteracija: ');
count=0;

pc=0.7;
pm=0.3;

Populacija=randi(4,B,A);
```

Slika 10. Definiranje početnih varijabli

Na slici 10. prikazana je inicijalizacija početne populacije. Parametar **B** određuje koliko će kromosoma biti u populaciji, a parametar **A** određuje koliko će koraka robot napraviti prije nego što se zaustavi. Osim populacije, unosi se broj iteracija (uvjet za **while** petlju u pseudokodu) koji želimo da program izvrti. Za kraj se unose koeficijenti križanja i mutacije.

5.4. Fitness funkcija

Zadnji je korak prije glavnog dijela programa određivanje fitness funkcije. Fitness funkcija je određena na osnovi analize i eksperimentiranja, te je odabrana ona koja je prikazivala najbolje rezultate.

$$F(i) = \frac{Udaljenost\_od\_starta(i)}{Udaljenost(i) + 0.1} + \frac{Korak\_sudara(i)}{20} + \frac{1}{Koraci(i)} \tag{5.3}$$

Jednadžba 5.3 prikazuje fitness funkciju. Funkcija sadrži parametre: *Udaljenost_od_starta*, *Udaljenost*, *Korak_sudara* i *Koraci*. Svaki kromosom u populaciji sadrži te parametre, te kad se oni uvrste u fitness funkciju dobije se ”dobrota” određenog kromosoma. Tim se postupkom procjenjuje kvaliteta pojedinog rješnja. U ovom je zadatku potrebno maksimizirati funkciju, što znači - što je veći fitness, to je bolje rješenje.

5.5. Glavni dio programa

Nakon što je definiran zadatak i svi parametri koji nam pomažu u rješavanju istog, slijedi glavni dio programa. U njemu se pomoću odabrane petlje (for, while) generiraju nove generacije s potencijalnim rješenjima. Tu se odvijaju postupci križanja i mutacija, kao i ocjenjivanje najboljeg fitnessa, te njegovo čuvanje pomoću elitizma. Nakon što prođe zadani broj generacija, analizira se najbolje rješenje.

6. Rezultati i analiza

U prethodnim smo poglavljima objasnili principe na kojima funkcioniraju genetski algoritmi. Sada ćemo stečena znanja iskoristiti kako bismo pronašli najbolju putanju za kretanje robota u labirintu. Program je napisan u programu MATLAB. Kao što je i prije navedeno, cijeli se program nalazi na kraju ovog rada u **Prilogu**. Za analizu su korištene tri vrste labirinta, te će se njihova zahtjevnost mijenjati od lakšega prema težemu. Svaki je labirint kreiran prema vlastitom nahođenju. U pravljenju ovoga rada korišteno je više vrsta labirinta, ali odabrani su oni najzanimljiviji. Svaki korisnik programa može kreirati vlastiti labirinit, ali se mora držati pravila koja su navedena u programu.

6.1. Fitness funkcija

Za rješavanje problema potrebno je definirati fitness funkciju. Koristit će se dvije funkcije, a one su:

$$F(i) = \frac{Udaljenost\_od\_starta(i)}{Udaljenost(i) + 0.1} + \frac{Korak\_sudara(i)}{20} + \frac{1}{Koraci(i)} \tag{6.4}$$

$$F(i) = \frac{1}{Udaljenost(i) + 0.1} + \frac{Korak\_sudara(i)}{20} + \frac{Udaljenost\_od\_starta(i)}{20} + \frac{1}{Koraci(i)} \tag{6.5}$$

U jednadžbama vidimo da fitness funkcije sadrže 4 parametra. Parametar *Udaljenost* mjeri koliko je robot udaljen od cilja odnosno od izlaska iz labirinta. Kako bi se ta udaljenost izračunala koristi se **Manhattan distance**. Ona se bazira kao zbroj horizontalne i veritaklne udaljenosti robota od cilja. S obzirom na to da robot nema dijagonalnih pomaka, ova je metoda učinkovita. Sljedeći parametar *Udaljenost_od_starta* tjera robota na kretanje. Postoje primjeri u kojima je robot postavljen blizu cilja, ali je problem što do njega ne može doći u malom broju koraka, nego mora ići okolnim putem. Tada ovaj parametar tjera robota da traži alternativna rješenja - da se ne vrti u krug. Treći parametar je *Korak_sudara*. On služi kako bi spriječio robota da radi sudare s preprekama i zidovima. Zadnja varijabla su *Koraci*. *Koraci* broje koliko je koraka trebalo robotu da dode do cilja. Ona služi kada imamo slučaj da robot radi više koraka od optimalnog broja. Na primjer, kada je robotu do cilja potrebno 10 koraka, a mi mu zadamo 12, taj parametar služi kako bi robot od 12 mogućih koraka iskoristio optimalnih 10 i došao do cilja.

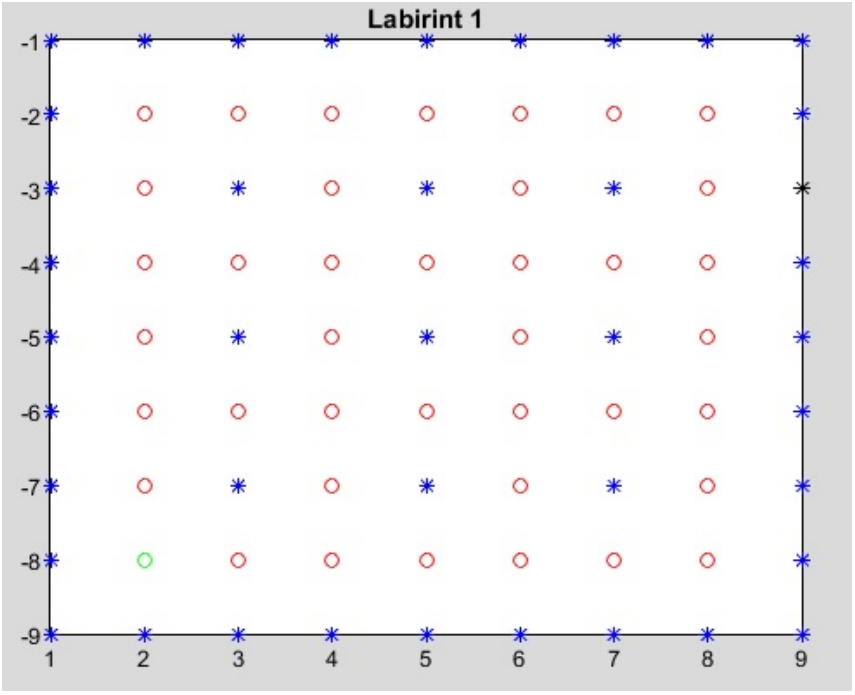
6.2. Križanja i mutacije

Nakon prikazanih fitness funkcija potrebno je odabrati koja će se vrsta križanja i mutacija koristiti, kao i njihovi koeficijenti. Na početku je testiranja korišteno križanje s jednom točkom prekida. To je križanje kontradiktorno problemu kretanja robota iz sljedećeg razloga; rješenje ovog programa zadano je nizom brojeva koji predstavljaju kretanje robota, na primjer (1 1 2 3 3 2 4 1 4 4). Prvi brojevi ovog niza predstavljaju kretanje robota na početku njegovog gibanja i ovom vrstom križanja ti koraci (pod pretpostavkom da je gibanje točno) odlaze na neko drugo mjesto u kromosomu što automatski uništava fitness te jedinke. Zato koristimo P-uniformno križanje gdje se geni između kromosoma križaju po pozicijama, što je za ovaj problem puno djelotvornije. Koeficijenti križanja su inače visoki. Najčešće se pojavljuju u rasponu između 0.7 i 1. Niski faktori nisu korisni za evoluciju jer onda nema miješanja gena između jedinki. U ovom programu koristimo faktor 0.7 kako bi poticali križanje između jediniki, a opet omogućili opstanak jedinkama kojima je potrebna jedna mutacija kako bi postigli željeno rješenje.

Poslije križanja slijedi mutacija. U programu se koristi mutacija koja može mijenjati više gena u kromosomu. Faktori mutacije su izrazito niski, te se obično kreću oko 0.05. Međutim, za ovaj problem su korisni malo viši faktori mutacije koji se kreću između 0.1-0.3. Pri tim vrijednostima mutacija nije toliko agresivna da kvari rješenje, a povećava mogućnost da popravi fitness jedinke koja sadrži jedan gen koji ju kvari.

6.3. Labirint 1

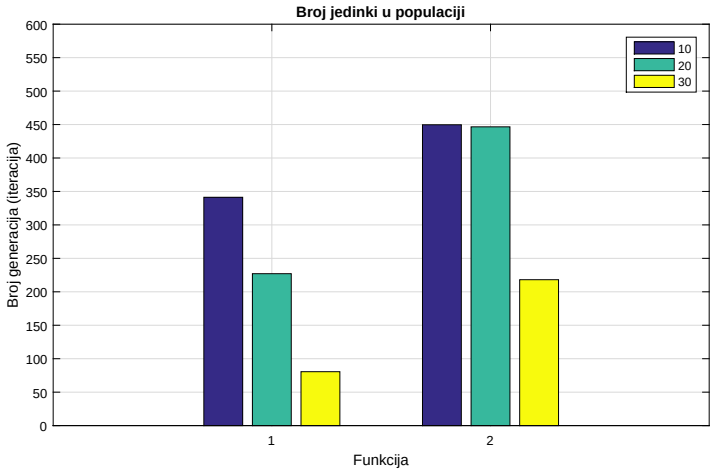
Na slici 11. prikazan je **labirint 1.** On će nam poslužiti kako bi testirali ispravnost fitness funkcija. Usporedit ćemo točnost rezultata kao i brzinu konvergencije. Odredit ćemo još jedan važan faktor u ovom algoritmu, a to je veličina populacije.



Slika 11. Labirint 1

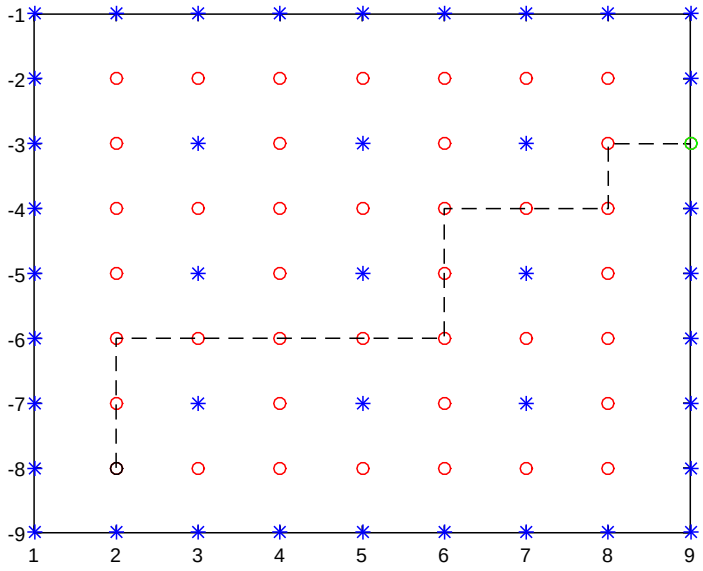
Prvo da objasnimo strukturu labirinta: plave zvijezdice (*) predstavljaju zidove i prepreke, dok crveni kružići (○) predstavljaju slobodan prolaz, zeleni kružić označava poziciju robota, a crna zvijezdica izlaz iz labirinta. Labirint nije zahtjevan jer ne sadrži veliki broj prepreka, ali je idealan za testiranje primarne funkcije zadatka, a to je izlaz iz labirinta. U ovom slučaju robotu je potrebno 12 koraka kako bi našao izlaz. Testirat ćemo dvije fitnes funkcije s različitim veličinama populacije i analizirati rezultate.

Rezultati su dobiveni na temelju 20 uzoraka za svaku populaciju. Korištene su populacije od 10, 20 i 30 jedinki. Da bi robot došao do cilja potrebno je 12 koraka, pa je svaka jedinka sadržavala 12 gena tako da robot mora pronaći optimalno rješenje inače će ostati u labirintu.



Slika 12. Dijagram veličine populacije

Slika 12. prikazuje prosječan broj generacija potrebnih za pronalazak rješenja, odnosno izlaza iz labirinta u ovisnosti o veličini populacije. Funkcija 1 predstavlja izraz (7.4), dok funkcija 2 predstavlja izraz (7.5). Iz grafa je vidljivo da obje funkcije uspješno obavljaju zadatak i pronalaze izlaz iz labirinta. Funkcija 1 postigla je bolje rezultate i puno brže konvergira u točno rješenje. Kod populacije od 10 jedinki za pronalazak rješenja trebala je oko 341 iteracija, dok je kod populacije od 30 jedinki taj broj pao na 80 iteracija. S druge pak strane kod funkcije 2 porast populacije s 10 na 20 jedinki nije donio ubrzanje u konvergenciji već se ono nastavila kretati na oko 446 iteracija. Tek kada smo populaciju povećali na 30 jedinki broj iteracija se smanjio na 218. Valja napomenuti kako veći broj jedinki u populaciji povećava raznolikost rješenja i u pravilu dovodi do brže konvergencije. Nedostatak kod korištenja veće populacije je što jako usporava izvođenje programa. Zato kod testiranja nije korištena veća populacija od 30 jedinki.

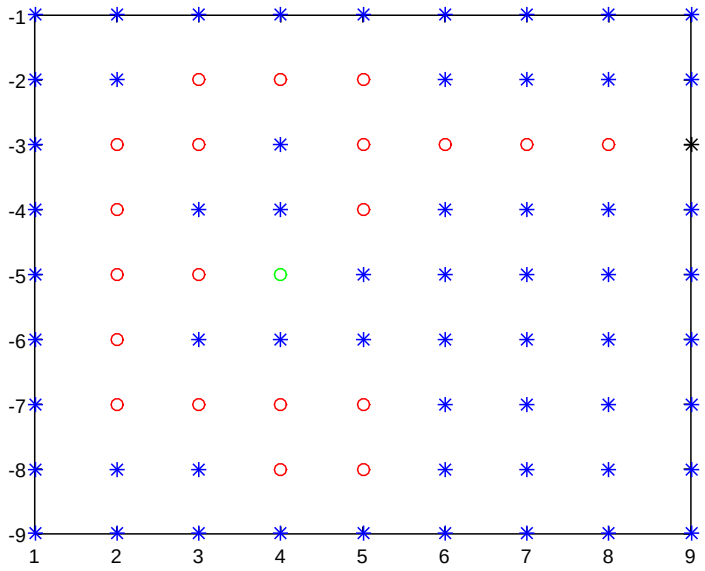


Slika 13. Putanja robota za labirint 1

Na slici 13. možemo vidjeti putanju koju je robot koristio kako bi došao do izlaza. Crtice predstavljaju njegovo kretanje. U ovom labirintu postoji više mogućnosti izlaska iz labirinta tako da ova putanja nije jedino moguće rješenje.

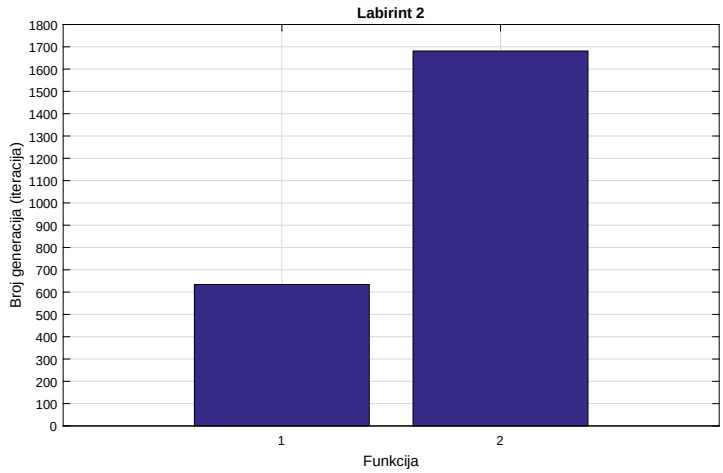
6.4. Labirint 2

Labirint 2 zahtjevniji je od prvog primjera. Da bi robot našao izlaz potrebno je 13 koraka. Sadrži više zidova i prepreka, te postoji samo jedna putanja koja vodi do izlaza. Na temelju rezultata dobivenih u prvom primjeru za labirint 2 ćemo koristiti populacije od 30 jedinki.



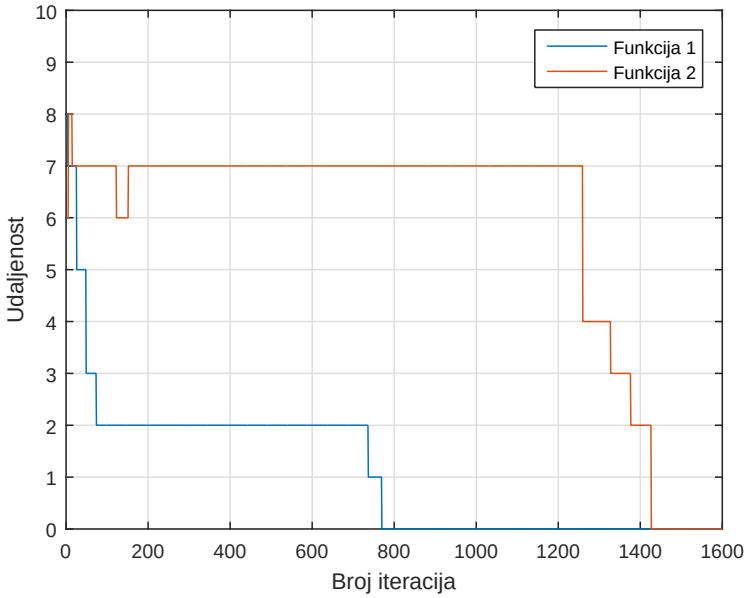
Slika 14. Labirint 2

Slika 14. prikazuje labirint 2. Analogija oznaka je ista kao i za prethodni primjer.



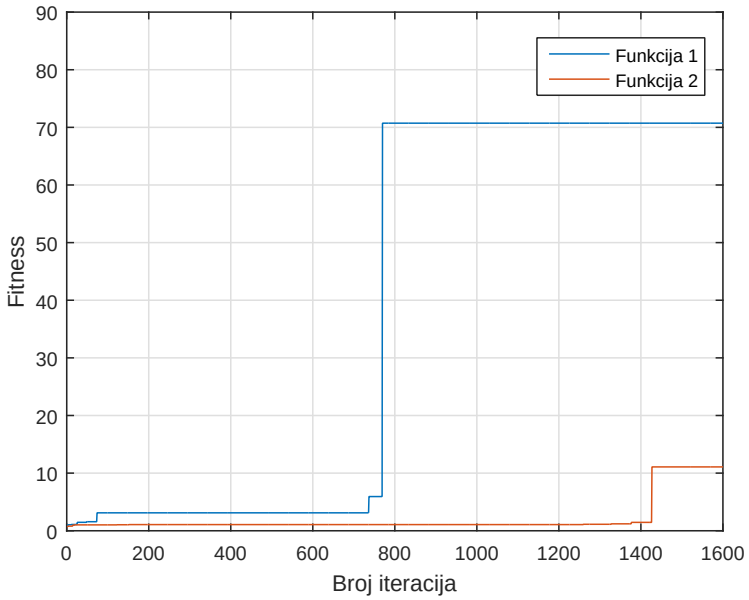
Slika 15. Dijagrami konvergencije

Dijagram na slici 15. prikazuje rezultate dobivene na temelju uzorka od 20 mjerenja. Vidljivo je da funkcija 1 opet brže konvergira od funkcije 2. Labirint 2 kompliciraniji je nego prvi primjer, te je razlika ovdje puno primjetnija. Prosječan broj iteracija za prvu funkciju je 634, dok je za drugu 1681. Sada sa sigurnošću možemo potvrditi da je prva funkcija bolja. Da ne bismo analizirali samo broj iteracija, prikazat ćemo dijagrame fitness funkcija kao i kretanje veličina određenih parametara funkcije tijekom iteracija.



Slika 16. Dijagram udaljenosti

Na slici 16. je dijagram koji prikazuje udaljenosti robota od cilja po iteracijama. S obzirom na to da udaljenost između robota i cilja mora biti nula, taj parametar treba minimizirati. Treba primijetiti kako udaljenost ipak ne pada monotono po iteracijama, nego u početnim iteracijama udaljenosti variraju gore - dolje. Razlog je u tome što *Udaljenost* nije jedini parametar u funkciji. Neka jedinka može biti udaljenija od cilja od prethodne, ali nema sudar u svome gibanju, pa njezin fitness ispadne bolji od one koja je bliže cilju, a radi sudar s preprekom na nekom koraku.

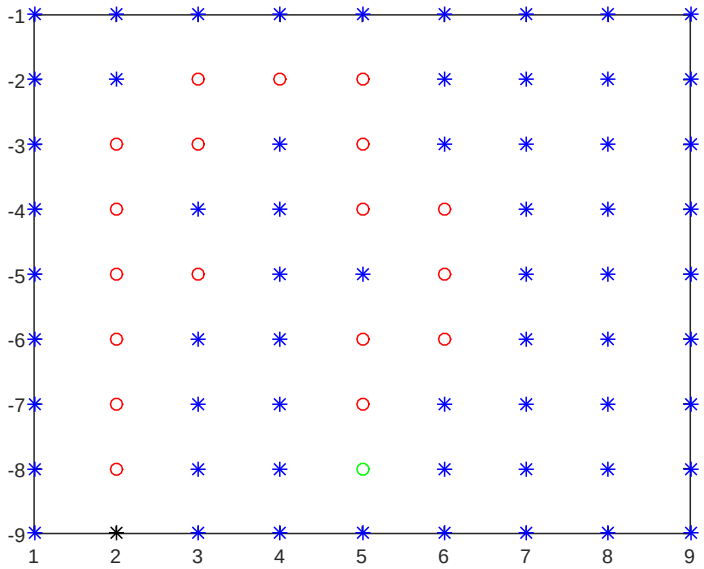


Slika 17. Dijagram fitnessa

Na slici 17. prikazan je rast fitness funkcija tijekom iteracija. Njihove vrijednosti nisu iste jer ni funkcije nisu iste, ali se jasno vidi skok u iznosima funkcija kada pronađu rješenje. Opet je vidljivo, kao i na drugim dijagramima, kako funkcija 1 brže pronalazi rješenje od funkcije 2. Fitness funkcije, za razliku od parametara koje ju čine, moraju biti monotone zato što program sadrži **elitizam**. U slučaju da program sadrži elitizam, a fitness funkcija nije monotona, program najvjerojatnije sadrži neku grešku koju treba ispraviti.

6.5. Labirint 3

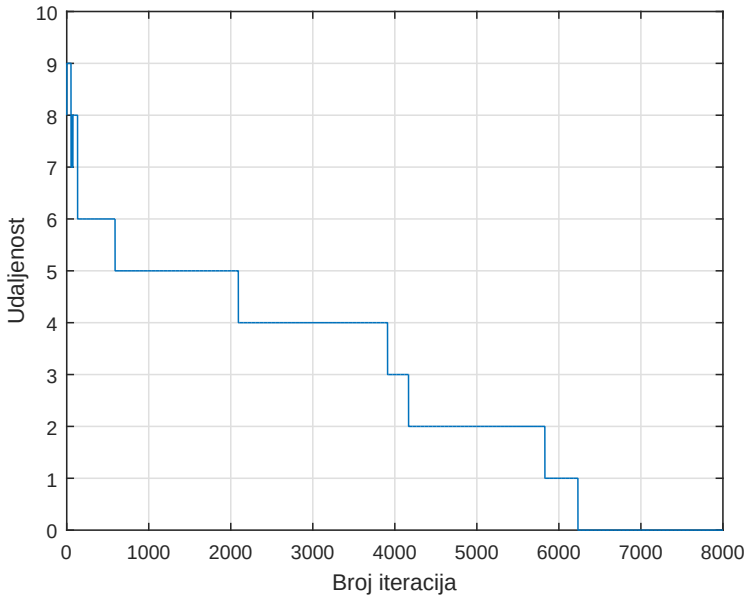
Labirint 3 je specifičan iz više razloga. Za početak, da bi robot došao do cilja mora napraviti 18 koraka, što je najviše od svih labirinata do sada. Druga stvar je što se robot na početku po Manhattan udaljenosti nalazi samo 4 koraka do izlaza. Naravno, ta 4 koraka ne može napraviti zbog zidova, nego mora ići okolnim putem. Zbog kompleksnosti problema, za ovaj će se primjer koristiti samo prva fitness funkcija.



Slika 18. Labirint 3

Problem ćemo pokušati riješiti na dva načina. Prvi način je da program pokuša naći rješenje u točno 18 koraka, dok je drugi način da robotu omogućimo fleksibilnost i dopustimo mu više koraka od optimalnog broja.

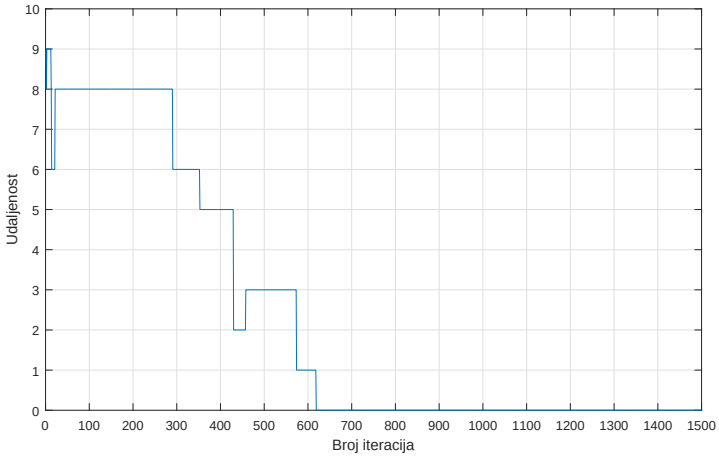
6.5.1. Prva metoda



Slika 19. Udaljenost u labirintu 3 (Prva metoda)

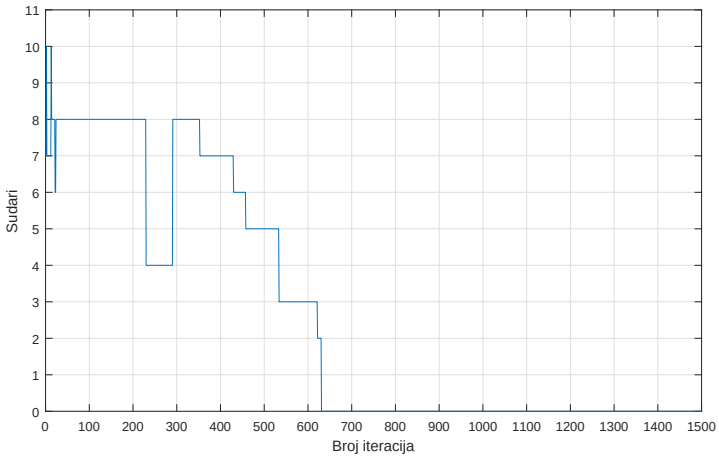
Na slici 19. vidimo kretanje udaljenosti robota kada mu zadamo da problem mora rješiti u optimalnih 18 koraka. Prvi je problem što robot u tom slučaju jako teško nalazi put do cilja, a i kada ga nađe, to bude u velikom broju iteracija. Na dijagramu vidimo da mu je bilo potrebno preko 6000 iteracija što ovaj proces čini jako sporim. Druga je stvar to što su za ovakve probleme potrebne i veće populacije kako bi se postigla što veća raznolikost rješenja. Veće populacije još više usporavaju program. Iz toga razloga pokušat ćemo dobiti rješenje drugom metodom, a ta je da povećamo broj koraka koje robot smije napraviti. Tim smo postupkom omogućili robotu fleksibilnost u kretanju i mogućnost da brže nađe rješenje.

6.5.2. Druga metoda



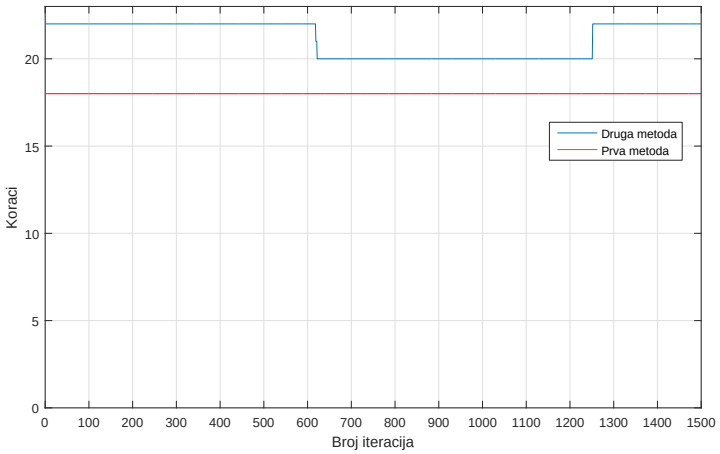
Slika 20. Udaljenost u labirintu 3 (Druga metoda)

Na slici 20. vidimo da davanje robotu veće fleksibilnosti u kretanju dovodi do toga da puno brže nađe rješnje, iako je problem kompleksiniji. Zbog većeg broja koraka, veće su šanse da robot napravi sudar s preprekom ili zidom. Dijagram koji analizira sudare robota vidimo na slici 21.



Slika 21. Sudari u labirintu 3 (Drugom metodom)

Na slici vidimo da robot u svom gibanju nema sudara nego se uspješno kreće kroz labirint dok ne nađe izlaz. U ovom slučaju, robotu su omogućena 22 koraka što je 4 više od optimalnih 18. To povlači pitanje o 4 koraka viška i tome gdje se točno nalaze, ako robot ne radi sudare. Time dolazimo do problema ove metode.



Slika 22. Broj koraka u labirintu 3

Na slici 22. vidimo da u prvoj metodi broj koraka robota iznosi optimalnih 18. U drugoj broj koraka malo varira, ali se zaustavlja na 22. To je zato što robot radi nepotrebne korake po labirintu u potrazi za izlazom. Većim smo mu brojem koraka omogućili fleksibilnost i on brže nađe rješenje, ali problem se javlja kada taj broj koraka treba spustiti na optimalni broj. Parametar *Koraci* u fitness funkciji trebao bi riješiti taj problem, ali se očito javljaju neki problemi koji to onemogućavaju. U tome bi se smjeru program mogao poboljšati.

7. Zaključak

Kroz ovaj završni rad vidjeli smo dio razvoja prirodno inspiriranih algoritama i to počevši od Darwinove teorije evolucije koju je John Holland iskoristio kako bi pokrenuo razvoj genetskih algoritama. Uveo je razne alate: od veličine populacije, selekcije najboljih jedniki do križanja i mutacija. Da ne bi sve ostalo samo na teoriji, neke su metode prikazane na kretanju robota. Vidjeli smo 3 vrste labiritna, svaki specifičan na svoj način i došli smo do nekih zaključaka. Potrebno je pažljivo izabrati početne parametre algoritma jer o njima ovisi krajnji rezultat. Iz labirinta 1 zaključujemo da veličina populacije diktira brzinu pronalaska rješenja, ali isto tako ima veliki utjecaj na brzinu izvođenja programa. Analizom rezultata dviju različitih fitness funkcija vidimo da je ona najvažniji dio programa; od razlike u brzini pronalaska rješenja, do njezine sposobnosti da nađe točno rješenje. U labirintu 3 zbog kompleksnosti problema korištena je samo fitness funkcija koja je pokazivala najbolja rješenja . Čak i najbolja funkcija može imati poteškoća u pronalasku rješenja i upravo zbog toga problemu treba pristupiti iz više kuteva gledanja. Potrebno je jasno definrati traži li se optimalno rješenje ili su prihvatiljva i manje dobra rješenja jer to sve ima utjecaj na brzinu, kvalitetu i točnost izvođenja programa.
Zbog nedostatka vremena u program nisu implementirana neka zamišljena poboljšanja, te se program i dalje može unapređivati u grafici, brzini i točnosti. Unatoč tome, program prikazuje točna rješenja i mogućnost rješavanja široke lepeze problema.

8. Literatura

[1.] Xin-She Yang, Nature-Inspired Metaheuristic Algorithams, Velika Britanija, 2008.

[2.] David E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, The University of Alabama, 1989.

[3.] $http://www.egr.msu.edu/\ goodman/GECSummitIntroToGA_Tutorial-goodman.pdf$, Erik D. Goodman, Introduction to Genetic Algorithms

[4.] Predavanja iz kolegija Umjetna inteligencija, Petar Ćurković, 2017.

[5.] $http://www.zemris.fer.hr/\ golub/ga/ga_skripta1.pdf$, Genetski algoritam Prvi dio, Marin Golub, 2010.

[6.] $https://www.mathworks.com/$, MATLAB

9. Prilog

9.1. Prikaz kompletnog programa u MATLAB-u

```
clear
clc
%ZADAVANJE VELICINE POPULACIJE, BROJA KORAKA ROBOTA I BROJA GENERACIJA
B=input('Broj kromosoma (broj jedinki u populaciji): ');
A=input('Duzina kromosoma (broj koraka robota): ');
L=input('Broj iteracija (broj generacija): ');
count=0;

pc=0.7;
pm=0.3;

Populacija=randi(4,B,A);

%DEFINIRANJE IZGLEDA LABIRINTA KORISTECI BROJEVE 0 I 9. BROJ 9 PREDSTAVLJA
%ZID ILI PREPREKU, A BROJ 0 SLOBODAN PROLAZ.
%
% D=[9 9 9 9 9 9 9 9 9 9;
%    9 9 0 0 0 9 9 9 9 9;
%    9 0 0 9 0 0 0 0 9 9;
%    9 0 9 9 0 9 9 9 9 9;
%    9 0 0 0 0 9 9 9 9 9;
%    9 0 9 9 9 9 9 9 9 9;
%    9 0 0 0 0 0 9 9 9 9;
%    9 9 9 0 0 9 9 9 9 9;
%    9 9 9 9 9 9 9 9 9 9;];

D=[9 9 9 9 9 9 9 9 9 9;
   9 9 0 0 0 9 9 9 9 9;
   9 0 0 9 0 9 9 9 9 9;
   9 0 9 9 0 0 9 9 9 9;
   9 0 0 9 9 0 9 9 9 9;
   9 0 9 9 0 0 9 9 9 9;
   9 0 9 9 0 9 9 9 9 9;
   9 0 9 9 0 9 9 9 9 9;
   9 9 9 9 9 9 9 9 9 9;]; %Labirint

% D=[9 9 9 9 9 9 9 9 9 9;
%    9 0 0 0 0 0 0 0 0 9;
%    9 0 9 0 9 0 9 0 0 9;
%    9 0 0 0 0 0 0 0 0 9;
%    9 0 9 0 9 0 9 0 0 9;
```

```
%     9 0 0 0 0 0 0 0 9;
%     9 0 9 0 9 0 9 0 9;
%     9 0 0 0 0 0 0 0 9;
%     9 9 9 9 9 9 9 9 9;];

%ODREDIVANJE POCETNE POZICIJE
startX=8;
startY=5;
D(startX,startY)=7;
%ODREDIVANJE IZLAZA IZ LABIRINTA
endX=9;
endY=2;
D(endX,endY)=8;

%ODREDIVANJE KOORDINATA KOJE KORISTIMO ZA GRAFICKI PRIKAZ
    LABIRINTA
[H,W]=size(D);
h=1;
w=1;
for i=1:H
    for j=1:W
        if D(i,j)==9
            prva(h)=j;
            druga(h)=-i;
            h=h+1;
        else
            treca(w)=j;
            cetvrta(w)=-i;
            w=w+1;
        end
    end
end

disp(D)
prekid=1;
brojac=0;

%FOR PETLJA KOJA BROJI BROJ GENERACIJA
for l=1:L;
% OVAJ DIO PROGRAMA JE KOMENTIRAN, ALI SLUZI ZA NJEGOVO
    PREKIDANJE
% UKOLIKO NE ZELIMO DA SE IZVRTE SVE ITERACIJE NEGO DA SE
    PROGRAM PREKINE
% CIM ROBOT DODE DO CILJA
%     if prekid==0;
%         fprintf('Robot je dosao do cilja u %d iteracija',
    brojac)
```

```
%
%
%           break
%       end

%KRETANJE ROBOTA KOJE SE IZVODI POMOCU DVIJE FOR PETLJE I SWITCH/CASE-a
        brojac=brojac+1;
          PolozajX=0;
          PolozajY=0;

          Korak_sudara=[];
          Udaljenost=[];
          Koraci=[];
          Sudari=[];



        for i=1:B;
            X=startX;
            Y=startY;
            Sudar=0;
            Korak=0;
            Xstari=startX;
            Ystari=startY;
            u=0;

            for j=1:A;
                Korak=Korak+1;
                switch Populacija(i,j)
                    case 1
                        if D(X-1,Y)==9;
                            Sudar=Sudar+1;

                        else
                            Xstari=X;
                            X=X-1;
                            Y=Y;


                        end

                    case 2
                        if D(X+1,Y)==9;
                            Sudar=Sudar+1;
```

```
131                     else
132                         X=X+1;
133                         Y=Y;
134
135
136                     end
137
138                 case 3
139                     if D(X,Y+1)==9;
140                         Sudar=Sudar+1;
141
142
143
144                     else
145                         X=X;
146                         Y=Y+1;
147
148
149                     end
150
151                 case 4
152                     if D(X,Y-1)==9;
153                         Sudar=Sudar+1;
154
155
156                     else
157                         X=X;
158                         Y=Y-1;
159
160
161                     end
162
163             end
164         %DIO PROGRAMA KOJI SLUZI ZA KRETANJE ROBOTA JE
                ISKORISTEN,
165         %KAKO BI ZA SVAKU JEDINKU U POPULACIJU RACUNAO
                NJEZINE
166         %PARAMETRE KOJI SE KORISTE KOD FITNESS FUNKCIJE.
167         if Sudar==1 && u==0;
168             Korak_sudara(i)=j;
169             u=1;
170         elseif Sudar==0
171             Korak_sudara(i)=Korak;
172         end
173
174         if X==endX && Y==endY
175             break
```

```
176                 end
177
178                 end
179                 PolozajX(i)=Y;
180                 PolozajY(i)=X;
181                 UdaljenostX=abs(endX-X);
182                 UdaljenostY=abs(endY-Y);
183                 Udaljenost(i)=UdaljenostX+UdaljenostY;
184                 Udaljenost_od_startaX=abs(startX-X);
185                 Udaljenost_od_startaY=abs(startY-Y);
186                 Udaljenost_od_starta(i)=Udaljenost_od_startaX+
                        Udaljenost_od_startaY;
187                 Sudari(i)=Sudar;
188                 Koraci(i)=Korak;
189
190
191             end
192
193
194 % FITNESS FUNCKIJA
195     F=0;
196     for i=1:B;
197  %          F(i)=((1/(Udaljenost(i)+0.1))+(Korak_sudara(i)/20)+(
        Udaljenost_od_starta(i)/20)+1/Koraci(i));
198                 F(i)=((Udaljenost_od_starta(i)/(Udaljenost(i)+0.1))
                        +(Korak_sudara(i)/20)+ 1/Koraci(i));
199
200     end
201
202     %OVAJ DIO KODA ODREDUJE NA KOJOJ SE POZICIJI NALAZI ROBOT S
            NAJBOLJIM
203     %FITNESSOM. SLUZI ZA CRTANJE GRAFA.
204     Fitness(1)=max(F);
205     E=0;
206     for i=1:B
207           robotX=0;
208           robotY=0;
209
210           if F(i)==max(F)
211                 E=i;
212                 robotX=PolozajX(E);
213                 robotY=-PolozajY(E);
214                 break
215           end
216     end
217
218     sumaF=sum(F);
```

```
219       Frel=0;
220
221      %RELATIVNI FITNESS
222       for i=1:B;
223 %     Frel(i)=((1/(Udaljenost(i)+0.1))+(Korak_sudara(i)/20)+(
      Udaljenost_od_starta(i)/20)+1/Koraci(i))/sumaF;
224       Frel(i)=((Udaljenost_od_starta(i)/(Udaljenost(i)+0.1))+(
          Korak_sudara(i)/20)+1/Koraci(i))/sumaF;
225
226       end
227 %
228
229  % ELITIZAM
230  % SLUZI OCUVANJE NAJBOLJE JEDINKE U POPULACIJI. TAKODER IZ
       ELITNE JEDINKE
231  % SE KORISTE PODACI ZA CRTANJE GRAFOVA
232        for i=1:B
233            if Frel(i)==max(Frel);
234                w=randi(B,1,1);
235                Elit=Populacija(i,:);
236                Kor(1)=Koraci(i);
237                UdaljStart(1)=Udaljenost_od_starta(i);
238                Udalj(1)=Udaljenost(i);
239                sud(1)=Sudari(i);
240                Sud(1)=Korak_sudara(i);%ZA CRTANJE GRAFA
241                break
242
243
244            end
245        end
246
247 %OVAJ DIO PROGRAMA SLUZI ZA GRAFICKI PRIKAZ LABIRINTA ZA
      VRIJEME
248 %IZVRSAVANJA KODA. ON NAKON SVAKE 100 ITERACIJE POKAZUJE
      TRENUTNU POZICIJU
249 %ROBOTA
250    count=count+1;
251
252  if count == 100
253     trenuniFitnes = Fitness(1)
254       count=0;
255       putanjaX=0;
256       putanjaY=0;
257       X=startX;
258       Y=startY;
259       putanjaX(1)=X;
260       putanjaY(1)=Y;
```

```
for j=1:A;
        switch Elit(1,j)
                    case 1
                        if D(X-1,Y)==9;

                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;

                        else
                            Xstari=X;
                            X=X-1;
                            Y=Y;
                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;


                        end

                    case 2
                        if D(X+1,Y)==9;

                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;

                        else
                            X=X+1;
                            Y=Y;
                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;


                        end

                    case 3
                        if D(X,Y+1)==9;

                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;

                        else
                            X=X;
                            Y=Y+1;
                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;

                        end

```

```
                    case 4
                        if D(X,Y-1)==9;

                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;


                        else
                            X=X;
                            Y=Y-1;
                            putanjaX(j+1)=-X;
                            putanjaY(j+1)=Y;

                        end

            end
            if X==endX && Y==endY
                break
            end
        end


        figure(2)
        plot(prva,druga,'b*',treca,cetvrta,'ro',robotX,robotY,'go',
            startY,-startX,'ko',putanjaY,putanjaX,'k--')
        pause(.05);
    ietracija = 1
    end

%SELEKCIJA NAJBOLJIH JEDINKI RULETNIM PRAVILOM
    for i=1:B;
        N=0;
        T=rand(1,1);
        for j=1:B;
            N=N+Frel(j);
                if T<=N;
                    Pop(i,:)=Populacija(j,:);
                    break

                end
        end
    end
%VARIJABLA Pop PREDSTAVLJA NOVO IZABRANU GENERACIJU RODITELJA

%KRIZANJE
    for i=2:2:B;
        S=rand(1,1);
```

```
354             Par=Pop(i-1:i,:);
355             if S<=pc
356                 O=randi(2,1,A);
357                 P1=Par(1,:);
358                 P2=Par(2,:);
359
360                 for j=1:A
361                     if O(1,j)==2
362                         P1novi(1,j)=P2(1,j);
363                         P2novi(1,j)=P1(1,j);
364                     else
365                         P1novi(1,j)=P1(1,j);
366                         P2novi(1,j)=P2(1,j);
367                     end
368                 end
369                 ParNovi=[P1novi;P2novi];
370
371             else
372                 P1novi=Par(1,:);
373                 P2novi=Par(2,:);
374                 ParNovi=[P1novi;P2novi];
375             end
376
377             POPkrizanje(i-1:i,:)=ParNovi;
378         end
379
380 %MUTACIJA
381         PopMutacija=POPkrizanje;
382
383         for i=1:B
384             for j=1:A
385                 m=rand(1,1);
386                 k=randi(4,1,1);
387                 if m<pm;
388                     PopMutacija(i,j)=k;
389                 else
390                     PopMutacija(i,j)=PopMutacija(i,j);
391                 end
392             end
393         end
394 %VARIJABLA Populacija PREDSTAVLJA NOVU POPULACIJU NASTALU
        KRIZANJEM I
395 %MUTACIJAMA, TE ONA IDE U SLJEDECU GENERACIJU.
396         Populacija=PopMutacija;
397         Populacija(w,:)=Elit;
398
399
```

```
400
401 end
402
403 %PLOTANJE GRAFOVA
404 figure(1)
405 subplot(2,3,1)
406 xx=1:1:brojac;
407 plot(xx,Udalj)
408 axis([0 brojac -1 10])
409 xlabel('Broj iteracija')
410 ylabel('Udaljenost')
411 grid on
412
413 subplot(2,3,2)
414 plot(xx,Fitness)
415 xlabel('Broj iteracija')
416 ylabel('Fitness')
417 grid on
418
419 subplot(2,3,3)
420 plot(xx,UdaljStart)
421 xlabel('Broj iteracija')
422 ylabel('Udaljenost od starta')
423 axis([0 L 0 (A+2)])
424 grid on
425
426 subplot(2,3,4)
427 plot(xx,Sud)
428 xlabel('Broj iteracija')
429 ylabel('Koraka do sudara')
430 axis([0 L 0 (A+2)])
431 grid on
432
433
434 subplot(2,3,5)
435 plot(xx,Kor)
436 xlabel('Broj iteracija')
437 ylabel('Koraci')
438 axis([0 L 0 (A+2)])
439 grid on
440
441 subplot(2,3,6)
442 plot(xx,sud)
443 xlabel('Broj iteracija')
444 ylabel('Sudari')
445 axis([0 L 0 (A+2)])
446 grid on
```