Primjena evolucijskog algoritma za planiranje trajektorije

Butković, Borna

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Borna Butković

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Primjena evolucijskog algoritma za planiranje trajektorije

Mentor:

Dr. sc. Petar Ćurković, dipl. ing.

Student:

Borna Butković

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se svom mentoru dr. sc. Petru Ćurkoviću na nesebičnoj pomoći i stručnim savjetima prilikom izrade završnog rada.

Zahvaljujem se svojoj obitelji na bezuvjetnoj podršci i razumijevanju jer bez njih ovo sve ne bi bilo moguće.

Borna Butković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske 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: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Borna Butković** JMBAG: **0035230452**

Naslov rada na hrvatskom jeziku: **Primjena evolucijskog algoritma za planiranje trajektorije**

Naslov rada na engleskom jeziku: **Evolutionary algorithm application for trajectory planning**

Opis zadatka:

Evolucijski algoritmi (EA) pripadaju skupu prirodom inspiriranih algoritama koji se uspješno koriste za rješavanje različitih optimizacijskih problema. Planiranje trajektorije općenito je problem koji nije rješiv u polinomnom vremenu (NP težak problem), pa je pronalazak optimalnih trajektorija računski kompleksno i vremenski zahtjevno. EA uspješno rješavanju niz NP teških problema dajući kvalitetna rješenja koja su primjenjiva u praksi.
U ovom radu potrebno je osmisliti dvodimenzionalnu okolinu koja uključuje niz prepreka te implementirati evolucijski algoritam koji će pronalaziti optimirane trajektorije uz poznatu polaznu i odredišnu točku.

Potrebno je napraviti sljedeće:

- osmisliti 2D okolinu, matematički ju zapisati i vizualno prikazati u programskom paketu Matlab,
- realizirati prikladan način kodiranja trajektorija za efikasnu evaluaciju evolucijskim algoritmom,
- osmisliti skup kriterija za evaluaciju trajektorija,
- napisati programski kôd za evolucijski algoritam koji omogućuje evaluaciju familije trajektorija i razvoj optimiranih trajektorija,
- analizirati utjecaj parametara evolucijskog algoritma na kvalitetu pronađenog rješenja,
- izvesti zaključke i dati preporuke za unaprjeđenja.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:
30. 11. 2023.

Datum predaje rada:
1. rok: 22. i 23. 2. 2024.
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
1. rok: 26. 2. – 1. 3. 2024.
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Zadatak zadao:
Izv. prof. dr. sc. Petar Ćurković

Predsjednik Povjerenstva:
Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U ovom završnom radu istražit će se primjena evolucijskih algoritama u rješavanju problema planiranja putanje robota u 2D okruženju. Bit će korišten MATLAB za implementaciju evolucijskog algoritma te će se analizirati utjecaj kombinacija koeficijenata križanja i mutacije na performanse algoritma. Cilj istraživanja je pronaći najučinkovitiju kombinaciju parametara koja će rezultirati bržim i preciznijim pronalaženjem optimalne putanje robota u 2D okruženju. Kroz eksperimentalno istraživanje, analizirat će se različite strategije optimizacije te će se identificirati moguća poboljšanja koja bi se mogla primijeniti u budućim istraživanjima.

Ključne riječi: evolucijski algoritam, umjetna inteligencija, trajektorija, 2D okolina

SUMMARY

In this final project, the application of evolutionary algorithms in solving robot path planning problems in a 2D environment will be explored. MATLAB will be used for the implementation of the evolutionary algorithm, and the impact of combinations of crossover and mutation coefficients on the algorithm's performance will be analyzed. The research goal is to find the most effective combination of parameters that will result in faster and more precise identification of the optimal robot path in a 2D environment. Through experimental research, different optimization strategies will be analyzed, and possible improvements that could be applied in future research will be identified.

Key words: evolutionary algorithm, artificial inteligence, trajectory, 2D enviroment

1. UVOD

Evolucijski algoritmi, inspirirani samim procesom evolucije u prirodi, predstavljaju snažan računalni alat koji se koristi za rješavanje širokog spektra optimizacijskih problema [1]. Ovi algoritmi simuliraju prirodnu selekciju, križanje i mutaciju kako bi iterativno generirali bolje rješenja tijekom vremena. U njihovoj srži leži koncept adaptivne optimizacije, gdje se populacija rješenja postupno prilagođava okolini kako bi se postigla bolja prilagodba na postavljene zahtjeve. Evolucijski algoritmi nalaze svoje korijene u biološkim sustavima gdje je proces evolucije ključan za opstanak i prilagodbu organizama na okolinu. U prirodi, organizmi se prilagođavaju promjenama u okolini kroz niz generacija, pri čemu su oni s najboljim prilagodbama skloni preživljavanju i reprodukciji. Analogno tome, evolucijski algoritmi primjenjuju sličan princip iterativne prilagodbe, ali umjesto bioloških organizama, optimiziraju niz rješenja u virtualnom prostoru. Bitna karakteristika evolucijskih algoritama je njihova sposobnost rada s velikim prostorima pretraživanja i rješavanja složenih problema optimizacije koji često nisu rješivi klasičnim analitičkim metodama. Evolucijski algoritmi postali su sveprisutni u istraživačkim područjima kao što su optimizacija, umjetna inteligencija, robotika, ekonomija, biologija i mnoga druga područja. Povijest evolucijskih algoritama seže unatrag u drugu polovicu 20. stoljeća, kada su znanstvenici poput Johna Hollanda i Ingo Rechenberga počeli istraživati koncepte evolucije u kontekstu računalnih modela. Ključni trenutak predstavlja rad Johna Hollanda iz 1960-ih koji je predložio koncept genetskog algoritma (GA), inspiriran procesom prirodne selekcije. Nakon toga, Richard E. Lenski i njegovi suradnici koristili su genetski algoritam za simuliranje evolucije bakterija u laboratorijskim uvjetima. Ovaj eksperiment pokazao je kako se genetski algoritmi mogu primijeniti za razumijevanje evolucijskih procesa na mikrobiološkoj razini. Primjena evolucijskih algoritama u različitim područjima potvrđuje njihovu svestranost i snagu kao alata za rješavanje kompleksnih problema optimizacije i prilagodbe na promjenjive uvjete. Ova područja kontinuirano se razvijaju, a evolucijski algoritmi ostaju ključni instrumenti u istraživanju i primjeni novih tehnologija. Ključni elementi evolucijskog algoritma su fitnes funkcija, koja ocjenjuje kvalitetu svakog rješenja u populaciji, te koeficijenti križanja i mutacije, koji kontroliraju vjerojatnost križanja i mutacije. Fitnes funkcija je srž algoritma, ocjenjujući "dobrotu" svakog rješenja u kontekstu problema optimizacije. Koeficijenti križanja i mutacije upravljaju procesima kombiniranja osobina roditeljskih rješenja i uvođenja slučajnih promjena radi očuvanja raznolikosti u populaciji. Pravilno podešavanje ovih parametara ključno je za uspjeh evolucijskog algoritma, omogućujući balans između intenzivnosti pretraživanja prostora rješenja i očuvanja raznolikosti, što vodi bržoj i efikasnijoj konvergenciji prema optimalnom rješenju. Navedeni parametri se odabiru ovisno o problemu koji treba biti riješen.

1.1. Cilj rada

Kroz ovaj rad, istražujemo primjenu evolucijskih algoritama u kontekstu planiranja putanje robota u 2D labirintu. Analiziramo kako se evolucijski pristup može primijeniti za generiranje optimalnih putanja kroz razne labirinte, istražujući različite algoritme, njihove prednosti i ograničenja. Kroz spajanje koncepta evolucije s računalnom optimizacijom, otvaramo vrata razumijevanju prirodnih procesa, dok istovremeno pružamo efikasne alate za rješavanje praktičnih problema u stvarnom svijetu. Ovaj rad oslikava interdisciplinarni pristup istraživanju, povezujući biološke principe s naprednim računalnim tehnikama kako bi se ostvarila optimizacija i inovacija.

2. Evolucijski algoritam

U prethodnom poglavlju napravljen je kratak uvod u evolucijski algoritam i sve bitno što se veže uz njega. Ono što je važno istaknuti vezano za evolucijski algoritam jest to da evolucijski algoritam (EA) funkcionira u nekoliko iteracija, tijekom kojih se generiraju, evaluiraju i modificiraju kandidatna rješenja sve dok se ne postigne zadovoljavajuće rješenje problema optimizacije. U ovom algoritmu, kandidatna rješenja predstavljaju populaciju koja se iterativno prilagođava prema optimalnom rješenju problema. Početna populacija rješenja generira se, a svako rješenje u populaciji ocjenjuje se koristeći fitnes funkciju. Fitnes funkcija služi za ocjenu kvalitete rješenja u kontekstu problema optimizacije. Rješenja s većom vrijednošću fitnesa smatraju se boljima. Nakon evaluacije fitnesa, rješenja se biraju za reprodukciju na temelju njihovog fitnesa. U procesu križanja, odabrani roditelji kombiniraju svoje osobine kako bi generirali potomstvo. Potomstvo se zatim podvrgava mutaciji, koja služi za održavanje raznolikosti u populaciji. Algoritam se izvršava kroz nekoliko iteracija, a terminira nakon određenog broja iteracija ili zadovoljenja određenih uvjeta zaustavljanja. Tijekom izvođenja algoritma, provjerava se konvergencija populacije, tj. stabilnost i poboljšanja rješenja tijekom vremena. Konačno rješenje može se odabrati iz posljednje generacije populacije ili između rješenja koja su izvučena tijekom izvođenja algoritma. Evolucijski algoritam može se prilagoditi različitim problemima optimizacije, uz različite strategije selekcije, operatora križanja i mutacije, te drugih parametara prilagodbe. Ključno je pažljivo podešavanje tih parametara kako bi se osigurala konvergencija prema optimalnom rješenju. Evolucijski algoritmi su se pokazali kao moćan alat za rješavanje širokog spektra optimizacijskih problema u različitim područjima. Nekoliko primjera primjene evolucijskih algoritama na rješavanje sličnih problema opisano je u radovima [2-6]. U ovom radu fokusirat ćemo se na 2D prostor, a robot će biti predstavljen točkom u tom prostoru, te je potrebno da pronađe optimalnu trajektoriju od početne do konačne pozicije uz izbjegavanje prepreka.

2.1. Odabir populacije i broja generacija

Populacija i generacija su ključni pojmovi u evolucijskim algoritmima, koji su temeljni u postupku optimizacije problema. U evolucijskim algoritmima, populacija se sastoji od skupa jedinki ili kandidatnih rješenja koja se koriste za pronalaženje optimalnog rješenja. Svaka jedinka u populaciji predstavlja moguće rješenje problema koji se optimizira. Populacija može biti fiksne veličine ili se može mijenjati tijekom izvođenja algoritma, ovisno o postavljenim parametrima. Broj generacija u evolucijskim algoritmima predstavlja broj iteracija ili prolaza kroz evolucijski proces populacije. Svaka generacija obuhvaća određeni broj iteracija u kojima se provodi proces selekcije, križanja, mutacije i evaluacije fitnesa kako bi se generirala nova populacija. Kako se evolucijski algoritam izvršava kroz više generacija, populacija se postupno prilagođava okolini problema optimizacije. Tijekom prvih generacija, populacija može biti raznolika, a algoritam može istraživati različite dijelove prostora rješenja. Kako algoritam napreduje kroz generacije, može se očekivati da će populacija postati sve prilagođenija problemu, a rješenja će postati sve bolja u smislu fitnesa. Broj generacija važan je parametar evolucijskog algoritma i treba ga pažljivo odabrati ovisno o konkretnom problemu optimizacije. Veći broj generacija može dovesti do duljeg vremena izvođenja algoritma, ali može pomoći u pronalasku boljih rješenja. S druge strane, premali broj generacija može dovesti do nedovoljne

konvergencije algoritma prema optimalnom rješenju ili može uzrokovati premalu raznolikost u populaciji. Kod odabira broja generacija, važno je uzeti u obzir kompleksnost problema, brojnost populacije, kao i resurse potrebne za izvođenje algoritma. U praksi, odabir broja generacija često je rezultat empirijskog testiranja i podešavanja algoritma na konkretnim problemima optimizacije.

2.2. Fitnes funkcija

Fitnes funkcija je ključan koncept u evolucijskim algoritmima koji ocjenjuje kvalitetu svakog rješenja u kontekstu problema optimizacije. Njena uloga je procijeniti koliko je svako rješenje "dobro" ili "loše" u odnosu na cilj koji se želi postići. Fitnes funkcija može biti definirana na različite načine, ovisno o prirodi problema i kriterijima optimizacije. Kada se koristi u evolucijskom algoritmu, fitnes funkcija služi kao osnovna mjerila prema kojoj se procjenjuju jedinke u populaciji. Cilj je pronaći rješenja koja maksimiziraju (ili minimiziraju, ovisno o problemu) vrijednost fitnesa, jer ta rješenja predstavljaju bolje prilagodbe okolini problema. Recimo da imamo problem raspoređivanja raspodjele resursa u proizvodnom procesu. Naša fitnes funkcija može ocjenjivati efikasnost svakog rasporeda resursa, gdje je veća efikasnost bolja. Primjer jedne takve fitnes funkcije može biti suma proizvoda resursa i vremena obrade za svaki proizvodni segment. Cilj je minimizirati ovu vrijednost, jer manja vrijednost ukazuje na učinkovitije raspoređivanje resursa. U ovom primjeru, fitnes funkcija ima ulogu kvantifikacije performansi svakog rješenja, omogućavajući algoritmu da usmjeri pretragu prema rješenjima koja najbolje zadovoljavaju postavljene kriterije. Što je rješenje bliže idealnom, to bi trebalo imati veću vrijednost fitnesa. Važno je napomenuti da je oblik i definicija fitnes funkcije specifična za svaki problem i zahtjeva pažljivo razmatranje kako bi odražavala stvarne zahtjeve i ciljeve problema optimizacije. Kvalitetna fitnes funkcija ključna je za uspjeh evolucijskog algoritma jer izravno utječe na sposobnost algoritma da pronađe optimalna rješenja u skladu s postavljenim kriterijima.

2.2.1. OneMax

OneMax problem je jednostavan problem optimizacije koji se često koristi kao testna platforma za evaluaciju performansi evolucijskih algoritama. Cilj OneMax problema je maksimizirati sumu vrijednosti binarnog vektora.
Uzet ćemo za primjer vektor:

$$[1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1] \qquad \textbf{(2. 1.)}$$

Izraz 2. 1. se sastoji od četiri jedinice i dvije nule. Vrijednost fitnes funkcije bismo računali u odnosu na jedinice te bi nam stoga za ovaj primjer fitnes iznosio 4.

2.2.2. OneMin

OneMin problem je suprotnost OneMax problema. Umjesto da maksimizira sumu vrijednosti binarnog vektora, OneMin problem se fokusira na minimiziranje te sume. Cilj je pronaći binarni vektor s najmanjim mogućim brojem jedinica.

Za ranije navedeni izraz 2. 1. u OneMin slučaju fitnes bi iznosio 2.

2.3. Selekcija

Selekcija predstavlja ključan korak u evolucijskim algoritmima koji simulira prirodni proces odabira u evoluciji. Ovaj proces omogućuje evolucijskom algoritmu da identificira i favorizira jedinke s boljom prilagodbom kako bi ih odabrao za sudjelovanje u reprodukciji, s ciljem postizanja optimalnog rješenja problema optimizacije. Metode koje se koriste pri procesu selekcije su ruletno pravilo i turnirska selekcija. Odabir metode ovisi o problemu s kojim se susrećemo.

2.3.1. Ruletno pravilo

Ruletno pravilo, poznato i kao proporcionalna selekcija, predstavlja ključnu fazu u evolucijskim algoritmima. Ovaj pristup selekciji jedinki simulira prirodni odabir u evoluciji, dajući prednost jedinkama s većom prilagodbom za sudjelovanje u reprodukciji. Postupak ruletnog pravila slijedi nekoliko koraka. Prvo, za svaku jedinku u populaciji izračunava se vjerojatnost njenog odabira, koja je proporcionalna njenom fitnesu u odnosu na ukupni fitnes populacije. Drugim riječima, što je jedinka prilagođenija, to će imati veću vjerojatnost biti odabrana. Nakon što se izračunaju vjerojatnosti odabira za sve jedinke, konstruira se "rulet kotač". Rulet kotač je virtualni kotač podijeljen u sektore, pri čemu je svaki sektor proporcionalan vjerojatnosti odabira odgovarajuće jedinke. Veća vjerojatnost odabira rezultira većim sektorom na rulet kotaču. Kada je rulet kotač konstruiran, algoritam generira slučajni broj između 0 i 1. Zatim se pronalazi sektor na kojem se zaustavlja "igla" rulet kotača. Jedinka čiji sektor obuhvaća iglu bira se za reprodukciju. Ovaj proces ponavlja se sve dok se ne odabere potreban broj jedinki za reprodukciju. Ruletno pravilo omogućuje odabir jedinki proporcionalno njihovoj prilagodbi, što rezultira u efikasnom usmjeravanju algoritma prema bolje prilagođenim rješenjima.

Tablica 1 Jedinke s pripadajućim fitnesima

Jedinka	Fitnes
A	10
B	15
C	8
D	20
E	12
UKUPNI FITNES	65

U tablici 1 imamo prikazane jedinke s njihovim fitnesima te ukupni fitnes prilagodbe. S pomoću fitnesa prilagodbe računamo vjerojatnost odabira za svaku jedinku.

Tablica 2 Jedinke s pripadajućim vjerojatnostima odabira

Jedinka	Vjerojatnost odabira
A	$\frac{10}{65} = 0.153$
B	$\frac{15}{65} = 0.230$
C	$\frac{8}{65} = 0.123$
D	$\frac{20}{65} = 0.307$
E	$\frac{12}{65} = 0.184$

U tablici 2 prikazane su vjerojatnosti odabira za svaku jedinku te sada konstruiramo rulet kotač koristeći ove vjerojatnosti. Na primjer, ako slučajno generirani broj padne unutar intervala od 0 do 0.153, odabiremo jedinku A. Ako padne između 0.153 i 0.383, odabiremo jedinku B, i tako dalje. Ovo je osnovni postupak ruletnog pravila koji omogućuje odabir jedinki proporcionalno njihovoj prilagodbi, čime se simulira prirodni odabir u evoluciji.

2.3.2. Turnirska selekcija

Turnirska selekcija je jedan od ključnih postupaka u evolucijskim algoritmima, koji simulira proces odabira jedinki za reprodukciju na način sličan sportskom turniru. Ovaj pristup omogućuje algoritmu da odabere jedinku s najboljom prilagodbom iz nasumično odabranog skupa jedinki, poznatog kao turnir. Kada se primjenjuje turnirska selekcija, prvo se nasumično odabire skup jedinki iz populacije, poznat kao turnir. Veličina ovog skupa može varirati ovisno o postavkama algoritma, ali obično je to mali broj jedinki. Nakon što se formira turnir, algoritam odabire jedinku s najboljom prilagodbom iz tog skupa, koja postaje roditelj sljedeće generacije. Prednost turnirske selekcije je u tome što omogućuje prilagodljivost u odabiru roditelja, jer nije potrebno sortirati populaciju prema prilagodbi. Osim toga, ovaj pristup omogućuje veću raznolikost u odabiru roditelja, jer se turniri formiraju nasumično.

Tablica 3. Jedinke s pripadajućim fitnesima

Jedinka	Fitnes
A	10
B	6
C	8
D	16

U tablici 3 prikazane su četiri jedinke s pripadajućim fitnesima. Sljedeći korak koji se treba napraviti u provedbi turnirske selekcije je odabir veličine turnira te zatim nasumični odabir jedinki.

Tablica 4 Odabrane jedinke s pripadajućim fitnesima

Odabrane jedinke	Fitnes
B	6
D	16

U tablici 4 vidimo odabrane jedinke s njihovim pripadajućim fitnesima. Uočavamo da je veličina turnira dvije jedinke. Sljedeće što uočavamo je da jedinka D ima veći fitnes od jedinke B te samim time pobjeđuje u turniru i postaje roditelj sljedeće generacije. Ovaj postupak ponavlja se sve dok ne odaberemo potreban broj roditelja za reprodukciju sljedeće generacije. U ovom primjeru, to bi moglo biti ponovljeno nekoliko puta dok se ne odaberu svi roditelji za sljedeću generaciju.

2.4. Rekombinacija ili križanje

Križanje ili rekombinacija omogućuje stvaranje novih jedinki kombiniranjem genetičkog materijala roditelja. Ovaj postupak simulira genetski proces u prirodi gdje se geni od roditelja prenose na potomstvo. Postoje različiti oblici križanja koji se koriste u evolucijskim algoritmima, a svaki od njih ima svoje specifičnosti i način djelovanja. Jedan od najčešćih oblika križanja je jednostruko križanje. U ovom obliku križanja, nasumično se odabire jedna pozicija u roditeljskim jedinkama. Geni s desne strane odabrane pozicije zamjenjuju se između roditelja, stvarajući dva nova potomka. Drugi oblik križanja je višestruko križanje, gdje se više od jedne pozicije koristi za zamjenu gena između roditelja. Ovisno o postavkama algoritma, moguće je odrediti više pozicija za križanje, što može rezultirati različitim razinama raznolikosti u potomstvu. Treći oblik križanja je uniformno križanje. U ovom slučaju, svaki gen u potomstvu nasumično se odabire iz jednog od roditelja, s jednakom vjerojatnošću odabira gena od oba roditelja. Ovaj oblik križanja može promicati veću raznolikost u potomstvu jer se geni biraju neovisno jedan o drugome.

Tablica 5 Tablica roditelja i potomstva nastalog križanjem

Roditelj A	10110
Roditelj B	01011
Potomstvo 1	10011
Potomstvo 2	01110

U tablici 5 uočavamo dva roditelja odabrana za križanje. Slučajnim odabirom odabrana je pozicija 3 za križanje te je obavljeno križanje. U posljednja trećem i četvrtom redu prikazano je potomstvo nastalo križanjem odnosno rekombinacijom.

2.5. Mutacija

Mutacija je još jedan ključni operator u evolucijskim algoritmima koji doprinosi raznolikosti populacije i omogućuje algoritmu da istraži širi prostor rješenja. Mutacija se javlja nasumičnim promjenama genetskog materijala jedinke tijekom reprodukcije. Osnovna svrha mutacije je osigurati divergenciju u populaciji, sprječavajući stagnaciju u pretrazi prostora rješenja. Iako je vjerojatnost mutacije obično mala, ona može biti ključna u otkrivanju novih, potencijalno boljih rješenja. U evolucijskim algoritmima, mutacija se obično primjenjuje na svaki gen jedinke s određenom vjerojatnošću. Ovisno o problemu i postavkama algoritma, mutacija može uključivati različite vrste promjena genetskog materijala, uključujući promjene u jednom bitu (za binarne vektore), promjene u vrijednostima atributa (za vektore s kontinuiranim vrijednostima) ili promjene u redoslijedu elemenata (za permutacijske probleme). Bitno je napomenuti da prevelika vjerojatnost mutacije može dovesti do gubitka konvergencije algoritma, dok premala vjerojatnost mutacije može dovesti do stagnacije u pretrazi prostora rješenja. Stoga je važno pažljivo podešavanje parametara mutacije kako bi se postigao ravnotežni odnos između istraživanja i iskorištavanja u evolucijskom algoritmu. Uz sve to, mutacija igra ključnu ulogu u očuvanju genetičke raznolikosti u populaciji tijekom evolucijskog procesa, što pomaže algoritmu da istraži širok spektar potencijalnih rješenja i pronađe optimalno rješenje problema optimizacije.

Tablica 6 Primjer mutacije

Prije mutacije	10110100
Nakon mutacije	10110000

U tablici 6 vidimo da je mutirao šesti član vektora iz jedinice u nulu. Ova promjena može doprinijeti divergenciji u populaciji i omogućiti istraživanje novih područja prostora rješenja. Važno je napomenuti da je sam proces mutacije nasumičan, stoga su moguće različite promjene na jedinkama u populaciji tijekom evolucijskog procesa.

2.6. Elitizam

Elitizam je strategija koja se često koristi u evolucijskim algoritmima, a koja osigurava da najbolje prilagođene jedinke iz trenutne generacije automatski prelaze u sljedeću generaciju bez promjena. Drugim riječima, elitizam čuva najbolje jedinke neizmijenjene tijekom evolucijskog procesa. Prednost elitizma je u tome što osigurava očuvanje najboljih rješenja tijekom generacija, čime se sprječava gubitak visoko prilagođenih jedinki zbog slučajnih procesa kao što su križanje i mutacija. Ovaj pristup pomaže u održavanju stabilnosti i bržem konvergiranju algoritma prema optimalnom rješenju, posebno u situacijama kada je prilagodba jasno definirana i kada su ciljevi optimizacije dobro poznati. Druga prednost elitizma je u tome što omogućava očuvanje diverziteta u populaciji. Dok se većina jedinki može mijenjati kroz procese križanja i mutacije, elitne jedinke pružaju stabilnu osnovu za istraživanje prostora rješenja. To može pomoći u sprječavanju konvergencije prema lokalnom optimumu i poticanju pretrage šireg prostora rješenja. Međutim, elitizam također može imati neke mane. Jedna od njih je povećanje rizika od previše uniformne populacije. Ako se elitizam primjenjuje bez odgovarajućih mehanizama diverzifikacije, populacija može postati previše homogena, što može ograničiti sposobnost algoritma da istražuje širi prostor rješenja. Također, elitizam može povećati vjerojatnost ranog konvergiranja prema lokalnom optimumu. Ako elitne jedinke previše dominiraju u populaciji, algoritam može postati previše usmjeren prema određenom dijelu prostora rješenja, čime se smanjuje sposobnost istraživanja i otkrivanja potencijalno boljih rješenja. U konačnici, iako elitizam ima svoje prednosti i mane, pravilno podešavanje njegove primjene u evolucijskim algoritmima može značajno poboljšati učinkovitost i sposobnost algoritma da pronađe optimalna rješenja za različite probleme optimizacije.

3. Problem projektiranja putanje robota

Problem projektiranja putanje robota u 2D labirintu predstavlja izazovno područje istraživanja u području robotike i umjetne inteligencije. Cilj je pronaći optimalnu putanju kojom će se robot kretati od početne do ciljne točke unutar labirinta, uz izbjegavanje prepreka i optimizaciju različitih kriterija poput udaljenosti ili vremena potrebnog za dolazak do cilja. U tom kontekstu, evolucijski algoritmi predstavljaju moćan pristup za rješavanje ovog problema. Evolucijski algoritmi su vrsta metaheurističkih optimizacijskih tehnika koje oponašaju proces evolucije u prirodi radi pronalaska optimalnih rješenja za različite probleme. Kroz iterativni proces selekcije, križanja i mutacije, evolucijski algoritmi pretražuju prostor rješenja i prilagođavaju se okolini kako bi pronašli rješenja koja su najbliža optimalnima. Načini rješavanja problema projektiranja putanje robota u 2D labirintu s pomoću evolucijskog algoritma su raznoliki i mogu se prilagoditi konkretnim potrebama i karakteristikama problema. Jedan od pristupa uključuje indirektno kodiranje putanje, gdje se putanja robota kodira s pomoću niza parametara ili gena, koji se zatim optimiziraju kroz evolucijski proces. Drugi pristup je direktno kodiranje putanje, gdje se sama putanja robota kroz labirint kodira s pomoću različitih tipova kôdova poput binarnih vektora ili matrica putanje. Također, postoje prilagodljive evolucijske strategije koje koriste evolucijski algoritam za optimizaciju parametara ili strategija planiranja putanje robota. Hibridni pristupi kombiniraju evolucijske algoritme s drugim tehnikama poput umjetne neuronske mreže ili algoritama pretrage putanje kako bi pronašli optimalna rješenja. Kroz kombinaciju ovih pristupa, evolucijski algoritmi mogu efikasno rješavati kompleksne probleme projektiranja putanje robota u 2D labirintima, pružajući prilagodljiv i robustan alat za navigaciju robota u nepoznatim okruženjima. U nastavku rada detaljnije će se istražiti primjena evolucijskih algoritama za rješavanje problema projektiranja putanje robota u 2D labirintu, analizirajući različite pristupe i njihove karakteristike.

3.1. Primjena MATLAB-a u problemu projektiranja putanje robota

Korištenje MATLAB-a u rješavanju problema projektiranja putanje robota uz pomoć evolucijskih algoritama pruža značajne prednosti i mogućnosti za istraživanje i razvoj u području robotike i umjetne inteligencije [7-10]. MATLAB je poznat po svojoj moćnoj skupini alata za programiranje, simulaciju, optimizaciju i vizualizaciju podataka. Ova kombinacija alata omogućuje nam da učinkovito istražujemo i rješavamo složene probleme od kojih je između ostalog i navigacija robota u labirintima. Evolucijski algoritmi su posebno prikladni za rješavanje problema projektiranja putanje robota zbog svoje sposobnosti pretraživanja prostora rješenja i pronalaženja optimalnih putanja. Kroz iterativni proces selekcije, križanja i mutacije, evolucijski algoritmi mogu generirati putanje robota koje minimiziraju udaljenost, vrijeme putovanja ili druge kriterije optimizacije. Jedna od ključnih prednosti MATLAB-a je u njegovim optimizacijskim alatima poput genetskog algoritma (GA) koji omogućuju jednostavnu implementaciju evolucijskih algoritama za rješavanje problema optimizacije, uključujući i projektiranje putanje robota. S pomoću ovih alata, možemo poprilično brzo prototipirati, testirati i optimizirati algoritme za planiranje putanje robota u simuliranom okruženju. Osim toga, MATLAB pruža moćne alate za vizualizaciju rezultata, što nam omogućuje da analiziramo ponašanje robota u labirintima, pratimo napredak algoritama kroz generacije te optimiziramo parametre algoritama kako bismo postigli što bolje rezultate. Fleksibilnost i prilagodljivost MATLAB-a omogućuje nam da eksperimentiramo s različitim varijantama algoritama, podešavamo parametre i analiziramo utjecaj tih parametara na performanse algoritama. To omogućuje dublje razumijevanje ponašanja robota u različitim scenarijima navigacije te olakšava razvoj novih tehnika i algoritama za planiranje putanje.

4. Tijek i ideja koda

Pri rješavanju problema pronalaska optimalne putanje robota kroz labirint koristit ćemo evolucijski algoritam. Osnovna ideja je simulirati evoluciju bića u prirodi kako bi se pronašlo rješenje. U prvom dijelu koda, definirat će se labirint kao matrica, te će se postaviti početna i ciljna pozicija robota unutar labirinta. Nakon toga, inicijalizirat će se parametri evolucijskog algoritma poput veličine populacije, duljine kromosoma i broja generacija. Glavna petlja evolucijskog algoritma sastoji se od petlje koja će obrađivati generacije populacije. Unutar svake generacije, svaka jedinka populacije ocijenit će se prema nizu kriterija koji odražavaju udaljenost od cilja, broj sudara, udaljenost od početne pozicije i druge faktore. Nakon ocjenjivanja, primijenit će se genetski operatori poput selekcije, križanja i mutacije kako bi se generirala nova populacija. Osim toga, uvest ćemo i elitizam, gdje će se najbolja jedinka iz svake generacije automatski prenesti u sljedeću generaciju. Tijekom izvođenja algoritma, praćenje fitness vrijednosti najbolje jedinke, kao i statistika poput udaljenosti, broja sudara i udaljenosti od početne pozicije, omogućit će procjenu učinkovitosti algoritma kroz generacije. Na kraju izvršenja, prikazat će se putanja koju je odabrao najbolji kromosom, zajedno s grafikonima koji ilustriraju ponašanje algoritma tijekom svih iteracija. Ovi rezultati pružit će nam uvid u proces evolucije rješenja i učinkovitost evolucijskog algoritma u pronalasku optimalne putanje robota kroz labirint.

5. Analiza koeficijenata križanja i mutacije u evolucijskim algoritmima za optimizaciju putanje robota

Optimizacija parametara u evolucijskim algoritmima, poput kombinacija koeficijenata križanja i mutacije, igra ključnu ulogu u postizanju uspješnih rezultata u različitim područjima, uključujući robotiku, strojno učenje i optimizaciju. Kombinacije ovih parametara imaju značajan utjecaj na performanse algoritma te mogu imati širok spektar posljedica na proces optimizacije. Pravilno podešavanje koeficijenata križanja i mutacije može rezultirati bržom konvergencijom algoritma, boljom raznolikošću populacije, tečnijim i preciznijim pronalaskom optimalnih rješenja te smanjenjem rizika od zaglavljivanja u lokalnim optimumima. Osim toga, ovi parametri mogu utjecati i na resurse potrebne za izvođenje algoritma, kao što su vrijeme izvođenja i memorijski zahtjevi. Napredak u optimizaciji parametara može rezultirati poboljšanjem performansi raznih algoritama za optimizaciju, što zauzvrat može imati značajan utjecaj na različite primjene, uključujući planiranje putanje robota, optimizaciju resursa, dizajn sustava, analizu podataka i druge složene probleme. Stoga je važno istraživati i razumjeti kako različite kombinacije koeficijenata križanja i mutacije utječu na performanse evolucijskih algoritama. Dublje razumijevanje ovih interakcija omogućuje razvoj boljih strategija optimizacije, te može rezultirati napretkom u raznim područjima primjene gdje se evolucijski algoritmi koriste za rješavanje složenih problema optimizacije i planiranja.

5.1. Definiranje koeficijenata mutacije i križanja

Definirat ćemo dva vektora koji će se sastojati od koeficijenata križanja i mutacije. Vrijednosti koje ćemo uzeti obično su korištene kod ovakvih problema. Nakon što smo definirali vrijednosti s kojima ćemo obaviti analizu potrebno je provesti određen broj izvođenja koda za svaku kombinaciju pc i pm.

```
% Definiranje koeficijenata krizanja i mutacije
pc = [0.6, 0.7, 0.8, 0.9];
pm = [0.05, 0.1, 0.2, 0.3];
```

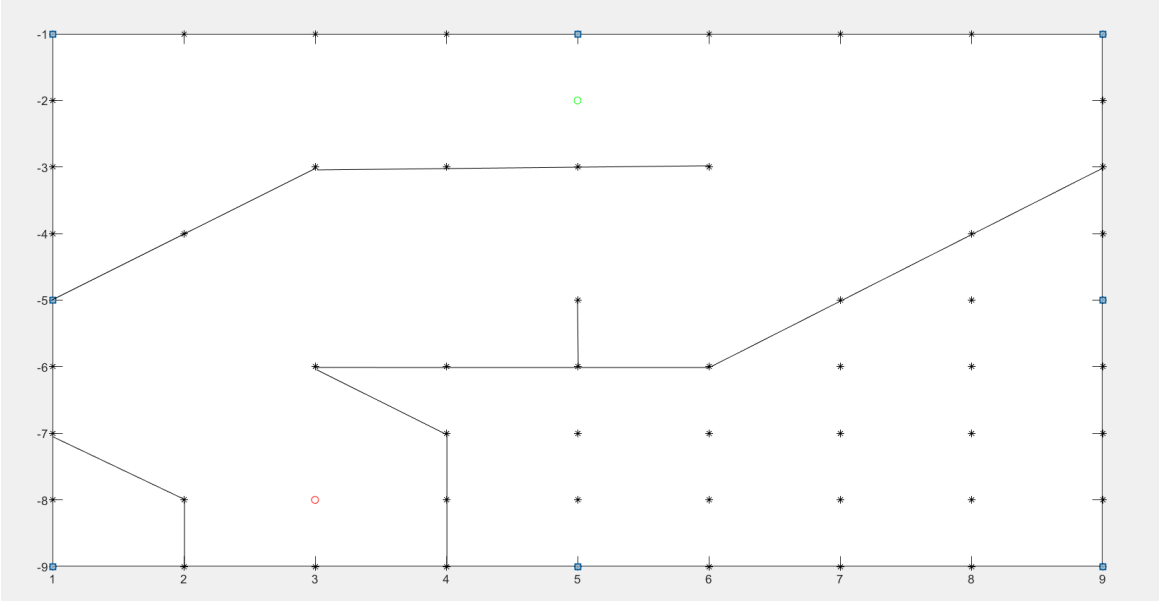
Slika 1 Vektori p_c i p_m

Slika 1 prikazuje vektore p_c i p_m na kojima će biti obavljena analiza. Za svaku kombinaciju vektora bit će provedeno 10 izvođenja te će na kraju biti prikazan prosječni rezultat kako bi u konačnici rezultat bio relevantan.

5.2. Rezultati analize

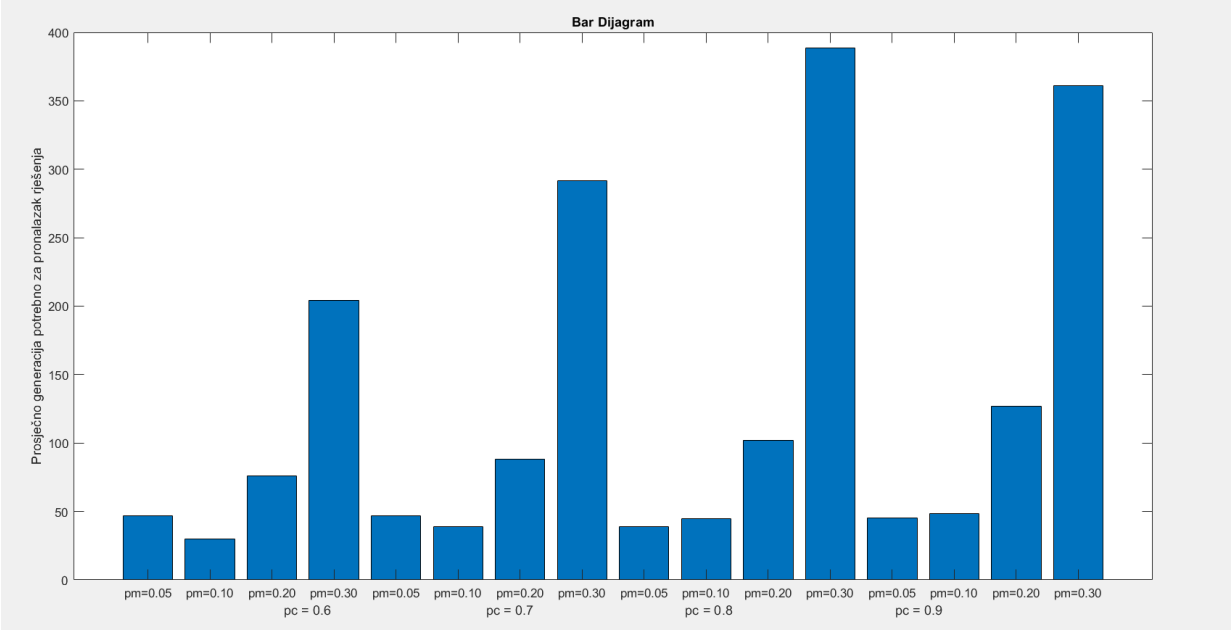
Analizu ćemo provesti na tri primjera labirinta različite složenosti kako bismo imali rezultat relevantan na više različitih modela labirinta.

5.2.1. Prvi primjer labirinta



Slika 2 Prvi primjer labirinta

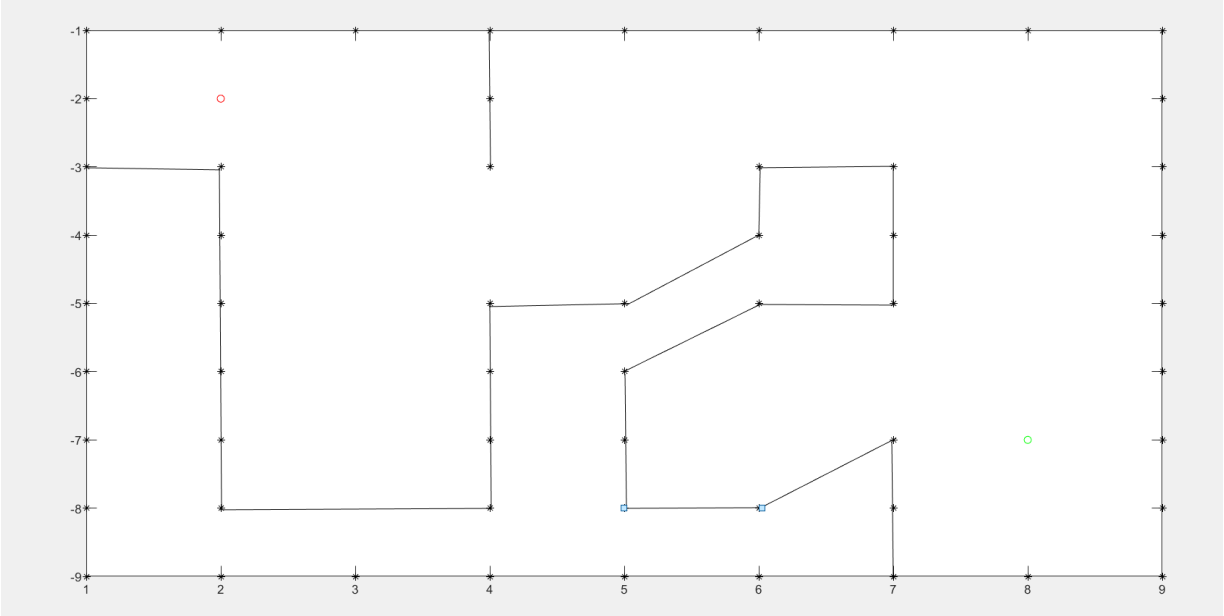
Na slici 2 vidimo početni izgled labirinta za koji ćemo provjeriti koja kombinacija koeficijenata mutacije i križanja nam najbolje odgovara.



Slika 3 Histogram 1

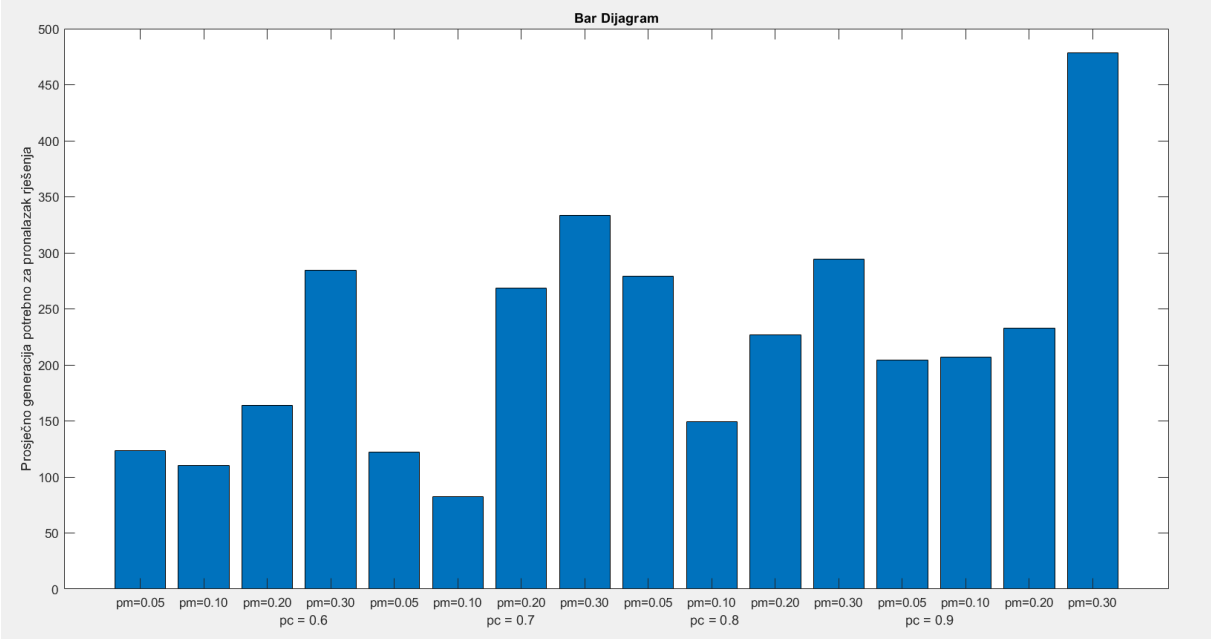
Na slici 3 prikazan je histogram sa svim mogućim kombinacijama za odabrane koeficijente mutacije i križanja te brojem generacije koja prva pronađe izlaz iz labirinta. Uočavamo kako postoji 16 mogućih kombinacija te sve kombinacije u većini slučajeva pronađu rješenje odnosno izlaz iz labirinta. Kao što je vidljivo iz dijagrama kombinacija $p_c = 0.6$ i $p_m = 0.1$ najbrže pronađe rješenje, a to je prosječno u 33. generaciji.

5.2.2. Drugi primjer labirinta



Slika 4 Drugi primjer labirinta

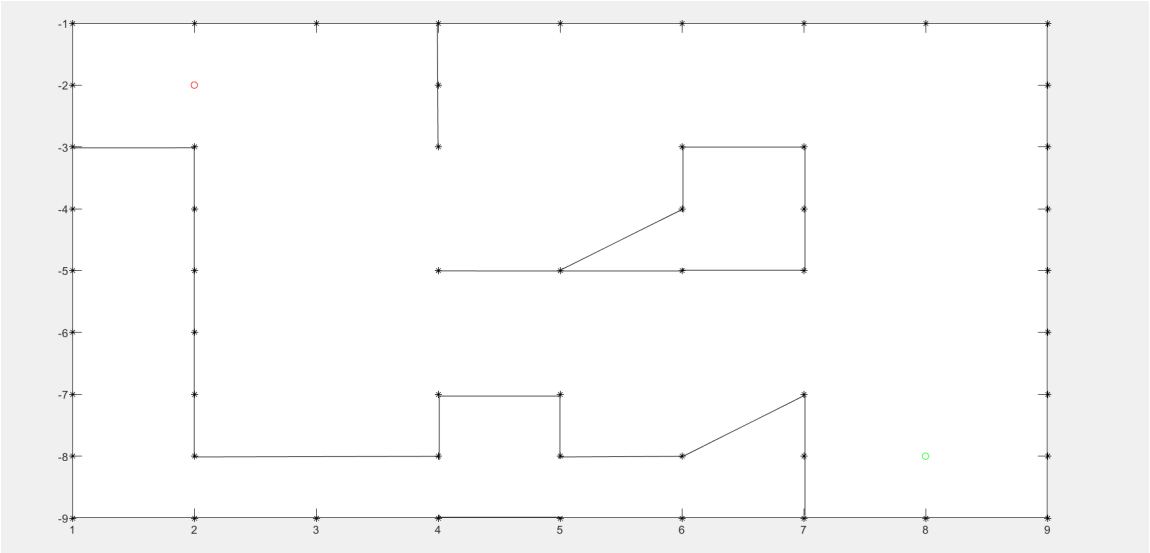
Na slici 4 vidimo početni prikaz drugog labirinta za koji ćemo također provjeriti koja kombinacija koeficijenata mu najbolje odgovara odnosno s kojom kombinacijom najbrže pronađe izlaz iz labirinta.



Slika 5 Histogram 2

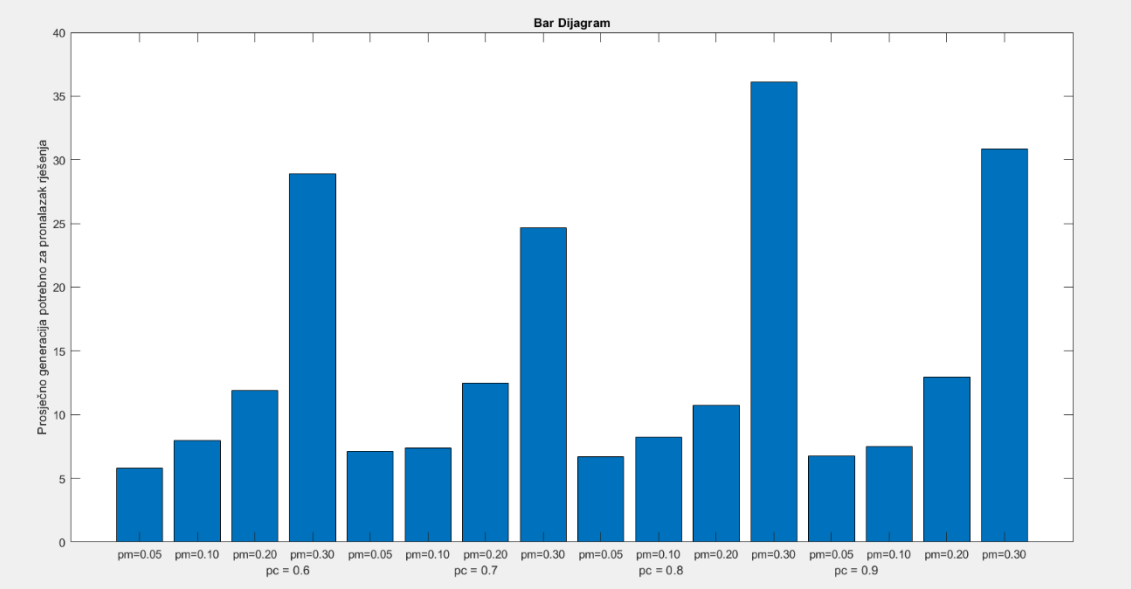
Na slici 5 uočavamo da je konvergencija do rješenja nešto sporija za ovaj primjer labirinta što je i logično s obzirom da je isti kompleksniji nego prvi primjer. Najbolja kombinacija za ovaj primjer labirinta ispostavilo se da je kombinacija gdje je pc = 0.7 i pm = 0.1 i za tu kombinaciju rješenje je pronađeno u prosječno 78. generaciji.

5.2.3. Treći primjer labirinta



Slika 6 Treći primjer labirinta

Na slici 6 vidimo treći primjer labirinta koji je ujedno i najsloženiji primjer labirinta od sva tri. Napravit ćemo i za njega analizu kako bismo dobili kombinaciju koeficijenata koja najbrže konvergira do rješenja.



Slika 7 Histogram 3

Na slici 7 je prikazan histogram za treći primjer labirinta i rezultati su pomalo iznenađujući s obzirom na kompleksnost labirinta. Kombinacija pc = 0.6 i pm = 0.05 najbrže konvergira do rješenja i to je već prosječno u 6.generaciji.

5.3. Zaključak analize

Nakon provedene analize tri primjera labirinta s različitim kombinacijama koeficijenata križanja i mutacije, utvrdili smo da najbolja kombinacija za naše potrebe jest ona koja uključuje koeficijent križanja od 0.7 i koeficijent mutacije od 0.1. Ova zaključna kombinacija proizlazi iz prosječnih najboljih vrijednosti dobivenih tijekom analize svakog primjera labirinta. Kombinirajući rezultate, primijetili smo da kombinacija pc=0.7 i pm=0.1 daje najkvalitetnije rezultate u smislu efikasnosti i brzine konvergencije algoritma. Ovaj rezultat sugerira da veći koeficijent križanja i manji koeficijent mutacije imaju pozitivan utjecaj na konvergenciju evolucijskog algoritma za našu specifičnu problematiku labirinta. Ova analiza bit će od velike koristi u daljnjem istraživanju i primjeni genetskih algoritama u rješavanju sličnih problema.

6. Rješenje problema u MATLAB okruženju

U prethodnom poglavlju obavili smo analizu najpovoljnije kombinacije koeficijenata križanja i mutacije kako bismo mogli ostvariti najbolje rezultate prilikom rješavanja problema projektiranja putanje robota u 2D labirintu. Prilikom rješavanja ovakvih problema koeficijenti i parametri su od velike važnosti te oni najviše utječu na rješenje. Problem projektiranja putanje robota u 2D labirintu riješen je u MATLAB okruženju s obzirom na sve mogućnosti i alate koje nam pruža. Kod za rješavanje problema je napisan na smislen način kako bi čitatelj najlakše mogao pratiti tijek radnji, baš kao što bi se i u prirodi odvijala evolucija. U ovom poglavlju bit će detaljno objašnjen kod i svaki njegov dio te će biti opisani neki načini rješavanja sličnog problema u povijesti.

6.1. Rješenja u prošlosti za problem planiranja trajektorije robota u labirintu

U procesu rješavanja problema projektiranja putanje robota u 2D labirintu, primijenjene su različite metode i algoritmi s ciljem pronalaska optimalne putanje. Ovaj problem je ključan u području robotike i umjetne inteligencije te zahtijeva sofisticirane tehnike kako bi se postiglo efikasno rješenje. Jedna od primijenjenih metoda je korištenje klasičnih algoritama pretraživanja grafa poput pretraživanja u širinu (BFS) i pretraživanja u dubinu (DFS). Ovi algoritmi omogućuju sustavno pretraživanje labirinta kako bi se identificirala optimalna putanja od početne točke do cilja. Uz to, implementiran je i algoritam A* koji se temelji na heurističkoj funkciji te omogućuje pronalazak najkraćeg puta uzimajući u obzir procjenu udaljenosti do cilja. Ova heuristička metoda posebno je učinkovita u situacijama gdje je potrebno brzo pronaći rješenje uz minimalnu potrošnju resursa. Dodatno, genetski algoritmi su korišteni kako bi se simulirao proces prirodne selekcije te optimizirao niz koraka ili akcija koje robot treba poduzeti. Ova heuristička optimizacijska tehnika pokazala se kao korisna u kontekstu kompleksnih labirinata gdje je potrebno pronaći optimalno rješenje [11]. Uz primjenu navedenih algoritama, razne heurističke metode poput algoritama "Right Hand Rule" ili "Left Hand Rule" također su razmatrane kako bi se pronašlo rješenje prilagođeno specifičnim uvjetima labirinta. Važno je istaknuti da su senzorske tehnologije, poput ultrazvuka, infracrvenih senzora ili kamera, također važan faktor u stvarnom okruženju robota. One omogućuju detekciju prepreka u labirintu i prilagodbu putanje robota u stvarnom vremenu, što dodatno doprinosi efikasnosti rješenja [12]. Kombinacija ovih metoda i tehnika pruža temelj za razvoj sofisticiranih sustava navigacije robota u 2D labirintima, otvarajući put daljnjim istraživanjima i primjenama u području robotike i umjetne inteligencije.

6.2. Početna inicijalizacija programa

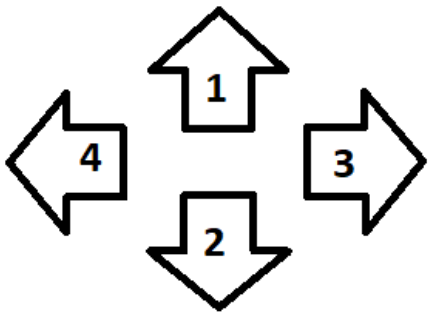
U početnom dijelu koda, prvo se inicijalizira labirint definiranjem matrice A, gdje su prepreke označene s 1, a prolazi s 0. Nakon toga, unosimo parametre koji određuju populaciju robota, duljinu njihovih kromosoma i broj generacija. Definiraju se i koeficijenti križanja i mutacije, koji su ključni parametri genetskog algoritma. Zatim se određuju početna i završna pozicija robota u labirintu, što se odražava promjenom vrijednosti odgovarajućih elemenata matrice A. Nakon inicijalizacije, generira se populacija robota s nasumično generiranim kromosomima. Svaki kromosom određuje korake koje robot može poduzeti unutar labirinta. Vrijednosti kromosoma odgovaraju koracima gore, dolje, lijevo i desno. Ovaj dio koda postavlja temelje za genetski algoritam koji će se koristiti za rješavanje problema pronalaženja optimalne putanje robota kroz labirint.

```
A = [1, 1, 1, 1, 1, 1, 1, 1, 1;
     1, 0, 0, 0, 0, 0, 0, 0, 1;
     1, 0, 1, 1, 1, 1, 0, 0, 1;
     1, 1, 0, 0, 0, 0, 0, 1, 1;
     1, 0, 0, 0, 1, 0, 1, 1, 1;
     1, 0, 1, 1, 1, 1, 1, 1, 1;
     1, 0, 0, 1, 1, 1, 1, 1, 1;
     1, 1, 0, 1, 1, 1, 1, 1, 1;
     1, 1, 1, 1, 1, 1, 1, 1, 1;];
```

Slika 8 Definiranje početnog labirinta

```
 % Inicijalizacija populacije
P = input('Broj kromosoma (broj jedinki u populaciji): ');
K = input('Duzina kromosoma (broj koraka robota): ');
G = input('Broj iteracija (broj generacija): ');
count =0;

% Definiranje koeficijenta krizanja i mutacije
pc = 0.7;
pm = 0.1;

% Odredivanje startne pozicije
Xulaz = 2;
Yulaz = 2;
A(Xulaz, Yulaz) = 7;

% Odredivanje izlazne pozicije
Xizlaz = 8;
Yizlaz = 8;
A(Xizlaz, Yizlaz) = 8;

% Generiranje populacije
Populacija = randi(4, P, K);
```

Slika 9 Definiranje parametara za algoritam



Slika 10 Logika kretanja robota

```
A = [1, 1, 1, 1, 1, 1, 1, 1, 1;
     1, 0, 0, 0, 8, 0, 0, 0, 1;
     1, 0, 1, 1, 1, 1, 0, 0, 1;
     1, 1, 0, 0, 0, 0, 0, 1, 1;
     1, 0, 0, 0, 1, 0, 1, 1, 1;
     1, 0, 1, 1, 1, 1, 1, 1, 1;
     1, 0, 0, 1, 1, 1, 1, 1, 1;
     1, 1, 7, 1, 1, 1, 1, 1, 1;
     1, 1, 1, 1, 1, 1, 1, 1, 1;];
```

Slika 11 Labirint s ulazom i izlazom

Na slikama 8 i 9 uočavamo početne parametre te oblik labirinta koji smo zadali. Parametri kao što su broj koraka, iteracija i populacija bit će zadani kada se pokrene program. Ta tri parametra su od velike važnosti i oni će biti zadani tek kad pokrenemo program zato što tako možemo iterirati više mogućih kombinacija na brži i lakši način. Na slici 5 prikazan je labirint u kojem je s brojem 7 označen ulaz, a s brojem 8 izlaz iz labirinta.

6.3. Dio programa za iteracije kroz generacije i populacije

U ovom dijelu koda, prvotno se obavlja iteracija po generacijama a zatim se obavlja iteracija kroz populaciju robota kako bi se utvrdilo kako se svaka jedinka kreće unutar labirinta. Iteracije se vrše s pomoću for petlji zato što je za ovaj slučaj to najlakši i ujedno čitatelju najpristupačniji način. Petlja se izvršava za svaku jedinku u populaciji, što je označeno varijablom P. Unutar ove petlje, robot se inicijalno postavlja na početnu poziciju (koordinate Xulaz i Yulaz). Zatim se provjerava svaki korak koji robot može poduzeti, što je određeno duljinom kromosoma (K). Za svaki korak se provjerava je li dozvoljen prema labirintu i ako jest, robot se pomakne u tom smjeru. Ključni dio petlje je switch naredba koja analizira vrijednost trenutnog gena u kromosomu. Ovisno o vrijednosti gena, robot se pomakne gore, dolje, lijevo ili desno, pri čemu se provjerava postoji li prepreka (vrijednost 1 u matrici A). Ako je prepreka detektirana, robot zabilježi sudar. Unutar unutarnje petlje, praćenje koraka i sudara omogućuje procjenu uspješnosti kretanja jedinke kroz labirint. Ako jedinka stigne do izlazne pozicije, petlja se prekida kako bi se optimiziralo vrijeme izvršavanja programa. Ovaj dio koda omogućuje simuliranje kretanja robota kroz labirint, uz praćenje broja koraka i sudara koji se događaju tijekom tog kretanja. Ovi podaci ključni su za procjenu uspješnosti pojedinih jedinki unutar populacije te za prilagodbu genetskog algoritma kako bi se postiglo optimalno rješenje.

6.4. Fitnes funkcija

U sljedećem dijelu koda obavlja se izračunavanje fitnesa za svaku jedinku unutar populacije robota. Fitnes predstavlja mjeru prilagođenosti jedinke za okolinu ili problem koji se rješava. Ovaj fitnes koristi se kao osnova za selekciju, križanje i mutaciju jedinki u genetskom algoritmu. Ključne komponente izračunavanja fitnesa uključuju različite parametre koji odražavaju ponašanje robota unutar labirinta. Težina za sudare (w_1) predstavlja kolizije koje robot doživljava tijekom kretanja labirintom, dok težina za broj koraka (w_2) odražava želju za minimiziranjem broja koraka potrebnih robotu da stigne do cilja. Funkcija je normalizirana tako da ne bi uslijed određenog parametra algoritam otišao u krivom smjeru. Nakon što su određene težine, izračunava se fitnes za svaku jedinku u populaciji. To se postiže iteracijom kroz sve jedinke. Formula za izračun fitnesa kombinira različite faktore u skladu s definiranim težinama. Nakon izračunavanja fitnesa za svaku jedinku, određuje se najveći fitnes unutar populacije. Najveći fitnes i pripadajuća jedinka koriste se za daljnje korake evolucijskog algoritma. Osim apsolutnog fitnesa, također se izračunava relativni fitnes za svaku jedinku. Relativni fitnes odražava udio apsolutnog fitnesa jedinke u ukupnom zbroju fitnesa svih jedinki unutar populacije. To omogućuje odabir roditelja za sljedeću generaciju genetskog algoritma na temelju njihove prilagodbe problemu koji se rješava.

$$Fitnes(i) = \frac{Korak\_sudara(i)}{w_1} + \frac{w_2}{Koraci(i)} + Udaljenost\_start(i) \cdot \frac{1}{Udaljenost(i)+0.1} \quad \textbf{(6. 1.)}$$

Izraz 6. 1. predstavlja formulu po kojoj se računa fitnes za svaku jedinku unutar populacije nakon čega se računa i relativni fitnes po formuli:

$$Relativni\ Fitnes(i) = \frac{Fitnes(i)}{sum(Fitnes)} \quad \textbf{(6. 2.)}$$

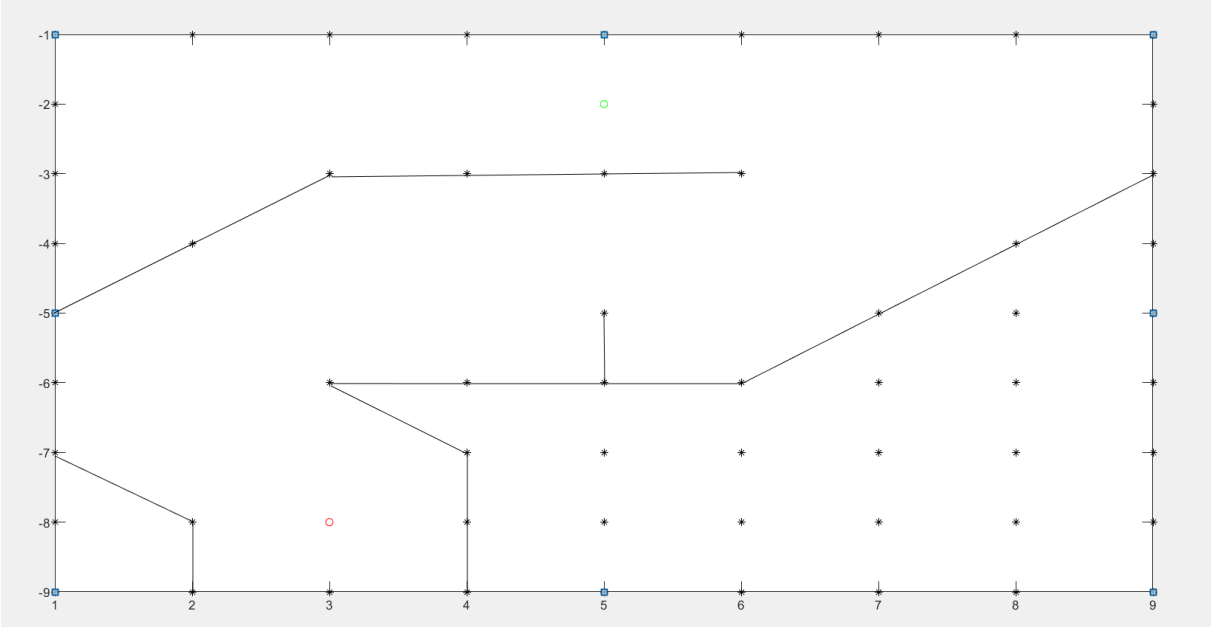
6.5. Glavni dio programa

Glavni dio koda sastoji se od selekcije, križanja, mutacije te su najbolja rješenja očuvana pomoću elitizma koji je od velike koristi u ovom našem problemu. U selekciji koristili smo se ruletnim pravilo što nam omogućuje odabir roditelja proporcionalno njihovoj relativnoj prilagodbi, što znači da jedinke koje imaju bolje prilagodbe imaju veću vjerojatnost da budu odabrane za reprodukciju. U kontekstu ovog problema, to znači da će se preferirati roditelji koji imaju bolje prilagodbe u pronalaženju optimalne putanje kroz labirint. Križanje s jednom točkom prekida omogućit će nam kombiniranje genetskog materijala (putanje) izabranih roditelja. U ovom kontekstu, to znači da se putanje robota mogu kombinirati kako bi se stvorile nove putanje koje možda imaju bolje osobine u smislu pronalaska cilja u labirintu. Uniformna mutacija mijenja gene (putanje) slučajnim odabirom s određenom vjerojatnošću. U problemu putanje robota, uniformna mutacija omogućuje nam uvođenje varijacija u putanji, što može pomoći u istraživanju različitih mogućnosti i izbjegavanju lokalnih optimuma. U mom kodu, proces započinje odabirom roditelja korištenjem ruletne selekcije, gdje se preferiraju jedinke s boljim prilagodbama. Zatim se primjenjuje križanje s jednom točkom prekida kako bi se stvorila nova populacija potomaka. Ova nova populacija potom prolazi kroz fazu uniformne mutacije, gdje se slučajno mijenjaju geni (putanje) s određenom vjerojatnošću. Kroz ove korake, algoritam traži optimalnu putanju kroz labirint promjenom i kombiniranjem putanja robota u svakoj generaciji, pri čemu se preferiraju putanje koje vode do cilja s najmanje prepreka ili najkraćim putem. Isto tako najbolje jedinke očuvane su elitizmom što nam doprinosi boljim rezultatima.

7. Rezultati

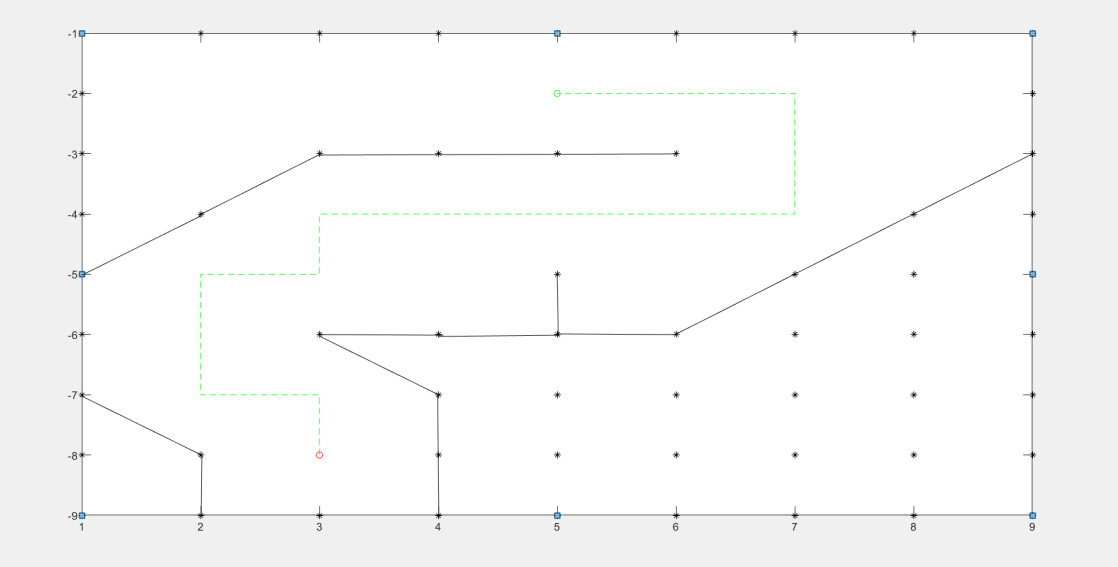
Glavna tematika ovog završnog rada jest planiranje trajektorije robota u 2D okruženju. Za 2D okruženje izabrali smo labirint. Kod napisan na temelju evolucijskog algoritma sa svim parametrima i koeficijentima koji su pomno odabrani treba rezultirati pronalaskom najpovoljnije putanje za izlaz iz labirinta. Napravit ćemo analizu koda na tri labirinta kako bismo u potpunosti provjerili ispravnost koda.

7.1. Prvi primjer labirinta



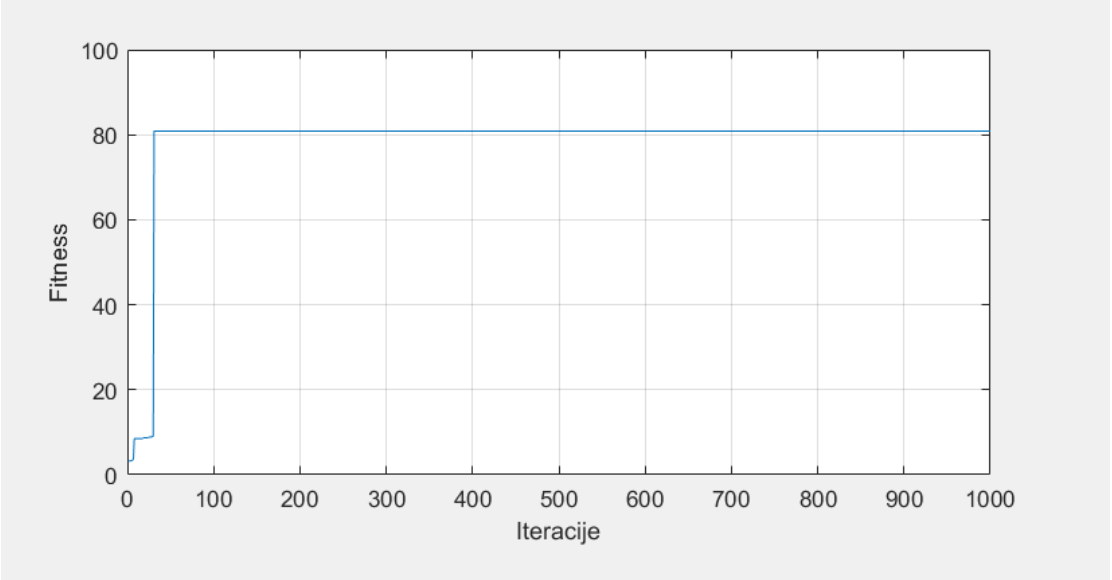
Slika 12 Početni labirint

Prvi labirint je jednostavne strukture kako bismo mogli provjeriti ispravnost cjelokupnog koda. Crveni kružić predstavlja ulaz, zeleni kružić predstavlja izlaz dok crne zvjezdice predstavljaju prepreke odnosno zidove. Zidovi su spojeni linijama kako nebi došlo do zabune da robot može proći između prepreka pod kutem. Kao što sam ranije spomenuo robotu je potrebno na početku zadati iznose parametara kao što su veličina populacije, broj koraka i broj generacija. Za ovaj primjer odabrali smo veličinu populacije 1000, broj koraka jednak je 16 što je više nego što je potrebno do izlaza te smo zadali broj generacija jednak 1000.



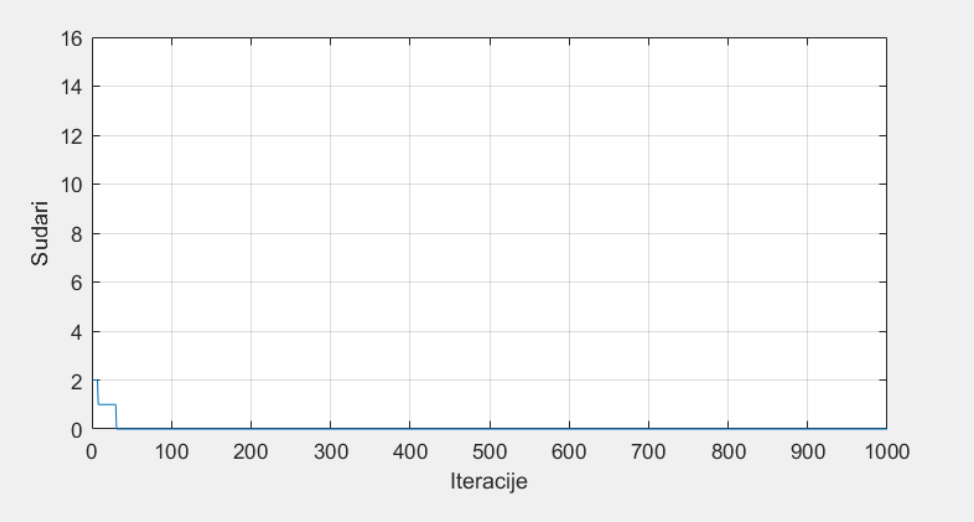
Slika 13 Putanja robota u početnom labirintu

Na slici 13 vidimo kako je robot uspio pronaći izlaz iz labirinta te nije imao nepotrebnih koraka što nam također govori o ispravnosti cjelokupnog programa.



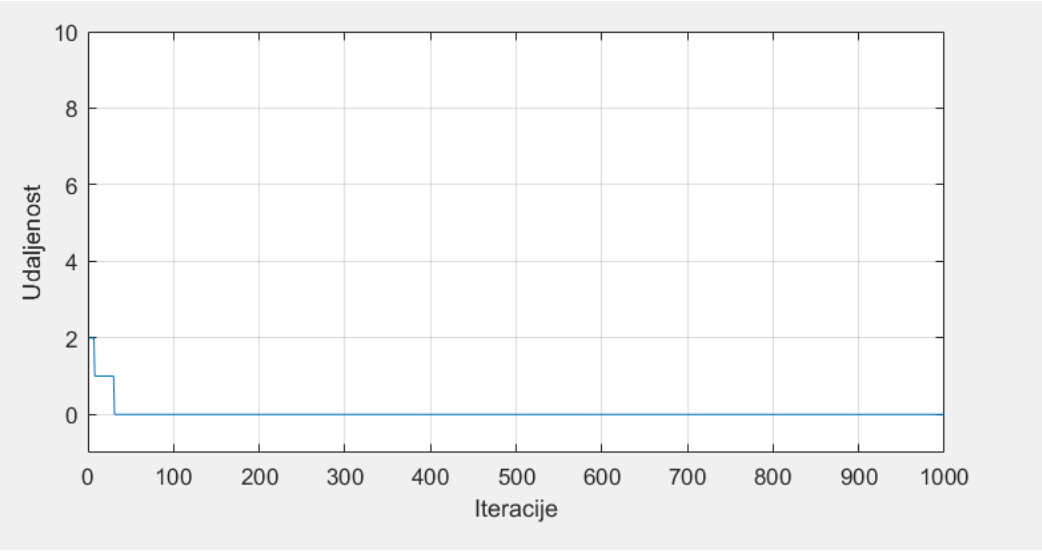
Slika 14 Fitnes tijekom generacija

Na slici 14 uočavamo kako je fitnes došao do određenog iznosa već u 30. generaciji te je taj fitnes ostao očuvan tijekom ostalih generacija iz razloga što je to bilo najbolje moguće rješenje. Rješenje koje smo dobili vrlo brzo je pronađeno što nam također govori i o kvaliteti koeficijenata koje smo odabrali ranije.

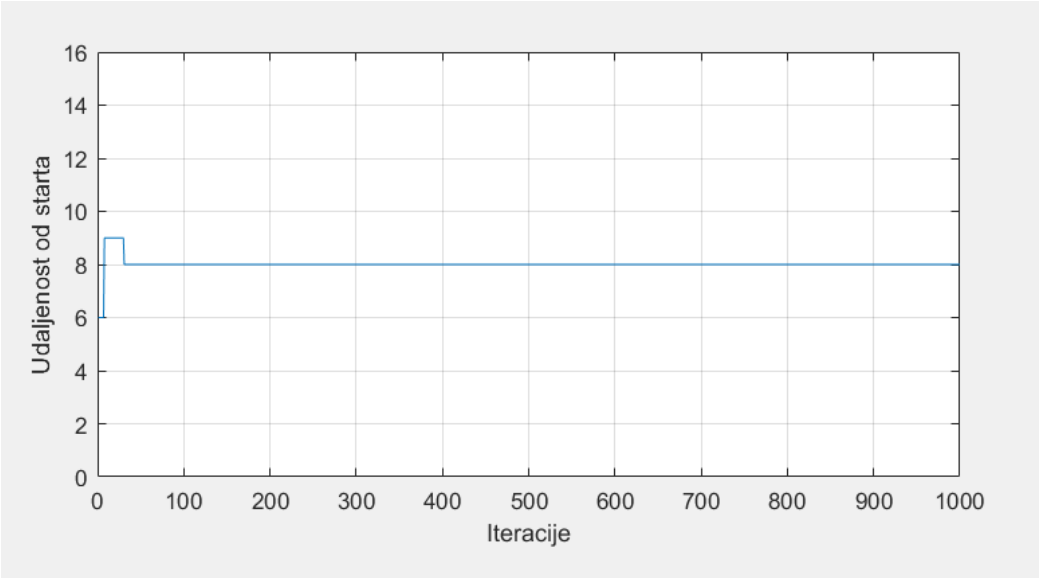


Slika 15 Broj sudara po generacijama

Na slici 15 prikazani su nam sudari kroz generacije. Uočavamo kako je u početku bilo nekoliko sudara, dok je u generaciji kada je pronađeno najbolje moguće rješenje prestalo biti sudara i zbog elitizma je ta jedinka očuvana do samog kraja iteracija.



Slika 16 Udaljenost od cilja po generacijama

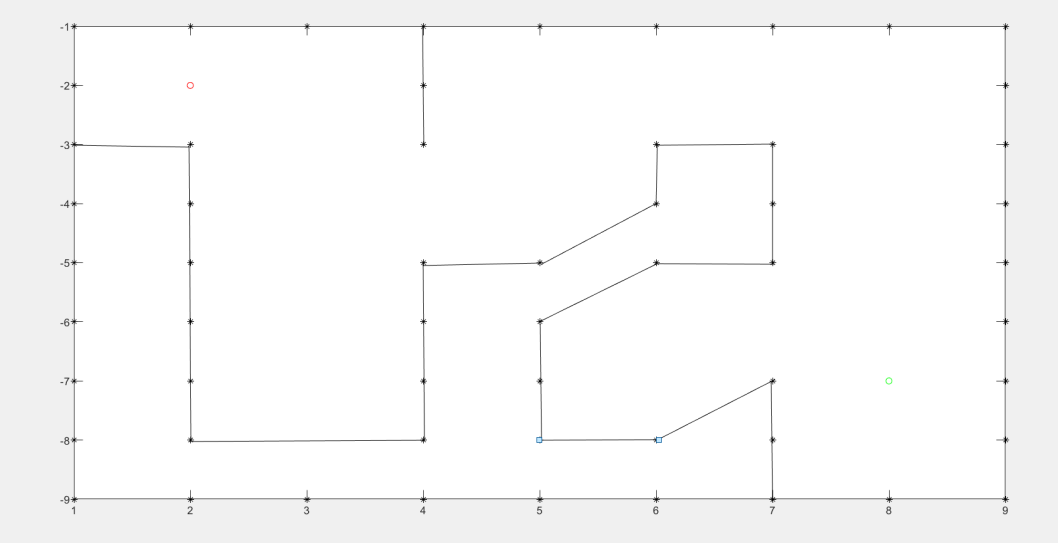


Slika 17 Udaljenost od starta po generacijama

Na slici 17 vidimo kako se mijenjala udaljenost od cilja po generacijama, dok na slici 12 vidimo kako se mijenjala udaljenost od starta. Udaljenost od starta nam je važna iz razloga što ukoliko ne bismo imali taj parametar, tada bi nam postojala šansa da se robot ne kreće. Iz grafova je također vidljiva tvrdnja kao što je i u prijašnjim grafovima navedeno da je najbolje moguće rješenje pronađeno u 30. generaciji te je očuvano do samog kraja iteracija.

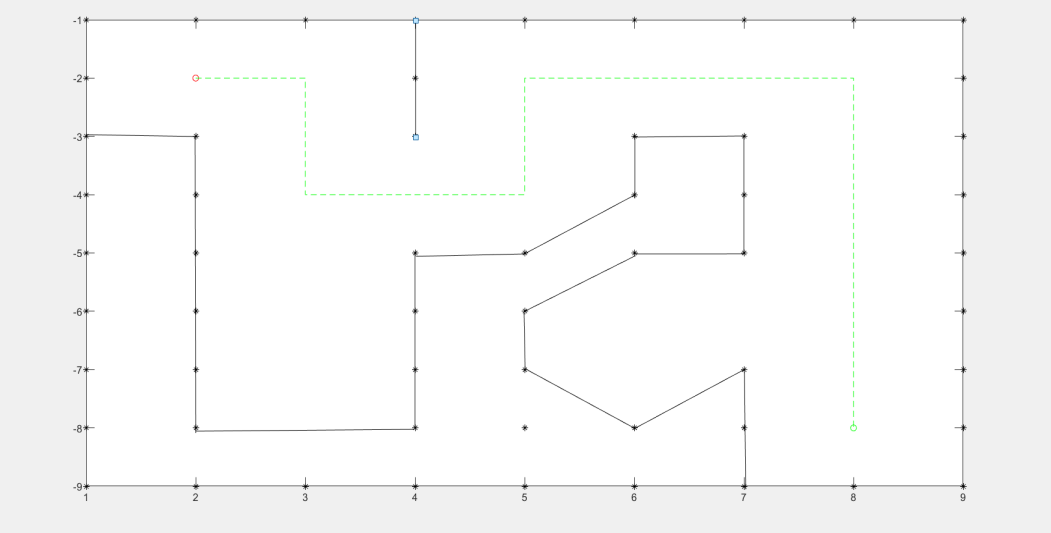
7.2. Drugi primjer labirinta

Drugi primjer labirinta je nešto kompleksniji te zahtjeva i veći broj koraka. U drugom labirintu potrebno je 16 koraka da bi robot izašao iz labirinta i potrebno je više promjena smjera.



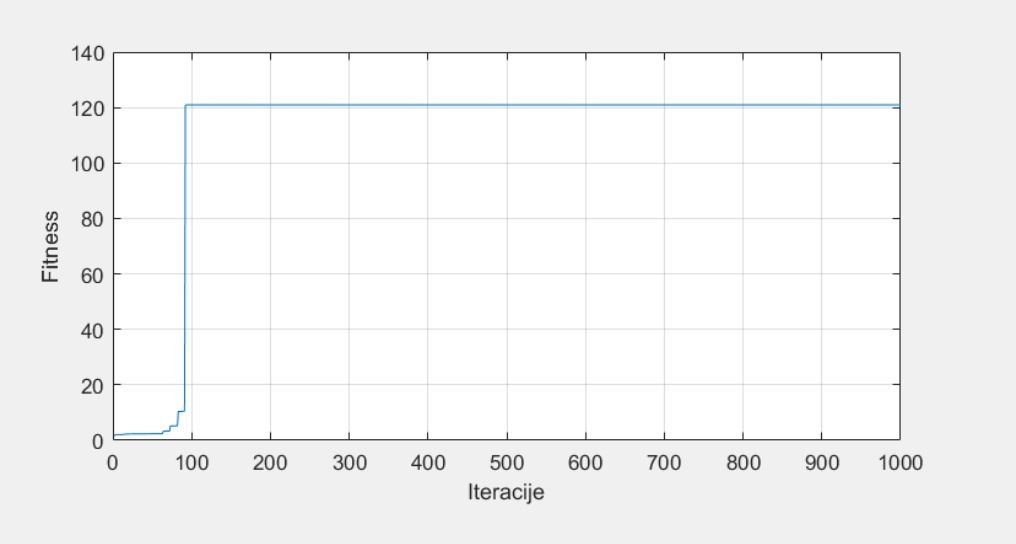
Slika 18 Drugi labirint

Na slici 18 prikazan je drugi labirint i na njemu crveni kružić predstavlja ulaz dok je zeleni kružić izlaz iz labirinta. Crne linije predstavljaju zidove odnosno prepreke kroz koje robot ne može proći. Za početne parametre zadali smo mu broj generacija jednak 1000, veličinu populacije jednaku također 1000 i broj koraka 18.

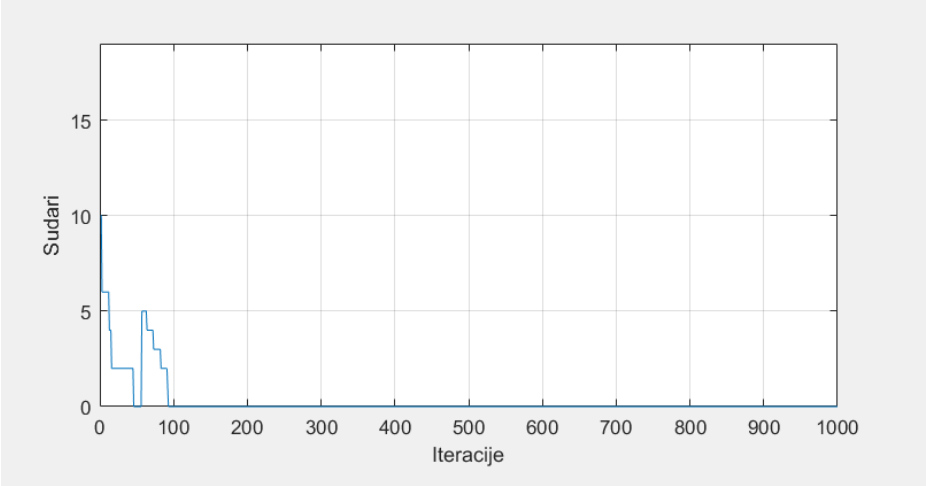


Slika 19 Putanja robota do izlaza u drugom labirintu

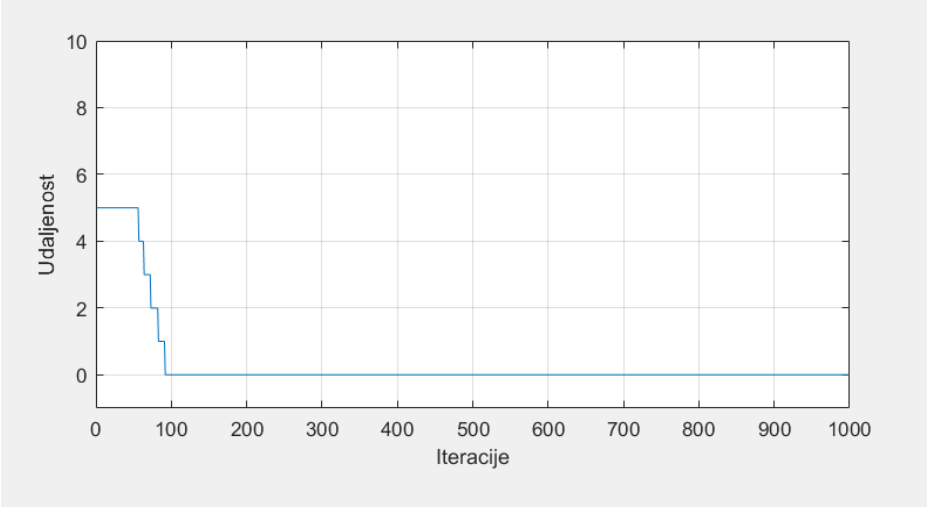
Na slici 19 vidimo kako je robot uspješno izašao iz labirinta s najmanjim mogućim brojem koraka što znači da nije radio nepotrebne korake odnosno pronađeno je najbolje moguće rješenje.



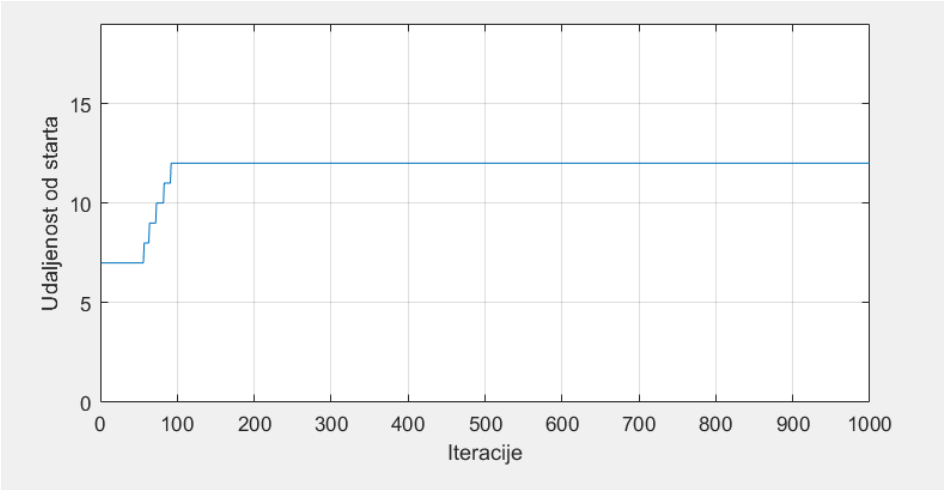
Slika 20 Fitnes drugog labirinta po generacijama



Slika 21 Sudari u drugom labirintu po generacijama



Slika 22 Udaljenost od cilja u drugom labirintu po generacijama

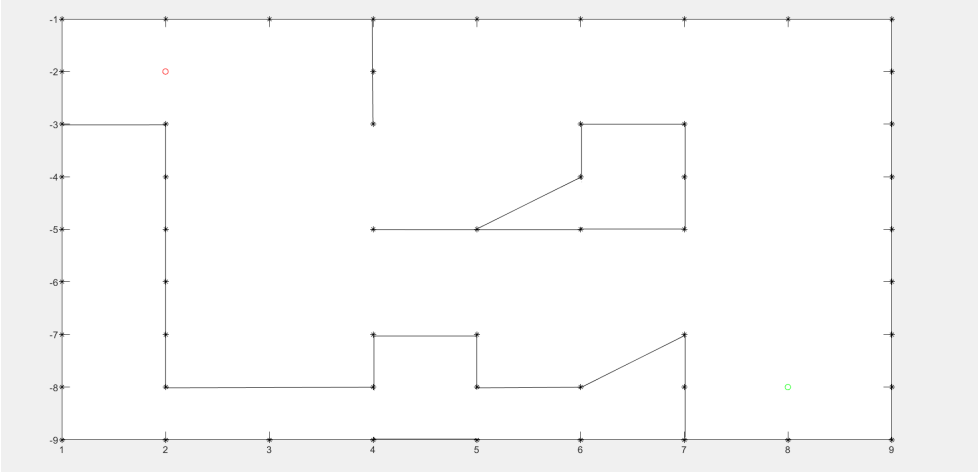


Slika 23 Udaljenost od starta u drugom labirintu po generacijama

Kao što vidimo u grafovima robot je u ovom labirintu pronašao rješenje nešto kasnije odnosno u 93. generaciji. Takav rezultat je i očekivan s obzirom na to da se radi o nešto kompliciranijem labirintu gdje je potreban veći broj koraka. Rješenje je i dalje vrlo dobro te je vrlo brzo robot pronašao rješenje. Iz grafova vidimo da je pri pronalasku najboljeg mogućeg rješenja očuvao tu jedinku do samog kraja s obzirom na to da nije mogao pronaći bolju od nje.

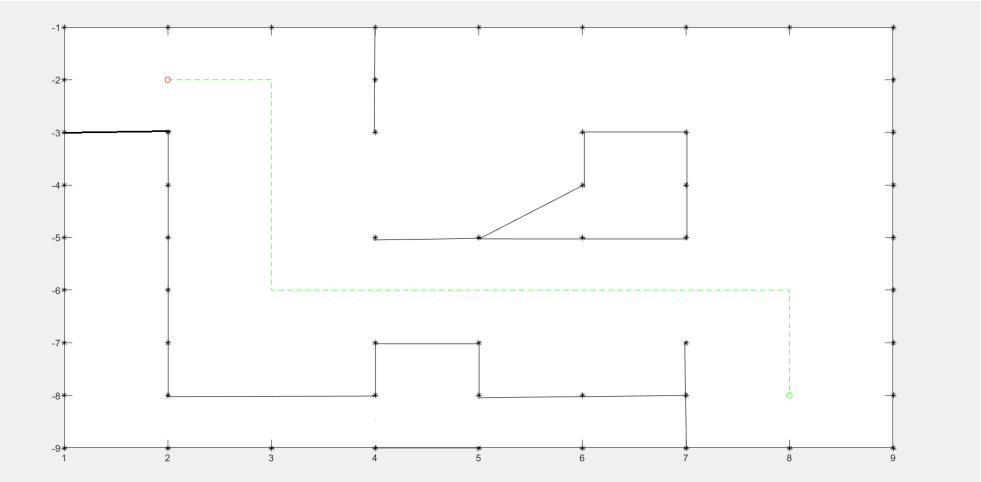
7.3. Treći primjer labirinta

Treći labirint je najkompleksniji od do sad prikazanih. Ima dvije moguće rute do cilja, a poanta ovog primjera labirinta je da on odabere onu najbržu rutu odnosno onu s najmanjim potrebnim brojem koraka.



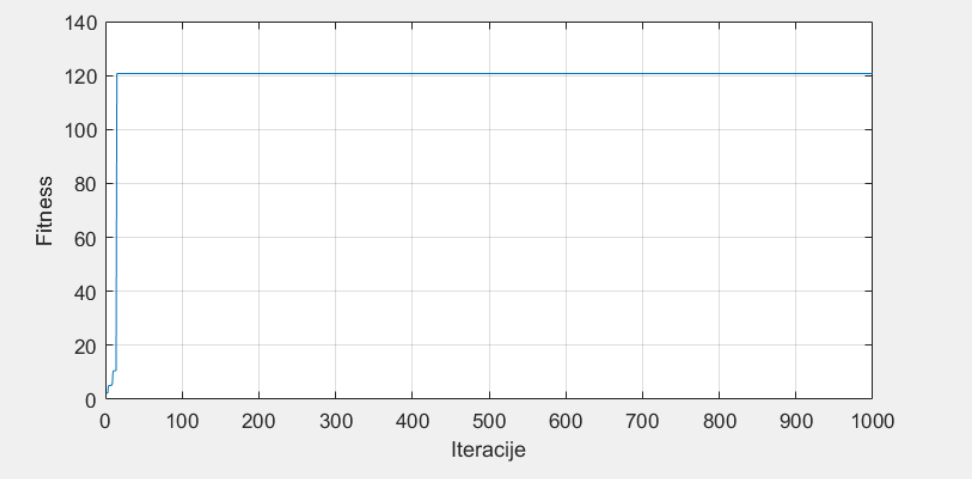
Slika 24 Treći primjer labirinta

Na slici 24 vidljiv je labirint. Do izlaza mu je potrebno minimalno 12 koraka dok može doći do izlaza i drugim putem koji je nešto duži. Za početne parametre zadat ćemo mu broj populacija 1000, broj generacija jednak 1000 i broj koraka 16 zbog toga što je 16 koraka dovoljno i da dužim putem dođe do labirinta.

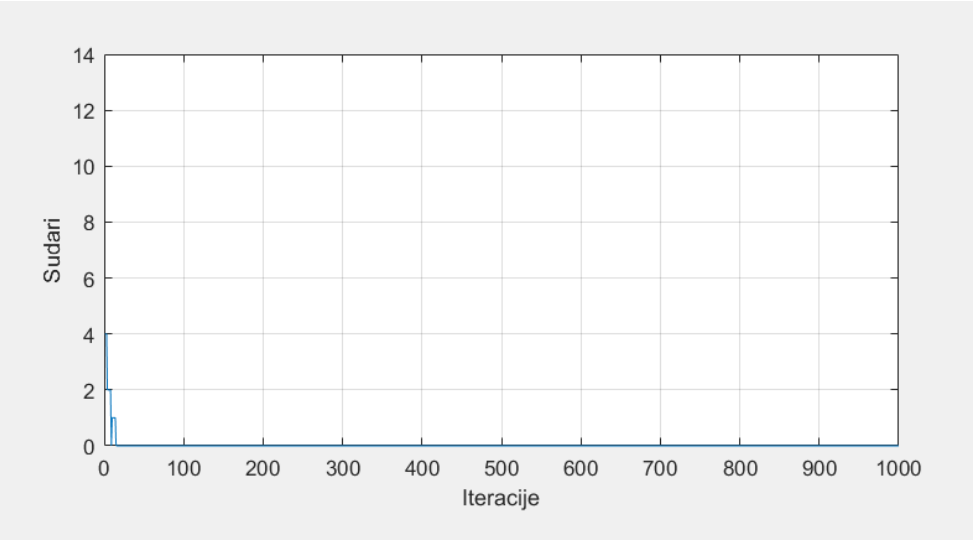


Slika 25 Putanja robota za izlaz u trećem labirintu

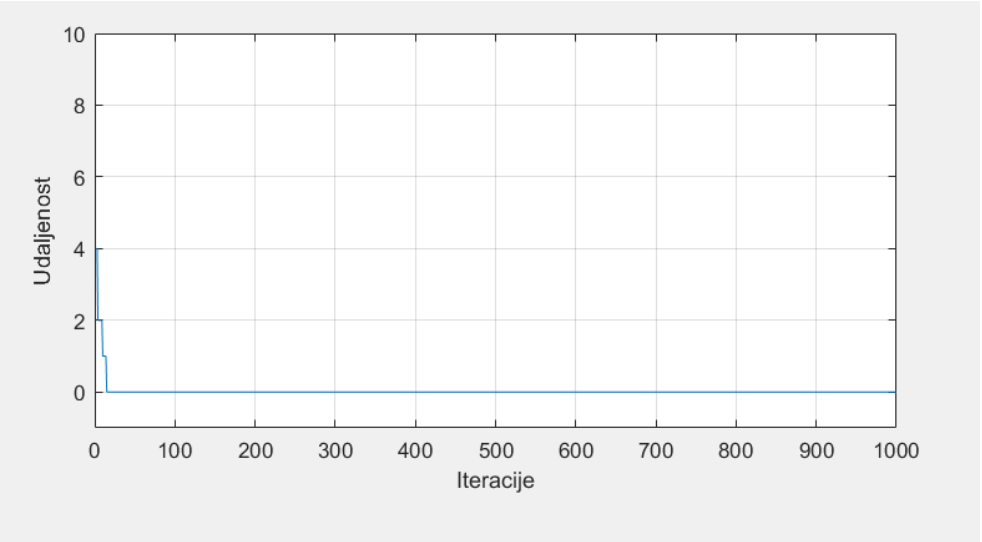
Na slici 25 vidimo kako je robot uspješno pronašao putanju do izlaza te je odabrao kraći put. Ispunili smo glavni cilj ovog labirinta, a to je da odabere kraći put od dva moguća.



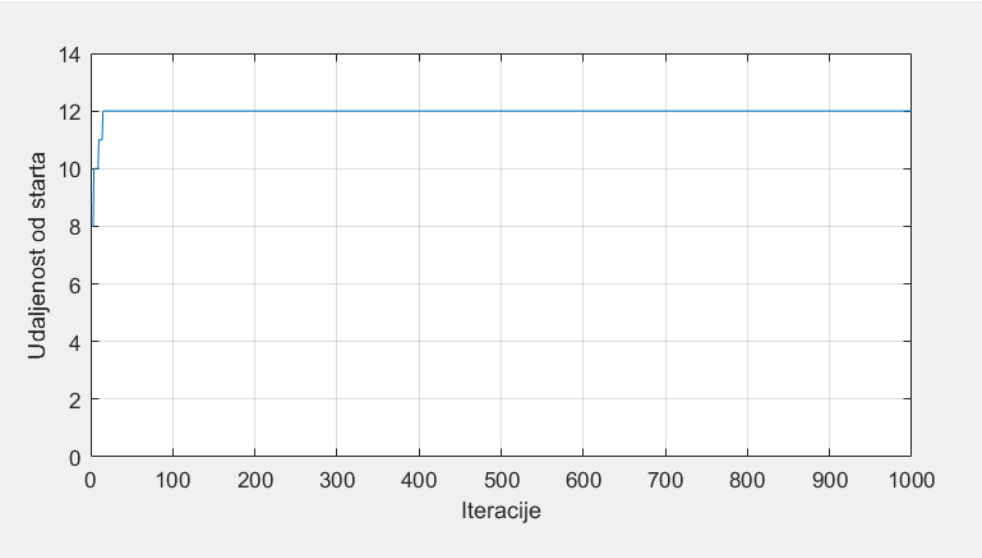
Slika 26 Fitnes trećeg labirinta po generacijama



Slika 27 Sudari u drugom labirintu po generacijama



Slika 28 Udaljenost od cilja u trećem labirintu po generacijama



Slika 29 Udaljenost od starta u trećem labirintu po generacijama

Na slikama 26, 27, 28 i 29 uočavamo kako je za ovaj labirint rješenje pronađeno brže nego što je kod prva dva labirinta. Taj podatak je pomalo iznenađujuć, no s obzirom na evolucijskim algoritam i njegove komponente moguće je da se dogodi ovakav slučaj. Rješenje je pronađeno već u 21. generaciji te je elitizmom očuvana jedinka.

8. Zaključak

Kroz ovaj rad, uspješno sam implementirao evolucijski algoritam za pronalaženje najbolje putanje robota u 2D labirintu s ciljem izlaska. Korištenje evolucijskih algoritama pokazalo se kao učinkovit pristup za rješavanje složenog problema planiranja putanje u dinamičkim okruženjima. Analizom rezultata eksperimenata, utvrdio sam da kombinacije koeficijenata križanja i mutacije imaju značajan utjecaj na brzinu konvergencije algoritma, raznolikost populacije te kvalitetu pronađenih putanja. Postignuti rezultati pružaju dublje razumijevanje mehanizama evolucijskih algoritama u kontekstu problema planiranja putanje robota. Unatoč postignutim uspjesima, projekt također otvara vrata za daljnja istraživanja i poboljšanja. U budućnosti, moglo bi se istražiti više strategija selekcije, križanja i mutacije, kao i optimizacija parametara algoritma, kako bi se postigla još bolja kvaliteta putanja i brža konvergencija. Također, implementacija naprednih tehnika za dinamičko prilagođavanje strategija pretraživanja mogla bi poboljšati prilagodljivost algoritma u različitim vrstama labirinta i okruženja. Nadalje, integracija dodatnih informacija o okruženju, poput dinamičkih prepreka ili promjenjivih uvjeta, mogla bi dodatno obogatiti proces planiranja putanje i rezultirati učinkovitijim rješenjima. Uvođenje tehnika dubokog učenja ili neuronskih mreža također bi moglo pružiti nove perspektive u rješavanju problema planiranja putanje robota u kompleksnim okruženjima. U konačnici, ovaj projekt predstavlja tek početak istraživanja u području optimizacije planiranja putanje robota s pomoću evolucijskih algoritama. Njegovi rezultati i zaključci pružaju temelj za daljnji razvoj i inovacije, otvarajući nove mogućnosti za primjenu u stvarnim robotskim sustavima i autonomnim vozilima.

LITERATURA

[1] Intoruction to evolutionary computing, A.E. Eiben. J.E. Smith, Springer Verlag, 2008

[2] Ćurković, P.; Čehulić, L. Diversity Maintenance for Efficient Robot Path Planning. *Appl. Sci.* **2020**, *10*, 1721. https://doi.org/10.3390/app10051721

[3] P. Curkovic and B. Jerbic, "Honey-bees optimization algorithm applied to path planning problem," *International Journal of Simulation Modelling*, vol. 6, no. 3, pp. 154–164, 2007.

[4] Ćurković, P., Jerbić, B., Stipančić, T.: Coordination of robots with overlapping workspaces based on motion co-evolution. Int. J. Simul. Model. **12**(1), 27–38 (2013)

[5] Curkovic P, Jerbic B, Stipancic T. Co-Evolutionary Algorithm for Motion Planning of Two Industrial Robots with Overlapping Workspaces. International Journal of Advanced Robotic Systems. 2013;10(1). doi:10.5772/54991

[6] Curkovic P. 2008. Hybridization of adaptive genetic algorithm and ART 1 neural architecture for efficient path planning of a mobile robot. Transactions of FAMENA, 32, 2 11-20.

[7] https://www.sciencedirect.com/science/article/pii/S095741742300756X, posjećeno 10.02.2024

[8] https://www.mathworks.com/ posjećeno 10.02.2024

[9] https://hr.wikipedia.org/wiki/ posjećeno 10.02.2024

[10] https://www.larksuite.com/en_us/topics/ai-glossary/evolutionary-algorithm posjećeno 10.02.2024

[11] J. H. Holland :"Adaptation in Natural and Artificial Systems," University of Michigan Press, 1975.

[12] S. Thrun, W. Burgard, and D. Fox : "Probabilistic Robotics," MIT Press, 2005.

PRILOZI

I. Kod u MATLAB okruženju

```
% DEFINIRANJE LABIRINTA

% Prvi primjer labirinta Ulaz(8,3) Izlaz(3,2)
%  A = [1, 1, 1, 1, 1, 1, 1, 1, 1;
%       1, 0, 0, 0, 0, 0, 0, 0, 1;
%       1, 0, 1, 1, 1, 1, 0, 0, 1;
%       1, 1, 0, 0, 0, 0, 0, 1, 1;
%       1, 0, 0, 0, 1, 0, 1, 1, 1;
%       1, 0, 1, 1, 1, 1, 1, 1, 1;
%       1, 0, 0, 1, 1, 1, 1, 1, 1;
%       1, 1, 0, 1, 1, 1, 1, 1, 1;
%       1, 1, 1, 1, 1, 1, 1, 1, 1;];

% Drugi primjer labirinta Ulaz(2,2) Izlaz(8,8)
%  A = [1, 1, 1, 1, 1, 1, 1, 1, 1;
%       1, 0, 0, 1, 0, 0, 0, 0, 1;
%       1, 1, 0, 1, 0, 1, 1, 0, 1;
%       1, 1, 0, 0, 0, 1, 1, 0, 1;
%       1, 1, 0, 1, 1, 1, 1, 0, 1;
%       1, 1, 0, 1, 1, 0, 0, 0, 1;
%       1, 1, 0, 1, 1, 0, 1, 0, 1;
%       1, 1, 0, 1, 1, 1, 1, 0, 1;
%       1, 1, 1, 1, 1, 1, 1, 1, 1;];

% Treci primjer labirinta Ulaz(2,2) Izlaz(8,8)
A = [1, 1, 1, 1, 1, 1, 1, 1, 1;
     1, 0, 0, 1, 0, 0, 0, 0, 1;
     1, 1, 0, 1, 0, 1, 1, 0, 1;
     1, 1, 0, 0, 0, 1, 1, 0, 1;
     1, 1, 0, 1, 1, 1, 1, 0, 1;
     1, 1, 0, 0, 0, 0, 0, 0, 1;
     1, 1, 0, 1, 1, 0, 1, 0, 1;
     1, 1, 0, 1, 1, 1, 1, 0, 1;
     1, 1, 1, 1, 1, 1, 1, 1, 1;];


% Inicijalizacija populacije
P = input('Broj kromosoma (broj jedinki u populaciji): ');
K = input('Duzina kromosoma (broj koraka robota): ');
G = input('Broj iteracija (broj generacija): ');
count =0;
```

```
% Definiranje koeficijenta krizanja i mutacije
pc = 0.7;
pm = 0.1;

% Odredivanje startne pozicije
Xulaz = 2;
Yulaz = 2;
A(Xulaz, Yulaz) = 7;

% Odredivanje izlazne pozicije
Xizlaz = 8;
Yizlaz = 8;
A(Xizlaz, Yizlaz) = 8;

% Generiranje populacije
Populacija = randi(4, P, K);

%Petlja koja služi za grafički prikaz kretanja
[N,R]=size(A);
n = 1 ;
r = 1 ;
for i = 1:N
    for j = 1:R
        if A(i,j) ==1
            pp(n) = j ;
            dd(n)=-i ;
            n=n+1;
        else
            ppp(r) = j ;
            ddd(r)=-i ;
            r=r+1;
        end
    end
end

prekid = 1 ;
brojac = 0 ;

% Petlja koja služi za brojanje generacija
for g =1:G;
    brojac=brojac+1;
    Xkordinata=0;
    Ykordinata=0;

    Korak_Sudara = [];
    Udaljenost = [];
    Koraci = [];
    Sudari = [];
```

```
%Petlja za brojanje populacija
for i = 1:P
    X=Xulaz;
    Y=Yulaz;
    Sudar = 0;
    Korak = 0;
    Xprosli = Xulaz;
    Yprosli = Yulaz;
    u = 0;
    %Petlja za brojanje koraka koji su dozvoljeni robotu
    for j = 1:K
        Korak = Korak + 1;
        switch Populacija(i,j)
            case 1
                if A(X-1, Y) == 1
                    Sudar = Sudar + 1;
                else
                    Xprosli = X;
                    X=X-1;
                    Y=Y;
                end
            case 2
                if A(X+1, Y) == 1
                    Sudar = Sudar + 1;
                else
                    X=X+1;
                    Y=Y;
                end

            case 3
                if A(X, Y+1) == 1
                    Sudar = Sudar + 1;
                else
                    X=X;
                    Y=Y+1;
                end

            case 4
                if A(X, Y-1) == 1
                    Sudar = Sudar + 1;
                else
                    X=X;
                    Y=Y-1;
                end
        end


        if Sudar ==1 && u ==0
```

```
                Korak_sudara(i) = j ;
                u = 1 ;
            elseif Sudar ==0
                Korak_sudara(i)=Korak ;
            end

            if X==Xizlaz && Y==Yizlaz
                break
            end

        end

        Xkordinata(i)=Y;
        Ykordinata(i)=X;
        UdaljenostX =abs(Xizlaz-X) ;
        UdaljenostY=abs(Yizlaz-Y) ;
        Udaljenost(i)=UdaljenostX + UdaljenostY ;
        Udaljenost_ulazX= abs(Xulaz-X) ;
        Udaljenost_ulazY =abs(Yulaz-Y) ;
        Udaljenost_start(i)=Udaljenost_ulazX +
Udaljenost_ulazY ;
        Sudari(i)=Sudar;
        Koraci(i)=Korak ;

    end

    % FITNESS
    w1 = 18; %Težina za sudare
    w2 = 0.8; %Težina za broj koraka

    F = 0 ;
    for i = 1:P

F(i)=((Korak_sudara(i)/w1)+w2/Koraci(i)+(Udaljenost_start(i)*1
/(Udaljenost(i)+0.1)));
    end

    Fitness(g)=max(F);
    E = 0;
    for i = 1:P
        robot_X = 0 ;
        robot_Y = 0 ;
        if F(i)==max(F)
            E=i;
            robot_X=Xkordinata(E);
            robot_Y=-Ykordinata(E);
            break
        end
    end
```

```
    sumaF=sum(F);

    %RELATIVNI FITNES
    Relativni_fitness=0;
    for i = 1:P;

Relativni_fitness(i)=((Udaljenost_start(i)/(Udaljenost(i)+0.1)
)+(Korak_sudara(i)/w1)+w2/Koraci(i))/sumaF;
    end

    % ELITIZAM
    for i=1:P
        if Relativni_fitness(i)==max(Relativni_fitness);
            r=randi(P,1,1);
            Elita=Populacija(i,:);
            Kor(g)=Koraci(i);
            UdaljenStart(g)=Udaljenost_start(i);
            Udaljen(g)=Udaljenost(i);
            ssudari(g)=Sudari(i);
            Ssud(g)=Korak_sudara(i) ;%ZA CRTANJE GRAFA
            break
        end
    end

     count = count+1;

     if count==100;
         trenutniFitnes=Fitness(g);
         count=0;
         kretnjaX=0;
         kretnjaY=0;
         X= Xulaz ;
         Y=Yulaz ;
         kretnjaX(1)=-X;
         kretnjaY(1)=Y;

     for j=1:K;
         switch Elita(1,j)
             case 1
                 if A(X-1,Y)==1;
                     kretnjaX(j+1)=-X;
                     kretnjaY(j+1)=Y;
                 else
                     Xstari=X;
                     X=X-1;
                     Y=Y;
                     kretnjaX(j+1)=-X;
                     kretnjaY(j+1)=Y;
```

```
                    end

                    case 2
                    if A(X+1,Y)==1;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    else
                        X=X+1;
                        Y=Y;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    end

                    case 3
                    if A(X,Y+1)==1;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    else
                        X=X;
                        Y=Y+1;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    end

                    case 4
                    if A(X,Y-1)==1;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    else
                        X=X;
                        Y=Y-1;
                        kretnjaX(j+1)=-X;
                        kretnjaY(j+1)=Y;
                    end

            end
            if X==Xizlaz && Y==Yizlaz
                break
            end
        end

        figure(1)
        plot(pp,dd,'k*', ppp, ddd,'wo', robot_X, robot_Y,
'go',Yulaz,-Xulaz,'ro')

        figure(2)
        plot(pp,dd,'k*', ppp, ddd,'wo', robot_X, robot_Y,
'go',Yulaz,-Xulaz,'ro',kretnjaY,kretnjaX,'g--')
        pause(.05);
```

```
    iteracija = 1;
    end

    %Ruletno pravilo
    for i=1:P
        N=0;
        T=rand(1,1);
        for j=1:P;
            N=N+Relativni_fitness(j);
            if T<=N;
                Popul(i,:)=Populacija(j,:);  %Novi roditelji
koji idu na krizanje
                break
            end
        end
    end

    %Križanje
    for i=2:2:P;
        S=rand(1,1);
        Par=Popul(i-1:i,:);
        if S<= pc
            O=randi(2,1,K);
            Par1=Par(1,:);
            Par2=Par(2,:);

            for j=1:K
                if O(1,j)==2
                    Par1novi(1,j)=Par2(1,j);
                    Par2novi(1,j)=Par1(1,j);
                else
                    Par1novi(1,j)=Par1(1,j);
                    Par2novi(1,j)=Par2(1,j);
                end
            end
            ParNovi=[Par1novi;Par2novi];
        else
            Par1novi=Par(1,:);
            Par2novi=Par(2,:);
            ParNovi=[Par1novi;Par2novi];
        end

        Populacija_krizanje(i-1:i,:)=ParNovi;
    end

    %Mutacija

    Populacija_Mutacija=Populacija_krizanje;
```

```
      for i=1:P
          for j =1:K
              m=rand(1,1);
              k=randi(4,1,1);
              if m<pm;
                  Populacija_Mutacija(i,j)=k;
              else

Populacija_Mutacija(i,j)=Populacija_Mutacija(i,j);
              end
          end
      end
      Populacija=Populacija_Mutacija;
      Populacija(r,:)=Elita;



end

broj_gen=1:1:brojac;
figure(3)

% Prikaz najboljeg fitnesa po generacijama
subplot(2,2,1)
plot(broj_gen,Fitness)
xlabel('Iteracije')
ylabel('Fitness')
grid on



% Prikaz Udaljenosti po generacijama
subplot(2,2,2)
plot(broj_gen,Udaljen)
axis([0 brojac -1 10])
xlabel('Iteracije')
ylabel('Udaljenost')
grid on

% Prikaz broja sudara po generacijama
subplot(2,2,3)
plot(broj_gen,ssudari)
axis([0 G 0 (K+2)])
xlabel('Iteracije')
ylabel('Sudari')
grid on

% Prikaz udaljenosti od starta po generacijama
subplot(2,2,4)
plot(broj_gen,UdaljenStart)
```

```
axis([0 G 0 (K+2)])
xlabel('Iteracije')
ylabel('Udaljenost od starta')
grid on
```