Elektrotehnički fakultet
Univerziteta u Sarajevu

Seminarski rad:

Primjena genetičkog algoritma za pronalazak izlaza iz lavirinta

Optimizacija Resursa

Sarajevo, 09.01.2017

Studenti: Kenan Mahmutović
Mehanović Irfan
Lučkin Anes

Rezime

Genetički algoritam je popularan metaheuristički algoritam koji ima veliku primjenu u rješavanju problema. Ovaj algoritam je nastao upravo na osnovu prirodnog ponašanja živih bića, njihove genetike prilagođavanja uvjetima u prirodi i bazira se na modeliranju procesa evolucije živih bića. Iako je ideja o genetičkom algoritmu nastala mnogo ranije, John H. Holland je zajedno sa kolegama i studentima na Univerzitetu Michigen razvio isti 1975. godine i pokazao se kao vrlo moćan i u isto vrijeme općenit alat za rješavanje čitavog niza problema inžinjerske prakse. Od nastanka algoritma pa do danas kreiran je veliki broj različitih varijanti genetičkog algoritma pri čemu se sve temelje na osnovnom izvornom algoritmu. Sama ideja ovog rada jeste da se koristeći genetičkim algoritom iskoriste njegove osobine stohastičkog algoritma organiziranog pretraživanja kako bi se pronašao izlaz iz lavirinta. Princip je da se ostave najbolja rješenja (najbolje jedinke) i na osnovu dobivenih informacija koje su pohranjene u populaciji da se tvore nova i bolja potencijalna rješenja kako bi se pronašao najbolji (najkraći i najbrži) put izlaska iz lavirinta. U radu je opisan genetički algoritam, prikazana je osnovna ideja genetičkog algoritma kao i njegova implementacija koja je izvršena u *python* programskom jeziku.

Abstract

Genetic algorithm is a popular metaheuristic algorithm that has great application in solving problems. This algorithm was created on the natural behavior of living organisms, their genetic adaptation conditions in nature and is based on the modeling of the process of evolution of living beings. Although the idea of a genetic algorithm originated much earlier, John H. Holland, together with colleagues and students at the University of Michigan developed the same in 1975. and proved to be a very powerful and at the same time a general tool for solving a series of issues of engineering practice. Since the creation of the algorithm until now, it has been created a large number of different variants of the genetic algorithm in which all are based on the basic original algorithm. The very idea of this paper is that by using genetic algorithm and with the help of its advantage, by the use of its features of stochastic algorithm organized search, to find a way out of the maze. The principle is to take the best solution (best individuals) and on the basis of the obtained information that are stored in the population to form a new and better potential solutions in order to find the best (shortest and fastest) way out of the maze. This paper describes a genetic algorithm, the basic idea of genetic algorithm and its implementation that was made in the *python* programming language.

Sadržaj

1. Uvod

Uvodni dio obuhvata opis ključnih genetičke terminologije čije razumijevanje olakšava razumijevanja rada samog genetičkog algoritma, te pojašnjenja osnovnog genetičkog algoritma, njegovih elemenata i mehaniyama korištenihnpri izradi projekta.

1.1. Genetika

Pojam *evolucija* se vrlo često upotrebljava kako bi se označila neka promjena. Evolucija u biološkom smislu ne podrazumijeva bilo kakvu promjenu nečega, već promjenu na genetskoj razini (u molekuli DNA, u genima), promjenu koja se može naslijediti na potomke. Evolucija je neprekidan proces prilagođavanja živih bića na svoju okolinu, tj. na uvjete u kojima žive. U prirodi vlada nemilosrdna borba za opstanak u kojoj pobjeđuju najbolji(najprilagodljiviji), a slabi nestaju. Da bi neka vrsta tijekom evolucije opstala, mora se prilagoditi uvjetima i okolini u kojoj živi, jer se i uvjeti i okolina mijenjaju. Priroda selekcijom jedinki regulira veličinu populacije. Dobra svojstva jedinki (otpornost na razne bolesti, sposobnost trčanja, itd.) pomažu preživjeti u neprestanoj borbi za opstanak. [1]

Mehanizam evolucije je na sistematičan način predstavio Charles Darwin: "*Može se metaforično reći da prirodno odabiranje svakodnevno i svakoga časa istražuje po cijelome svijetu i najmanje varijacije; ono odbacuje loše, a održava i sabire one koje su dobre; ono radi mirno i neprimjetno, kad god i gdje god se ukaže prilika, na usavršavanju svakog organskog bića u odnosu na njegove organske i neorganske uslove života...*". [1]

Genetika je grana biologije. Genetika je nauka o nasljeđu - prenošenju karakteristika od jedne generacije drugoj. Tijela koja su oruđa u ovom procesu zovu se hromosomi. Svaki hromosom sastavljen je od gena - "šifrovanih" instrukcija za izgled i sastavne dijelove organizma. Važno je napomenuti, da bi došlo do evolucije, mora doći do promjene u genima, a te promjene mogu biti od velike koristi (nekada i veoma štetne) za jedinke koje ih imaju. Živa bića evoluiraju kroz procese mutacije (slučajnih promjena u genima odnosno DNK koja je nositelj nasljednih informacija), geentskog otklona (eng. *Genetic drift*) (slučajnih promjena u frekvenciji pojedinih genetskih varijacija unutar populacije tokom vremena) i prirodne selekcije (ne-slučajnog i postepenog procesa kojim pojedine genetske varijacije i mutacije postaju manje ili više zastupljene u nekoj populaciji).[1]

1.2. Genetički algoritam

Genetički algoritmi (eng. *Genetic Algorithms - GA*) predstavljaju efikasan alat za rješavanje problema optimizacije. Pripadaju porodici heurističkih metoda optimizacije koji su inspirisani procesom prirodne selekcije gdje samo najbolje vrste opstaju, te na takav način rješavaju određene računarske probleme. Genetički algoritam je stohastički algoritam organiziranog pretraživanja problemskog prostora, a sama evolucija predstavlja robustan proces prilikom pretraživanja prostora rješenja.[2]

Kao što je već navedeno, genetički algoritam radi na principu opstanka najboljih jedini, a inspiracija za nastanak ovakvog algoritma upravo leži u tome što se živa bića prilikom procesa evolucije prilagođavaju uvjetima u životnoj okolini tj. prirodi. Genetički algoritam se koristi principe evolucije koje primjenljuje u svojim metodama optimizacije. Ti principi se ogledaju u procesu selekcije i genetskim operatorima baš kao i kod živih bića u prirodi. Procesom selekcije, kao što sama riječ kaže, vrši se ocjenjivanje jedinki po određenim kriterijima, pa se vrši odabir tj. selekcija najboljih jedinki koje će učestvovati u formiranju sljedeće generacije. Na osnovu selekcije jedinki prevashodno njihovom kvalitetu dobro geni se prenose u slijedeću generaciju, a loši geni se odbacuju tako da izgeneracije u generaciju se dobija bolja populacija. Moguće je i sortiranjem izvršiti odabir najboljih jedinki, međutim to dovodi do brze konvergencije algoritma koji se završava nakon par iteracija i ne može da napreduje. Zato se uvodi i mogućnost lošijih jedinki da pređu u slijedeću generaciju kako bi se održala raznolikost populacije i mogućnost algoritma da i dalje napreduje. Osim toga, nad genetičkim algoritmima se primjenljuju genetički operatori koji će osigurati što kvalitetniji rad istog.[2]

Iako se sama ideja o ovom tipu algoritma javila nekoliko desetljeća prije, genetički algoritmi predloženi su u ranim sedamdesetim godinama prošlog vijeka. Razvijeni su od strane Johna H. Hollanda i njegovih kolega studenata (DeJong i dr.) kao osnovni model genetičkog algoritma na kojem se baziraju i drugi modeli istog. Ovi algoritmi se koriste kao općeniti alat rješavanja čitavog niza problema iz inžinjerske prakse jer su se pokazali kao veoma moćni algoritmi rješavanja različitih klasa problema. Počeli su se intenzivnije koristiti tokom nešto više od dva desetljeća, a sve više i više u posljednjim godinama. Iz razloga što se baziraju na principima iz prirode, genetički algoritmi su jednostavni za upotrebu i imaju široku primjenu zbog prilično opšte prirode. Imaju veliki doprinos znanstvenicima i inžinjerima u njihovim primjenama na prilagođavanju velikog broja problema i povećanju efikasnosti. Nasuprot svemu, problemi genetičkih algoritama tj. većine njih su samo teorijske prirode koji se ogleda u nedostatku matematičkih dokaza da oni rade zaista ono što rade. Nažalost, rezultati postignuti na teorijskom području su dvojbeni, a genetski algortimi ostaju i do danas u osnovi heurističke metode.[2]

1.3. Osnovni genetički algoritam

Osnovni genetički algoritam (eng. *Simple Genetic Algorithm - SGA*) predstavlja jednostavni oblik genetičkog algoritma koji koristi binarnu reprezentaciju kako bi predstavio tj. kodirao hromozome. Takođe, ovaj algoritam koristi prostu selekciju, ukrštanje sa jednom tačkom prekida i prostu mutaciju. Zbog svoje jednostavnosti, ovaj algoritam je korišten kao osnova za razvijanje drugih varijanti genetičkih algoritama. Također je često i predmet studija i još uvjek se koristi kao mjerilo za novije genetičke algoritme. Iako je jednostavan za razumijevanje i za rad, algoritam ipak ima mnoge nedostatke. Osnovni genetički algoritam koristi binarne stringove kako bi predstavio potencijalna rješenja, a binarna reprezentacija je suviše restriktivna, pa se onda umjesto njega koristi genetički algoritam s realnim kodiranjem. Ograničenje postoji i u pogledu mutacije i ukrštanja koje koristi algoritam, a mogu da se primjenljuju samo za binarne i cjelobrojne reprezentacije. Što se tiče problema konvergencije, selekcija je osjetljiva na konvergirajuće populacije sa bliskim vrijednostima fitnesa i konvergencija ka rješenju je jako spora.[2]

1.4. Osnovne odlike genetičkog algoritma

Poznato nam je da genetički algoritmi simuliraju prirodni evolucijski proces pa zbog te povezanosti među evolucijskim procesima i genetičkih algoritama, slobodno se može, za te algoritme, ustanoviti da:

- postoji populacija jedinki;
- neke jedinke su bolje prilagođene okolini u odnosu na druge;
- bolje jedinke imaju veću verovatnoću preživljavanja i reprodukcije;
- osobine jedinki zapisane su pomoću genetskog koda;
- potomci nasljeđuju osobine roditelja;
- jedinke mogu mutirati.[3]

Individue (ili jedinke) kod genetičkih algoritama predstavljaju trenutne aproksimacije rješenja, i one se kodiraju kao hromozomi (u vidu stringova, binarno kodiranje, realno kodiranje). Još jedna od naziva za stringove jeste genotip, a svakom genotipu odgovara jedinstveni fenotip (drugi naziv za individuu). Hromozom se sastoji od više gena koji predstavlja varijablu konkretnog problema. Svaka jedinka se kodira i nakon toga joj se pridružuje određena osobina mjerila kvaliteta koji se naziva fitnes (eng. *fitness*) koji se dobiva pogodnom transformacijom rezultata evaluacije. Evaluacija predstavlja dodjelu vrijednosti fitnesu, a vrši se nakon inicijalizacije početne populacije nad kojom se radi. Početna populacija se najčešće generiše slučajnim izborom rješenja iz domena, mada se može i generisati koristeći određene pogodnosti

za poboljšanje performansi. Nakon toga, proces se ponavlja dok se ne zadovolji jedan od uslova zasutavljanja koji može biti ukupan broj jedinki. U ovom procesu se primjenjuju operatori genetičkog algoritma kao što su selekcija, ukrštanje i mutacija. Nad kodiranim jedinkama se izvršavaju sve operacije genetičkog algoritma osim procjene kvalitete koja se radi nad samim jedinkama. Prilikom ponavljanja procesa primjene operatora selekcije, omogućuje se da loše jedinke izumru dok bolje jedinke opstaju kako bi se u slijedećem koraku ukrštale. Ukrštanje se koristi kako bi se odobine roditelja uspješno prenijele na potomke. Mutacija je operator koji se koristi kako bi se nasumično izmijenile osobine gena. Koristi se kako bi se izvršila prevencija rane konvergencije algoritma. Jedan ovakav postupak omogućuje da iz generacije u generaciju raste prosječan kvalitet populacije.[3]

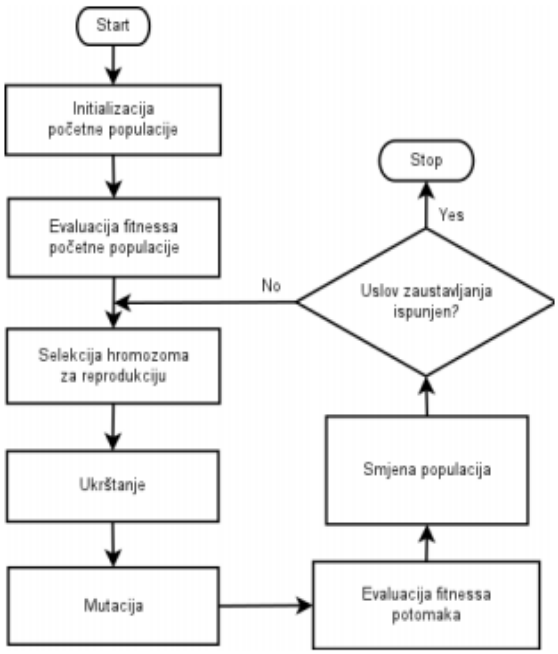
Genetički algoritmi se sastoje od sljedećih komponenti:

- **Inicijalna populacija**
- **Fitnes funkcija**
- **Selekcija**
- **Križanje**
- **Mutacija**

Genetički algoritam mora da ima:

- Domen potencijalnih rješenja
- Funkciju cilja kako bi se došlo do optimalnog rješenja

Osnovni koraci genetičkog algoritma prikazani su na sljedećoj slici 1.4.1:



Slika 1.4.1. Osnovni koraci genetičkog algoritma

Pseudo kod rada genetičkog algoritma je prikazan na slici 1.4.2.:

```
procedure GA
begin
    t = 0;
    inicijalizuj X(t);
    ocijeni X(t);
    radi do nekog uslova
    begin
        t = t + 1;
        odaberi X(t) iz X(t-1);
        reprodukuj parove u X(t);
        ocijeni X(t);
    end
end
```

Slika 1.4.2. Pseudo kod genetičkog algoritma

1.5. Osnovni operatori genetičkog algoritma

Spomenuli smo da svaka jedinka se kodira u formu hromozoma i operatori genetičkog algoritma se vrše nad hromozomima. Najvažniji operatori koji se susreću kod gotovo svakog genetičkog algoritma jesu ukrštanje i mutacija. Osim ova dva osnovna operatora, postoji niz drugih operatora od kojih ćemo u nastavku ukratko objasniti samo neke od njih.[4]

1.5.1. Selekcija

U prirodi preživljavaju samo one individue (jedinke) koje su najbolje u borbi za životne resurse a sama sposobnost prilagođavanja promjenljivoj okolini je ključna kvaliteta nužna za preživljavanje jedinki pojedine vrste.

Sam proces selekcije imitira prirodnu selekciju u kojoj individue koje su uspješne preživljavaju dok neuspješne (neprikladne) izumiru. Na taj način boljim jedinkama se daje veća šansa učetvovanja u procesu pravljenja nove generacije dok lošije jedinke u potpunosti nestaju tj. zanemaruju se. Kod osnocnog genetičkog algoritma koristi se proporcionalna selekcija. Kod

proporcionalne selekcije je vrijednost preživljavanja i-te individuoe proporcionalna f_i / f_{av} gdje je f_i prikladnost i-te individue, a f_{av} srednja prikladnost populaciji.[4]

Postoji više metoda selekcije koje na različite načine vrše odabir jedinki na osnovu njihovog fitnesa modelirajući različite prirodne procese. Neke od tih metoda selekcije su: selekcija skraćivanjem, selekcija proporcionalno fitnesu, selekcija na bazi ranga, turnirska selekcija i druge.

1.5.2. Ukrštanje

Nakon procesa selekcije obavlja se ukrštanje roditelja prilikom čega se prenose njihove osobine na potomke. Ukrštanje je slučajni proces odabira roditelja. Postoji više varijanti ukrštanja. Osnovni genetički algoritam koristi ukrštanje u jednoj tački (eng. *single-point crossover*). Takođe postoji i ukrštanje u više tačaka (eng. *multi-point crossover*), jer ukrštanje u jednoj tački nije uvijek najbolje rješenje iz razloga što uvijek razdvaja krajeve hromozoma. Daljim proširivanjem dobija se ujednačeno tj. uniformno ukrštanje (eng. *uniform crossover*).[4]

1.5.3. Mutacija

Mutacija je jedan od najvažnijih operatora genetičkih algoritama. Mutacijom se vrši izmjena slučajno odabranih gena neke jedinke. Omogućava se vraćanje izgubljenog, a korisnog genetskog materijala, prilikom selekcije i ukrštanja. Takođe, mutacija održava raznolikost jedinki i same populacije. Na taj način algoritam neće brzo konvergirati. Ako je mutacija velika onda algoritam vodi ka slučajnom pretraživanju. Mutacija daje mogućnost održavanja raznolikosti i ključan je operator genetskih algoritama jer uvodi nedostajuće vrijednosti i radi konvergencije algoritma.[4]

1.6. Primjena genetičkih algoritama

Na početku su se genetički algoritmi koristili većinom za rješavanje optimizacijskih problema, ali danas se oni koriste za rješavanje raznih klasa problema. Načelno, oni se koriste kod problema optimizacije - nalaženje optimalnih parametara nekog sistema. Razlog tome je to što se genetički algoritmi mogu lahko prilagoditi problemu. Ovi algoritmi našli su svoju primjenu u organizaciji, hemiji, biologiji, neurologiji, tehnici itd.

Genetički algoritam se primjenljuje i daje dobre rezultate u području učenja kod neuronskih mreža, pri traženju najkraćeg puta, problemu trgovačkog putnika, strategiji igara, problemima sličnim transportnom problemu, problemu raspoređovanja procesa, problemu određivanja parametara sistema, optimizaciji upita nad bazom podataka, itd.

Još neki od problema u čijem se rješavanju koriste genetički algoritmi su[4]:

- automatizirani dizajn
- izračunavanje optimalnog broja prijava mobilnih telefona baznim stanicama
- dijeljenje grafova
- problem raspoređivanja procesa
- razbijanje šifri
- dizajniranje sistema opskrbe vodom
- učenje kretanja likova u 3D simuliranom svijetu
- dizajn univerzalnog motora
- sinteza CMOS operacijskih pojačala

2. Korišteni algoritam

Postoji mnogo algoritama za rješavanje lavirinta. Takvi algoritmi se smatraju automatiziranim rješenjima za pronalaženje izlaza iz lavirinta. Veličina i kompleksnost lavirinta, te upotrijebljeni algoritam znatno utječu na performanse izvršenja programa. Rješenje problema pronalaska izlaza iz lavirinta koje nema beskonačne petlje se naziva perfektnim rješenjem. Mnoga rješenja za pronalazak izlaza iz lavirinta su iz teorije grafova.

Neki od algoritama za rješavanje problema pronalaska izlaza iz lavirinta su:

- *Random mouse*
- *Wall follower*
- *Pledge*
- *Trémaux's algorithm*
- *Dead-end filling*
- *Recursive algorithm*
- *Maze-routing algorithm*
- *Shortest path algorithm*

Dok su sva rješenja zasnovana na specificnim algoritmima, ne znači da su u svim slučajevima optimalna, te se teži pronaći optimalno rješenje za pronalaženje izlaza iz lavirinta.[5]

2.1. Opis rada korištenog algoritma

Genetički algoritmi su pronašli primjenu pri pronalaženju optimalnih rješenja problema i problema pretrage. Kod genetičkih algoritama populacija (fenotip) iz grupe potencijalnih rješenja (domena funkcije) kroz niz iteracija evoluira ka boljem rješenju. Svako potencijalno rješenje (*solution candidate*) ima niz atributa (*properties*) koji se nazivaju hromosomima ili genotipom potencijalnog rješenja. Genotip može mutirati i mijenjati se. [6]

Evolucija obično počinje od populacije koja nastaje od slučajno generisanih individua. Evolucija je iterativni proces.Populacija se u svakoj od iteracija naziva generacijom (generation). U svakoj od generacija, fitnes svake individue u populaciji se evaluira. Fitnes je vrijednost funkcije cilja optimizacionog problema. Individue se stohastički biraju iz trenutne generacije, mutiraju i mijenjaju kako bi se kreirala nova generacija. Tako kreirana nova generacija (potencijalno rješenje) se koristi u narednoj iteraciji evolucije. Algoritam završava ukoliko su sve generacije isprocesirane i dosegnut je maksimalni njihov broj ili ukoliko je pronađenja zadovoljavajuća vrijednost funkcije cilja. [6]

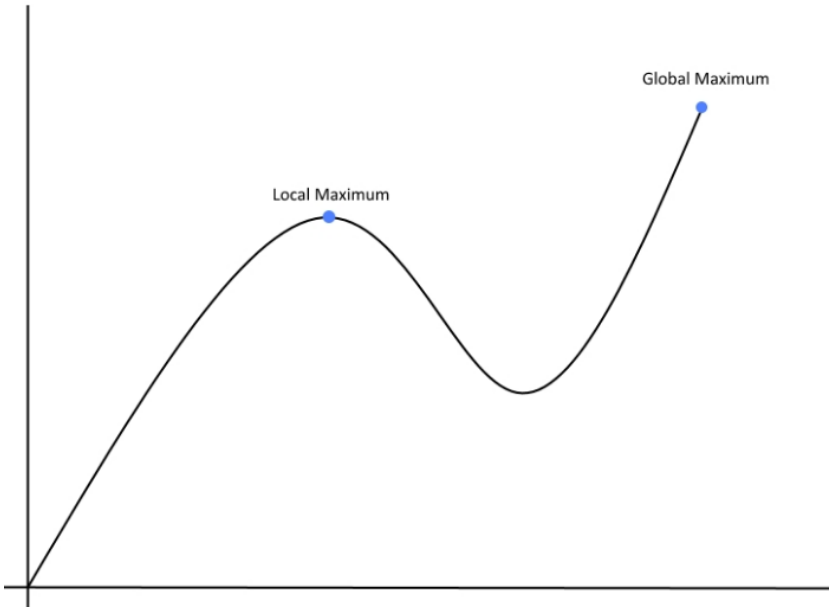
Standardna reprezentacija potencijalnog rješenja je niz bitova.

Kada je definirana funkcija cilja tj. fitnes funkcija i domen potencilajnih rjesenja ili populacija, genetički algoritam počinje sa inicijalizacijom i evolucijom trenutne početne populacije. Poboljšsanje populacije se postiže kroz niz mutacija, crossover, inverzija i selekcija.

S obzirom da su genetički algoritmi optimizacini algoritmi, imaju ograničenja koja vode ka kreiranju opitmalnog rješenja.

Genetički algoritmi su po prirodi više optimizacioni algoritmi. Optimizacija podrazumijeva da se može naći dovoljno dobro rješenje, ali to isto rješenje ne mora nužno biti i najbolje. Optimizacione algoritme je dobro koristiti u slučajevima kada postoji veči broj parametara koji mogu testirati valjanost rješenja.

Optimizacija se može prikazati funkcijom y= f(x), slika 2.1.1.

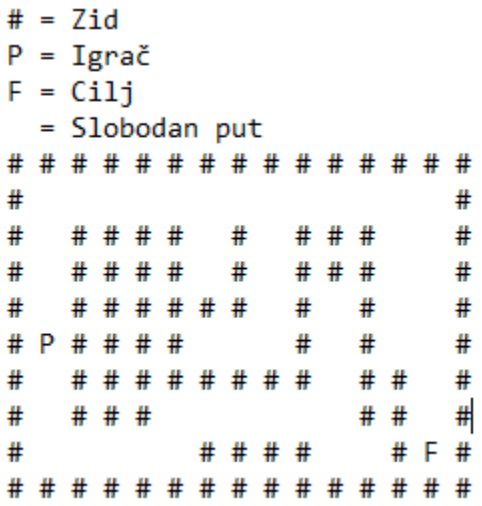


Slika 2.1.1. Optimizacija prikazana funkcijom [6]

Računar se iskorištava u svrhe iteriranja kroz veliki broj kombinacija vrijednosti f(x), pri čemu se pokušava pronaći rješenje koje je dovoljno dobro, tj. optimalno rješenje.

Postavlja se pitanje, kakvo je to rješenje dovoljno dobro? Dovoljno dobro rješenje se određuje na osnovu niza parametara i subjektivne je prirode, te ovisi o problemu koji se treba riješiti. Razlika izmešu genetičkog algoritma i brute force algoritma je u tome što genetički algoritmi se kroz evoluciju baziraju na poboljšanju rješenja.

Lavirint koji se rješava genetičkim algoritmom je prikazan na slici 2.1.2.:



Slika 2.1.2. Lavirint [6]

2.2. Svođenje opisanog problema u formu korištenog algoritma

Na početku izvršavanja algoritma se kreira populacija hromosoma, u ovom slučaju od 20000 članova(hromosoma), ukrštenje je šansa da se dva gena ukrste, to se radi u funkciji *evolve*. *Elitism* je mali procenat najboljih iz generacije sa najmanjim fitnessom koji se dugo neće mijenjati odnosno ostaju isti. *Mutation* je opet procenat, u funkciji *evolve* se koristi da li će se neki gen na hromosomu promijeniti, sa neke vrijednosti od 1 do 4 na neku drugu.

```
def __init__(self, size=20000, crossover=0.8, elitism=0.1, mutation=0.03):
        self.elitism = elitism
        self.mutation = mutation
        self.crossover = crossover
        self.size = size

        buf = []
        for i in range(size): buf.append(Chromosome.gen_random())
        self.population = list(sorted(buf, key=lambda x: x.fitness))
```

Na početku se kreira prva populacija u konstruktoru *init*. Kad se kreira populacija njoj se šalju gore navedene vrijednosti mutacija, elitizam i ostale, ukoliko se u konstruktoru ne pošalju gore navede vrijednosti, uzimaju se ove predefinisane. Generacija se popunjava hromosomima, koji su liste dužine u nasem slučaju 30 gena. Tih 30 gena se generiše nasumično gdje geni poprimaju vrijendnosti izmedju 1 i 4. Vrijednost 1 označava da se kreće gore, 2 da ide dole, 3 da ide lijevo, a 4 da ide desno.

Kada se svaki novi hromosom generiše poziva se funkcija *gen_random* iz klase *Chromosome*.

```
def gen_random():
        """
       A convenience method for generating a random chromosome with a random
       gene.
       """
        gene = []
        for x in range(30):
            gene.append(randint(1, 4))

        return Chromosome(gene)
```

Tada se populacija popuni do one vrijednosti koju smo zadali parametrom size u konstruktoru klase *Population*. Kada se generiše hromosom u njemu se generiše i fitness, tj.vrijednost koja predstavlja koliko je dobar taj hromosom. Kada se pozove konstruktor fitness se dobije tako što se pozove funkcija *update_fitness* kojoj šaljemo taj gen i lavirint kao parametre.

```
def _update_fitness(gene,maze):
        """
       Helper method used to return the fitness for the chromosome based
       on its gene.
       """

        fitness = 0
        for i in range(0,30):
            if(gene[i] == 1):
                maze.moveUp()
            elif(gene[i] == 2):
                maze.moveDown()
            elif(gene[i] == 3):
                maze.moveLeft()
            elif(gene[i] == 4):
                maze.moveRight()
            else:
                continue

        fitness = abs(maze.finishPosition[0] -
 maze.playerCurrentPosition[0]) + abs(maze.finishPosition[1] -
 maze.playerCurrentPosition[1]) + maze.penalties;
        maze.resetPlayer()

        return fitness
```

U zavisnosti kakvi su generisani geni u hromosomu tako se i krećemo po lavirintu, gore, dole, lijevo, desno. Na kraju računamo vrijednost fitness tako sto računamo razdaljinu izmedju finish i nase tačke, i još dodavajući offset ukoliko je hromosom „udarao“ u zidove lavirinta. Na kraju tu vrijednost vratimo.

Ispisom u konzoli prikazujemo najbolji gen iz te generacije, njegov fitness, i najgori fitness iz te generacije i prikazujemo gdje je završio nas kromozom oznakom "P" u lavirintu.
Nakon toga se gleda ukoliko je fitness najboljeg jednak nuli, to znači da je došao do kraja i prekida se pretraga. Ukoliko nije, nad tom populacijom se izvršava funkcija evolve.

```
def evolve(self):
        """
        Method to evolve the population of chromosomes.
        """
        size = len(self.population)
        idx = int(round(size * self.elitism))
        buf = self.population[:idx]

        while (idx < size):
            if (random() <= self.crossover):
                (p1, p2) = self._selectParents()
                children = p1.mate(p2)
                for c in children:
                    if random() <= self.mutation:
                        buf.append(c.mutate())
                    else:
                        buf.append(c)
                idx += 2
            else:
                if random() <= self.mutation:
                    buf.append(self.population[idx].mutate())
                else:
                    buf.append(self.population[idx])
                idx += 1

        self.population = list(sorted(buf[:size],key=lambda x: x.fitness))
```

U funkciji *evolve* prvo se najbolji hromosomi od ranije sačuvaju, pa se onda nad ostatkom najprije vrši selekcija dva partnera ukoliko je neka random vrijednost gora od našeg parametra crossover, na kraju ta dva partnera generišu dvoje djece. Nakon što se generišu djeca moguće je da se nad njima desi mutacija nekog gena na neku vrijednost od 1 do 4, ukoliko ne, djeca se samo dodaju u populaciju. Ukoliko se ne ukrštaju ta dva roditelja, onda taj kromozom ili mutira ili se samo dodaje u populaciju, u zavisnosti od parametra mutation.
I na kraju kad se sve to odradi, sortiraju se po fitnessu i nastaje nova populacija. Ovo se ponavlja toliko puta, koliko smo zadali da može biti generacija ili ukoliko želimo možemo staviti da se izvršava dok ne dobijemo optimalno ili dovoljno dobro rješenje, te onda uslovno završavamo pretragu po lavirintu.
Kada se biraju dva roditelja za ukrštanje, oni se biraju tako što se za svakog roditelja posebno izabere prva vrijednost kao najbolja, i onda se biraju 3 ili više vrijednosti u zavisnosti od parametra tournament, te se ta najbolja vrijednost poredi sa drugima. Ukoliko je neki drugi bolji on postaje jedan od mogućih roditelja. Ukoliko porazi sve ostale kao roditelj, te se onda i za

drugog roditelja radi isto. Ovo koristi Tournament algoritam gdje se hromosomi bore za mjesto roditelja.
Ovaj algoritam daje dobra rješenja, ali mu je moguće prilagoditi još fitness funkciju, da bi možda za nas primjer lavirinta dalo optimalno rješenje. Ali u većini slučajeva daje dovoljno dobro rješenje.
Ukoliko bi ove neke parametre prilagodili našem lavirintu, onda ovaj algoritam ne bi bio koristan za druge lavirinte, tako da i to nije pogodno. Ukoliko bi povećali mutaciju, to bi onda bilo potpuno nasumično pretraživanje sto ne predstavlja genetički algoritam.
Naš lavirint je veoma težak lavirint, jer kraća strana koja je bliža Finishu je zatvorena, tako da postoji nekoliko lokalnih ekstrema u koje može upasti. Moguće je da u nekom slučaju algoritam dadne optimalno rješenje, ali u većini slučajeva daje dovoljno dobro rješenje. Fitness bi se još mogao prilagoditi, ali bi tada naveli da algoritam radi za naš lavirint, dok za druge lavirinte ne bi radio dovoljno dobro.
Na nekoliko slika možemo predstaviti kako algoritam iz generacije u generaciju daje sve bolje i bolje rješenje, gdje u nekim slučajevima za posljednje generacije najbolji fitness i najgori imaju istu vrijednost, te je tu moguće uvesti neke promjene poput da se poveća mutacija, da bi se ta vrijednost postaknula da izađe iz lokalnog ekstrema.

3. Simulacija

3.1. Postavka simulacija

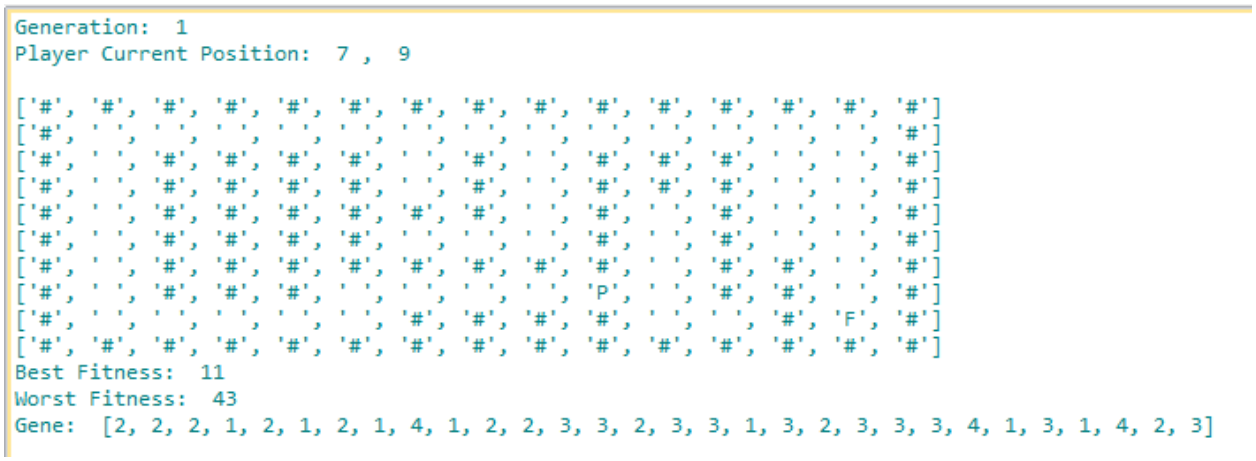
Parametri koji se moraju postaviti su veličina populacije, broj generacija, crossover, elitism, mutation. U ovom demonstracijskom slučaju iznose 200000, 200, 0.8, 0.001, 0.03 respektivno.

Također se postavlja i struktura lavirinta, koja je prikazana na slici 3.1.1.

```
Board =[
['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#'],
['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#'],
['#', ' ', '#', '#', '#', '#', ' ', '#', ' ', '#', '#', '#', ' ', ' ', '#'],
['#', ' ', '#', '#', '#', '#', ' ', '#', ' ', '#', '#', '#', ' ', ' ', '#'],
['#', ' ', '#', '#', '#', '#', '#', '#', ' ', '#', ' ', '#', ' ', ' ', '#'],
['#', 'P', '#', '#', '#', '#', ' ', ' ', ' ', '#', ' ', '#', ' ', ' ', '#'],
['#', ' ', '#', '#', '#', '#', '#', '#', '#', '#', ' ', '#', '#', ' ', '#'],
['#', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', ' ', '#'],
['#', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#', ' ', ' ', '#', 'F', '#'],
['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']]
```

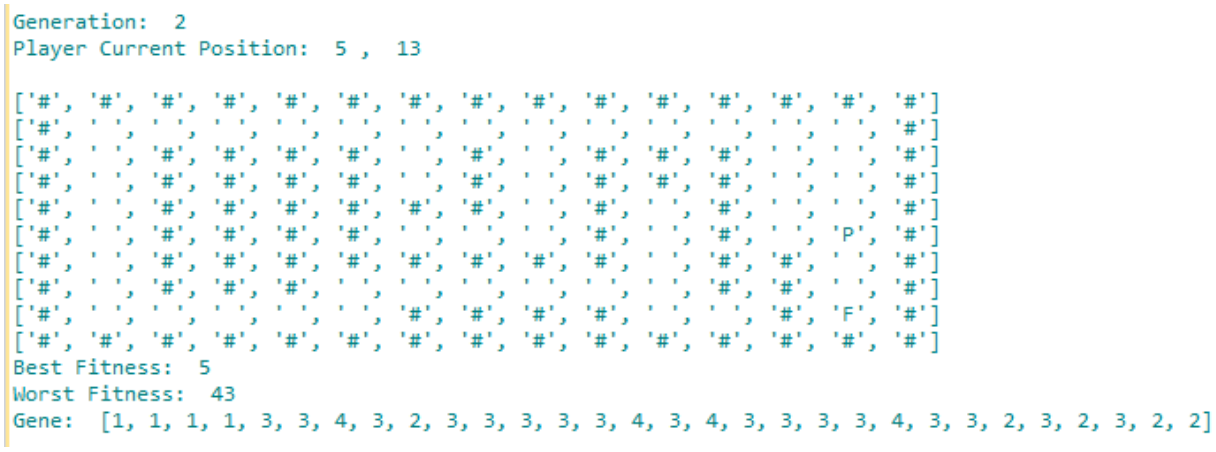
Slika 3.1.1. Izgled lavirinta

Pokretanje algoritma daje ispis na konzoli za svaku iteraciju. Na slici 3.1.2. je prikazan izgled jednog ispisa.



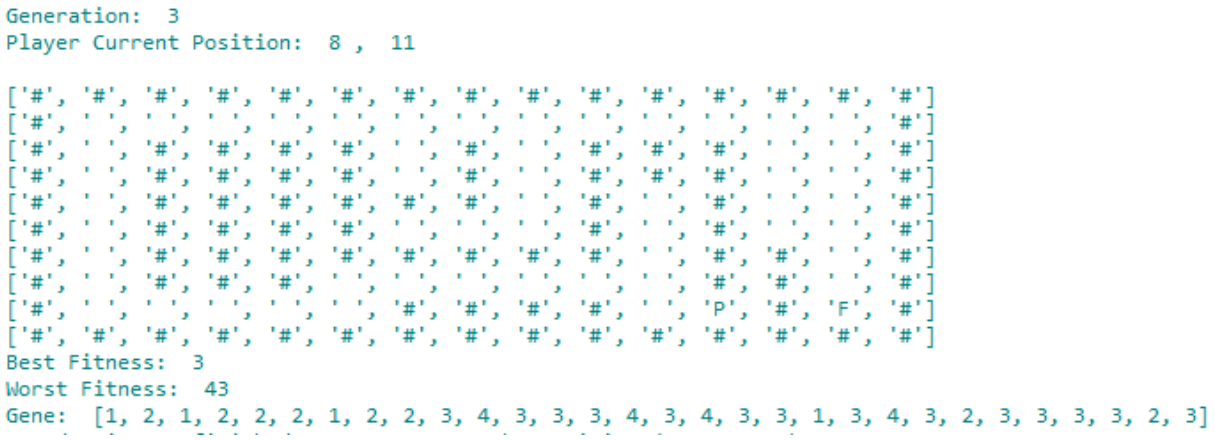
Slika 3.1.2. Izgled jednog ispisa u toku izvršavanja algoritma

Kao što na Slici 3.1.2. iznad možemo vidjeti prvu generaciju u izvršavanju algoritma, već možemo primijetiti da algoritam polagano zapada u lokalni ekstrem, jer se kreće donjom putanjom, a da bi došao do optimalnog rezultat morao bi se kretati gornjom putanjom. Također možemo vidjeti ispis najboljeg hromozoma, poziciju na kraju pretrage najboljeg hromozoma, njegov fitness i najgori fitness u generaciji. U nastavku ćemo vidjeti šta se dešava u narednoj generaciji.



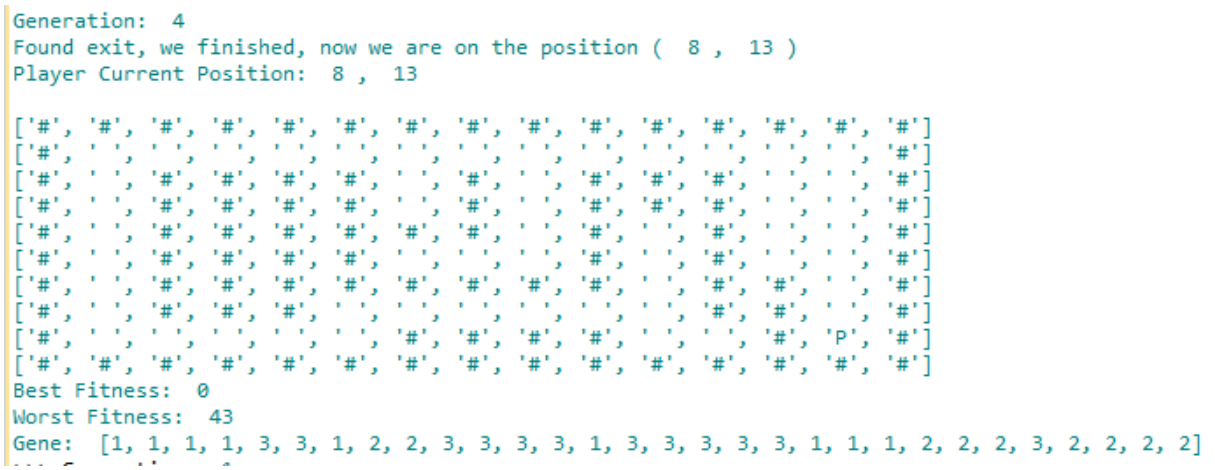
Slika 3.1.3. Druga generacija u izvršavanju algoritma

Algoritam se već u drugoj generaciji poboljšao, gdje vidimo da sada trenutno najbolji fitness iznosi 5, odnosno veoma smo blizu optimalnom rješenju, ali već u narednoj generaciji, nakon mutacija i ukrštanja algoritam nas je iznenadio, jer je opet zapao u lokalni ekstrem i dobio veoma dobro rješenje. To možemo vidjeti na narednoj slici.



Slika 3.1.4. Algoritam zapada u lokalni ekstrem u 3. generaciji

Iako je algoritam bio veoma blizu optimalnom rješenju, ipak nas je iznenadio, jer je u ovoj generaciji bolji fitness dobio na drugoj strani lavirinta, gdje se opet vratio u lokalni ekstrem, ali u svakom slučaju dijeli ih jedna pregrada što je veoma dobro rješenje.



Slika 3.1.5 Algoritam pronalazi hromozom koji nas je odveo do rješenja lavirinta

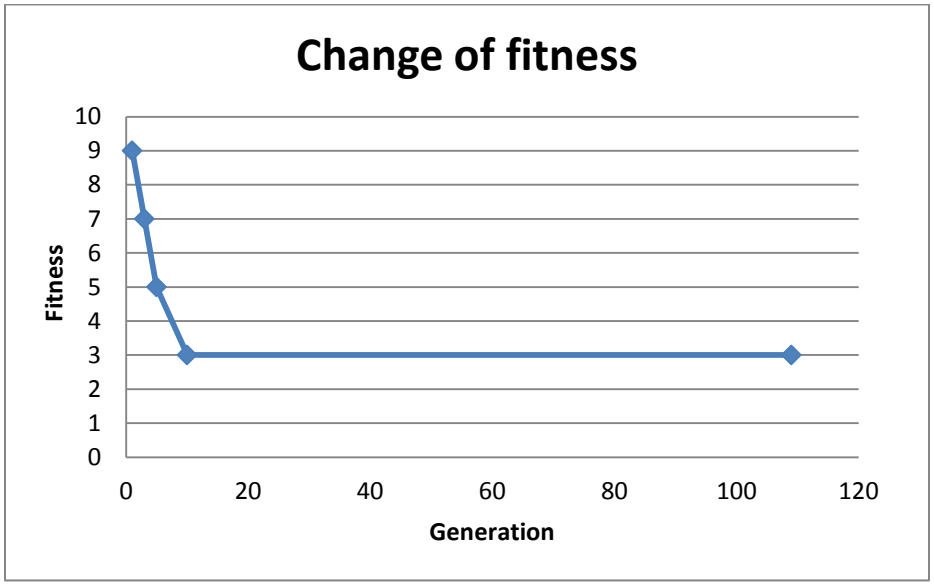
Ovaj potez nas je prosto iznenadio, jer smo mislili kako se kretao algoritam, da će opet zaglaviti u lokalnom ekstremu. Ali kao što vidimo veoma je moguće da se dobije i optimalno rješenje, ali u većini slučajeva bi zapadao u lokalni ekstrem, te bi mu se poboljšavao fitness, ali bi i dalje ostajao u lokalnom ekstremu, dok ne ispuni vrijednost da je fitness manji ili jednak 2 nakon čega bi se zaustavio. Na narednom slici možemo vidjeti kako i nakon 109 generacija, algoritam i dalje ostaje u lokalnom ekstremu, dok ne ispuni uslov za dovoljno dobro rješenje odnosno fitness manji ili jednak 2.

```
Generation:  109
Player Current Position:  8 ,  11

['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']
['#', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '#']
['#', ' ', '#', '#', '#', '#', ' ', '#', ' ', '#', '#', '#', ' ', ' ', '#']
['#', ' ', '#', '#', '#', '#', ' ', '#', ' ', '#', '#', '#', ' ', ' ', '#']
['#', ' ', '#', '#', '#', '#', '#', '#', ' ', '#', ' ', '#', ' ', ' ', '#']
['#', ' ', '#', '#', '#', '#', ' ', ' ', ' ', '#', ' ', '#', ' ', ' ', '#']
['#', ' ', '#', '#', '#', '#', '#', '#', '#', '#', ' ', '#', '#', ' ', '#']
['#', ' ', '#', '#', '#', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', ' ', '#']
['#', ' ', ' ', ' ', ' ', ' ', ' ', '#', '#', '#', '#', ' ', 'P', '#', 'F', '#']
['#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#', '#']
Best Fitness:  3
Worst Fitness:  43
Gene:  [2, 2, 2, 3, 4, 3, 3, 3, 3, 1, 2, 3, 1, 3, 3, 3, 4, 4, 3, 3, 3, 4, 3, 4, 3, 3, 4, 3, 2, 3]
```

Slika. 3.1.5. Algoritam je zapao u lokalni ekstrem, pokušavajući naći dovoljno dobro rješenje

U nastavku ćemo prikazati graf sa rezultatima kakve je rezultate algoritam davao, kroz nekoliko generacija. U ovom izvršenju korišteni su parametri od 200000 hromozoma populacije, postotak ukrštanja od 0.8, elitizam od 0.001, mutacija od 0.3 i dužina jednog hromozom jednaka 30.



Graf 1. Rezultati fitnessa prolaskom kroz generacija

Dobijeni rezultati zavise od generisanja hromozoma, ukoliko dobijemo dobro generisane hromozome veoma je moguće, da u prvih nekoliko generacija već dobijemo dobre rezultate, u nekim slučajevima dobijemo početne katastrofalne rezultate koji se vremenom popravljaju u zavisnosti da li se kreću prema lokalnim ekstremima ili prema izlazu iz lavirinta. Rezultati mogu varirati i u zavisnosti od različitih lavirinata koji se mogu zadati, ali korišteni lavirint je dovoljno dobar, jer posjeduje i dosta lokalnih ekstrema i globalni ekstrem, te kao takav ima dobre prepreke.

3.2. Rezultati simulacije

Kao što možemo vidjeti sa slike, već u prvih nekoliko generacija možemo vidjeti promjene *fitnessa*, i to ne samo najboljih hromosoma, nego i svih ostalih. Parametri poput *elitizma, mutacije*, veličine populacija i ostali mogli bi se dalje optimizirati za pretežno naš slučaj, ali to ne bi pogodovalo za neki drugi lavirint. Kako se populacija mijenja iterativno, dešava se da neka od potencijalnih rješenja budu odbačena, jer imaju veliki broj penala odnosno slučaj kada dolazi do zida, te tako možda neka od rješenja koja bi se kretala prema potencijalnom optimalnom rješenju prevazilažeći lokalne ekstreme, budu odbačena zbog povećavanja *fitnessa*. Zbog ovoga se dešava da naš algoritam zapada u lokalne ekstreme, te dobijamo "dovoljno" dobra rješenja u većini situacija. Jedna mogućnost koju smo kasnije uveli poboljšavajući fitness je i ta da ukoliko se u nekoj generaciji generiše hromozom koji u svojoj dužini posjeti optimalno rješenje, daje mu se mogućnost da upadne u „elitu“, tako što mu se penali zbog „udaranja u zid“ resetuju, te kao takav nastavlja daljnje kretanje kao da nije nikako udarao u zidove prethodno. Moguće je u fitness funkciju uvrstiti i to da se za ovakve slučajeve ukoliko neki hromozom posjeti optimalno rješenje, da se takvo rješenje uzme odmah kao rezultat, a moguće je podesiti i da se uzme kao dobar rezultat, te da se u narednim generacijama na osnovu tog hromozoma, generišu novi hromozomi, tako što će se taj hromozom uvrstiti u elitu. U drugom slučaju ukoliko se uvrsti u elitu, moguće je da se taj hromozom nastavi ukrštati dok se ne dobije rješenje u kojem nećemo imati nijedno udaranje u zid, tj. najbolje moguće rješenje, ali opet to zavisi od situacije do situacije. U našem slučaju to nije bilo potrebno, ali recimo ako bi smo ovaj algoritam koristili za realnu situaciju npr. kretanje robota kroz lavirintu bez da skenira je li zid ili nije u narednom koraku, ukoliko ne bi imali ovakvo rješenje robot bi se slupao sav udarajući u zidove. Velika je mogućnost, da bi naš algoritam za druge slučajeve lavirinta davao dovoljno dobra rješenja, isto kao i za ovaj lavirint, tako da nismo baš u mogućnosti da podesimo idealne parametre za sve vrste lavirinata. Najbolji zaključak bi bio da bi podešavanjem funkcije *fitnessa,* mogli dovesti do drastičnog poboljšanja algoritma, ali i naša vrsta funckije *fitnessa* je uveliko korištena i od strane drugih istraživača za ovakve vrste algoritma. Kao *fitness* funkciju koristili smo *Manhattan* distancu, koja predstavlja razmak između dva polja u mreži zajedno sa brojem penala nekog hromosoma.

4. Zaključak i diskusija

U ovom radu smo nastojali da prikažemo genetički algoritam za pronalaženje izlaza iz lavirinta rješavanjem
problema optimizacije. Primjena ovih algoritama je široka jer zbog svojih karakteristika mogu se primjenjivati u problemima gdje postoji potreba za robusnim, višedimenzionim pretragama prostora rješenja. Genetički algoritmi su područje aktivnog istraživanja s puno otvorenih problema te je postalo zasebno područje. Potrebno je još puno teoretskih i praktičnih problema riješiti prije nego što genetički algoritam može postati alternativa konvencionalnim opmtimizacijskim tehnikama.

Detaljno smo opisali primjenu genetičkog algoritma za pronalazak izlaza iz lavirinta. Rad sadrži teorijski opis genetičkog algoritma, ali i implementaciju samog problema kao praktični dio prikaza primjene algoritma. Genetički algoritam je testiran na javno dostupnim instancama.

Pronalaženje izlaza iz lavirinta je problem iz NP-domena problema, pa za njihovo rješavanje je potrebno koristii neku od heurističkih metoda, gdje se genetički algoritam pokazao kao snažno oružje u rješavanju istog.

Na našem primjeru može se vidjeti kako primjena genetičkih algoritama, na ovakvu vrtu problema, je vrlo korisna jer klasične metode optimizacije ne mogu rješiti ili njihovo izvođenje bi trajalo jako dugo. Praktični rad je pokazao da odabir parametara genetičkog algoritma znatno utiče na efikasnost i performancu rješenja. Samim tim, pravilan odabir parametara može bitno da utiče na kvalitet rješenja, na brzinu pronalaska izlaza i samim tim najkraćeg i najbržeg puta za izlazak iz lavirinta. Zbog toga je bitno odabrati optimalnu kombinaciju parametara. Jedan od nedostataka genetičkih algoritama je u tome što ne postoji sistematski postupak za određivanje dobrih parametara pa se traženje ispravne kombinacije svodi na eksperimentisanje, pa smo i mi u svom radu vršili eksperimentisanje sa varijablama i parametrima kako bismo dobili što bolje rješenje, blizu optimuma.

U praktičnom radu se može vidjeti kako se vrijeme izvođenja i broj generacija povećava s povećanjem broja jedinki u populaciji, a duljina puta se smanjuje. Zbog toga treba pronaći kompromis između kvaliteta rješenja i brzine izvođenja.

5. Sažetak

Tema ovog rada jeste primjena genetičkog algoritma kao algoritma optimizacije za pronalazak optimalnog puta izlaska iz lavirinta. Rad opisuje kakvi su to genetički algoritmi i kako oni tačno rade. Kod takvih algoritama problemi se rješavaju evolucijskim procesom koji rezultira najboljim rješenjem. Naglasak je stavljen na načine prikaza ili predstavljanja rješenja, genetske operatore (selekcija, križanje i mutacija) te parametre (vjerojatnosti ukrštanja i mutacije i veličina populacije). U drugom dijelu rada opisana je primjena genetičkog algoritma za pronalazak izlaza iz lavirinta. Detaljnije se opisuju problemi traženja izlaza iz lavirinta i same pretrage najboljeg i najkraćeg mogućeg rješenja tj. puta kroz lavirint. Ovaj problem pripada skupini NP-problema za čije se rješavanje genetički algoritmi često koriste.

Literatura i reference

[1] Corinne Stockley, Rječnik biologije, Izdaje: SP "Svjetlost", izdavačko preduzeće, Sarajevo (ISBN 86-01-02805-5)
[2] Marin Golub, Genetski algoritam, Prvi dio, (Verzija 1.0. izašla 6. listopada 1997. godine)
[3] Edin Salković: Genetički algoritmi i njihova praktična primjena, Diplomski rad, Elektrotehnički fakultet, Univerzitet Crne Gore, Podgorica, 2005.
[4] Biljana Borak: Genetski algoritmi za rešavanje lokacijskog problema snabdevača ograničenog kapaciteta u više nivoa, Diplomski - master rad, Matematički fakultet, Univerzitet u Beogradu, Beograd, 2009.
[5] Marko Čupić, Prirodom inspirirani optimizacijski algoritmi, Skripta namijenjena uporabi na kolegiju *Umjetna inteligencija*, Fakultet elektrotrhnike i računarstva, Zagreb 2009-2010.
[6] http://www.tonytruong.net/solving-a-2d-maze-game-using-a-genetic-algorithm-and-a-search-part-2/