UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

IMPLEMENTACIJA NEAT GENETSKEGA ALGORITMA ZA NAVIGACIJO PO 2D POLIGONU

DOMEN VAKE

2019

ZAKLJUČNA NALOGA

VAKE

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Implementacija NEAT genetskega algoritma za navigacijo po 2D poligonu

(Implementation of NEAT genetic algorithm for navigation in 2D space)

Ime in priimek: Domen Vake
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Jernej Vičič
Somentor: asist. Aleksandar Tošić

Koper, avgust 2019

Ključna dokumentacijska informacija

Ime in PRIIMEK: Domen VAKE

Naslov zaključne naloge: Implementation of NEAT genetic algorithm for navigation in 2D space

Kraj Koper:

Leto: 2019

Število listov: 43 Število slik: 33 Število tabel: 1

Število referenc: 23

Mentor: doc. dr. Jernej Vičič

Somentor: asist. Aleksandar Tošić

Ključne besede: NEAT, genetski algoritem, strojno učenje, optimizacija, umetne nevronske mreže

Izvleček:
Zaključna projektna naloga predstavlja implementacijo genetskega algoritma NEAT za navigacijo po prostoru. Ker je področje umetne inteligence zelo obširno, je bralcu najprej predstavljeno področje umetne inteligence in strojnega učenja. Opisani so tipi stojnega učenja in modeli, ki se pogosto pojavljajo na področju. Nato je bralcu podrobneje opisano področje genetskih algoritmov, ki imitirajo evolucijo za reševanje problemov. Izbran problem, ki je implementiran in testiran v projektu, je navigacija agentov v dvodimenzionalnem okolju. Ker pa izbrana implementacija, genetski algoritem NEAT, optimizira umetne nevronske mreže, so opisani tudi njihovi koncepti. Nato je opisana implementacija algoritma ter optimizacijske metode uporabljene v algoritmu. Na koncu naloge so bralcu predstavljeni še rezultati testov, ki so bili zbrani.

Key words documentation

Name and SURNAME: Domen VAKE

Title of final project paper: Implementation of NEAT algorithm for navigation in 2D space

Place: Koper

Year: 2019

Number of pages: 43 Number of figures: 33 Number of tables: 1

Number of references: 23

Mentor: Assist. Prof. Jernej Vičič, PhD

Co-Mentor: Assist. Aleksandar Tošić

Keywords: NEAT, genetic algorithms, machine learning, optimization, artificial neural networks

Abstract: The final project presents the implementation of the NEAT genetic algorithm for the optimization of navigation through the two-dimensional space. Since the field of artificial intelligence is vast, the reader is first presented with the basic concepts of the machine learning and and the types of learning in the field. We touch upon different learning models and go into more details with the artificial neural networks. The the reader is presented with the concepts of genetic algorithms and it's optimization methods. We describe the neuroevolutionary concepts in theory and implementation of the algorithm. At the end the reader is presented with the tests that were performed by the implemented algorithm and the results that it provided.

Zahvala

Zahvalil bi se mentorju doc. dr. Jerneju Vičiču in somentorju mag. Aleksandru Tošiću za vso podporo, strokovno pomoč in usmeritve tako pri zaključnem delu, kot v času izobraževanja. Prav tako bi se zahvalil družini in prijateljem za vso podporo, ki so mi jo izkazali na izobraževalni poti.
Hvala!

Kazalo vsebine

Kazalo tabel

Kazalo slik

Seznam kratic

NEAT	Neuroevolution of augmenting topologies (Nevroevolucija spreminjajočih se topologij)
NORMAL	Ime testnega primera, pri katerem je omogočen celoten sistem.
NO EFS	Ime testnega primera, pri katerem je onemogočen podsistem za deljenje ocen.
NO SPECIATION	Ime testnega primera, pri katerem je onemogočen podsistem za razporeditev v vrste.
NO SPECIATION AND EFS	Ime testnega primera, pri katerem sta onemogočena sistema za deljenje ocen in razporeditev v vrste.

1 Uvod

Problem zaključne naloge je optimizacija nevronske mreže s pomočjo genetskega algoritma. Optimizacija temelji na strojnem učenju, pri katerem se subjekti naučijo premikati po začrtanem 2D poligonu. Pri implementaciji uporabim različico algoritma NEAT (Neuro Evolution of Augmenting Topologies), ki se od klasičnih genetskih algoritmov razlikuje po tem, da genom predstavlja enostavna nevronska mreža, ki odloča o delovanju subjekta. Tako se v zaključni nalogi poleg področja umetne inteligence in računalniškega strojnega učenja dotaknemo tudi področij genetike, predvsem evolucije nevronskih mrež.

Zaključna naloga je razdeljena na teoretični in praktični del. V prvem delu se bralec seznani z osnovnimi koncepti umetne inteligence in strojnega učenja. Nato se spoznamo z genetskimi algoritmi, katere je že v 1960 utemeljil John Henry Holland. Njegova teorija predstavlja temeljno delo za razširjen algoritem NEAT, ki drži visok potencial na področju vzpodbujevalnega učenja, saj pri optimizaciji nevronskih mrež poleg uteži upošteva in optimizira tudi topologijo nevronske mreže, prav tako pa omogoča zmanjševanje kompleksnosti in hkratno iskanje rešitev.

2 Strojno učenje

Strojno učenje je področje v računalništvu, ki se ukvarja z uporabo statističnih metod za doseganje raznolikih ciljev. Leta 1959 je Arthur Samuel prvi definiral strojno učenje, in sicer kot iskanje rešitev na problem brez eksplicitno določenih poti ali načinov [1]. Strojno učenje je podmnožica področja umetne inteligence, ki temelji na matematični logiki. S pomočjo takšne logike poskušamo na različne načine formalizirati splošno znanje. To formalizirano znanje pa lahko potem uporabimo tako, da probleme rešujemo s pomočjo logičnega sklepanja. Navsezadnje umetna inteligenca temelji na konceptu, da imamo ljudje sposobnost reševanja problemov, ki jo lahko formaliziramo s pomočjo konceptov s področij psihologije in nevrofiziologije [2].
Izraz umetna inteligenca je najpogosteje uporabljen v navezavi na računalnike in druge stroje, ki posnemajo človekove kognitivne sposobnosti. Kognitivne sposobnosti pa znotraj strojnega učenja ponavadi predstavljajo zmožnosti reševanja problemov, zaznavanje vzorcev in učenje, kar pa v splošnem predstavlja statistično optimizacijo danega problema [2].

2.1 Vrste strojnega učenja

Na področju strojnega učenja poznamo več vrst ali tehnik. Te se med sabo razlikujejo predvsem po pristopu do problema, po vrstah podatkov, ki jih potrebujejo in vračajo, in po značilni obliki problema, ki ga rešujemo.

2.1.1 Nadzorovano učenje

Algoritmi v nadzorovanem učenju sestavijo matematični model podatkov, kjer so znani vhodni in želeni izhodni podatki. Naloga algoritmov pri nadzorovanem učenju je optimizacija objektne funkcije, ki predstavlja preslikavo vhodnih podatkov v izhodne [3]. Tu se funkcija iterativno izboljšuje, dokler ne preslika novih vhodnih podatkov v pravilne izhodne podatke z dovolj visoko natančnostjo. Podatkom, na katerih se funkcija optimizira, rečemo učni podatki. Predstavljeni so z matriko vhodnih podatkov in z vektorjem želenih izhodnih podatkov [3].

2.1.2 Nenadzorovano učenje

Kot je opisano v knjigi o nenadzorovanem učenju [4], je naloga algoritma poiskati objektno funkcijo, ki preslika vhodne podatke v izhodne, vendar pri učenju želeni izhodni podatki niso na voljo. Algoritem v vhodnih podatkih išče podobnosti in vzorce, s pomočjo katerih poskuša napovedati izhodne podatke. Objektna funkcija pa se uravnava glede na prisotnost ali odsotnost vzorcev in podobnosti v novih vhodnih podatkih, ki so dani modelu [13]. Večinoma se nenadzorovano učenje uporablja za grupiranje skupin in podatkov, v statistiki pa na področju ocenjevanja gostote podatkov.

2.1.3 Spodbujevalno učenje

Po članku *Reinforcement Learning: A Survey* [5] je spodbujevalno učenje področje strojnega učenja, kjer imamo agenta, kateri je del določenega okolja. Okolje je predstavljeno z izbranim stanjem, dejanje agenta pa predstavlja prehod iz nekega stanja v drugo stanje. Naloga algoritma je poiskati najboljšo odločitev za agenta v njegovem okolju. Algoritem se uči glede na uspešnost agenta v novem stanju. Glede na končno nagrado ali kazen je v prihodnosti to dejanje bolj, ali pa manj verjetno. Zaradi splošnosti omenjenega algoritma, to področje raziskujejo raznovrstne stroke, kot so teorija iger, statistika, teorija informacij in genetski algoritmi.

2.1.4 Učenje značilnosti

Učenje značilnosti je skupek tehnik, ki sistemu omogočajo zaznavanje značilnosti in vzorcev v podatkih. Tako lahko na podlagi pridobljenega zaznavamo posamezne vzorce in jih klasificiramo. [6]. Takšno učenje značilnosti vse bolj zamenjuje običajni inženiring značilnosti. Ta je vsekakor potreben za pripravo podatkov, za učenje ter uporabo sistemov strojnega učenja. Pa vendar predvsem priprava podatkov zajema zahtevne matematične in je ponavadi časovno zahtevna. Od tu tudi izhaja motivacija za razvijanje pristopa [6].

2.1.5 Zaznavanje anomalij

Zaznavanje anomalij je statistično zaznavanje neobičajnih predmetov, dogodkov ali kakršnih koli drugih opazovanj, ki predstavljajo neobičajne vzorce v primerjavi z večino podatkov [7]. Po članku *Anomaly detection: A survey* [8] delimo zaznavanje anomalij na tri glavne veje. To iskanje je podobno nenadzorovanemu učenju, kjer z neoznačenimi podatki iščemo vzorce. Glavna razlika je v tem, da pri učenju proces poteka s predpostavko, da večina podatkov predstavlja urejene podatke. Pri anomalijah pa iščemo podatke, ki se v podatkovni zbirki najmanj ujemajo z zaznanimi vzorci.

Druga veja procesa pa, kot pri nadzorovanem učenju, zajema učenje klasifikatorja. Ta potrebuje podatke, ki so označeni z "običajno"ali "neobičajno". Delno nadzorovano učenje zajema konstrukcijo modela, ki poskuša sestaviti testne podatke in model, ki "običajne"podatke poskuša klasificirati kot običajne ali sestavljene s strani kreacijskega modela. Te modeli so ponavadi uporabljeni za zaznavanje bančnih goljufij, medicinske analize, napake v tekstih ipd.

2.1.6 Asociacijska pravila

Asociacijska pravila so področje strojnega učenja, ki na podlagi pravil poskuša odkriti povezave med podatki v velikih podatkovnih bazah [9]. Po George W. [10] strojno učenje na podlagi pravil zajema vse algoritme, ki se učijo ali spreminjajo pravila v namen shranjevanja, manipuliranja ali uporabe znanja. Te sistemi želijo najti in izraziti pravila, ki izražajo vzorce relacij podatkov podatkovne baze. Te pristopi so največkrat uporabljeni na področjih kot so marketing, odkrivanje vdorov, proizvodnja in bioinformatika.

2.2 Modeli

2.2.1 Genetski algoritmi

Genetski algoritmi kot model strojnega učenja jemljejo idejo iz evolucije. Hevristični pristop algoritma simulira naravno selekcijo s križanjem in mutacijo. Algoritmi niso uporabljeni zgolj za reševanje problemov, temveč tudi za izboljšavo učenja drugih algoritmov v strojnem učenju [15]. Ker v tej nalogi govorimo predvsem o genetskih algoritmih, bomo več o genetskem modelu spoznali v poglavju 3.

2.2.2 Umetne nevronske mreže

Umetne nevronske mreže so modeli, ki so sposobni strojnega učenja in zaznavanja vzorcev v danem naboru podatkov. Sestavljene so iz umetnih nevronov, ki ohlapno simulirajo nevrone. Ti nevroni sestavljajo živalske možgane [19]. Več o umetnih nevronskih mrežah bomo prav tako spoznali v poglavju 4.

2.2.3 Odločitvena drevesa

Odločitvena drevesa so napovedni modeli, ki z drevesno vejo predstavijo odločitveno pot, po kateri pridemo do rešitev v drevesnih listih. Ponavadi se uporabljajo za predstavitev klasifikacijskih oznak preko analiziranja vrednosti podatkov. Krmarjenje po

drevesu je sekvenca odločitev glede na neko vrednost, kjer vsaka odločitev vodi do novega vozlišča ali pa do lista. Vsaka vrednost je v drevesu lahko uporabljena tudi večkrat [11]. Ta drevesa so pogosto uporabljena v statistiki, podatkovnem rudarjenju in pri strojnem učenju.

2.2.4 Podporni vektorski stroji

Podporni vektorski stroji ali podporne vektorske mreže, so skupek metod nadzorovanega učenja, ki so pogosto uporabljene za klasifikacijo in regresijo. Podporni vektorski stroji potrebujejo set podatkov, ki pripadajo eni od dveh kategorij [12]. Algoritem tako zgradi model, ki napove, ali novi primeri padejo pod eno ali drugo kategorijo. Poleg izvajanja linearne klasifikacije lahko vektorski stroji učinkovito izvedejo tudi nelinearno klasifikacijo z implicitno preslikavo svojih vhodov v višje dimenzijske prostore [12].

3 Genetski algoritmi

Genetski algoritmi spadajo pod spodbujevalno učenje. Leta 1960 jih je skupaj s svojimi učenci in sodelavci izumil John Henry Holland. Vendar pa jih Holland ni utemeljiv z namenom reševanja, temveč zaradi opazovanja fenomena adaptacije organizmov okolju. Kljub temu so se genetski algoritmi kasneje izkazali za uporabno orodje v strojnem učenju. Ideja genetskih algoritmov prihaja iz narave, bolj natančno iz načela evolucije. Organizmi se v naravi skozi generacije prilagajajo svojemu okolju preko spreminjanja svojega genetskega zapisa. Ta koncept pa lahko predstavimo tudi z logičnim modelom in simuliramo agentovo delovanje in njegovo okolje. [14]

3.1 Struktura algoritma

Genetski algoritmi so sestavljeni iz treh ključnih faz evolucije. To so selekcija, križanje in mutacija. Te se v iterativnem procesu ponavljajo dokler ne dosežejo želenega ustavitvenega pogoja. Ena simulacija algoritma predstavlja sekvenco generacij, ki so sestavljeni iz enega omenjenega cikla selekcije, križanja in mutacije [14]. V vsaki generaciji se faze izvedejo na skupini agentov, ki jo imenujemo populacija. Agenti v danem okolju poskušajo doseči željen cilj in so glede na svojo uspešnost ocenjeni. Ko populacija doseže želeno oceno uspešnosti, velja za razvito in ustavitveni pogoj je izpolnjen [14].

3.1.1 Genom

Skupek genov agenta je imenovan genom ali kromosom. Navadno je genom predstavljen kot vektor genov/podatkov, ki vplivajo na njegovo odločitev za dejanje v trenutnem stanju in v trenutnem okolju [15]. Vsi agenti so med sabo enaki, razlikujejo pa se samo po genomu. Naloga genetskega algoritma je najti najbolj optimalen genom, ki bo pripeljal agenta do odločitev, ki mu bodo prinesli najvišjo oceno. To pa pomeni, da bo kar se da najbolje opravljal nalogo ob predpostavki, da sposobnostna funkcija oceni agente glede na uspešnost izvedene naloge [16].

3.1.2 Evalvacija

Vsako generacijo agenti v okolju izvajajo svoje odločitve do ustavitvenega pogoja. Po ustavitvenem pogoju je pomembno, da lahko ocenimo uspešnost agenta v okolju, za kar pa je v genetskih algoritmih implementirana sposobnostna funkcija. Ta funkcija oceni agenta, glede na njegovo obnašanje v okolju. Za namene lažje predstave podatkov to oceno ponavadi normaliziramo glede na najboljšega agenta v generaciji [14]. To pomeni, da se vse ocene agentov gibljejo med 0 in 1, kjer je 0 najslabša ocena, ocena 1 pa predstavlja najboljšega agenta (ali več agentov, v primeru enakih ocen). Pomembno je, da sposobnostna funkcija predstavlja problem čim boljše, saj se skozi generacije agenti razvijajo tako, da povečujejo oceno, ki jo ta generira.

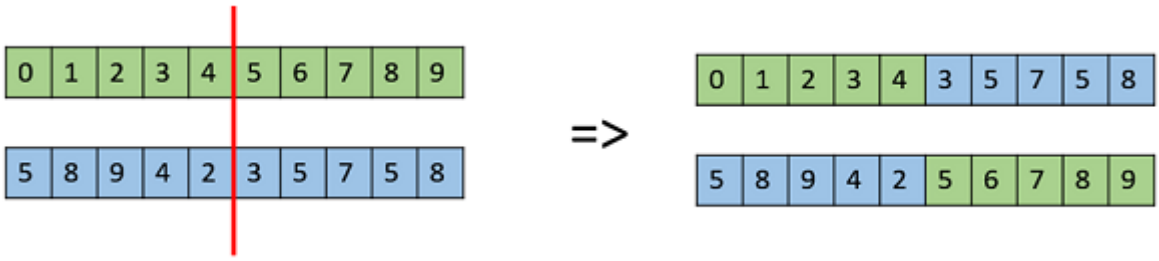
3.1.3 Selekcija

V fazi selekcije želimo izbrati agente, kateri bodo kasneje preko križanja predali svoje gene v naslednjo generacijo. Izbrati želimo agente, ki imajo dobre gene, kajti tako bodo vsako generacijo geni prinesli inkrementalno boljše rezultate [14]. Ker pa posamezen gen neposredno ne predstavlja, kako dobro se genom obnaša v okolju, ne moramo enostavno izbrati agentov, ki so bili najbolje ocenjeni v generaciji. Obstaja namreč možnost, da določen gen nosi velik potencial, vendar se genom ni dobro izrazil v okolju. Želimo imeti možnost prenosa teh genov v naslednjo generacijo, zato uporabimo uteženo naključno izbiro. Vsak agent ima možnost, da bo naključno izbran za prenos genov, vendar je ta sorazmerno izražena z njegovo oceno [15]. Tako zagotovimo, da se v splošnem prenašajo boljši genomi, kljub temu pa imajo agenti, ki še niso razviti, možnost prenosa genoma.

3.1.4 Križanje

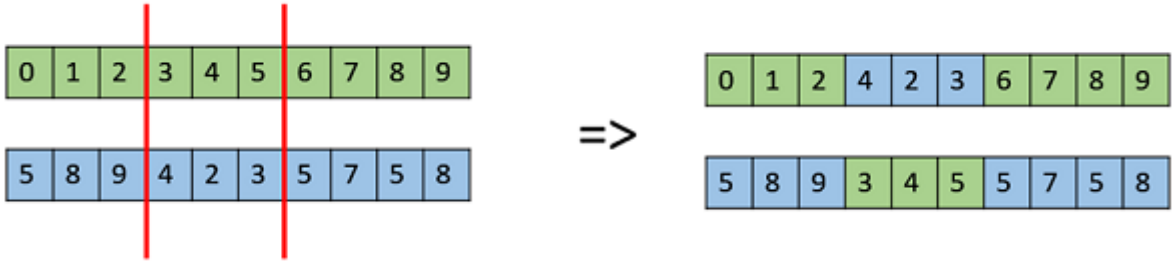
Pri križanju iz genomov dveh izbranih agentov, želimo narediti nov genom, ki bo predstavljen kot potomec teh agentov. Poznamo več načinov križanja. Najbolj pogosti načini so eno točkovno, več točkovno in enotno [16] križanje. Ti načini se razlikujejo po izbiri starša, ki bo prenesel gen na specifičnem delu genoma.

Pri algoritmu za eno točkovno križanje izberemo naključen gen znotraj genoma. Ta gen predstavlja točko, do katere so geni v novem genomu geni prvega starša in za njo geni drugega starša.



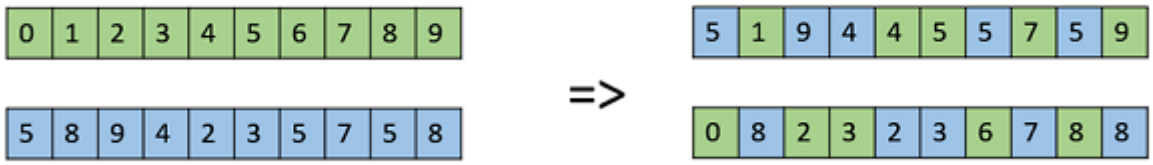
Slika 1: Eno točkovno križanje[1]; Genoma na sliki se križata na petem indeksu

Nasprotno kot pri eno točkovnem križanju, pri več točkovnem križanju izberemo več genov, pri katerih se starš, ki preda gen, zamenja. Tu začnemo pri poljubnem staršu in po vrsti prenašamo njegove gene. Ob doseženem izbranem genu, zamenjamo starša in do naslednjega izbranega gena prenašamo njegove gene [16].



Slika 2: Več točkovno križanje[3]; Slika predstavlja križanje z dvema točkama. Prvič se genoma križata na tretjem indeksu in potem še enkrat na šestem.

Pri enotnem križanju začnemo pri prvem genu in izberemo naključnega starša. Od izbranega agenta prenesemo gen in nadaljujemo na naslednji gen v zaporedju. Na ta način nadaljujemo do zadnjega gena v zaporedju. Potomec agentov ima tako genom, ki je naključno sestavljen iz dveh starševskih genomov [16].



Slika 3: Enotno križanje[5]; Na vsakem indeksu v novem genomu je izbran naključen starš, katerega gen se prenese.

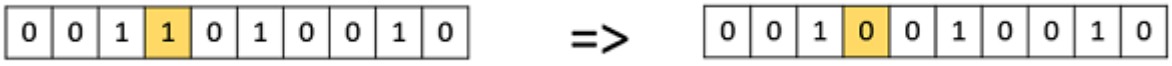
[1]Vir:https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

[3]Vir:https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

3.1.5 Mutacija

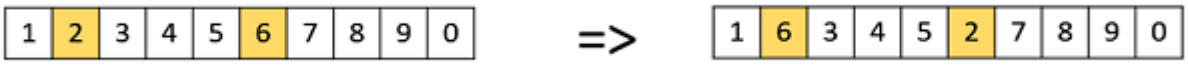
Ob kreaciji začetne populacije ima vsak agent naključno generiran genom, kar predstavlja končno mnogo različnih genov v celotni populaciji. Skozi selekcijo in križanje se zaradi izločanja šibkih genov to število dodatno manjša. Naše možnosti iskanja optimalnega genoma so omejene le na najboljšo permutacijo genov, ki so bili generirani v začetni populaciji. Ker so geni generirani naključno, to pomeni, da rešitev nikoli ni zagotovljena. Mutacija genoma tako vpelje v sistem nove možnosti rešitev. Pa si najprej poglejmo več različnih vrst mutacij [14].

Kot je prikazano na sliki 4, vzamemo pri bitni mutaciji naključen gen v genomu nekega agenta in ga spremenimo. Tako zagotovimo, da v genomu nikoli ne bo zmanjkalo možnosti za razvoj zaradi pomanjkanja diverzitete genov na določenih mestih [15].



Slika 4: Mešanje genov[6]Na četrtem indeksu genoma se gen spremeni iz 1 v 0.

Zamenjalna mutacija je pojav, pri katerem izberemo dva naključna gena v genomu in jih med seboj zamenjamo. Tako smo efektivno prinesli diverziteto na dve različni mesti v genomu, kot prikazuje slika 5. Vendar pa ta mutacija ne zagotavlja, da se genom lahko z gotovostjo razvije, saj v genom nismo vpeljali novih genov [16].



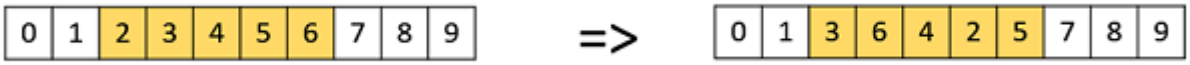
Slika 5: Mešanje genov[8]; Naključno sta bila izbrana gen na indeksu dve in gen na indeksu 6. Gena sta zamenjana znotraj genoma.

Mutacijo pa lahko razširimo tudi v mešanje večje količine genov. Vzamemo lahko poljubno podmnožico genov genoma in jih premešamo med sabo, kot prikazuje slika 6. Mutacija bo tako potencialno močno povečala raznolikost genov na specifičnih mestih, vendar obstaja velika verjetnost, da se bo agent v naslednji generaciji izkazal slabše [16].

[5]Vir:https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm

[6]Vir:https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm

[8]Vir:https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm



Slika 6: Mešanje genov[10]; Geni v genomu od indeksa dve do indeksa 6 so med seboj premešani.

3.2 Paralelizem

Osnovna ideja paraleliziranja programov je razdeliti program na kose, ki jih lahko hkrati izvajamo na večjem številu procesorskih jeder in pri tem prihranimo čas za poganjanje programa. Princip "deli in vladaj"je pri genetskih algoritmih aplikativen na več načinov. Nekatere paralelizacijske metode uporabljajo eno populacijo, medtem ko jo druge razdelijo na več, med seboj neodvisnih, podmnožic populacije [17].
Pri algoritmih, ki uporabljajo samo eno populacijo, je pogosto deljenje nalog na podprocese. Naloge, ki so v našem primeru evalvacija, križanje in mutacija so razdeljene v podprocese, od katerih je vsak zadolžen za podmnožico začetne populacije. Tu pa se deli algoritem glede na sinhronizacijo populacij [17]. Večina implementacij se po fazah sinhronizira vsako generacijo, medtem ko se pri nekaterih implementacijah med seboj ne sinhronizirajo, temveč samo komunicirajo spremembe glavni niti.
Implementacije, ki delijo populacije na več podmnožic, so zaradi svoje delitve ponavadi implementirane na sistemih z več računalniki. Zaradi svoje modularnosti omogočajo velike začetne populacije, kar predstavlja širše področje iskanja rešitve [17]. Kljub temu pa velikost celotne populacije ne predstavlja velike prednosti, saj so geni izolirani v svoje skupine in se mešajo z drugimi populacijami v le redkih implementacijah.

[10]Vir:`https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm`

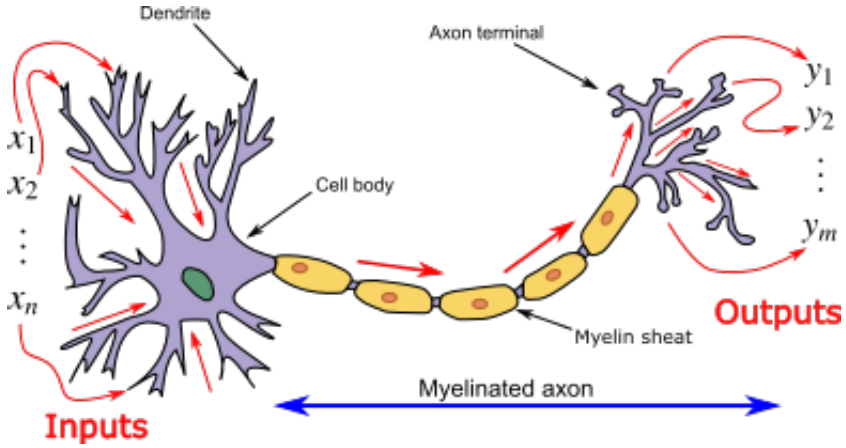
4 Nevronske mreže

Nevronske mreže so računski modeli. Njihova naloga je simuliranje živalskega centralnega živčnega sistema, katerega glavna komponenta je nevron [18]. Nevronske mreže so sposobne strojnega učenja in zaznavanja vzorcev v izbranem naboru podatkov. Ti prepleteni sistemi povezujejo nevrone, ki s pomočjo vhodnih podatkov izračunajo izhodni podatek in ga podajo drugim povezanim nevronom. Nevronske mreže lahko predstavimo z usmerjenim grafom brez ciklov [19].

4.1 Struktura

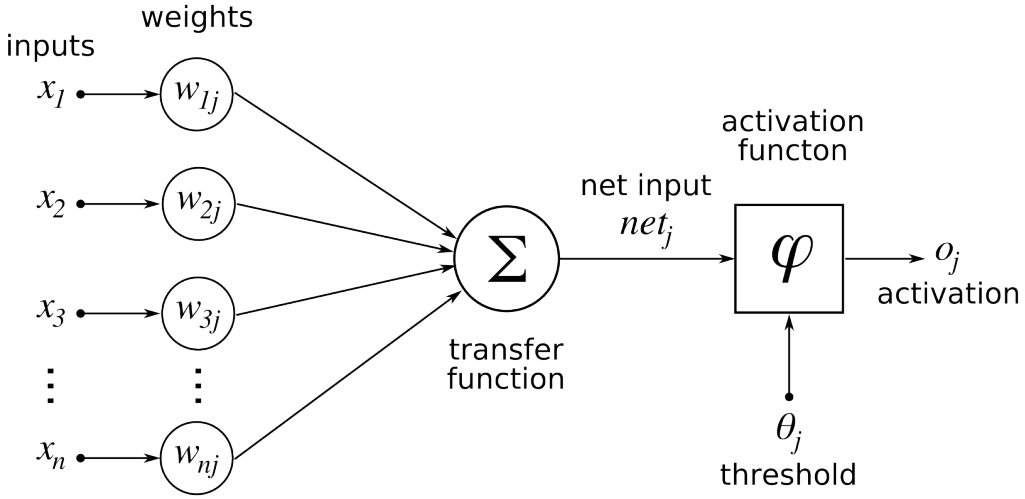
4.1.1 Vozlišča in povezave

Koncept nevrona je prevzet s področja nevroznanosti. Nevroni v ljudeh in živalih so v glavnem sestavljeni iz treh glavnih gradnikov. Dendriti predstavljajo vhodne stimualacije nevrona, aksoni predstavljajo izhodne stimulacije, jedro pa glavni del nevrona [18]. Glavna gradnika umetnih nevronskih mrež sta vozlišče in povezava. Povezave povezujejo nevrone in predstavljajo vhodne in izhodne podatke umetnega nevrona, kar v naravi prestavlja dendrite in aksone. Umetni nevron sam pa predstavlja jedro, ki vhodne podatke pretvori v izhodne podatke, ter jih preko povezav oddaja drugim povezanim nevronom [19].



Slika 7: Nevron[2];Slika prikazuje nevron živalskih možganov ter njihovo predstavo v umetnih nevronskih mrežah. Dendriti(dendrite) predstavljajo vhodne povezave, medtem ko aksoni (axon terminal) prikazujejo izhodne povezave.

Vsi podatki v nevronski mreži so predstavljeni s števili. Vsaka povezava ima svojo utež. Ko povezava prejme nek podatek, ki ga mora prenesti v izhodni nevron, ta podatek najprej uteži s privzeto utežjo. Ko so v vozlišču zaznani vhodni podatki, ki jih prenesejo povezave, vozlišče vzame uteženo vsoto vseh vhodnih povezav in jo preslika s pomočjo aktivacijske funkcije. Ta rezultat potem preko izhodnih povezav prenese v naslednji nevron [18].



Slika 8: Struktura umetnega nevrona[4]; x_m na sliki prikazuje vhodne podatke, ki so vstavljeni v nevron. Vsi vhodni podatki so pomnoženi z ustrezno utežjo w in sešteti (transfer function). Nato je vsota podana v aktivacijsko fukncijo φ (activation function), ki nam poda rezultat nevrona (označeno z o).

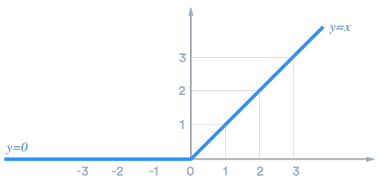
[2]Vir: `https://en.wikipedia.org/wiki/Artificial_neural_network`

Poleg vhodnih podatkov ima vsak nevron pristranski vhodni podatek, ki je vedno nastavljen na 1 [20]. Namen pristranskega vhodnega podatka je omogočiti nevronu neničelni izhodni podatek, tudi če so vsi vhodni podatki enaki 0. Rezultat nevrona je označen z y in predstavlja vsoto m-tih vhodnih podatkov pomnoženih z utežmi w, ki je podana v aktivacijsko funkcijo φ [20].

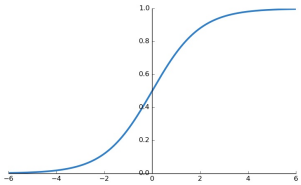
$$y = \varphi\left(\sum_{j=0}^{m} w_j x_j\right)$$

Najpogostejši aktivacijski funkciji sta usmerjena linearna enota in sigmoidna funkcija [20]. Umerjena linearna enota slika vhodno število x v $y \in [0, \infty)$, kjer se vsa števila manjša od 0 slikajo v 0, vsa ostala števila pa se slikajo sama vase [21]. Sigmoidna funkcija pa je funkcija, ki slika x v $y \in (0, 1)$ [21], po formuli prikazani spodaj.

$$f(x) = \frac{1}{1 + e^{-x}}$$



Slika 9: Graf usmerjene linearne enote



Slika 10: Graf sigmoidne funkcije

[5]

4.1.2 Sloji in globoke nevronske mreže

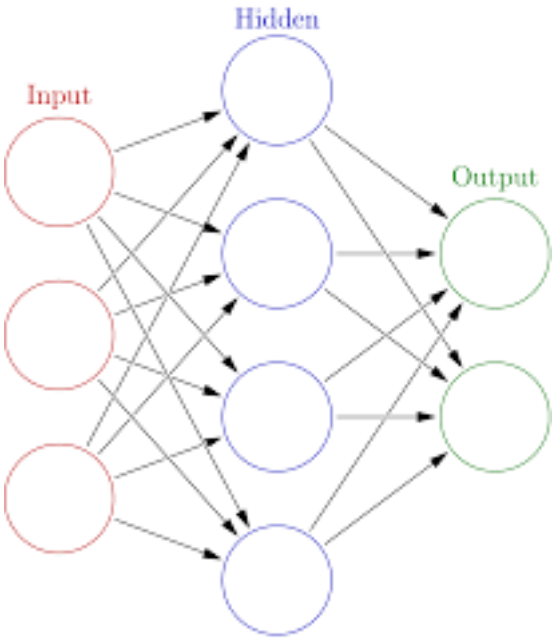
Nevronske mreže so razdeljene na vhodni, izhodni in skriti sloj, kateri pa so sestavljeni iz nevronov. Umetni nevroni v slojih med seboj niso povezani, temveč so povezani le z nevroni v naslednjem sloju. Vhodni sloj predstavlja sloj, kjer je v mrežo podan vektor vhodnih podatkov, iz katerih se mreža uči. Izhodni sloj predstavlja možne napovedi mreže, kjer vsak nevron predstavlja enega od možnih odgovorov [18]. Aktivacijska funkcija umetnih nevronov v izhodnem sloju je navadno usmerjena linearna enota. V skritem sloju pa se nahaja večina računskih operacij, ki predelajo vhodne podatke in jih prenesejo v izhodni sloj, navadno s pomočjo sigmoidne finkcije [20].
Nevronske mreže, ki imajo več skritih slojev, imenujemo globoke nevronske mreže. Te

[4]Vir: `https://m.tau.ac.il/~tsirel/dump/Static/knowino.org/wiki/Artificial_neural_network.html`

[5]Vir slike 9: `http://ronny.rest/blog/post_2017_08_10_sigmoid/`; vir slike 10: `https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7`

mreže so zaradi števila komutacijskih korakov bolj kompleksne in so sposobne reševati bolj kompleksne probleme. Vendar pa so zaradi velikega števila uteži težje za učenje [20].

Umetne nevronske mreže ne podajo enega odgovora, temveč nam dajo raven zaupanja v določen odgovor. Napoved umetne nevronske mreže je nato izbrana tako, da poiščemo umetni nevron, ki ima največjo vrednost in nato pogledamo katerega od odgovorov predstavlja dani nevron [19].



Slika 11: Slika [7]prikazuje enostavno umetno nevronsko mrežo, ki je sestavljena iz treh slojev: vhodni sloj (Input, označeno z rdečo barvo), izhodni sloj (output, označeno z zeleno barvo) in skriti sloj (hidden, označeno z modro barvo).

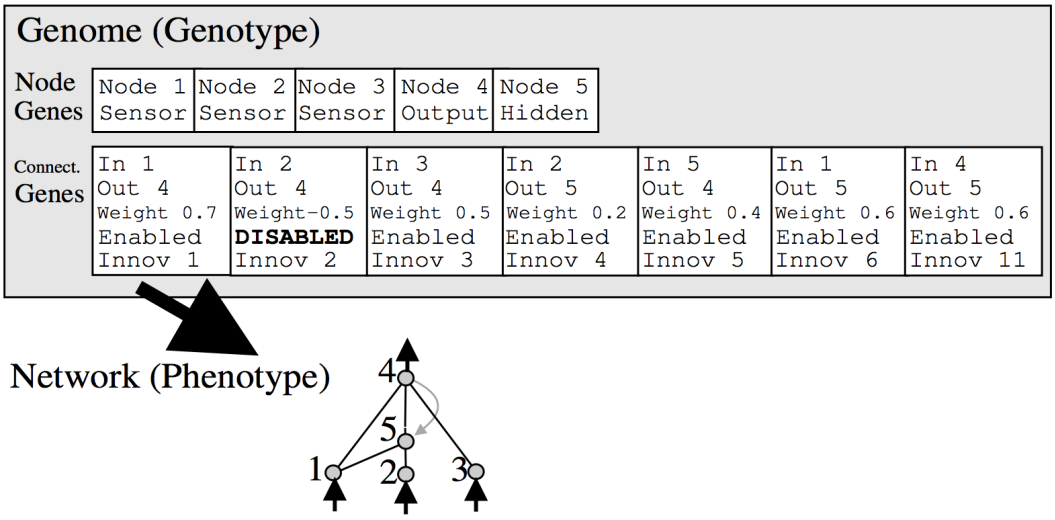
[7]Vir: `https://en.wikipedia.org/wiki/Artificial_neural_network`

5 NEAT algoritem

Nevro-evolujucija je pristop genetskih algoritmov k optimizaciji nevronskih mrež, ki je na področju vzpodbujevalnega učenja prikazal velik potencial, saj se je pri nalogah kot je uravnavanje ravnotežja palice izkazal bolje, kot Q-učenje in prilagodljiva hevristična kritika [22]. Gre za optimizacijo umetnih nevronskih mrež preko optimizacije topologije mreže, ne le optimizacije uteži. V prostoru rešitev išče najmanjšo optimalno topologijo za rešitev danega problema [23].

5.1 Genom

Genom je v nevro evolucijskih genetskih algoritmih predstavljen kot skupek vozlišč in povezav, ki sestavljajo nevronsko mrežo agenta v okolju [22]. Vsak agent začne z enako topologijo mreže in naključnimi utežmi povezav. Ponavadi začetna topologija zajema en skriti sloj ali pa sploh nobenega skritega vozlišča [23].
Vsakemu genu je ob kreaciji inkrementalno dodeljeno inovacijsko število. To število nam omogoča kronološko ločevanje genov [22]. Prav tako lahko preko inovacijskih števil primerjamo podobnost dveh genomov. Nižje kot je največje skupno inovacijsko število, bolj sta bila gena ločena v preteklem razvoju [22]. Povezavi z enakim inovacijskim številom predstavljata enak gen in topologijo, tudi če imata drugačne uteži. Povezav v mreži ne smemo brisati, saj bi tako lahko izgubili inovacijsko število in s tem pregled nad zgodovinskim razvojem genoma, zato se neuporabljene povezave onemogoči in se ne izražajo v fenotipu, kljub temu, da obstajajo v genotipu [23].

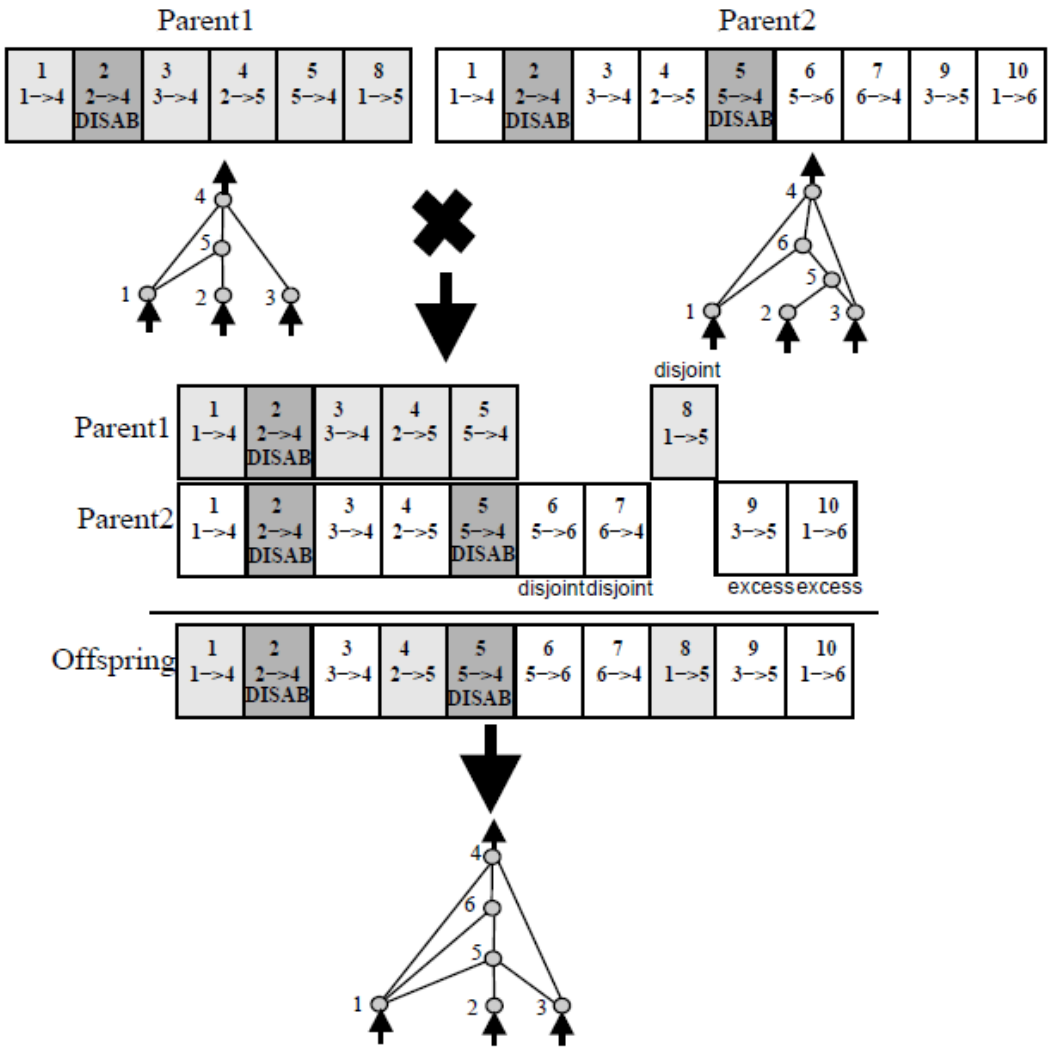


Slika 12: [2]Genom umetne nevronske mreže je zapisan kot skupek vozlišč (node genes) in skupek povezav (connect. genes). Vsako vozlišče ima inovacijsko število in tip. Lahko je vhodno (sensor), izhodno (output) ali skrito (hidden) vozlišče. Povezave imajo pet atributov. Kazalca na začetno (in) in končno (out) vožlišče, utež povezave (weight), inovacijsko število (innov) in atribut, ki ponazarja, če je povezava izražena v fenotipu (enabled/disabled).

5.2 Križanje

Križanje algoritma NEAT je malo drugačno kot pri klasičnih genetskih algoritmih, saj genom predstavlja umetna nevronska mreža. Za križanje so inovacijska števila genov izjemno pomembna, saj nam omogočijo, da med dvema genomoma poiščemo ujemajoče, ločene in odvečne gene s pomočjo primerjanj inovacijskih števil. Ta števila se prenašajo iz generacije v generacijo. Ko preko mutacije v mrežo pride nov gen, mu je dodeljeno globalno inovacijsko število, ki se poveča za ena, za naslednji gen. Mogoč problem predstavlja večkratna pojavitev enake mutacije bodisi v različnih genomih ali pa v primeru, ko je bila mutacija enkrat prej že zavržena. Tako bi lahko genu, ki mu je v preteklosti že bilo dodeljeno inovacijsko število, dodali novo. To bi vodilo do podvojitve inovacij in genov v genomih, zato moramo enakim mutacijam vedno dodeliti enako inovacijsko število. Ujemajoči geni so geni, ki jih vsebujeta oba agenta in se dedujejo naključno. Ločeni geni so geni, katere vsebuje samo eden od staršev in se prenašajo na potomca samo z bolje ocenjenega starša. Enako velja za odvečne gene, ki predstavljajo zadnje ločene gene v sekvenci genoma. V primeru, da imata starša enako oceno, lahko vzamemo gen poljubnega starša. [22]

[2]Vir: članek [22]



Slika 13: [4]Slika prikazuje križanje genomov umetnih nevronskih mrež, kjer je predpostavljena enaka ocena obeh staršev. V primeru prvih štirih genov imata oba starša enak genom, zato vzamemo povezavo naključnega starša, saj imata kljub enakemu inovacijskem številu lahko različne uteži. Ločeni geni in odvečni geni se dedujejo od bolje ocenjenega starša. V tem primeru je predpostavljena enaka ocena, zato so prevzeti vsi geni.

5.3 Razporeditev v vrste

Ker se manjše topologije optimizirajo hitreje kot večje topologije ter dodajanje novih vozlišč ali povezav v mreži zniža kvaliteto delovanja v okolju, to pomeni, da imajo novi geni izjemno majhno možnost preživetja v okolju za več kot eno generacijo. Kljub temu, da lahko njihova inovacija predstavlja ključen gen za rešitev problema v prihodnosti

[4]Vir: članek [22]

[23]. Rešitev tega problema predstavlja razdelitev populacije v vrste.

5.3.1 Genetska razdalja

Če želimo razdeliti populacijo na skupine glede na topologijo genoma agenta, moramo predstaviti koncept genetske razdalje, ki predstavlja oceno različnosti dveh genomov. V vsaki generaciji lahko vsakega od agentov uvrstimo v vrsto. Vsaka vrsta ima agenta, ki je predstavnik vrste. Če je genetska razdalja med agentom, ki ga uvrščamo, manjša od združljivostne razdalje, ga uvrstimo v vrsto. Drugače pa naredimo novo vrsto in tega agenta postavimo za predstavnika vrste [22]. Genetska razdalja je število, ki opisuje podobnost genomov. Formula za genetsko razdaljo δ upošteva število odvečnih genov(E), število ločenih genov(D), povprečno absolutno razliko v utežeh ujemajočih genov ($\overline{W}$), kar vključuje tudi onemogočene gene, ter normalizator N, ki predstavlja število genov v večjem genomu [22]. Koeficienti c_1, c_2 in c_3 predstavljajo uteži, ki jih lahko priredimo glede na pomembnost člena enačbe.

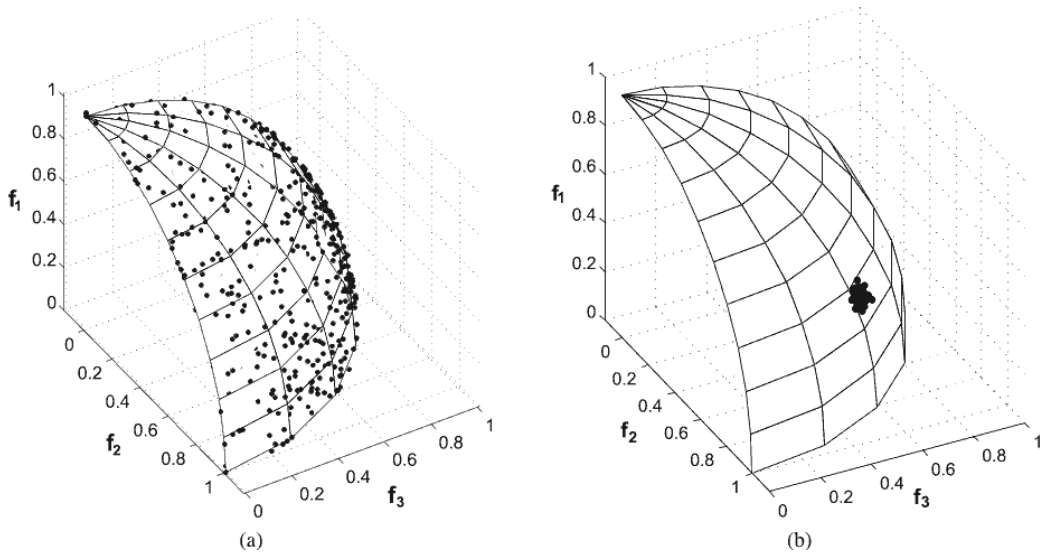
$$\delta = \frac{w_1 E}{N} + \frac{w_2 D}{N} + w_3 \overline{W}$$

5.3.2 Selekcija

Kot že omenjeno lahko predstavitev novih genov v genom agentu drastično zniža kvaliteto opravljanja naloge, zato je prezervacija novih genov zahtevna. V protiutež temu selekcioniramo agente za križanje znotraj vsake vrste posebej. Agent se mogoče v primerjavi s celotno populacijo ne izraža najbolje, vendar mora za prezervacijo gena tekmovati le z agenti, s katerimi si deli vrsto. V vsaki generaciji izločimo iz vsake vrste polovico agentov, ki so bili znotraj svoje vrste najslabše ocenjeni [22]. Tako omogočimo novim genom več časa za optimizacijo.

5.3.3 Eksplicitna delitev ocen

Eksplicitna delitev ocen je optimizacijska metoda, ki spodbuja populacijo za iskanje rešitev v širšem prostoru. Optimizacija za vsakega agenta izračuna razdaljo od drugih agentov v genetskem prostoru ter njegovo oceno uteži glede na njegovo bližino. Bližje kot je agent drugim agentom v prostoru, nižja je njegova otežena ocena, saj ta spodbuja iskanje in optimizacijo novih genov. Tako so geni bolj razpršeni po prostoru rešitev, kar populaciji predstavlja manjšo možnost, da obtiči v lokalnem minimumu.



Slika 14: [6]Slika predstavlja vpliv eksplicitne delite ocen v prostoru rešitev. Desna stran predstavlja populacijo, ki je ujeta v lokalnem minimumu in optimizira rešitev, ki pa potencialno ni najboljša. Leva polovica slike pa predstavlja populacijo, ki enakomerno porazdeljeno išče rešitev po celotnem prostoru.

5.4 Mutacije

Mutacije v okolju predstavljajo inovacijo in s tem možnost razvoja. V sistemu lahko pride do pomanjkanja genomov, saj skozi selekcijo in križanje gene izločujemo iz populacije. Da bi preprečili to pomanjkanje genov v genomih v sistemu, vpeljemo mutacijo na umetnih nevronskih mrežah [23].

5.4.1 Mutiranje povezav

Povezave lahko v genomu mutiramo na nekaj načinov. Povezavi mutiramo utež tako, da jo pomnožimo z koeficientom med 0 in 2, kar predstavlja optimizacijo uteži. Pomembnost podatka, ki ga prenaša povezava, se tako potencialno podvoji, ali pa se izniči. Ta povezava še vedno ostane pozitivna ali negativna, zato kot drugo mutacijo ponavadi vpeljemo še obračanje uteži, pri kateri utež na povezavi pomnožimo z -1 [**?**]. Tako povezava lahko predstavi podatke kot pozitivne ali negativne, ne glede na dano začetno vrednost ob inicializaciji. Prav tako lahko povezave dodajamo ali deaktiviramo. Ob dodajanju povezav predstavimo v genomu nov gen, ki povezuje dve nepovezani vozlišči [23]. Takšna mutacija v sistemu predstavi kompleksnost. Kot protiutež te kompleksnosti zadnja mutacija na povezavah predstavlja onemogočenje gena.

[6]Vir:`https://stackoverflow.com/questions/37836751/what-are-fitness-sharing-and-niche-count-in-e`

Če povezavo onemogočimo, potencialno odstranjujemo nepotrebno povezana vozlišča, ki agentu ne omogočata dobrega odločanja v okolju.

5.4.2 Mutiranje vozlišč

Pri dodajanju novega vozlišča izberemo poljubno aktivno povezavo, na katero bo dodano vozlišče. Ker zaradi inovacije ne smemo spreminjati genov, dodamo genomu dve novi povezavi, kjer prva povezuje začetno vozlišče z dodanim vozliščem in druga dodano vozišče s končnim vozliščem izbrane povezave. Izbrano povezavo nato deaktiviramo, saj jo dodani geni nadomestijo [22]. Ker bi dodani povezavi z naključnimi utežmi verjetno poslabšali uspešnost agenta v okolju, prvo povezavo nastavimo na 1. To pomeni, da na podatek, ki ga prenaša v novo vozlišče ni spremenjen, drugo povezavo pa inicializiramo na vrednost, ki jo je imela povezava, katero smo deaktivirali [22]. Tako smo v genom prinesli potencial za nadaljnji razvoj, brez da bi spremenili njegovo uspešnost. Zaradi genetske razdalje bo verjetno agent spadal pod drugo vrsto, ki mu pomaga pridobiti čas za optimizacijo nove topologije [23].

6 Implementacija

6.1 Opis problema

V tem poglavju bo opisana naša implementacija genetskega algoritma NEAT. Problem, ki ga algoritem rešuje je navigacija po dvodimenzionalnem poligonu. Naloga algoritma je poiskati optimalno nevronsko mrežo za premikanje po prostoru. Agent predstavlja vozilo in okolje predstavlja proga. Cilj je dosežen, ko vsaj eden od agentov naredi poln krog po progi. Tudi če je cilj dosežen, imajo agenti nekaj generacij časa za morebitno optimizacijo rešitve. Agent je upoštevan kot neuspešen če se zaleti v steno ali pa se neha premikati. Če se agent ustavi, lahko z gotovostjo rečemo, da se ne bo nikoli več premaknil, saj bi za premik potreboval drugačne vhodne podatke, vendar pa na vhodne podatke vpliva le njegova lokacija v prostoru, ki pa se zaradi ustavitve ne spreminja. Zato je agent zaklenjen v smrtnem objemu.

6.2 Okolje

Proga je sestavljena iz šestkotnikov. V središču enega od njih se nahaja začetna točka (štart), kjer je začetna pozicija vseh agentov. Vsi agenti so ob začetku obrnjeni v isto smer. Na progi so postavljene kontrolne točke, ki se nahajajo na stičiščni stranici poljubnih dveh sosednjih šestkotnikov. Namen kontrolnih točk je usmerjati agenta v pravo smer po progi. Ob inicializaciji aktiviramo nekaj zaporednih kontrolnih točk, na začetku proge. Te kontrolne točke usmerjajo agenta v pravo smer po progi. Z vsako prevoženo kontrolno točko se le-ta onemogoči in se omogoči naslednja v vrsti.

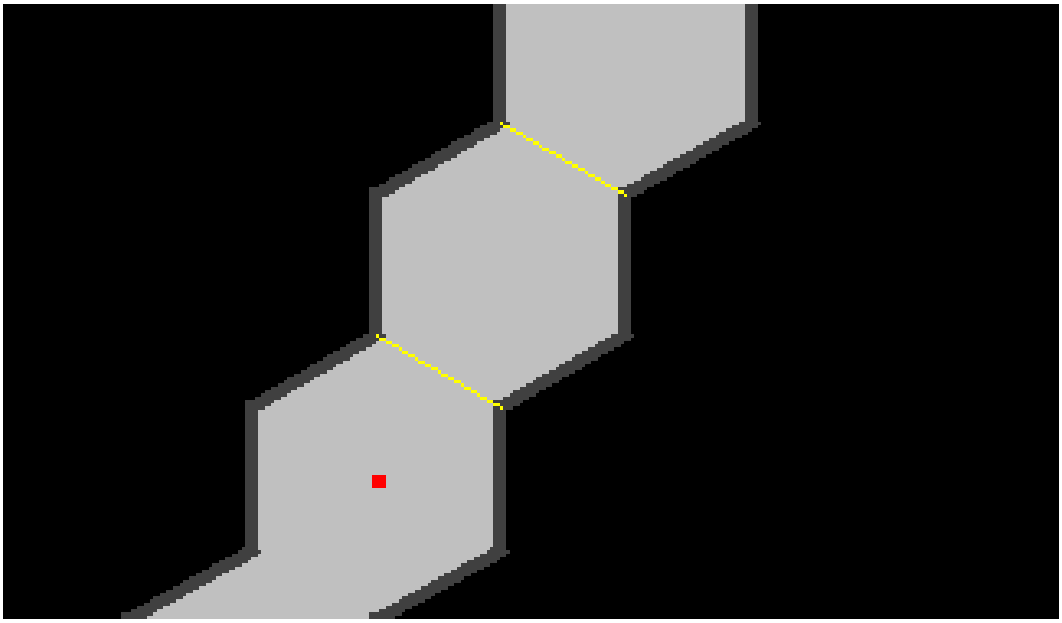
6.2.1 Zapis proge

Za sestavljanje prog smo implementirali urejevalnik prog, v katerem lahko uporabnik nariše progo in jo shrani v datoteko. Program shrani progo v lastnem formatu, ki je sestavljen iz seznama koordinat v prostoru. Tako shranjena proga se lahko kadarkoli uporablja za učenje. Zapis proge je sestavljen iz treh delov: začetna točka, zaporedje

točk zidov in zaporedje parov koordinat kontrolnih točk.

$$
\begin{aligned}
&ST\\
&x_{start}, y_{start}\\
&WL\\
&x_{11}, y_{11}; x_{12}, y_{12}; x_{13}, y_{13}; ...; x_{1i}, y_{1i}\\
&x_{21}, y_{21}; x_{22}, y_{22}; x_{23}, y_{23}; ...; x_{2j}, y_{2j}\\
&...\\
&x_{n1}, y_{n1}; x_{n2}, y_{n2}; x_{n3}, y_{n3}; ...; x_{nk}, y_{nk}\\
&CP\\
&x_{11}, y_{11}; x_{12}, y_{12}\\
&x_{21}, y_{21}; x_{22}, y_{22}\\
&...\\
&x_{m1}, y_{m1}; x_{m2}, y_{m2}
\end{aligned}
\tag{6.1}
$$

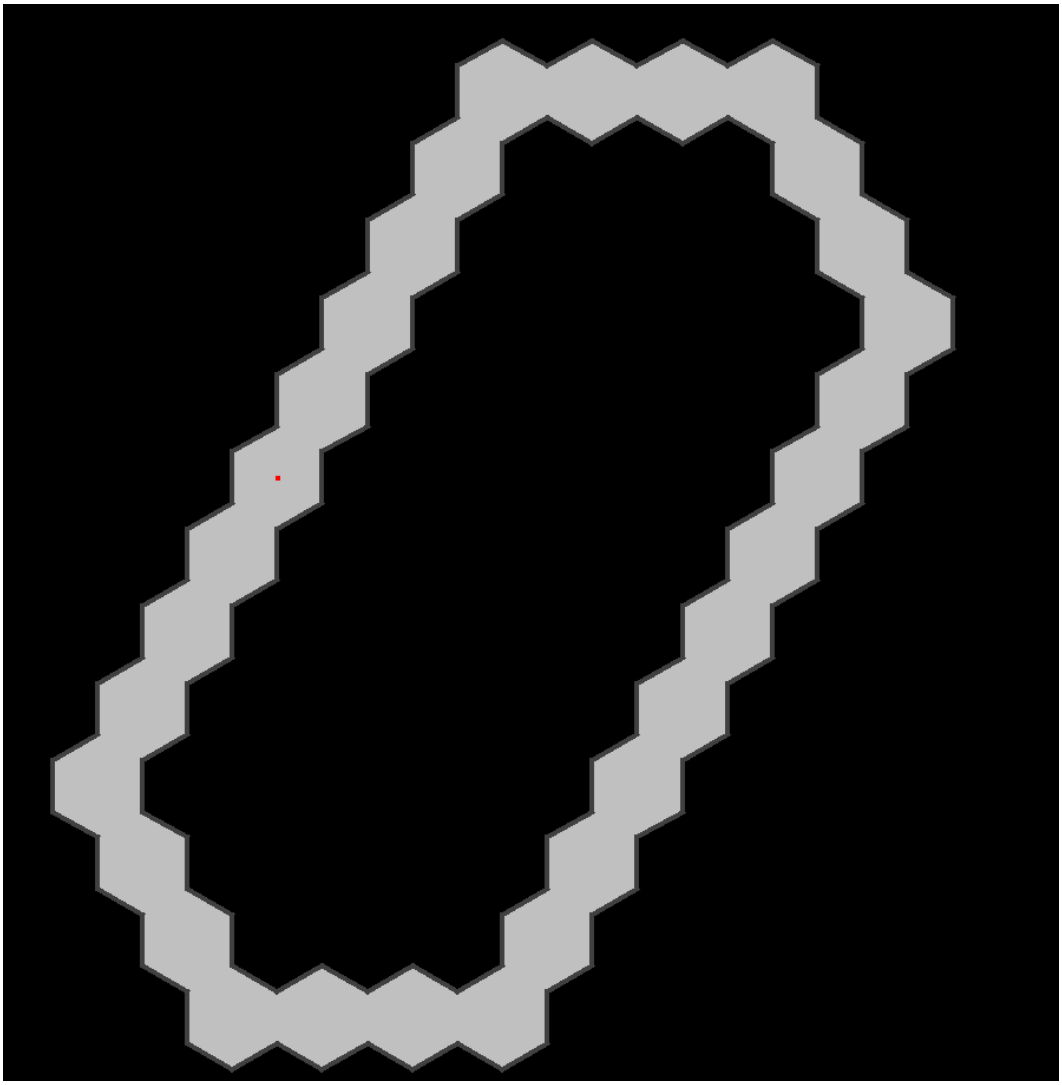
Kot je predstavljeno v poglavju 6.1, je vsaka točka zapisana kot par dveh celih števil, ki sta ločeni z vejico (x, y), točke pa so med seboj ločene s podpičjem $(x_1, y_1; x_2, y_2)$. Začetna točka je predstavljena samo z eno točko, medtem ko so zidovi in kontrolne točke predstavljeni kot zaporedje točk (dolžine zaporedja kontrolnih točk je 2). Te so med seboj ločeni z znakom za novo vrstico.



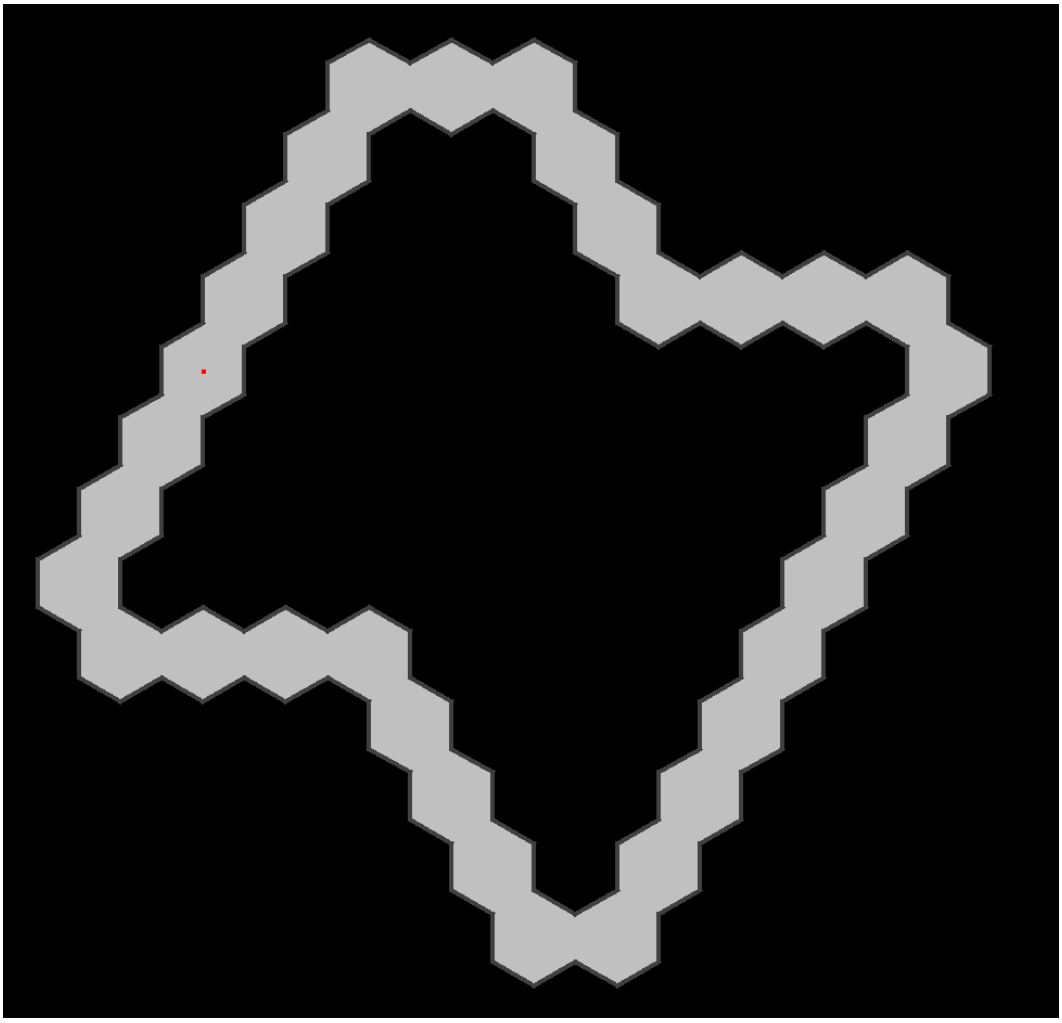
Slika 15: Na sliki je vidna začetna točka (v rdeči), kjer začnejo vsi agenti v okolju. Kontrolne točke so predstavljene z rumenimi črtami, ki povezujejo bližnji oglišči šestkotnika.

6.2.2 Testni primeri

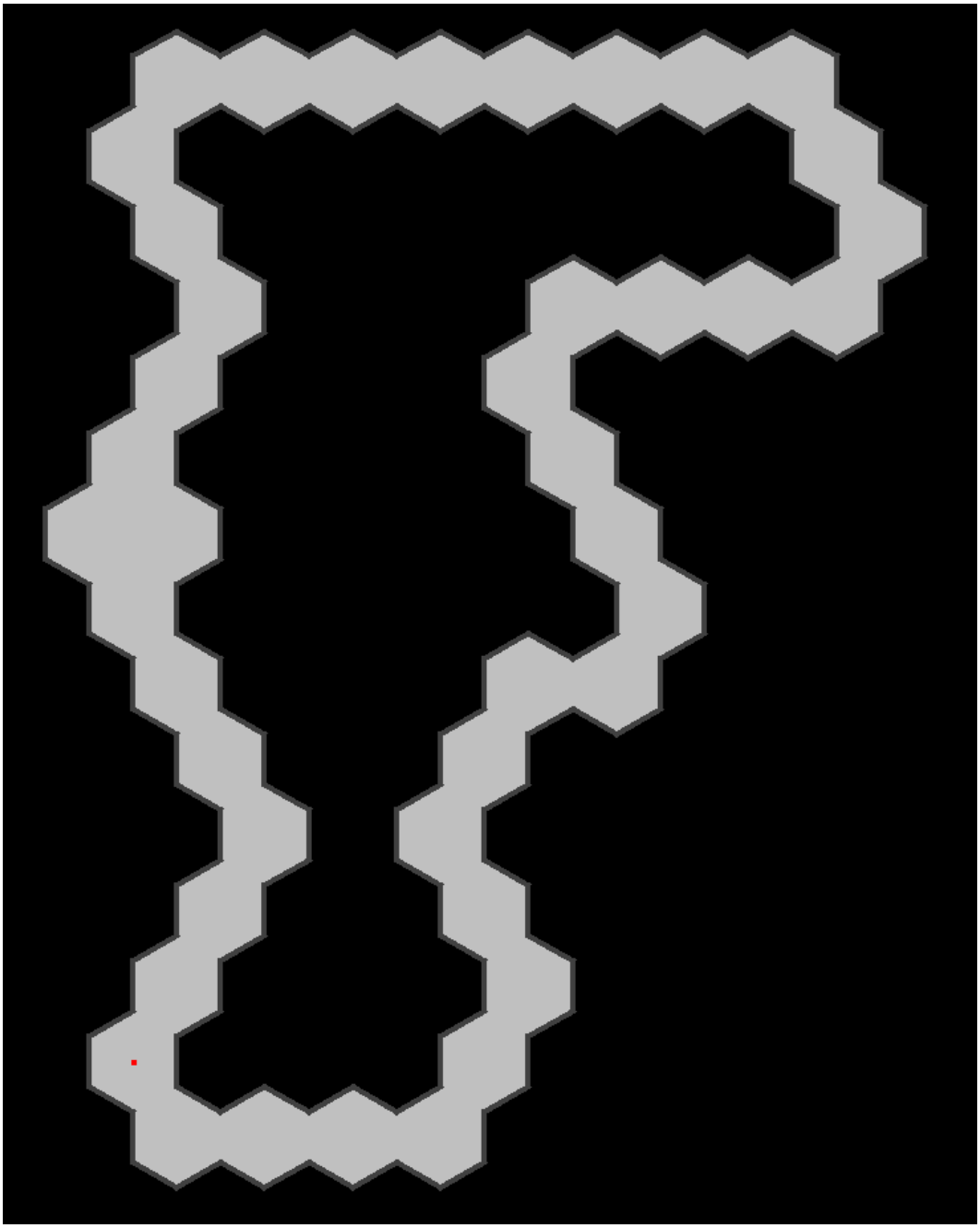
Za namen empiričnega testiranja algoritma so bile izdelane štiri različna okolja, ki po ocenjujejo uspešnost modela, kot opisano pri slikah 16, 20, 18 in 19. Rezultati testov, bodo kasneje predstavljeni v poglavju 7.



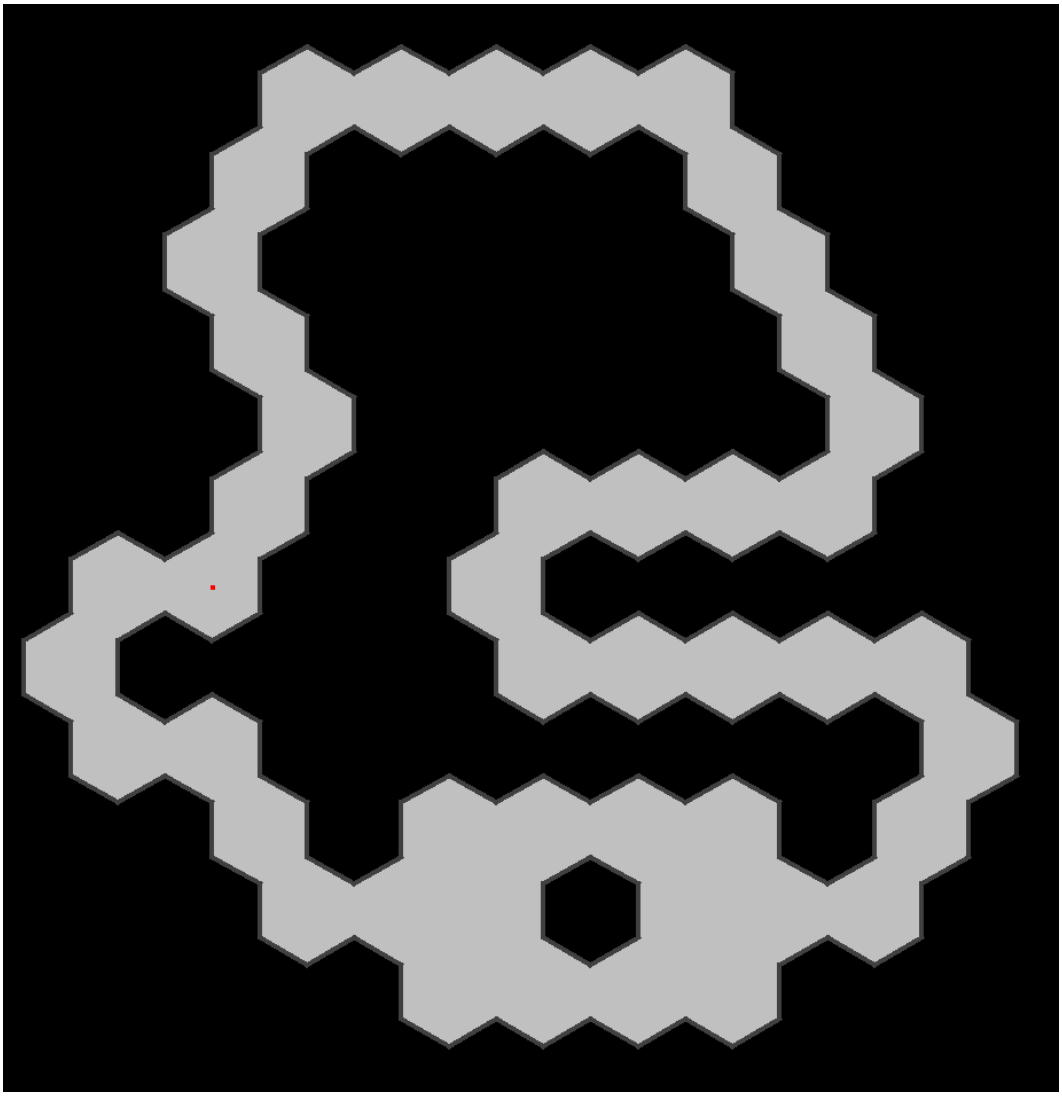
Slika 16: Na sliki je prikazana prva proga, ki predstavlja najenostavnejšo progo izmed testnih okolij. Enostavnost predstavlja odsotnost levih ovinkov, kar pomeni, da je optimalna rešitev predvideno manj kompleksna.



Slika 17: Na sliki je prikazana druga proga, ki je na prvi pogled enostavna, vendar pa je sestavljena na način, ki lahko algoritem zavede. Proga na začetku predstavlja le ravnine in desne ovinke, kar pomeni, da preden agent doseže prvi levi ovinek, ki bi testiral razvoj ustreznih genov, so lahko te že večinoma izločeni iz genoma. Prav tako dolge ravnine testirajo agentovo kontrolo hitrosti.



Slika 18: Na sliki je prikazana tretja proga, katere je testiranje vsesplošnega obnašanja modela na ne trivialnem poligonu. Proga je razgibana in vsebuje veliko ovinkov, kljub temu pa ne vsebuje večjih pasti za algoritem.



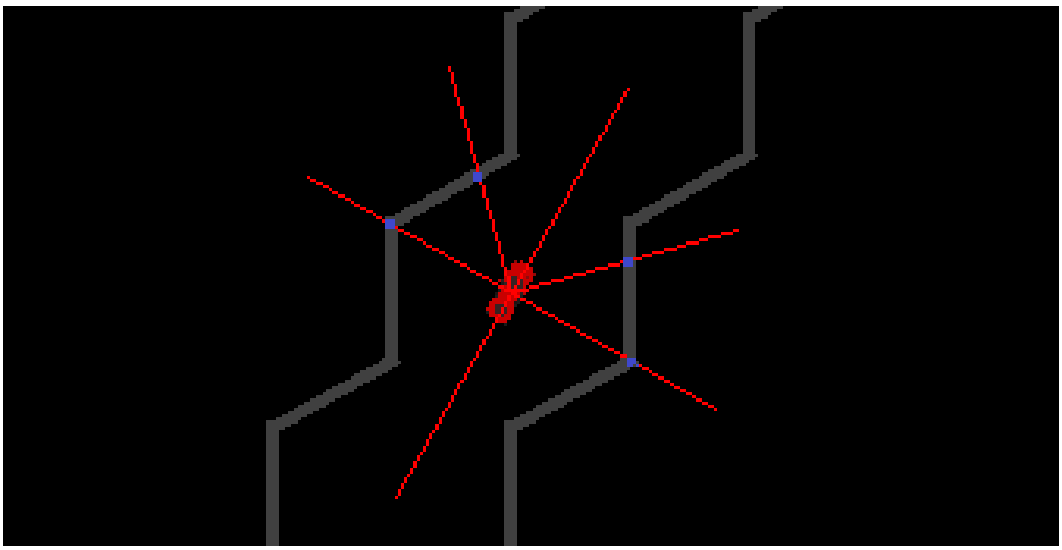
Slika 19: Na sliki je prikazana četrta proga, ki je izmed štirih testnih prog predvidoma najzahtevnejša. Vsebuje nekaj težjih ovinkov, vendar največjo pregrado v razvoju predstavlja predel proge, ki se proti koncu razdeli na dva hodnika in nato spet združi. Namen predela je zmesti agente, da krožijo okrog sredinskega stebra.

6.3 Agenti

Vozilo v implementaciji predstavlja agenta, ki v okolju optimizira svoje obnašanje. V implementaciji populacijo predstavlja 1024 agentov. Vsako iteracijo vsak od agentov izračuna vhodne podatke za svoj genom, s pomočjo katerega nato izbere enega od dejanj, ki so mu na voljo. Ta dejanja vplivajo na njegovo lokacijo znotraj okolja in bodo bolje opisana v poglavju 6.3.2.

6.3.1 Navigacija

Vsak agent se zaveda svoje lokacije v okolju. To doseže s šestimi vidnimi linijami, ki predstavljajo vid agenta. Agent vidi naravnost, nazaj, levo, desno ter naravnost-levo in naravnost-desno. Vrednost, ki jo predstavlja vidna linija, je število med 0 in 1, ki ponazarja razdaljo, do najbližje stene v smer linije. Če v vidnem polju agenta presečišča s steno ni, ima vidna linija maksimalno vrednost (ena). Vsako iteracijo se agent lahko odloči za premik. Lahko pospeši, vendar le do določene maksimalne hitrosti, ali pa upočasni. Upočasnjevanje istočasno deluje kot zavora ter vzvratna prestava, saj je hitrost agenta lahko tudi negativna. Poleg tega lahko agent zavije levo ali desno. Zavoj je linearno sorazmeren s hitrostjo agenta z omejeno maksimalno želeno hitrostjo zavoja. Če avto stoji pri miru, agent ne mora zaviti. Zadnja možnost, ki jo ima agent pa je, da ne naredi nič. To bo agenta premaknilo v trenutni smeri, glede na trenutno hitrost, na katero pa deluje nekaj upora.

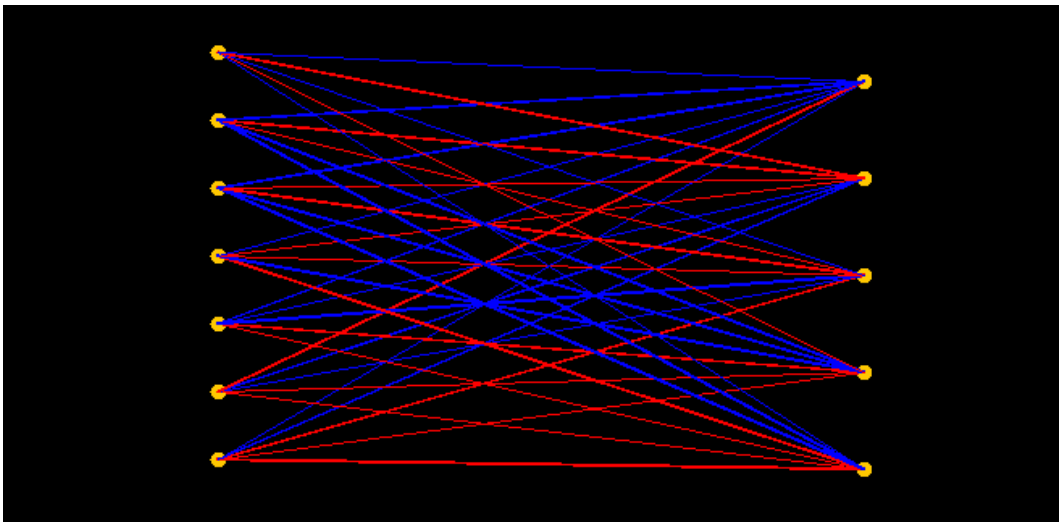


Slika 20: Na sliki je prikazan agentovo dojemanje okolja. Agent je prikazan kot rdeče vozilo. Ima šest vidnih linij, ki so narisane kot rdeče črte, ki izvirajo v središču in gledajo vsaka v svojo smer. Vse linije so enake fiksne dolžine. Vidne linije, ki se sekajo z zidom, prestavljajo točke zidu, ki jih agent vidi in so predstavljene kot modre točke na presečišču zidu in vidne linije.

6.3.2 Genom

Genom agentu predstavlja proces odločanja za izvedbo določenega dejanja v okolju. Začetni genom je v implementaciji sestavljen iz vhodnega sloja in izhodnega ter ne vsebuje skritega sloja. Vhodni sloj je sestavljen iz sedmih vozlišč. Vhodni vektor pa je sestavljen iz sedmih podatkov, ki so: hitrosti agenta, vidna linija pred agentom, vidna

linija za agentom, vidni liniji levo in desno od agenta ter vidni liniji naravnost-levo in naravnost-desno od agenta. Da vsi vhodni uravnoteženi, vrednost hitrosti agenta normaliziramo z maksimalno hitrostjo agenta. Te podatki se potem propagirajo po mreži dokler ne dosežejo izhodnega sloja, ki je sestavljen iz petih vozlišč, ki predstavljajo potencialne odločitve agenta. Prvo vozlišče izhodnega vektorja predstavlja odločitev za nobeno od aktivnih dejanj, tako da agent ne naredi ničesar. Drugo in tretje vozlišče predstavljata zavoj levo in zavoj desno. Četrte vozlišče predstavlja pospeševanje, peto pa pojemek.



Slika 21: Slika prikazuje začetni genom agenta, ki ima naključno inicializirane uteži. Genom ima 7 vhodnih vozlišč, ki jih vidimo kot levi stolpec rumenih pik na sliki in pet izhodnih vozlišč, ki pa so predstavljene kot stolpec rumenih pik na desni polovici slike. Nevronska mreža je polno povezana in vsaka povezava ima naključno utež. Vsaka povezava je narisana kot črta, ki povezuje dve vozlišči. Levo vozlišče je vhodno in desno izhodno. Debelina črte povezave nam pove absolutno velikost uteži. Večja kot je utež, debelejša je črta, s katero je narisana. Barva uteži pa nam pove ali je povezava pozitivna (rdeča) ali negativna (modra).

7 Testiranje

7.1 Opis testov

V namen preverjanja algoritma smo implementirali štiri teste. Cilj je bil oceniti kako določeni deli algoritma vplivajo na uspešnost razvoja populacije. Kritična dela algoritma, ki smo jih izbrali sta eksplicitno deljenje ocen in delitev v vrste. V prvem testu bo algoritem povno delujoč z vsemi sklopi (poimenovan NORMAL), v drugem testu smo onemogočili sistem za eksplicitno deljenje ocen (poimenovan NO EFS), kot smo v tretjem onemogočili razporeditev v vrste (poimenovan NO SPECIATION). V zadnjem testu pa smo onemogočili oba podsistema algoritma (poimenovan NO SPECIATION AND EFS). Pričakujemo da se bo najboljše rezultate prinesel test, kjer delujejo vsi podsistemi, ter najslabše test, kjer ni omogočen noben od podsistemov. Pravtako pa pričakujemo, da se bo bolje odrezal test, kjer je onemogočeno deljenje ocen, na pram testu, ki ima onemogočeno razporeditev v vrste. Oba podsistema vplivata na zaščito inovacije in s tem preprečevanje algoritmu, obtičanje v lokalnem minimumu. Vse štiri teste smo pognali na štirih progah, ki so opisane v poglavju 6.2.2. Ker algoritem predstavlja zelo dinamičen sistem in njegov razvoj vlkjučuje naključne pojave, smo vsak test zagnali desetkrat. Izvorna koda in testni primeri so na voljo na spletni platformi Github[1].

7.2 Rezultati

7.2.1 Pregled

Pri vsakem testu smo za vsako generacijo shranjevali naslednje podatke: ocena najbolj uspešnega agenta, povprečna ocena agentov ter standardna deviacija ocen agentov. Prav tako smo vsakih pet generacij spremljali podatke vrst. Podatki, ki smo jih spremljali so velikost populacije, predstavnik populacije in uspešnost populacije (povprečna ocena agentov v populaciji). Te podatke smo uporabili za grafični prikaz in spodnjih tabelah.

[1] Vir:`https://github.com/VakeDomen/NEAT-driving`

Tabela 1: V tabeli so predstavljeni rezultati testov na progah. Vsaka vrstica predstavlja povprečne podatke desetih simulacij testa. Ocena agentov je predstavljena z oznako x.

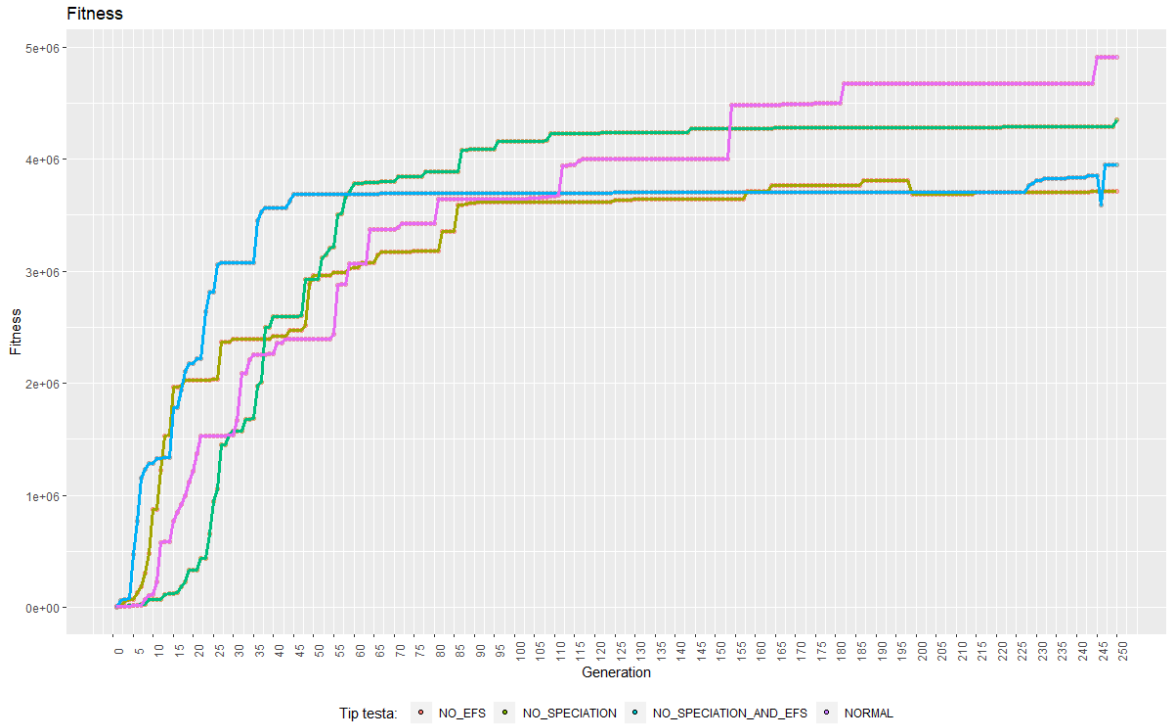
	Proga	Test	$max(x)$	$\mu(x)$	$\sigma(x)$	Razvitost[%]
1	Proga 1	NORMAL	0,99	0,12	0,58	100
2	Proga 1	NO EFS	0,98	0,18	0,456	100
3	Proga 1	NO SPECIATION	1	0,13	0,57	100
4	Proga 1	NO SPECIATION AND EFS	0,98	0,21	0,46	100
5	Proga 2	NORMAL	0,96	0,03	0,15	50
6	Proga 2	NO EFS	1	0,05	0,20	50
7	Proga 2	NO SPECIATION	0,99	0,04	0,24	80
8	Proga 2	NO SPECIATION AND EFS	0,88	0,05	0,19	60
9	Proga 3	NORMAL	0,98	0,05	0,24	80
10	Proga 3	NO EFS	1	0,12	0,37	80
11	Proga 3	NO SPECIATION	0,93	0,06	0,26	80
12	Proga 3	NO SPECIATION AND EFS	0,83	0,09	0,28	60
13	Proga 4	NORMAL	1	0,01	0,16	40
14	Proga 4	NO EFS	0,61	0,03	0,14	50
15	Proga 4	NO SPECIATION	0,64	0,01	0,05	20
16	Proga 4	NO SPECIATION AND EFS	0,84	0,06	0,24	50

Tabela 1 predstavlja podatke zbrane pri testih modela. V namen lažje berljivosti, so podatki ocen (x) normalizirani z največjo oceno doseženo na tisti progi. Takoj lahko vidimo, da so različne proge modelom predstavljale različne stopnje težavnosti, glede na procent razvitih modelov. Proga 1 modelu ni predstavljala večjih težav, saj so se na njej razvili vsi modeli. Sledijo ji proga 3, na kateri se je razvilo 75% modelov, proga 2, na kateri se je razvilo 60% modelov in najtežja, proga 4, na kateri se je razvilo le 40% modelov. Test NORMAL se je razvil v 67.5% in deli drugo mesto z NO SPECIATION AND EFS, medtem ko najboljše rezultate sta prinesla NO EFS in NO SPECIATION z 70% razvitostjo. Modeli prinesli zelo podobne rezultate ko merimo frekvenco razvitosti. Vendar pa je test, ki je onemogočeno deljenje ocen na dveh od štirih prog našel najoptimalnejšo rešitev. Podatki v stolpcu $\mu(x)$ nam povedo povprečno oceno modela skozi vse generacije. Večje kot je število, prej je v povprečju model našel dobro rešitev problema. Tu se je najbolje izkazal test NO SPECIATION AND EFS. Ta rezultat je smiseln, saj je ta model zaradi pomanjkanja ohranjanja

inovacije nagnjen k hitri optimizaciji, vendar je zato tudi nagnjen k padcem v lokalni minimum, kar se tudi izraža, saj model nikoli ni našel najoptimalnejše rešitve.

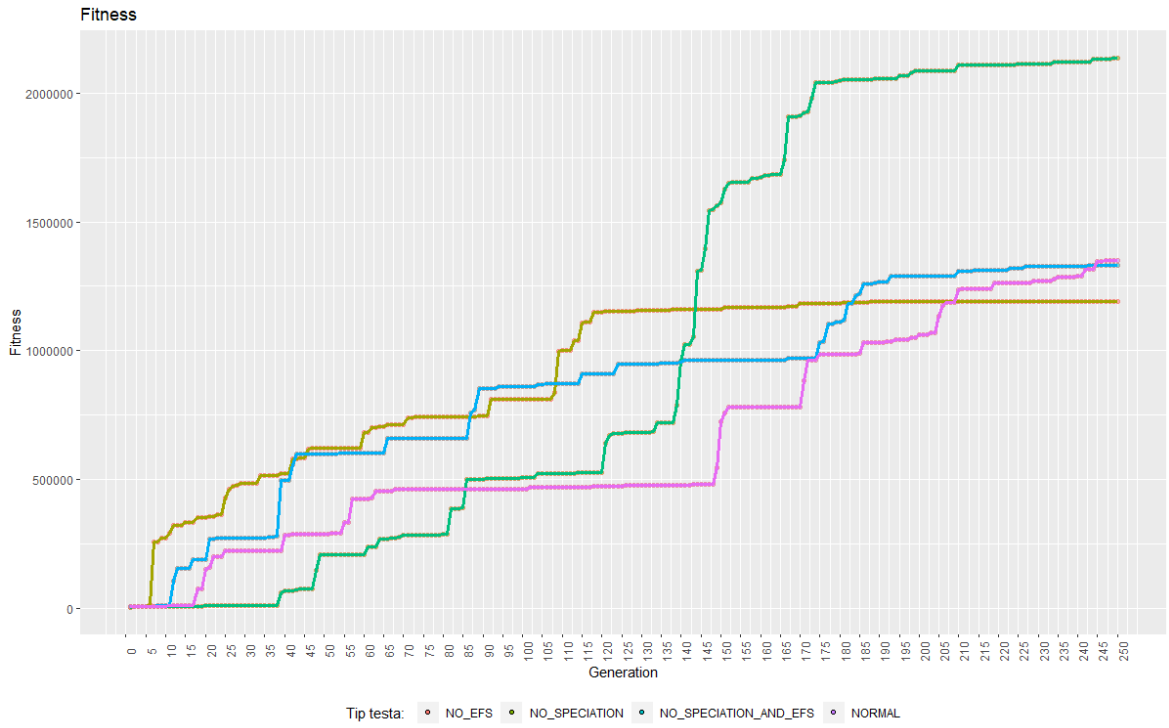
7.2.2 Razvoj agentov

Na spodnjih grafih je predstavljen razvoj agentov skozi čas. Na vsakem grafu so prikazani povprečni podatki vseh simulacij(N=10) za vsakega od testov. V grafih 22, 23, 24 in 25 so podatki testa NORMAL označeni z vijolično črto, podatki NO EFS z rjavo, NO SPECIATION z zeleno in test NO SPECIAITON AND EFS z modro barvo.



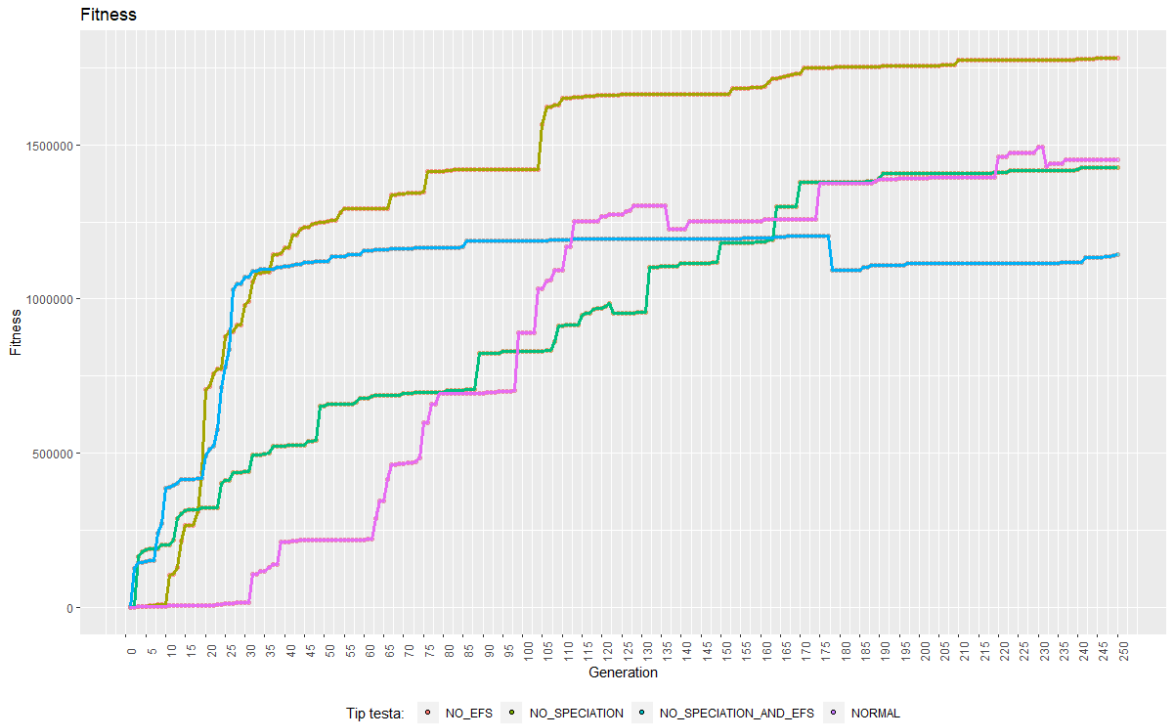
Slika 22: Graf prikazuje razvoj ocen agentov v odvisnosti od časa(generacij). Vsak od testov je predstavljen z svojo barvo. Prikazani testi so bili izvedeni na progi 1.

Graf 22 prikazuje teste, ki so bili izvedeni na prvi progi. Kot je razvidno, povprečna maksimalna ocena agentov najprej hitro narašča, potem pa hitro konvergira proti najdenem optimumu. Vsi testi so se izkazali za uspešne, kar nam pove da v tem okolju najbrž ni veliko lokalnih minimumov, v katere bi lahko agenti padli. Opazimo pa lahko da testi, ki so bili izvedeni brez deljenja ocen in razporejanja v vrste po najdeni rešitvi le-te niso uspeli izboljšati. To lahko pripišemo pomanjkanju inovacije v genih, saj s temi nastavitvami algoritem hitro zavrže nove inovacije, če se te takoj ne izkažejo za dobre.



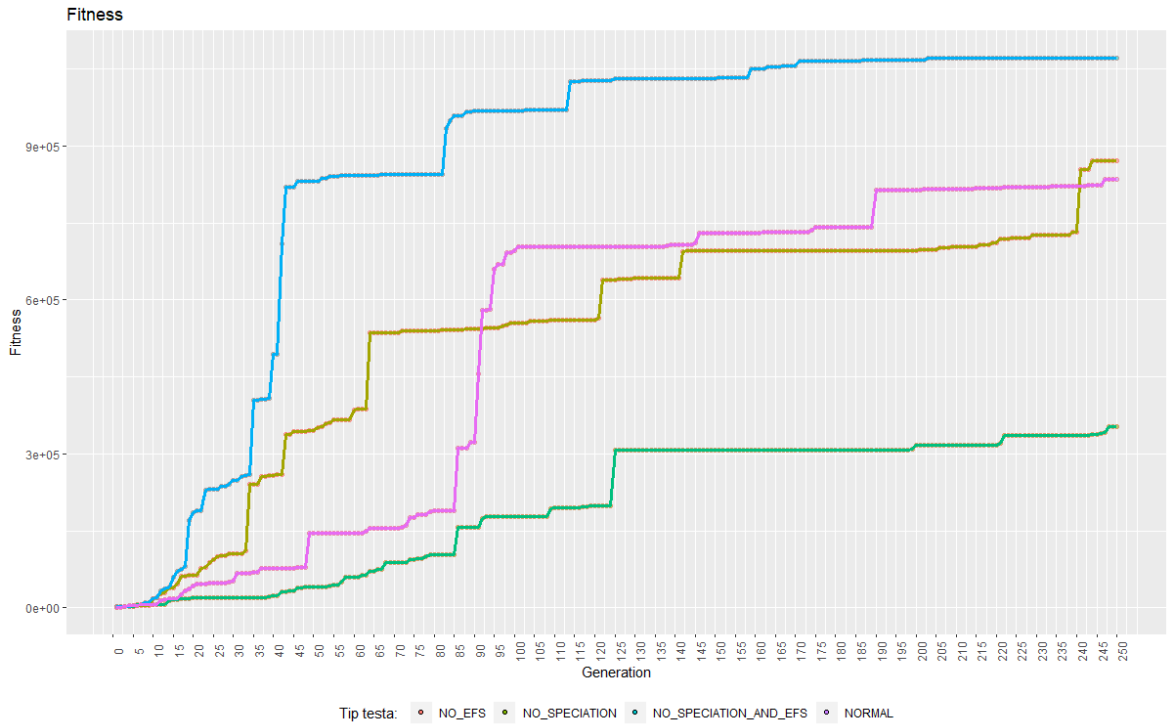
Slika 23: Graf prikazuje razvoj ocen agentov v odvisnosti od časa(generacij). Vsak od testov je predstavljen z svojo barvo. Prikazani testi so bili izvedeni na progi 2.

V nasprotju s prvo progo, lahko iz grafa 23 razberemo, da je proga 2, modelu predstavljala večje težave. Povprečna ocena agentov narašča veliko počasneje, kot pri prvi progi. Na tej progi se je v začetnih generacijah najbolje izkazal test NO EFS, vendar pa svoje rešitve v povprečju ni uspešno optimiziral. Podatki prav tako prikazujejo, da je model brez razporejanja v vrste sposoben močno optimizirati rešitev, vendar počasi išče nove inovacije.



Slika 24: Graf prikazuje razvoj ocen agentov v odvisnosti od časa(generacij). Vsak od testov je predstavljen z svojo barvo. Prikazani testi so bili izvedeni na progi 3.

Graf 24 prikazuje rezultate tretje proge, katerih oblika je podobna rezultatom proge 1 (graf 22). Vendar pa po naklonu naraščanja ocen vidimo, da proga tri kljub temu ni bila tako trivialna, kot proga 1. Najboljše rezultate je imel test NO EFS, ki je hitro poiskal rešitev in jo optimiziral.

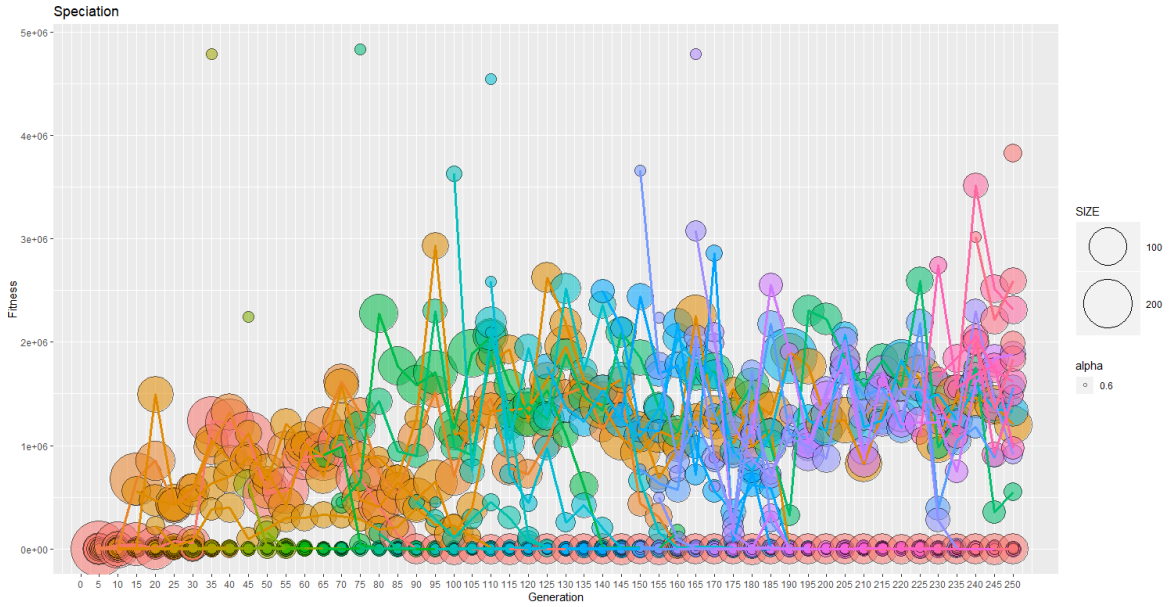


Slika 25: Graf prikazuje razvoj ocen agentov v odvisnosti od časa(generacij). Vsak od testov je predstavljen z svojo barvo. Prikazani testi so bili izvedeni na progi 4.

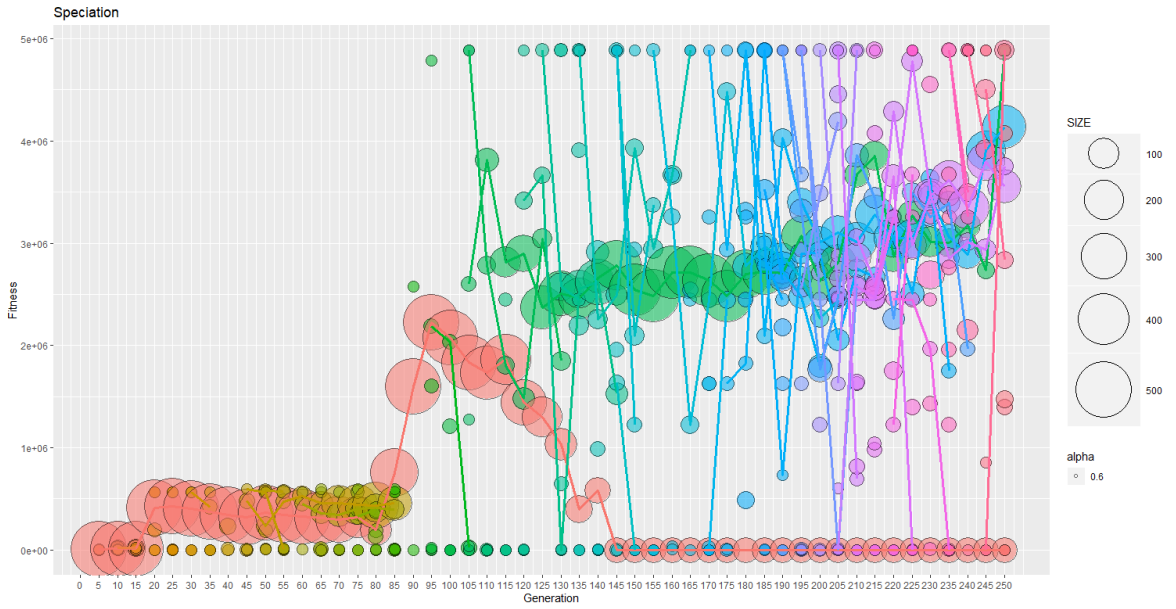
Proga 4 je izmed vseh prog rezultate najbolj polarizirala. To lahko pripišemo predelu proge, kjer se pot razdeli, kot je razvidno s slike 19, saj predstavlja velik lokalni minimum, v katerega se lahko ujamejo agenti. Najslabše rezultate je tu prinesel test NO SPECIATION, medtem, ko sta se testa NORMAL in NO EFS odrezala precej podobno. Najbolj nepričakovani rezultati pa pridejo s strani testa NO SPECIATION AND EFS, za katerega smo predvidevali, da bo prinesel najslabše rezultate, vendar pa se je na te progi izkazal najbolje.

7.2.3 Razvoj vrst

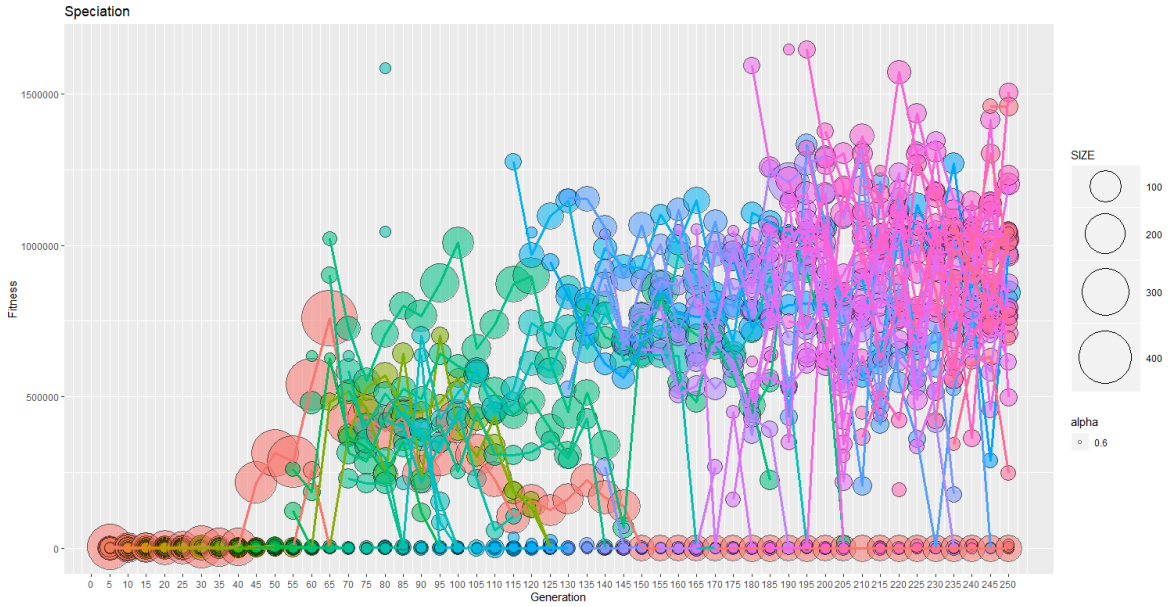
V tem poglavju bodo predstavljeni podatki, ki predstavljajo razvoj vrst skozi generacije v vseh okoljih. Ker testa NO SPECIATION in NO SPECIATION AND EFS ne omogočata modelu razvoja večih vrst, te podatki v tem poglavju ne bodo predstavljeni. Spodnji grafi (26, 27, 28, 29, 30, 31, 32, 33) prikazujejo razvoj vrst skozi čas. Za vsako progo smo za oba testa v namen predstavitve vzeli naključno simulacijo. Vrste so označene s krogom, katerega velikost prikazuje število agentov, ki pripadajo vrsti. Grafi prikazujejo več vrst, katere so predstavljene vsaka s svojim barvnim odtenkom. Vrsta je skozi generacije povezana s črto, v namen lažjega sledenja njenemu razvoju.



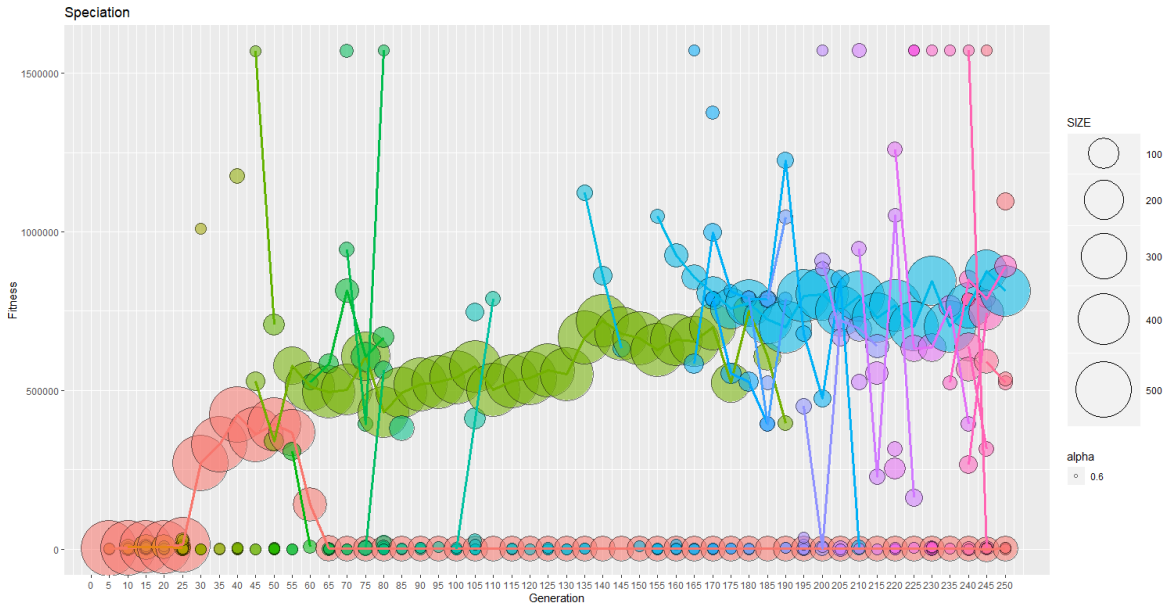
Slika 26: Graf testa (NORMAL) na progi 1. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



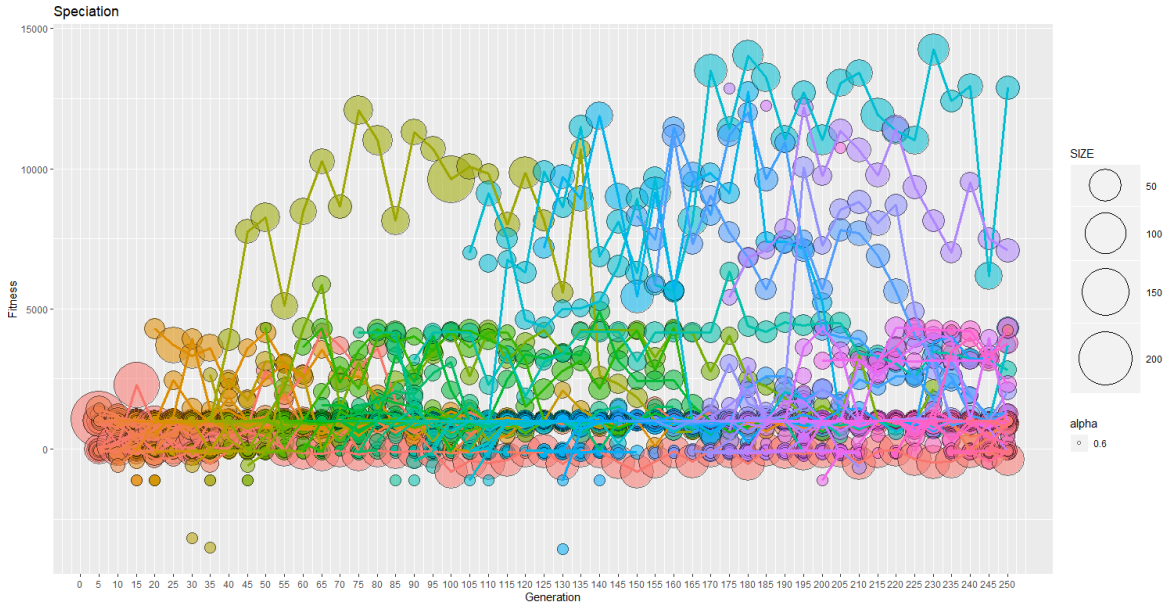
Slika 27: Graf testa (NO EFS) na progi 1. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



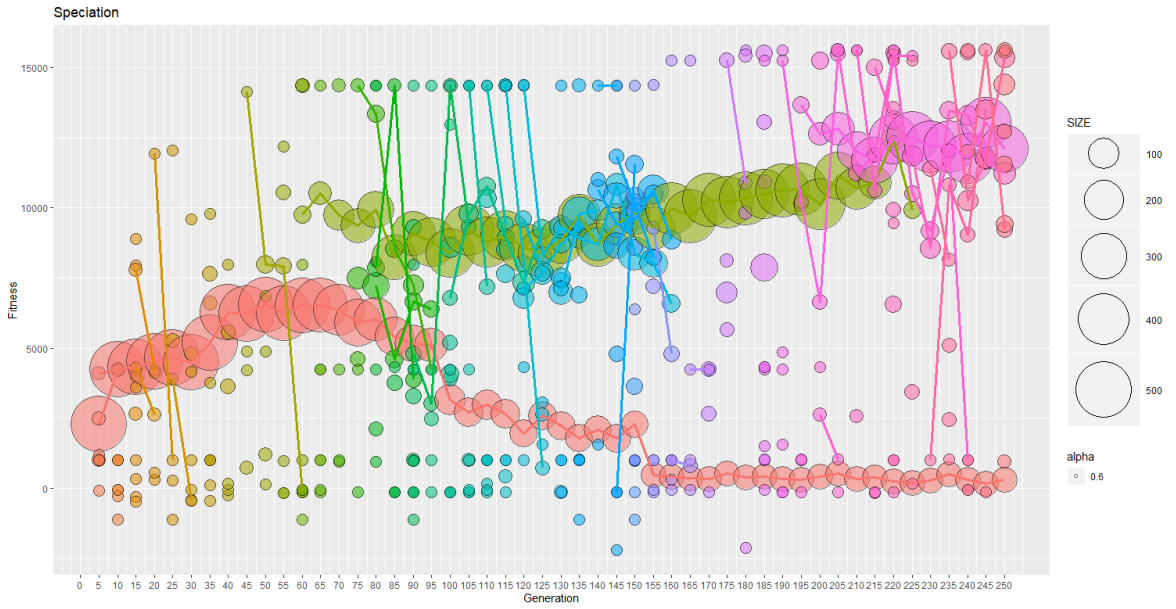
Slika 28: Graf testa (NORMAL) na progi 2. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



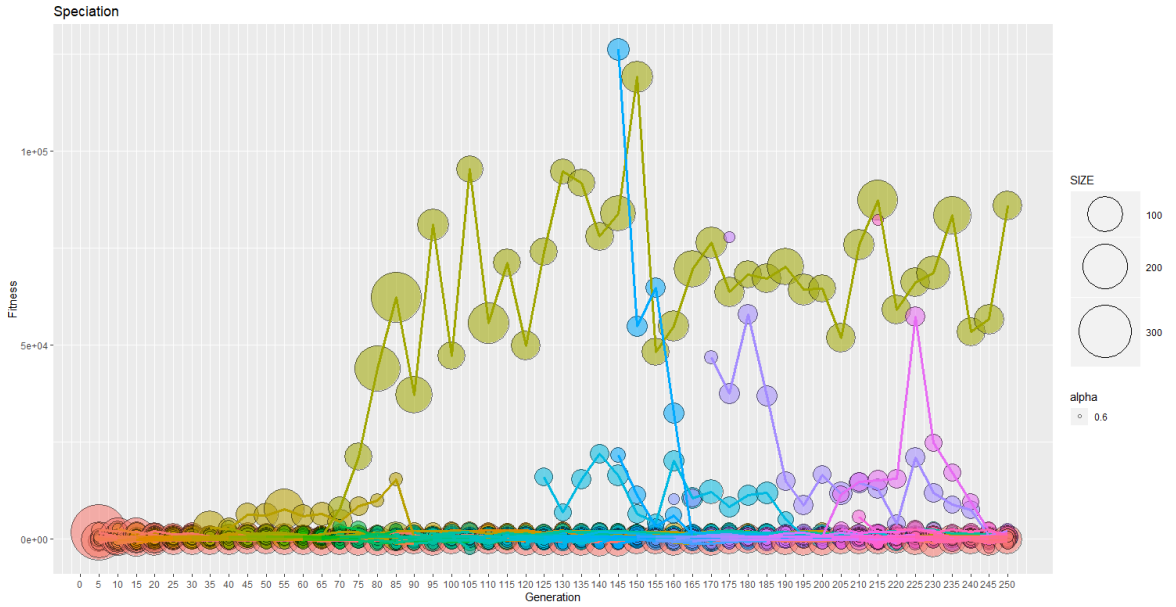
Slika 29: Graf testa (NO EFS) na progi 2. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



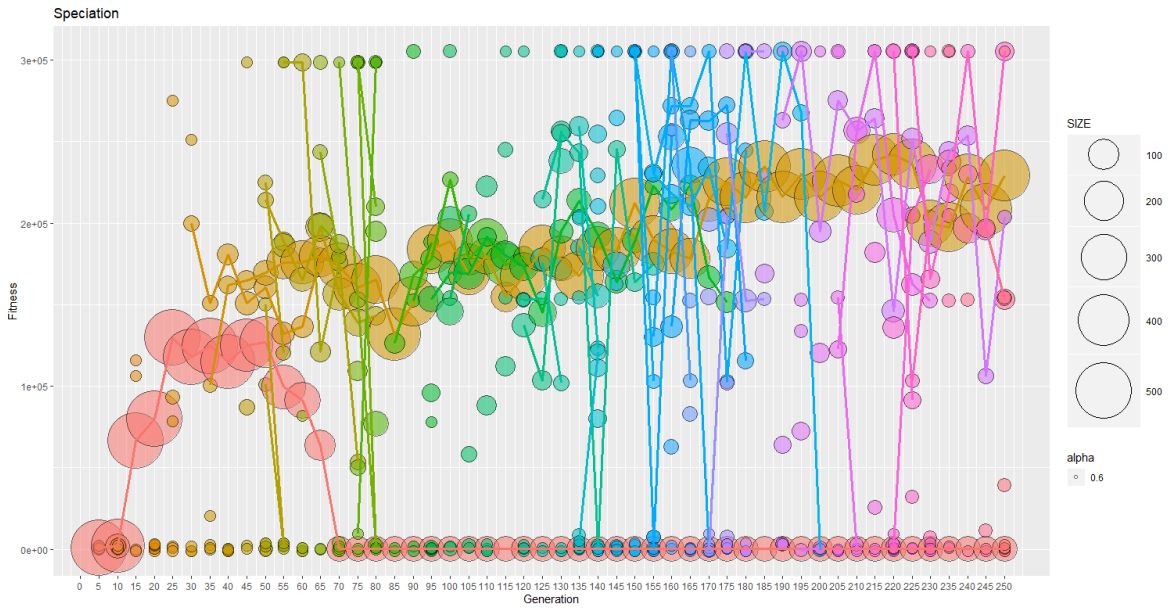
Slika 30: Graf testa (NORMAL) na progi 3. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



Slika 31: Graf testa (NO EFS) na progi 3. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



Slika 32: Graf testa (NORMAL) na progi 4. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.



Slika 33: Graf testa (NO EFS) na progi 4. Graf prikazuje vrednosti ocen vrst v odvisnosti od časa (generacij). Vrste so prikazane s krogi, katerih polmer je sorazmeren s številom agentov v vrsti. Vsaka vrsta je predstavljena s svojim barvnim odtenkom.

Iz grafov je razvidno, da so simulacije, pri katerih model ni imel omogočenega deljenja ocen, preiskale manjše območje v prostoru potencialnih rešitev, saj vsi grafi 27, 29, 31, 33 prikazujejo vzorec, pri katerem ena vrsta predstavlja takratno najboljšo

rešitev in le manjši del populacije raziskuje druge rešitve v prostoru. Ko neka vrsta naleti na primerno rešitev, ki poda boljše rezultate, kot trenutna najmočnejša vrsta, večina populacije prevzame gene te vrste. Prav tako opazimo trend, da se začetna (rdeča) vrsta, ki ponavadi predstavlja variacijo začetnega genoma, v večini nekoliko razvije, potem pa zapade, vendar nikoli ne odmre. To predstavlja nove gene, ki so vsako generacijo predstavljeni v populacijo, kot naključni agenti z naključno obliko začetnega genoma.

V grafih 26, 28, 30, 32 je prikazan razvoj vrst pri testu NORMAL, kjer je deljenje ocen med agenti omogočeno. Razvidno je, da populacija širše preiskuje prostor rešitev saj večje število vrst predstavlja več inovacije v genomih. Široko raziskovanje prostora je najbolj razvidno iz simulacije na progi 2 (graf 28).

Na grafu na na sliki 32 je prikazana simulacija, v kateri se je populacija razvila že v prvi tretjini simulacije. Kljub deljenju ocen skoraj nobena od novih vrst ni našla rešitve, ki bi bila na nivoju najbolj razvite vrste. Mogoča razlaga za ta pojav je, da je rešitev optimalna ali pa je minimum v katerem je populacija zelo blizu optimalni rešitvi. Tako nove vrste odkrivajo le slabše rešitve.

8 Zaključek

Delo predstavlja pregled konceptov strojnega učenja in implementacijo genetskega algoritma NEAT, ki zaradi optimizacije nevronskih mrež v strojnem učenju predstavlja velik potencial. Implementacija algoritma je bila uspešna, vendar je še veliko prostora za izboljšave. Algoritem je kaotičen, vsebuje veliko konstant, vsaka od njih pa močno vpliva na obnašanje modela v okolju. Nadaljnja optimizacija spremenljivk bi lahko povišala uspešnost algoritma. To še posebej velja za uporabo optimizacijskih metod, kot je eksplicitno deljenje ocen, saj se je v implementaciji izkazalo, da je model med testiranjem večkrat deloval bolje brez deljenja ocen, kot z deljenjem. Prav tako bi lahko v prihodnje algoritem testirali na bolj kompleksnih problemih, saj je izbran problem mogoče rešiti le z optimizacijo uteži v začetnih umetnih nevronskih mrežah. Tako bi optimizacije za širše iskanje po prostoru rešitev v algoritmu igrale pomembnejšo vlogo.

9 Literatura

[1] SAMUEL, Arthur L., Some studies in machine learning using the game of checkers. II—recent progress.. *IBM journal of research and development, 1959*, sprejeto v objavo. *(Citirano na strani 2.)*

[2] MCCARTHY, John, Artificial intelligence, logic and formalizing common sense.. *Philosophical logic and artificial intelligence. Springer, Dordrecht, 1989. p. 161-190.*, sprejeto v objavo. *(Citirano na strani 2.)*

[3] Russell, Stuart J. in Norvig, Peter., Artificial Intelligence: A Modern Approach. *Malaysia; Pearson Education Limited*, sprejeto v objavo. 2016 *(Citirano na strani 2.)*

[4] HINTON, Geoffrey E.; SEJNOWSKI, Terrence Joseph; POGGIO, Tomaso A. (ed.) in Unsupervised learning: foundations of neural computation, *MIT press*. 1999. *(Citirano na strani 3.)*

[5] KAELBLING, Leslie Pack; LITTMAN, Michael L.; MOORE, Andrew W. in Reinforcement learning: A survey, Journal of artificial intelligence research. *1996*, sprejeto v objavo. *(Citirano na strani 3.)*

[6] BENGIO, Yoshua; COURVILLE, Aaron; VINCENT, Pascal in Representation learning: A review and new perspectives, IEEE transactions on pattern analysis and machine intelligence. *2013*, sprejeto v objavo. *(Citirano na strani 3.)*

[7] ZIMEK, Arthur; SCHUBERT, Erich in Outlier detection, Encyclopedia of Database Systems. *2017*, sprejeto v objavo. *(Citirano na strani 3.)*

[8] CHANDOLA, Varun; BANERJEE, Arindam; KUMAR, Vipin in Anomaly detection: A survey, ACM computing surveys (CSUR). *2009*, sprejeto v objavo. *(Citirano na strani 3.)*

[9] PIATETSKY-SHAPIRO, Gregory, Discovery, analysis, and presentation of strong rules. *Knowledge discovery in databases*, sprejeto v objavo. 1991 *(Citirano na strani 4.)*

[10] BASSEL, George W., Functional network construction in Arabidopsis using rule-based machine learning on large-scale data sets. *The Plant Cell*, sprejeto v objavo. 2011 *(Citirano na strani 4.)*

[11] QUINLAN, J. Ross, Induction of decision trees. *Machine learning, 1986*, sprejeto v objavo. *(Citirano na strani 5.)*

[12] CORTES, Corinna; VAPNIK, Vladimir in Support-vector networks, Machine learning. *1995*, sprejeto v objavo. *(Citirano na strani 5.)*

[13] Jordan, Michael I. in Bishop, Christopher M., *Neural Networks*. In Allen B. Tucker (ed.). Computer Science Handbook, Second Edition, 2004. *(Citirano na strani 3.)*

[14] HOLLAND, John Henry, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press, 1992. *(Citirano na straneh 6, 7 in 9.)*

[15] GOLDBERG, David E.; HOLLAND, John H. in Genetic algorithms and machine learning, Machine learning. *1988*, sprejeto v objavo. *(Citirano na straneh 4, 6, 7 in 9.)*

[16] MITCHELL, Melanie, An introduction to genetic algorithms. *MIT press, 1998*, sprejeto v objavo. *(Citirano na straneh 6, 7, 8 in 9.)*

[17] CANTÚ-PAZ, Erick, A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis, 1998, 10.2: 141-171*, sprejeto v objavo. *(Citirano na strani 10.)*

[18] BASHEER, Imad A.; HAJMEER, Maha in Artificial neural networks: fundamentals, computing, design, and application, Journal of microbiological methods. *2000*, sprejeto v objavo. *(Citirano na straneh 11, 12 in 13.)*

[19] HASSOUN, Mohamad H., *Fundamentals of artificial neural networks*, MIT press, 1995. *(Citirano na straneh 4, 11 in 14.)*

[20] YAO, X, *Evolutionary artificial neural networks*, International journal of neural systems, 1993. *(Citirano na straneh 13 in 14.)*

[21] MAYER, Helmut A.; STRAPETZ, Marc; FUCHS, Roman in Simultaneous evolution of structure and activation function types in

GENERALIZED MULTI-LAYER PERCEPTRONS, Proc. WSES International Conference on Neural Networks and Applications. *2000*, sprejeto v objavo. *(Citirano na strani 13.)*

[22] STANLEY, KENNETH O.; MIIKKULAINEN, RISTO in EVOLVING NEURAL NETWORKS THROUGH AUGMENTING TOPOLOGIES, Evolutionary computation. *2002*, sprejeto v objavo. *(Citirano na straneh 15, 16, 17, 18 in 20.)*

[23] STANLEY, KENNETH O.; MIIKKULAINEN, RISTO in EFFICIENT REINFORCEMENT LEARNING THROUGH EVOLVING NEURAL NETWORK TOPOLOGIES, Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation. Morgan Kaufmann Publishers Inc.. 2002, poslano v objavo. *(Citirano na straneh 15, 18, 19 in 20.)*