Primjena algoritma NEAT u optimiziranju logičkih sklopova

Pristav, Mihael

Undergraduate thesis / Završni rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Electrical Engineering and Computing / Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:168:180562

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-14**

Repository / Repozitorij:

FER Repository - University of Zagreb Faculty of Electrical Engineering and Computing repozitory

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1529

PRIMJENA ALGORITMA NEAT U OPTIMIZIRANJU LOGIČKIH SKLOPOVA

Mihael Pristav

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 1529

PRIMJENA ALGORITMA NEAT U OPTIMIZIRANJU LOGIČKIH SKLOPOVA

Mihael Pristav

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 4. ožujka 2024.

ZAVRŠNI ZADATAK br. 1529

Pristupnik:	**Mihael Pristav (0036535862)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	prof. dr. sc. Domagoj Jakobović

Zadatak: **Primjena algoritma NEAT u optimiziranju logičkih sklopova**

Opis zadatka:

Opisati problematiku oblikovanja kombinatoričkih logičkih sklopova s obzirom na različite kriterije ocjene performansi. Opisati postupak NEAT za evoluciju umjetnih neuronskih mreža i navesti neka područja primjene algoritma. Implementirati programsko ostvarenje algoritma NEAT s mogućnošću primjene na proizvoljni problem. Primijeniti ostvareni algoritam na problem sinteze i optimizacije logičkih sklopova. Ocijeniti učinkovitost algoritma s obzirom na različite kriterije optimizacije i složenost logičih sklopova. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 14. lipnja 2024.

Zahvaljujem mentoru prof. dr. sc. Domagoju Jakoboviću na slobodi u izboru teme i podršci tijekom izrade ovog rada.

Sadržaj

Uvod

Booleova algebra je fundamentalna disciplina u matematici i logici koja se bavi binarnim varijablama i operacijama nad njima. Iako postoje mnogi složeni operatori nad vrijednostima u Booleovoj algebri. Svaka složena funkcija može se prikazati kao kombinacija triju osnovnih operacija: konjunkcija, disjunkcija i negacija. Logičke funkcije također predstavljaju i logičke sklopove, koji kao domenu i kodomenu koriste vrijednosti 0,1 te kao operatore koriste logička vrata AND, OR i NOT. Logički sklopovi se koriste u gotovo svakom djelu elektronike te samim time postoji potreba za pojednostavljivanjem sklopa i njegove odgovarajuće logičke funkcije.

Svaki logički sklop je moguće prikazati kao neuronsku mrežu s odgovarajućim brojem ulaza i izlaza. Samim time postoji i način da se potrebni sklop može modelirati, na temelju potrebnih izlaza za određene ulaze, koristeći metode za razvoj i treniranje neuronskih mreža.

U ovom radu, za razvoj neuronske mreže, odabran je algoritam NEAT (*NeuroEvolution of Augmenting Topologies*). Algoritam NEAT ima mogućnost razvoja strukture neuronske mreže i vrijednosti samih neurona. Ovime NEAT teži prema pronalasku rješenja s jednostavnijom strukturom. Predstavljajući logičke sklopove kao neuronske mreže, možemo provjeriti jesu li pronađena rješenja uistinu najjednostavnije moguće strukture, te drže li se u kojem slučaju ikakve norme o pojednostavljivanju logičkih sklopova.

U početnom djelu rada obrađen je teorijski princip rada NEAT algoritma. Zatim je definirana domena i kodomena problema kojeg će algoritam rješavati. Ovi elementi su potrebni za daljnje razumijevanje promjena u osnovnom algoritmu te implementaciju istog.

Zaključni dio rada daje uvid u postignute rezultate te različite pristupe testirane kod razvoja algoritma.

1. NEAT algoritam

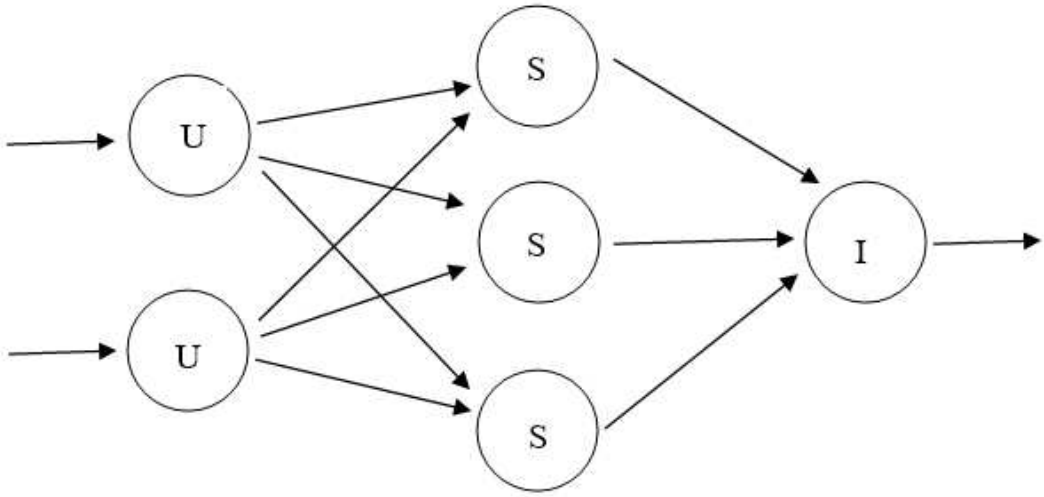
1.1. Povijest

NEAT (*NeuroEvolution of Augmenting Topologies*) je algoritam za evoluciju neuronskih mreža koji je razvijen kako bi unapredio proces optimizacije i evolucije složenih struktura neuronskih mreža. Ovaj algoritam je prvobitno predložen od strane Kennetha O. Stanleyja i Risto Miikkulainena 2002. godine u radu pod nazivom "*Evolving Neural Networks through Augmenting Topologies*". Cilj NEAT-a je da evoluira topologiju i težine neuronskih mreža simultano, omogućavajući da se složenost mreže povećava postepeno tokom evolucije.[1]

1.2. Originalna implementacija algoritma

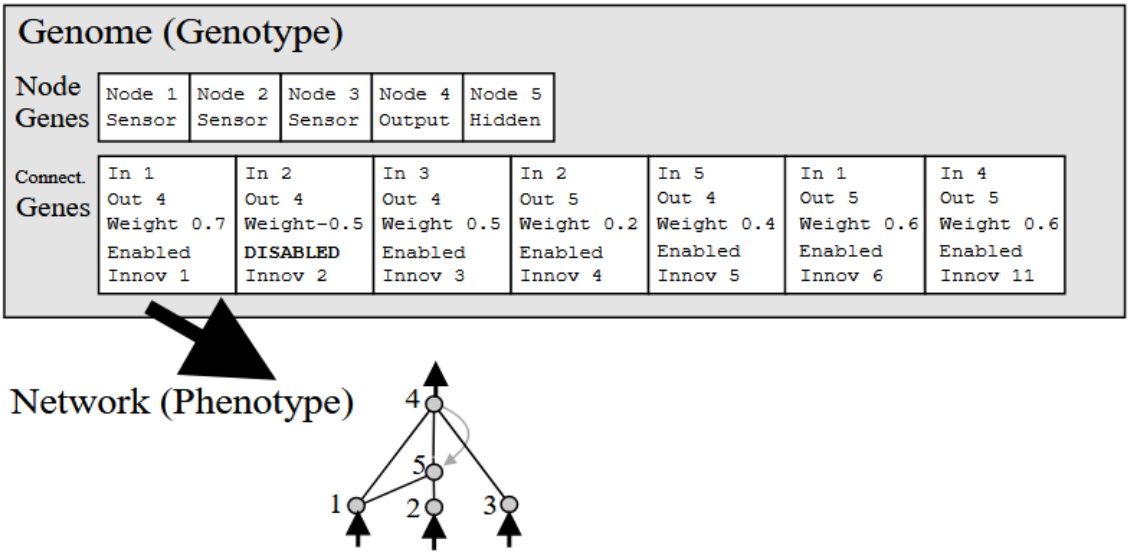
1.2.1. Jedinka

Svaka jedinka u originalnom algoritmu predstavlja jednu neuronsku mrežu. Svaka Neuronska mreža sačinjena je od 3 vrste neurona i bridova koji ih povezuju. Razlikujemo 3 vrste neurona (Slika 1.1). Neuroni označeni slovom U su ulazni neuroni, te u njih redom unosimo ulazne podatke za našu jedinku. Skriveni neuroni označeni su slovom S. Oni kao ulaz primaju vrijednosti iz drugih neurona povezanih usmjerenim bridom. Vrijednost prijašnjeg neurona se množi vrijednošću koja pripada bridu koji ih spaja. Izlazni neuroni označeni su slovom I, oni nemaju izlazni brid već se njihova vrijednost uzima kao rezultat. Vrijednost svakog neurona je jednak zbroju vrijednosti koje dobiju na ulaz, te je taj zbroj proveden kroz odabrani izlazni operator.



Slika 1.1 Prikaz jednostavne neuronske mreže

U algoritmu jedinka je prikazana kao lista neurona i lista bridova (Slika 1.2). Svaki brid u sebi sadrži informacije o genotipu i fenotipu. Svaki brid u sebi ima označene početni i završni neuron tog brida. Također zapisana je i informacija o aktivnosti brida. Ove tri stavke u svakom bridu su dovoljne da se odredi kompletna struktura i izgled neuronske mreže(fenotip). Svaki brid također sadrži i svoju težinu te inovacijski broj. Sve ove stavke su genotip.



Slika 1.2 Prikaz genotipa i fenotipa u originalnom NEAT algoritmu

1.2.2. Mutacija i križanje

Mutacija

Mutacije su oblik aseksualne reprodukcije kod koje je potrebna samo jedna originalna jedinka. Mutacije mijenjaju strukturu ili težine unutar jedinke kako bi se generirala nova jedinka.

Moguće mutacije

- Promjena težine brida
- Deaktivacija brida
- Aktivacija brida
- Dodavanje nepostojećeg brida između 2 postojeća neurona
- Dodavanje neurona na jedan od aktivnih bridova
    - Deaktivacija postojećeg brida
    - Dodavanje neurona
    - Dodavanje 2 brida koji povezuju novi neuron s krajevima starog brida

Kod dodavanje bridova i neurona, stvorenim elementima dodjeljuju se inovacijski brojevi. na temelju njih je moguće praćenje novonastalih mutacija te prepoznavanje jednakih i sličnih jedinki nastalih drugačijim reprodukcijama.

Križanje

Križanje je oblik seksualne reprodukcije koji zahtjeva dvije jedinke.

Proces započinje kopiranjem bolje jedinke. Zatim se za svaki brid u kopiji pokušava pronaći odgovarajući brid s jednakim inovacijskim brojem u drugom roditelju. Zatim se slučajnim odabirom težina podesi ili na težinu drugog roditelja ili ostaje ista. Elementi drugog roditelja koji nisu pronađeni u kopiji mogu biti preneseni na kopiju, ako su dobrote oba roditelja podjednaka. Svaki brid koji je u bilo kojem od roditelja deaktiviran, ima određenu šansu da bude deaktiviran u kopiji.

1.2.3. Populacija i vrste jedinki

Populacija

Populacija jedinki započinje s predefiniranim brojem jedinki koji se održava istim kroz cijeli proces učenja jedinki. Prva generacija u populaciji sadrži samo jedinke koje imaju minimalnu moguću strukturu. Ulazni neuroni su direktno bridovima spojeni na izlazne neurone.

Kroz proces učenje ove jedinke mutiraju i razvijaju različite strukture. Kako bi se moglo kvalitetno birati jedinke koje se smiju reproducirati u novu generaciju, potrebno je sve jedinke u trenutnoj populaciji podijeliti na vrste.

Vrste

Vrsta je skupina jedinki, koje imaju toliko sličnu strukturu i težine, da se nijedna jedinka u grupi ne razlikuje dovoljno od ostalih da bi se smatrala značajnim napretkom u procesu traženja rješenja problema. Svaka jedinka se uspoređuje s već postojećim vrstama, te ako nije dovoljno slična nijednoj, stvara se nova vrsta u kojoj je ta jedinka jedini član.

U originalnom algoritmu razlika između dvaju jedinki se računa s pomoću dane jednadžbe (1). N predstavlja ukupni broj bridova u boljoj jedinki. E predstavlja broj bridova koji se nalaze isključivo u jednoj od dvije jedinke, a ne postoje bridovi s većim inovacijskim brojem u toj jedinki. D predstavlja broj bridova koji se nalaze isključivo u jednoj od dvije jedinke, ali isključivo nakon zadnjeg zajedničkog brida obiju jedinki. W predstavlja prosječnu razliku u težini zajedničkih bridova. C1 i C2 su konstante s pomoću kojih može se dati veću ili manju važnost razlikama u strukturi jedinki kod razvrstavanje u vrste. C3 određuje istu važnost za razlike težina bridova unutar jedinki iste strukture. Ovisno o tome prelazi li razlika između jedinki predefiniranu konstantu, jedinke pripadaju u istu vrstu.

$$\delta = \frac{C_1 D}{N} + \frac{C_2 D}{N} + C_3 W \qquad (1)$$

1.2.4. Selekcija

Kod odabira jedinki za reprodukciju u sljedeću generaciju, potrebno je zaštiti nove vrste koje su nastale nedavno, ali još nisu razvile zadovoljavajuća rješenja jer im nedostaje broj generacija koje su imale dosadašnje vrste. Jedinka s potencijalno zadovoljavajućom strukturom može imati gore rezultate od jedinke koja ima lošiju strukturu za rješavanje danog problema, ali je prilagodila težine kako bi maksimizirala svoju svoju učinkovitost. Iz tog razloga selekciju se radi zasebno unutar svake vrste. Pola svake vrste odbacuje se, dok se bolja polovica koristi za generiranje novih jedinki korištenjem prije navedenih metoda (1.2.2). Koliko se jedinki generira korištenjem koje vrste, određuje se dobrotom svake vrste. Koristi se eksplicitno dijeljenje dobrote. To je tehnika u evolucijskim algoritmima koja se koristi za očuvanje raznolikosti u populaciji jedinki tijekom procesa selekcije. Ova metoda je prvi put uvedena od strane Davida E. Goldberga i Jonathana Richardsona 1987. godine. Cilj dijeljenja dobrote je da spriječi dominaciju nekoliko najboljih jedinki i da potakne istraživanje šireg spektra mogućih rješenja[2]. Dobrota svake jedinke u vrsti se dijeli veličinom vrste te se zatim sve na taj način prilagođene dobrote zbroje u dijeljenu dobrotu vrste. Tako određena dobrota vrste se dijeli zbrojem prilagođenih dobrota svih vrsta, kako bi se dobio postotak populacije koji će se popuniti jedinkama iz te vrste.

2. Problem optimizacije logičkih sklopova

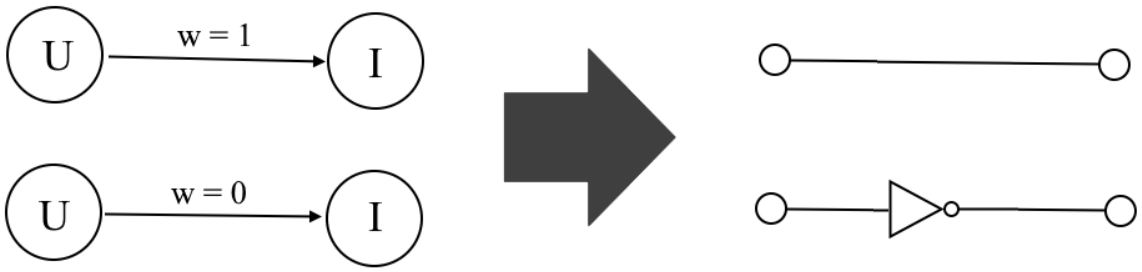
Cilj algoritma je razviti neuronsku mrežu koja predstavlja logički sklop. Taj logički sklop mora za svaku moguću kombinaciju ulaza dati odgovarajući izlaz. Ulaz je niz nula ili jedinica, dok je izlaz isključivo jedna nula ili jedna jedinica. Algoritam će kao svoj ulazni podatak dobiti broj ulaza u željeni logički sklop i listu vrijednosti koje predstavljaju rezultate za svaku od mogućih kombinacija nula i jedinica na ulazu sklopa.

2.1. Mapiranje rješenja

Svaki logički sklop moguće je prikazati kao neuronsku mrežu na nekoliko načina. Jedna od prvih prepreka ovog problema je mapiranje dobivenih neuronskih mreža u logičke funkcije, kako bismo mogli ocijeniti uspješnost algoritma. Način na koji je odlučeno mapirati rješenja, omogućuje vrlo jasno iščitavanje same funkcije golim okom. Kao jedine dozvoljene elemente u sklopovima odabrana su AND, OR i NOT vrata.

2.1.1. NOT vrata

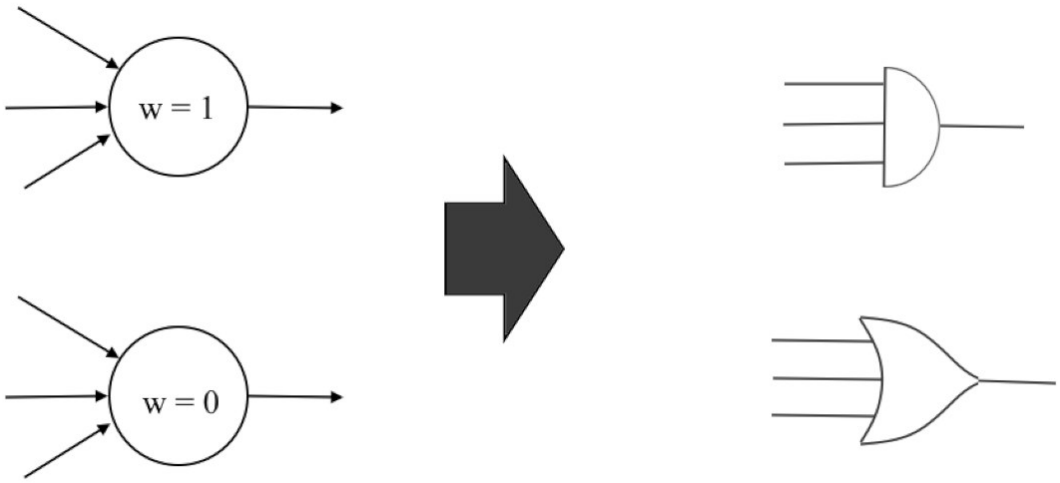
Zbog sličnosti u prikazu kod neuronskih mreža i logičkih sklopova, NOT vrata su prikazana vrijednostima na bridovima neuronske mreže. Težina brida 0 označava negaciju vrijednosti koja bi trebala proći tim bridom, dok težina 1 predstavlja prijenos vrijednosti bez promjene (Slika 2.1).



Slika 2.1 Mapiranje NOT vrata kod neuronske mreže

2.1.2. AND i OR vrata

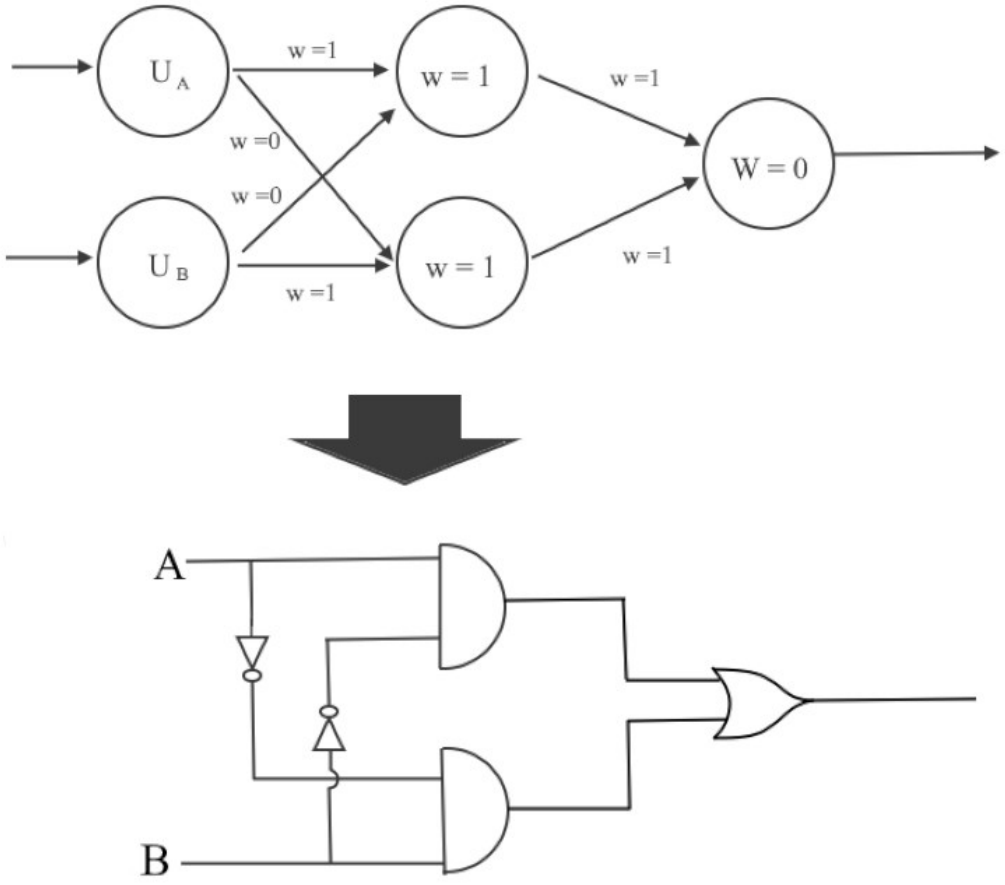
Zbog višestrukog ulaza i jednog izlaza svaki neuron je savršen za predstavljanje ili OR ili AND sklopa(Slika 2.2). Ako neuron ima pristranost težine 1, tretira se kao AND vrata, dok se neurone s pristranost 0 smatra OR vratima. Sama implementacija ovih sklopova objašnjena je u kasnijem poglavlju (3.1.3).



Slika 2.2 Mapiranje AND i OR vrata kod neuronske mreže

2.1.3. Primjer

Kao primjer u nastavku (Slika 2.3) se nalazi sklop XOR, prikazan s pomoću osnovnih vrata u obliku (NOT(A) AND B) OR (NOT(B) AND A). Neuronska mreža sadrži dva skrivena neurona koji predstavljaju AND vrata i jedan izlazni neuron koji predstavlja OR vrata. Na ovom primjeru je vrlo lagano za uočiti sličnosti neuronske mreže i samog sklopa koji mreža predstavlja. Također, mapiranjem AND i OR vrata na neurone, a NOT vrata na bridove, omogućeno je stvaranje struktura koje se vrlo lagano mogu prilagođavati potrebama zadatka. Ako je potrebno negirati određeni izlaz neurona, nema potrebe za strukturnom mutacijom, već se rezultat može postići mutacijom težine, koja se odvija češće. Isti princip omogućuje i jednostavno mijenjanje vrata AND u vrata OR i obrnuto. Ove karakteristike uvelike pomažu kod implementacije samog algoritma i podešavanja parametara.



Slika 2.3 Prikaz mapiranja potpune neuronske mreže u logički sklop

3. Implementacija

Implementacija algoritma objavljena je u programskom jeziku Java 21, dok se za obradu i prikaz rezultata koriste programski jezici Java 21 i Python 3. Zbog same specifičnosti problema koji bi algoritam trebao rješavati, korišteni se algoritam u puno dijelova razlikuje od originalnog. U nastavku je opisana implementacija koja je korištena kod dobivanja funkcionalnih rješenja i razne varijacije dijelova algoritma koji su testirane ali na samom kraju nisu korištene zbog svojih nedostataka. Svaka varijacija ima svoju oznaku, te je njezin razlog nekorištenja uz ostala opažanja zapisan u tablicu (Tablica 5.1) na kraju poglavlja.

3.1. Model neuronske mreže

```
public class Network {
    private double fitness;
    private HashMap<Integer, Edge> edges;
    private HashMap<Integer, Node> nonOutputNodes;
    private Node outputNode;
    private Integer inputDimension;

    private static HashMap<Pair, Integer> edgeInnovations;
    private static HashMap<Integer, Trio> nodeInnovations;

    private static Integer nextEdgeInnovationNumber;
    private static Integer nextNodeInnovationNumber;
...
}
```

Svaka neuronska mreža prikazana je kao objekt koji sadrži mapu bridova, izlazni neuron i mapu ostalih neurona. Indeksi u obje mape odgovaraju inovacijskim brojevima bridova i neurona. Također za svaku neuronsku mrežu pamti se njena dobrota i dimenzija ulaza koji podržava. Klasa sama po sebi ima statičke elemente koji usklađuju inovacijske brojeve kod svih do sad stvorenih neuronskih mreža. Inovacije kod bridova se pamte na temelju inovacijskih brojeva neurona između kojih su stvorene. Inovacije kod neurona se pamte na temelju brida u koji smo stavili taj neuron, također na ovaj način moguće je i saznati inovacijske brojeve novostvorenih bridova kod stvaranja novih neurona. Na ovaj način moguće je osigurati da, kojim god slijedom mutacija se razvije struktura neuronske mreže,

ako već postoji neuronska mreža takve strukture, one se tretiraju kao ista struktura, neovisno o načinu na koji su se razvile.

3.1.1. Brid

```
public class Edge{
    private Node start;
    private Node finish;
    private double weight;
    private int innovationNumber;
    private boolean enabled;

    private static Double absoluteTreshold = 0.35;
...
}
```

Svaki brid ima zabilježena oba neurona koja povezuje, vlastitu težina koja određuje hoće li se tretirati kao negacija ili obični prijenos vrijednosti. Strukturno su bitni inovacijski marker i oznaka aktivnosti brida. Statička varijabla *absoluteTreshold* je potrebna kasnije kod samog izračuna vrijednosti (3.1.3).

3.1.2. Neuron

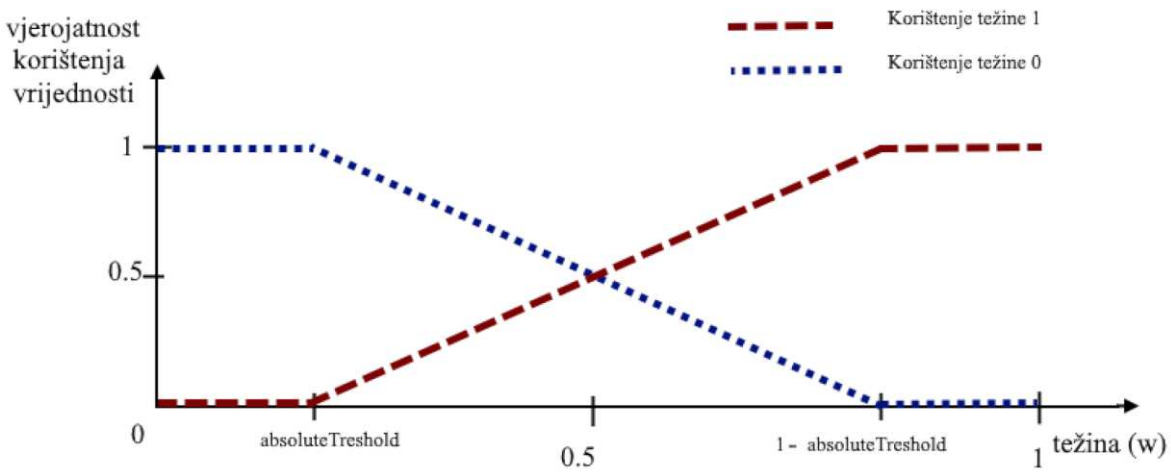
```
public class Node {
    private double biasWeight;
    private double edgeWeightForOutputFromOutputNode;
    private int innovationNumber;
    private Type type;
    private List<Edge> inputEdges = new ArrayList<>();

    private static Double absoluteTreshold = 0.35;
...
}
```

Neuroni su prikazani objektima klase Node. Svaki objekt ima vlastiti inovacijski broj i listu bridova koji ulaze u njega. U ovu listu su uključeni svi bridovi, neovisno o svojoj aktivnosti. Tip neurona može biti INPUT,HIDDEN ili OUTPUT, te se ovisno o tome na taj način izračunava njegova vrijednost. Iznos pristranosti određuje vrstu vrata koja neuron predstavlja. Jedan od problema prikaza NOT vrata bridovima je nemogućnost negiranja cijelog izraza, jer nakon izlaznog neurona, ne postoji brid. Ovaj nedostatak ispravljen je

dodatnim poljem *edgeWeightForOutputFromOutputNode*, koje dolazi u obzir samo kad se računa vrijednost izlaznog neurona. Ovo polje ima isti funkcionalnost kao i težina ostalih bridova, te omogućuje negaciju cijelog sklopa, bez dodavanja novog nepotrebnog neurona i brida prije izlaznog neurona. Statička varijabla *absoluteTreshold* je potrebna kasnije kod samog izračuna vrijednosti.

3.1.3. Izračun vrijednosti u neuronskoj mreži

Izračun samog izlaza iz neuronske mreže ili logičkog sklopa je poprilično jednostavan proces. Nažalost ovaj proces nije prikladan za učenje u genetskom algoritmu. S obzirom na to da svaka težina u neuronskoj mreži ima samo dvije mogućnosti (NOT vrata ili ništa, AND ili OR vrata), mijenjanje vrijednosti težina bi se svelo na apsolutnu promjenu u funkcionalnosti logičkog sklopa. Takve promjene svele bi učenje unutar jednake strukture na pogađanje vrijednosti dok se ne pojavi zadovoljavajuća kombinacija. Kako bi se omogućilo postepeno učenje unutar istih struktura, uvodi se novi operator koji se koristi kod odabira akcije na bridu i kod računanja vrijednosti neurona. Operator dodaje postepeni prijelaz između dvije moguće akcije predstavljen brojčanom vrijednošću između 0 i 1 (težina). Ako je vrijednost težine manja od vrijednosti varijable *absoluteTreshold.* vrijednost težine se uvijek tretira kao 0. Ako je vrijednost veća od *(1 - absoluteTreshold)*, težina će se uvijek tretirati kao 1. Sve vrijednosti između *absoluteTreshold* i *(1 - absoluteTreshold),* predstavljaju vjerojatnost da težina tretira na odabrani način. Vrijednosti bliže *absoluteTreshold* imaju veću šansu da se tretiraju kao 0, ali mogu i dalje biti 1 (Slika 3.1). Na ovaj način jedinka koja ima težinu 0, može putem mutacije pomaknuti svoju vrijednost bliže vrijednosti 1. Promjena u dobroti će biti malena, i samim time neće u potpunosti poremetiti dijeljenu dobrotu vrste, kao kod potpune promjene na 1. Jedinke na ovaj način zadržavaju priliku da promijene vrijednost svojih težina na suprotnu vrijednost kroz nekoliko generacija, ali ne gube stabilnost dobrote vrste. U trenutnom algoritmu varijabla *absoluteTreshold* je postavljena na 0.35. Tijekom razvoja algoritma isprobane su vrijednosti 0.1 (VA001), 0.4 (VA002) i 0 (VA003).



Slika 3.1 Grafički prikaz vjerojatnost korištenja vrijednosti 0/1 u ovisnosti o težini

Izračun vrijednosti koja se pojavljuje na kraju bridova stoga izgleda ovako:

1) Spremamo vrijednost ulaznog neurona
2) Prethodno opisanim postupkom određuje se korištenje vrijednosti 0 ili 1
    a. Ako je vrijednost 0, negiramo vrijednost ulaznog neurona `Math.abs(result-1)`
3) Spremljenu vrijednost vraćamo

Izračun izlaznih vrijednosti započinje pokretanjem rekurzivne funkcije koje izračunava vrijednost neurona. Kod pokretanja izračuna funkciji se predaje izlazni neuron. Vrijednost neurona računa se na sljedeći način:

1) Provjeravamo tip neurona
    a. Ako je neuron ulazni, funkcija automatski vraća ulaznu vrijednost
2) Lista ulaznih bridova se filtrira kako bi ostali samo aktivni bridovi
3) Zbrajamo vrijednosti koje vrati računanje vrijednosti nad preostalim bridovima
4) Na temelju težine pristranosti, određuje se vrsta vrata koja predstavlja neuron
    a. OR vrata, sprema se rezultat uvjeta: zbroj > 0
    b. AND vrata, sprema se rezultat uvjeta: zbroj = broj aktivnih bridova
5) Ako je u pitanju izlazni neuron, spremljenu vrijednost provedemo kroz postupak jednak izračunu brida, koristeći vrijednost *edgeWeightForOutputFromOutputNode*
6) Vraćamo spremljenu vrijednost

3.1.4. Funkcija dobrote

Dodavanjem nedeterminističkih operatora u neuronsku mrežu, više nije dovoljno provjeriti samo svaku moguću kombinaciju ulaza. Kako bismo znali koje jedinke su bliže željenom rješenju, potrebno je svaku kombinaciju ulaza provjeriti u neuronskoj mreži više puta. Trenutni algoritam provjerava svaku kombinaciju 100 puta. Kod logičke funkcije koja ima 2 ulazna parametra, i testovi se ponavljaju 100 puta, ukupni broj izračuna neuronske mreže je 400. što nam određuje maksimalnu moguću dobrotu za vrijeme ovog učenja. Također za vrijeme razvoja algoritma, testirana je verzija koja ponavlja testove 200 puta (VA004)

3.2. Vrste

Podjela jedinki u vrste omogućava nam bolju kontrolu nad reprodukcijom jedinki. Kako bi podijelili jedinke, potrebno je izračunati razliku između njih. Tijekom razvoja korišteno je nekoliko različitih načina izračuna. Kod podjele jedinki u vrste, jedinka se uspoređuje s najboljom do sad pronađenom jedinkom u vrsti. Ako nije pronađena strukturno bitna razlika, jedinka se dodaje u vrstu. Ako ne postoji zadovoljavajuća vrsta, nova vrsta se stvara s trenutnom jedinkom kao najboljom. Kod dodavanja jedinke u vrstu, njezina dobrota se uspoređuje s dobrotom najbolje do sad pronađene jedinke u toj vrsti. Ako je pronađena nova najbolja jedinka, zabilježi se njezina dobrota, i generaciju u kojoj je napredak ostvaren. Ova informacija je bitna, jer je u bilo kojem trenutku moguće provjeriti stagnira li napredak vrste. Vrsta i njezina najbolja jedinka ostaju zabilježene do kraja izvođenja programa, čak iako u trenutnoj populaciji nema nijedne jedinke koja pripada toj vrsti. Ako se u vrsti dobrota najbolje jedinke nije povećala u zadnjih 30 generacija, vrsta više nije namijenjena za repopulacija. Ovaj limit se povećava na 50 generacija, ako dobrota najbolje jedinke veća od određenog izraza. U ovom radu se predlaže:

$$f_{najbolja\ jedinka} > \left(2^{dimenzija\ ulaza} - 1\right) * broj\ ponavljanja\ testova \qquad (2)$$

Ovaj izraz je dobiven eksperimentalno te je kasnije, matematičkim operacijama njegov oblik sveden na prikazani. Vidljivo je da jedinke, čija logička funkcija zadovoljava sve osim jedne kombinacije ulaza, ne zadovoljavaju izraz. Što bi značilo da samo jedinke s izrazito visokom dobrotom i barem jednom točno određenom vrijednošću za svaku moguću kombinaciju ulaza, imaju limit od 50

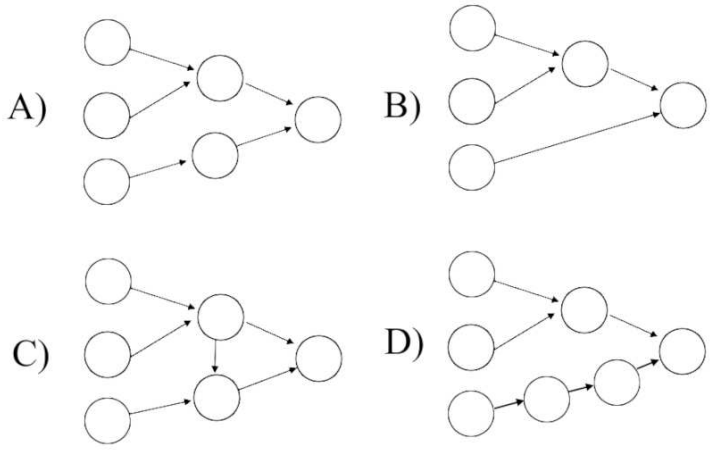
generacija. Također, u trenutku kada vrsta stvori predstavnika s maksimalnom mogućom dobrotom, dozvoljava joj se reprodukcija u svim daljnjim generacijama.

3.2.1. Korištena funkcija razlike

Razlika dviju jedinki računa se tražeći aktivne bridove i neurone, koji se nalaze u isključivo jednoj jedinki. Broj ovakvih elemenata je korišten kao razlika dvaju jedinki. Kako bi se smanjio broj nepotrebnih vrsta koje imaju strukturno različite jedinke ali svojom strukturom ne pridonose razlici u ponašanju, dodan je naknadni uvjet. Bridovi i neuroni koji tvore lanac bez dodatnih ulaza, tretiraju se kao jedan brid. Ovu usporedbu je moguće raditi, jer znamo način na koji su nastali ovi lanci. Prateći inovacijske brojeve, moguće je ovakve inovacije izuzeti iz brojenja različitih elemenata. Primjeri ovakvih lanaca prikazani su na kraju odjeljka (Slika 3.2). Jedinke prikazane pod slovima A,B,D bi sve pripadale istoj vrsti. Jedinka pod slovom C ne bih pripadala toj vrsti te bi označava početak nove vrste.

Kroz proces razvoja ovog algoritma, testirane su i ove opcije:

- Originalno računanje razlike (VA005) [1]
- Suma različitih elemenata (VA006)
- Usporedba na temelju sklopa koji predstavlja neuronska mreža (VA007)



Slika 3.2 Prikaz neuronskih mreža koje mogu nastati u algoritmu

3.3. Reproduciranje

Reproduciranje jedinki se odvija u dva koraka, aseksualno i seksualno. Također, dio buduće populacije se puni već postojećim jedinkama. U sljedećih nekoliko poglavlja opisan je potpuni postupak popunjavanja iduće generacije. Kod stvaranja jedinke, ona se odmah evaluira na prethodno spomenuti način (3.1.4), te se zabilježi njezina dobrota.

3.3.1. Elitizam

Elitizam je postupak prenošenja najboljih jedinki iz trenutne populacije u sljedeću. U algoritmu je podešeno da samo vrste koje su veće od predodređene veličine (10), smiju prenijeti svoju najbolju jedinku u sljedeću generaciju. Ovime je spriječeno da relativno mlade vrste prebacuju jednu jedinku više, kad znamo da još nisu imale vremena razviti dobre jedinke, ili su imale dovoljno vremena ali su stagnirale u svom razvoju. Vrste koje nisu namijenjene za repopulaciju također ne mogu ući u elitizam. Također, kod razvoja algoritma testirane su još 3 alternative vezane uz elitizam:

- Provjera veličine vrste i minimalna dobrota najbolje jedinke (VA008)
- Bez elitizma (VA009)
- Dodavanje najbolje jedinke iz svake vrste (VA010)

3.3.2. Mutacija

Mutacija uvijek stvara isti broj jedinki za slijedeću generaciju, koji je određen postotkom ukupne populacije populacije. Trenutni algoritam koristi vrijednost 0.45 . Svaka vrsta će generirati sljedeći broj jedinki:

$$\frac{0.45 * f_{dijeljeni\ fitnes\ vrste}}{\sum_i f_{dijeljeni\ fitnes\ vrste}(vrsta_i)} \tag{3}$$

Na taj način se vrstama koje imaju lošiju dobrotu omogućava da imaju vrstu dovoljne veličine kako bi se desio napredak u sljedećoj generaciji. Unutar vrste, jedinka čija će se kopija mutirati i dodati u sljedeću generaciju bira se između bolje polovice unutar vrste. Unutar bolje polovice, jedinke s boljom dobrotom imaju proporcionalno veću šansu biti odabrane. Nakon odabira jedinke, ona se kopira, te se nad njom obavlja jedna od mutacija. Svaka mutacija ima svoju vjerojatnost izvođenja i nikad se ne obavljaju dvije mutacije na

istoj jedinki. Ove vjerojatnosti moguće je promijeniti. U kodu su korištene sljedeće varijable i vrijednosti.

Tablica 3.1 Vjerojatnost odabira aseksualnih mutacija

Ime varijable	Trenutna vrijednost	VA011	VA012
chanceForWeightShift	0.999	0.991	0.91
chanceForAddingNode	0.0003	0.003	0.03
chanceForAddingEdge	0.0004	0.003	0.03
chanceForDisablingEdge	0.0004	0.003	0.03

Šanse za promjenu težine bridova i pristranosti u neuronu su znatno veće od šansi za strukturnu mutaciju, kako bi se spriječilo stvaranje prevelikog broja vrsta, prije nego li su trenutne vrste dobile priliku za napraviti napredak. Prerano stvaranje velikog broja vrsta ograničilo bi broj jedinki u svakoj vrsti za sljedeću generaciju, te bi se samim time šansa za kreiranjem novih boljih jedinki iste vrste smanjila. Ako nije moguće dodati novi brid u neuronsku mrežu, u nju se dodaje novi neuron. Na ovaj način, u prvih nekoliko generacija, jedinke osnovnih struktura, imaju duplo veće šanse za stvaraju novog čvora. Ova metoda omogućuje da se u početku učenja ubrza stvaranje novih vrsta, bez da se mijenjaju šanse u kasnijim generacijama. Nakon odrađene mutacije provjerava se jesu li ulazi povezani s izlazom neuronske mreže. Ako nisu, u populaciju se dodaju nova osnovna neuronska mreža bez skrivenih neurona.

3.3.3. Deaktivacija bridova

Svaki aktivirani brid ima jednaku šansu biti aktiviran. Bridovi koji su jedini ulaz u svoj završni neuron ne mogu biti deaktivirani.

3.3.4. Promjena težine

Mutacija s najvećom vjerojatnošću događanja je promjena težine. Težina se mijenja na svim bridovima i neuronima. Svaka težina ima 3 opcije, čije vjerojatnosti se mogu promijeniti u datoteci *aSexualReproductionValues.java.* Varijabla *chanceForWeightShift_small* određuje vjerojatnost da se odabrana težine promjeni za slučajnu vrijednost u rasponu

[- (1 / *weigthShiftStrength*) , (1 / *weigthShiftStrength*)] . Varijabla *chanceForWeightShift_random* određuje vjerojatnost da se težina postavi na slučajnu vrijednost u rasponu [0 , 1]. Ako nije odabrana nijedna od te dvije opcije, težina se ne mijenja. Ako se bilo koja vrijednost postavi izvan raspona [0,1], postavljena je na bližu od dvije granice. U kodu su korištene sljedeće varijable potrebne za podešavanje težina. Kod jedinki koje su dobrotom prošle vrijednost `(2^(dimenzija ulaza)-1)*broj ponavljanja testova`, mutiranje vrijednosti na nasumičnu vrijednost je onemogućeno, kako bi se dala veća vjerojatnost širenju dobrih vrijednosti kroz vrstu.

Tablica 3.2 Vjerojatnosti potrebne za podešavanje težine

Ime varijable	Trenutna vrijednost	VA013	VA014	VA015	VA016
chanceForWeightShift_small	0.40	0.40	0.40	0.50	0.40
chanceForWeightShift_random	0.40	0.40	0.40	0.50	0.10
weigthShiftSrength	4.4	8	2.7	24.4	4.4

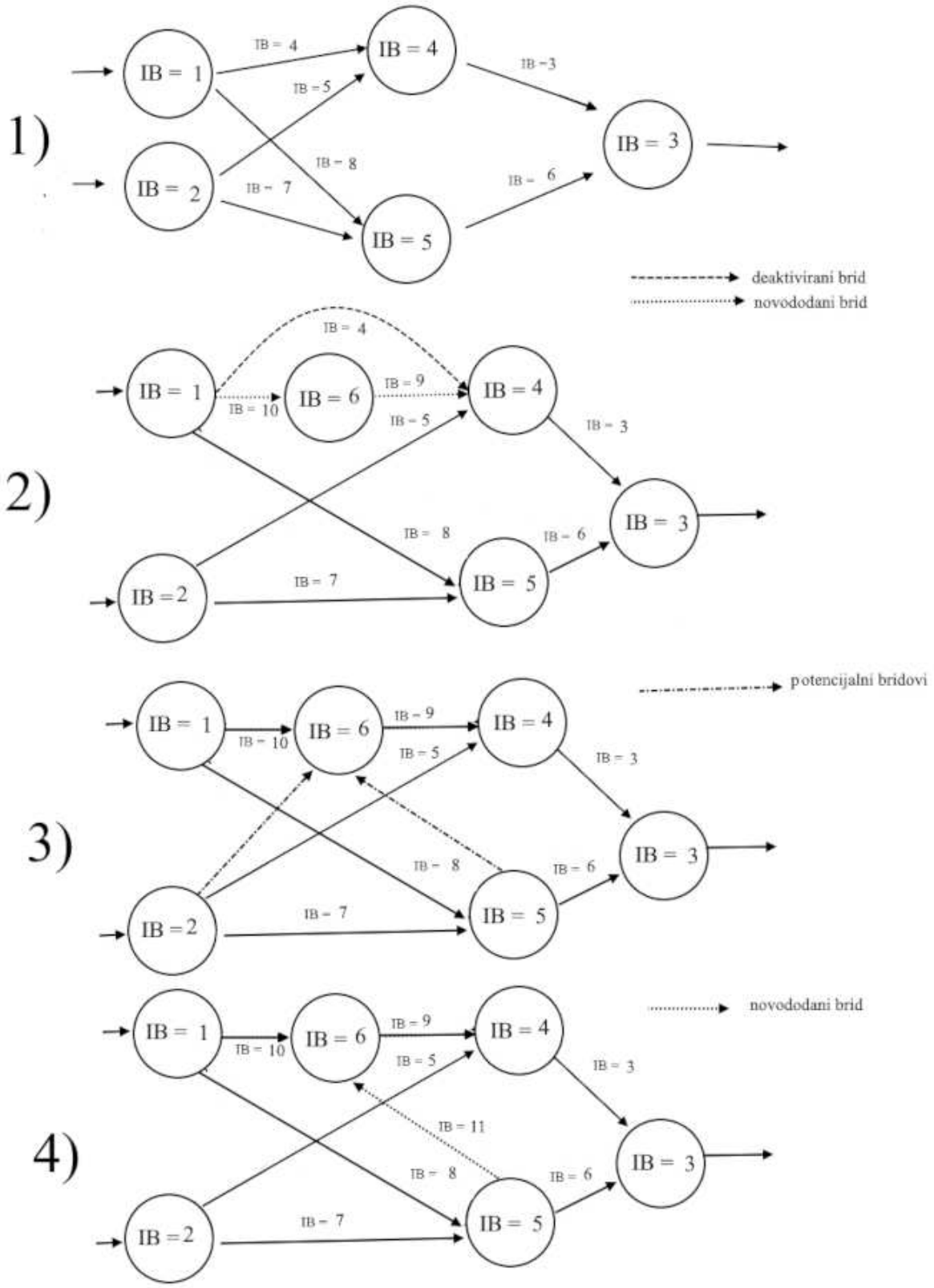
3.3.5. Dodavanje novog brida

Kako bi se dodao novi brid, potrebno je odabrati dva neurona, početni i završni neuron. Početni neuron ne smije biti tipa OUTPUT, dok završni neuron ne smije biti tipa INPUT. Također zabranjeno je dodavanje bridova koji bi stvorili cikluse u neuronskoj mreži. Bridovi koji se već koriste u neuronskoj mreži ne mogu se dodati opet. Kako bi se ovi mogući bridovi pronašli, algoritam prvo isprobava svaku kombinaciju neurona, te nakon toga iz liste mogućih bridova odabire jedan koji će dodati. Odabrani brid kao početnu težinu ima vrijednost 0.5. Dodavanje brida odvija se istim principom kao i u originalnom NEAT algoritmu. Tijekom razvoja algoritma isprobano je još nekoliko varijacija dodavanja brida:

- Reaktivacija deaktiviranih bridova (VA017)
- Promjena početne težine dodatnog brida na 0 ili 1 (VA018)

3.3.6. Dodavanje novog neurona

Dodavanje novog neurona moguće je samo na aktivnim bridovima. Nakon filtriranja bridova po aktivnosti, algoritam ih sortira po inovacijskim brojevima u silaznom redoslijedu. Zatim je svakom od tih bridova dodijeljena vjerojatnost da se na njemu stvori novi neuron. Vjerojatnosti se određuju geometrijskim redom s koeficijentom r, što znači da svaki sljedeći brid ima r puta veću šansu od prošlog brida. Na ovaj način je moguće dati poticaj algoritmu da istražuje rješenja u širinu ili dubinu. Kod ravnomjerne distribucije vjerojatnosti, svakim dodavanjem brida, šansa za istraživanje rješenja koja vode u dubinu se povećava, jer imamo jedan brid više u dijelu neuronske mreže koji smo upravo mutirali. Kako dodavanje neurona, povlači sa sobom i dodavanje tri nova brida, to znači da ćemo sljedeći neuron imati još veću šansu staviti u njegovu okolinu. U trenutnom algoritmu korišten je r s vrijednošću 2.7. Tijekom razvoja algoritma, testirane su i vrijednosti 1 (VA019), 1.5 (VA020), 5 (VA021). Sam proces dodavanja neurona razlikuje se od originalnog algoritma u tome da se uz dva brida, koji povezuju početni i završni neuron zamijenjenog brida s novim neuronom, dodaje i treći brid. S obzirom na to da neuroni predstavljaju logička vrata AND ili OR, neuron sa samo jednim izlazom ne predstavlja ništa, to je samo prijenos dobivene vrijednosti, neovisno o vrsti vrata koja predstavlja. Kako bi se ubrzao proces učenja, umjesto da algoritam čeka da se slučajno stvori brid koji spaja ovaj neuron s nekim drugim, algoritam će ga stvoriti na početku. Ovdje vrijede ista pravila za dodavanje bridova kao i kod dodavanja nasumičnog brida. Algoritam provjerava sve ostale neurone, te stvara listu onih koji mogu predati svoju vrijednost u novi neuron. Svi ovako odabrani neuroni imaju jednaku vjerojatnost da budu izabrani za početak novog brida. Primjer ovog procesa nalazi se na slici niže (Slika 3.3). Oznaka IB označava inovacijski broj. U ovom primjeru prikazano je dodavanje novog neurona na brid s oznakom IB = 4. U trećem koraku prikazani su svi mogući bridovi koji se mogu dodati kao drugi ulaz u novi neuron. Težina ulaznog brida postavlja se na težinu zamijenjenog brida. Izlazni brid se postavlja na težinu 0.5, a dodatni ulazni brid se postavlja na težinu 0.5. Kod razvoja algoritma, testirano je postavljanje težine 0.5 na prvi ulazni brid (VA022).



Slika 3.3 Ilustrirani postupak dodavanja neurona

3.3.7. Križanje jedinki

Križanje jedinki je seksualni dio reprodukcije za koji su potrebne dvije jedinke. Križanje jedinki generira toliko jedinki, koliko je potrebno da se nakon mutacije i elitizma popuni sljedeća generacija. Postoje dvije vrste križanja, križanje unutar iste vrste i križanje dvije jedinke različitih vrsta. Trenutni algoritam koristi samo križanje unutar jedne vrste, iako je križanje između različitih vrsta testirano tijekom razvoja algoritma (VA023). Svaka vrsta generira dio jedinki predviđenih za dobivanje križanjem. Taj broj je određen udjelom dijeljene dobrote vrste u odnosu na zbroj svih dijeljenih dobrota vrsta. Križanje dvije jedinke unutar iste vrste započinje uzimanjem bolje polovice vrste. Unutar te polovice svaka jedinka ima šansu da bude odabrana proporcionalno svojoj dobroti u usporedbi s ostalim jedinkama u boljoj polovici. Druga jedinka odabrana je slučajnim odabirom, kod kojeg sve jedinke u vrsti imaju jednaku šansu biti odabrane. Postupak se nastavlja kopiranjem bolje jedinke. Zatim se za svaki brid i neuron koji se nalazi u kopiranoj jedinki pronalazi njegov odgovarajući par u goroj jedinki od početne dvije. Za svaku težinu u kopiji postoji 0.50 vjerojatnost da će se pretvoriti u odgovarajuću vrijednost iz goreg roditelja. Također za svaki brid koji je deaktiviran u gorem roditelju, postoji 0.2 šansa da će se biti deaktiviran i u kopiji. Ako bi se ovom deaktivacijom u potpunosti odvojili ulazni neuroni od izlaznog neurona, brid ostaje aktivan.

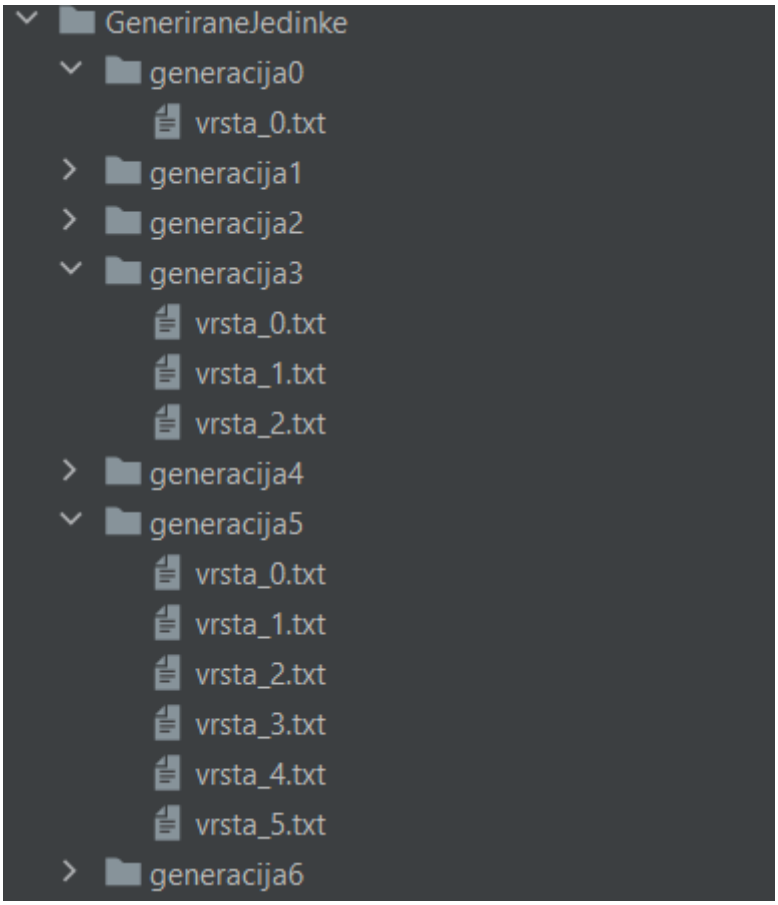
3.3.8. Postavljanje sljedeće generacije

Nakon popunjavanja populacije za sljedeću generaciju, potrebno je isprazniti sve vrste, kako bi se u sljedećoj iteraciji mogla ponovno računati dijeljena dobrota. Koristeći isti objekt vrste, osigurano je da se napredak vrste može lagano pratiti kroz cijeli tok algoritma. Podaci o vrsti koji ostaju zapamćeni su: najbolja do sad pronađena vrsta i broj generacija od posljednjeg napretka vrste. Nakon pražnjenja vrsta, slijedi prebacivanje svih novostvorenih jedinki u trenutnu populaciju. Zatim se postupak ponavlja: razdvajanje u vrste, podešavanje fitnesa, reprodukcija…

3.4. Spremanje rješenja

Algoritam sprema podatke o razvijenim jedinkama i vrstama u .txt datoteke, koje se kasnije mogu koristiti za analizi rješenja. Algoritam će prvo stvoriti mapu „Generirane jedinke.“ Zatim će za svaku generaciju stvoriti mapu „generacija_<broj generacije>“. U toj mapi će

se nalaziti datoteke naziva „vrsta_<broj vrste>.txt“. Svaka od ovih datoteka u sebi sadrži broj jedinki koje se nalaze u vrsti, broj generacija od posljednje dobre inovacije, logički izraz koji predstavlja najbolja jedinka u vrsti i dobrota najbolje jedinke u vrsti. Primjer ovakve strukture moguće je vidjeti na niže prikazanoj slici (Slika 3.4) Algoritam je moguće podesiti da se zaustavi nakon određenog broja generacija, nakon što je pronađena jedinka s više od 0.95 maksimalne dobrote ili bez uvjeta. Algoritam će obavijestiti korisnika o broju generacija od početka učenja, također korisnik će biti obaviješten svaki puta kada se završi učenje generacije u kojoj je neka jedinka imala veću dobrotu od 0.95 maksimalne dobrote ili kada je ostvarena maksimalna dobrota.



Slika 3.4 Prikaz strukture spremljenih rezultata

4. Rezultati

Kako bi se moglo proučiti ponašanje algoritma, odabrana su 4 problema različitih težina. Promatrajući ponašanje jedinki i algoritma kod rješavanja ovih problema određeni su parametri i metode koje se koriste u trenutnom algoritmu. U nastavku prikazani rezultati odnose se na jedinke dobivene korištenjem tih postavki.

4.1. A OR notB

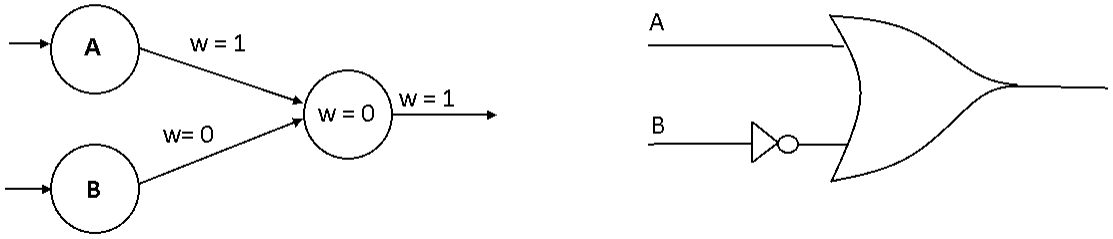
Problem najmanje težine na kojemu je odlučeno testirati osnovnu funkcionalnost algoritma je logička funkcija A OR notB s logičkom tablicom prikazanom u nastavku (Tablica 4.1). Očekuje se da svaka varijacija algoritma može pronaći ovo rješenje u relativno malom broju generacija, jer je struktura potrebna za rješavanje ovog problema, istovjetna početnoj strukturi jedinki u prvoj generaciji. Jedine potrebne promjene su težine na bridovima i težine pristranosti u neuronima. Ovo jedini problem koji je svaka varijacija algoritma uspjela riješiti.

Tablica 4.1 Ulazi i odgovarajući izlazi logičke funkcije prvog problema

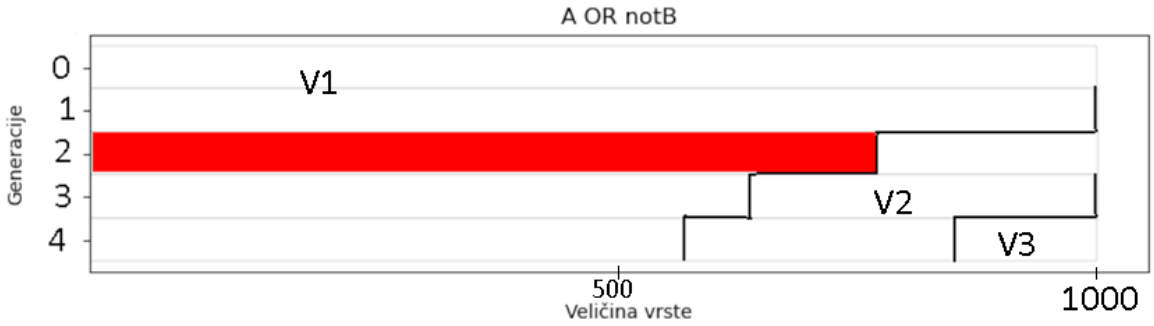
A	B	A OR notB
0	0	1
0	1	0
1	0	1
1	1	1

Trenutni algoritam unutar nekoliko generacija pronalazi rješenje koje ima maksimalnu moguću dobrotu. Na slici ispod (Slika 4.1) nalazi se grafički prikaz neuronske mreže tog rješenja i njegova reprezentacija kao logički sklop. Algoritam je uspio pronaći rješenje koje koristi najmanji mogući broj logičkih vrata. Opaženo je da se kod ovog rješenja, ali i rješenja ostalih problema, završne vrijednosti težina uvijek nalaze unutar *absoluteTreshold* udaljenosti od vrijednosti 0 ili 1. To znači da unatoč postojanju vrijednosti koje daju nedeterminističke rezultate tijekom samog procesa učenja, algoritam na putu stvaranja najboljeg rješenja uspijeva podesiti težine do te razine da se neuronska mreža ponaša potpuno deterministički. Razvoj vrsta i njihovog napretka je zabilježen u datoteke s informacijama o njima. Na temelju tih datoteka, korištene su programske skripte i ručna

obrada, kako bi se vizualizirao napredak populacije (Slika 4.2). Ova vrsta grafa prikazuje veličinu svake vrste u populaciji kroz generacije. Kod stvaranja nove vrste, ona se dodaje na desni kraj retka. Svaki redak prikazuje jednu generaciju, dok širina polja u tom retku prikazuje veličinu vrste u tom trenutku. Crvenom (tamno sivom) bojom je prikazano prvo pojavljivanje savršene jedinke unutar vrste. Na samom desnom kraju retka koji predstavlja generaciju 1, vidljivija je okomita linija koja odvaja vrste. To znači da je već u toj generaciji započeta nova vrsta V2. Njezina širina je jedva vidljiva, jer velik dio vrsta započinje samo s jednom ili dvije jedinke.



Slika 4.1 Prikaz neuronske mreže prvog rješenja i njegova reprezentacija kao logički sklop



Slika 4.2 Grafički prikaz veličina i dobrota vrsta kroz generacije za prvi problem

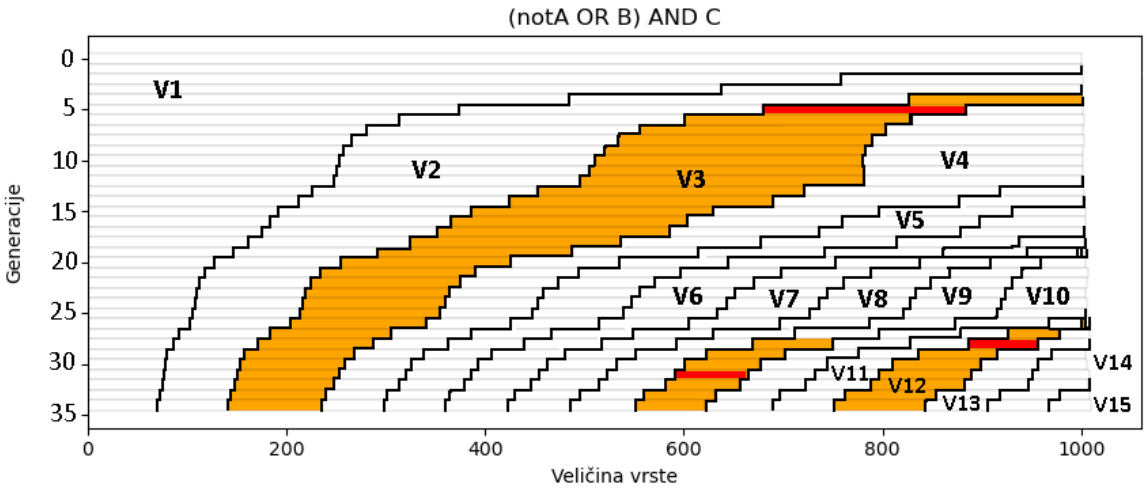
4.2. (notA OR B) AND C

Drugi problem na kojemu je odlučeno testirati algoritam je logička funkcija (notA OR B) AND C s logičkom tablicom prikazanom u nastavku (Tablica 4.2). Ovaj problem zahtjeva 3 ulazna neurona i dodavanje barem jednog neurona kako bi se dostigla jedinka s maksimalnom mogućom dobrotom.

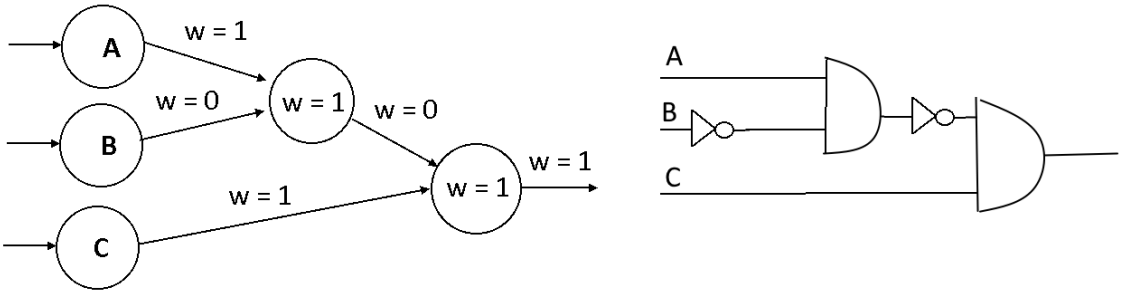
Tablica 4.2 Ulazi i odgovarajući izlazi logičke funkcije drugog problema

A	B	C	(notA OR B) AND C
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Na grafu veličine vrsta (Slika 4.3) vidljive su 3 vrste koje su razvile jedinke s maksimalnim mogućim dobrotama. To su vrste V3,V9 i V12. narančastom(svijetlo sivom) bojom označene su vrste koje u toj generaciji imaju dobrotu najbolje jedinke veću od medijana svih vrsta za barem jednu standardnu devijaciju. Trenutni algoritam dolazi do rješenja s maksimalnom dobrotom već u generaciji 6. Ovo rješenje nalazi se u vrsti V3. Neuronska mreža ovog rješenja ima minimalnu moguću strukturu, iako logički sklop koji ona predstavlja nije najjednostavnija moguća struktura (Slika 4.4). Generirana struktura je not(A AND notB) AND C, što skraćivanjem odgovara traženoj logičkoj funkciji. Ova vrsta u malom broju generacija pronalazi svoje najbolje rješenje, ali joj je potrebno 20 generacija kako bi se primijetila značajna razlika u veličini te vrste. Razlog tome je taj što se veličina vrste dodjeljuje na temelju dijeljene dobrote. Tijekom tih 20 generacija, gotovo sve jedinke u vrsti V3 dostignu maksimalnu dobrotu, te je oko generacije 25 vidljivo da je vrsta V3 duplo veća od okolnih vrsti. Vrste V9 i V12 su se razvile strukturnim mutacijama nad vrstom V3. One predstavljaju logičke funkcije : (C AND not(A AND notB) AND (B OR notA)) i (C AND not(A AND notB AND C)).

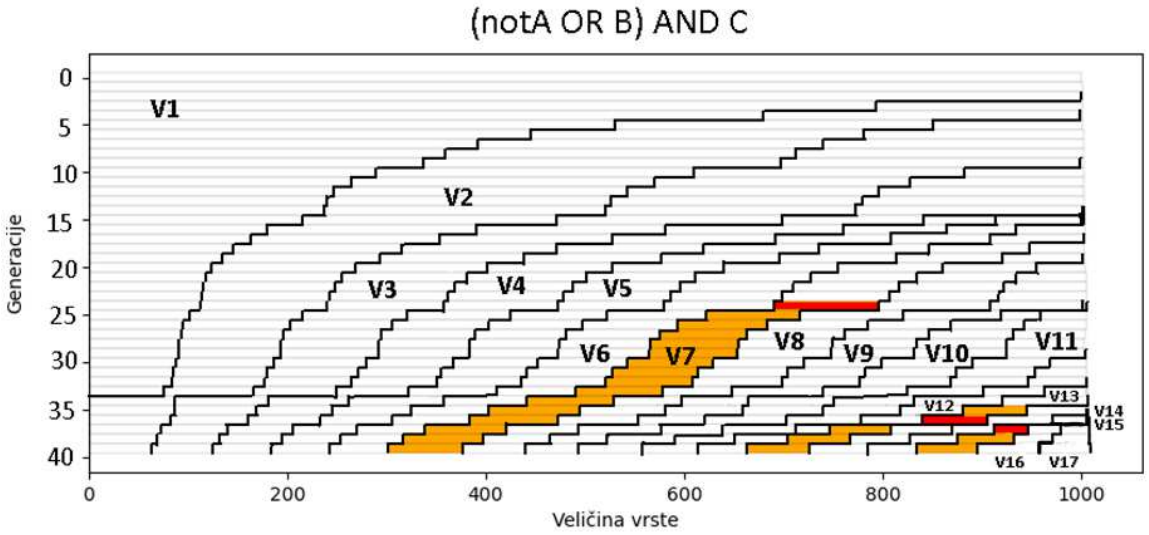


Slika 4.3 Grafički prikaz veličine vrsta kroz generacije za drugi problem



Slika 4.4 Neuronska mreža i odgovarajući logički sklop najbolje jedinke vrste V3

Postupak učenja je ponovljen više puta, niže je prikazan jedan od pokušaja u kojem nije pronađeno strukturno minimalno rješenje(Slika 4.5). U ovom pokušaju također su pronađene 3 vrste s jedinkama maksimalnih mogućih dobrota. Prva takva jedinka pronađena je u vrsti V7 u generaciji 24. Ta jedinka predstavlja logičku funkciju ((B AND C) OR (C AND notA)). Za vrijeme prvih 150 generacija, ni jedna vrsta nije razvila strukturu koja bi mogla predstavljati najjednostavniju logičku funkciju. Graf prikazuje samo prvih 40 generacija jer se u generaciji V15 razvila struktura vrlo bliska najjednostavnijoj mogućoj. Vrsta V15 predstavljena je logičkom funkcijom (C AND (B OR notC OR notA)). Ova vrsta razvila se dvjema uzastopnim mutacijama kod kojih se deaktivirao brid. Ova situacija potvrđuje da je moguće razviti jednostavnije strukture iz kompleksnijih. Također u generaciji 34 vidljivo je izumiranje vrste V1, posljednja dobra inovacija nad ovom vrstom dogodila se u 4 generaciji.



Slika 4.5 Veličine vrsta kroz generacije za drugi problem s gorim rješenjem

4.3. XOR

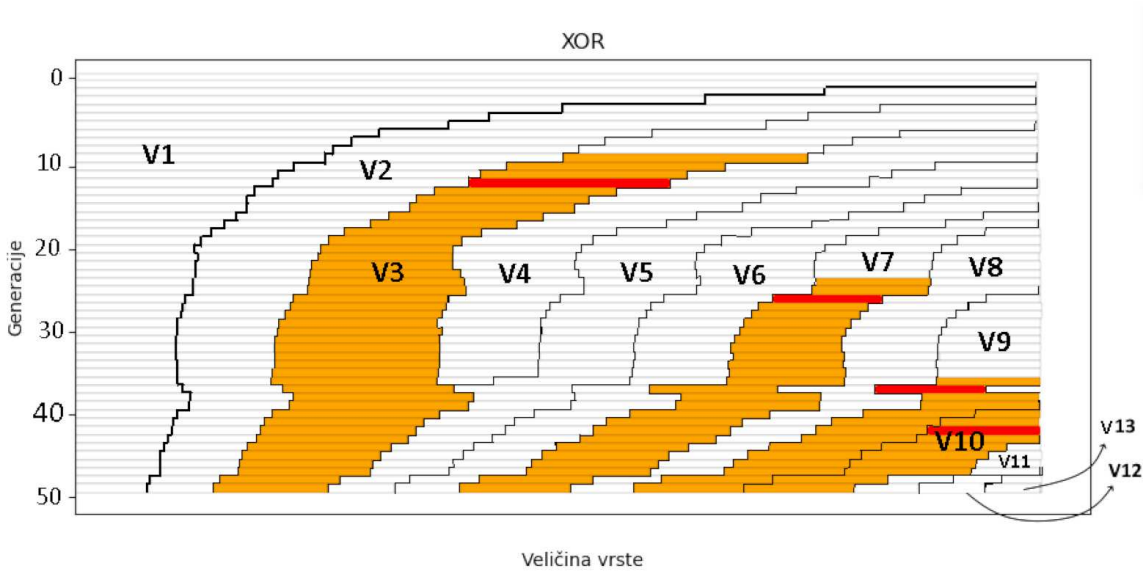
Treći problem na kojemu je odlučeno testirati algoritam je logička funkcija XOR s logičkom tablicom prikazanom u nastavku (Tablica 4.3). Ovaj problem je korišten kod testiranja original NEAT algoritma 2002. godine [1]. Iako ovaj problem ima samo dva ulaza, za njegovo rješenje i negaciju istog, algoritmu je potrebno gotovo duplo više generacija, nego ostalim problemima s dva ulazna neurona(Slika 4.6).

Tablica 4.3 Logička tablica funkcije XOR

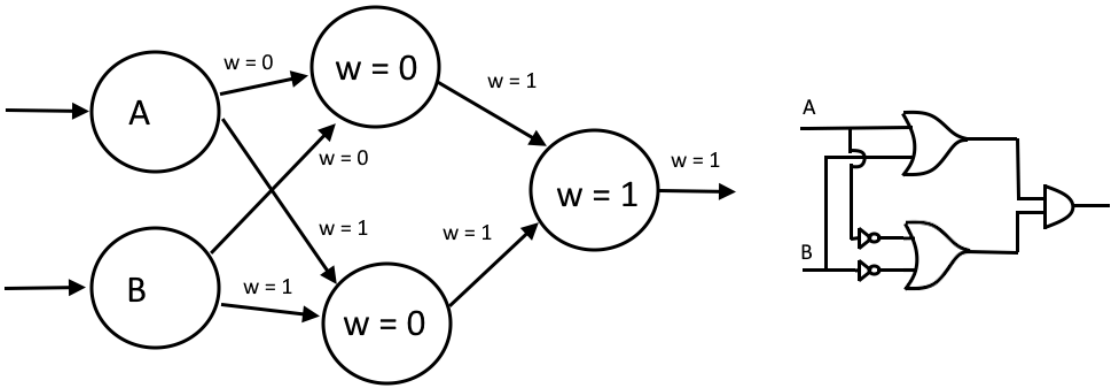
A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Rješenje pronađeno u postupku prikazanom na donjem grafu, ima minimalnu strukturu i kao neuronska mreža i kao logički sklop (Slika 4.7). Ovo rješenje pronađeno je u vrsti V3 u generaciji 12. Ova vrsta unutar 10 generacija uspijeva povećati svoju brojnost, na duplo od okolnih vrsta. Vrste V7 i V9 su nastale strukturnim mutacijama vrste V3, dok je vrsta V9 nastala strukturnom mutacijom vrste V8, koja je nastala kao mutacija vrste V2, pokazujući

opet primjer kada je točno rješenje dobiveno na dva nepovezana načina. V7 sadrži dodatni brid, dok V9 sadrži dodatni neuron. Sve četiri vrste uspijevaju proširiti dobre težinske vrijednosti unutar svojih jedinki, do te mjere da u generaciji 50 sve imaju više nego duplo veće brojnosti od ostalih vrsta.



Slika 4.6 Veličina vrsta po generacijama za XOR problem



Slika 4.7 Neuronska mreža i odgovarajući logički sklop pronađen rješenja za XOR problem

4.4. (A AND B AND notC) OR (notB AND C)

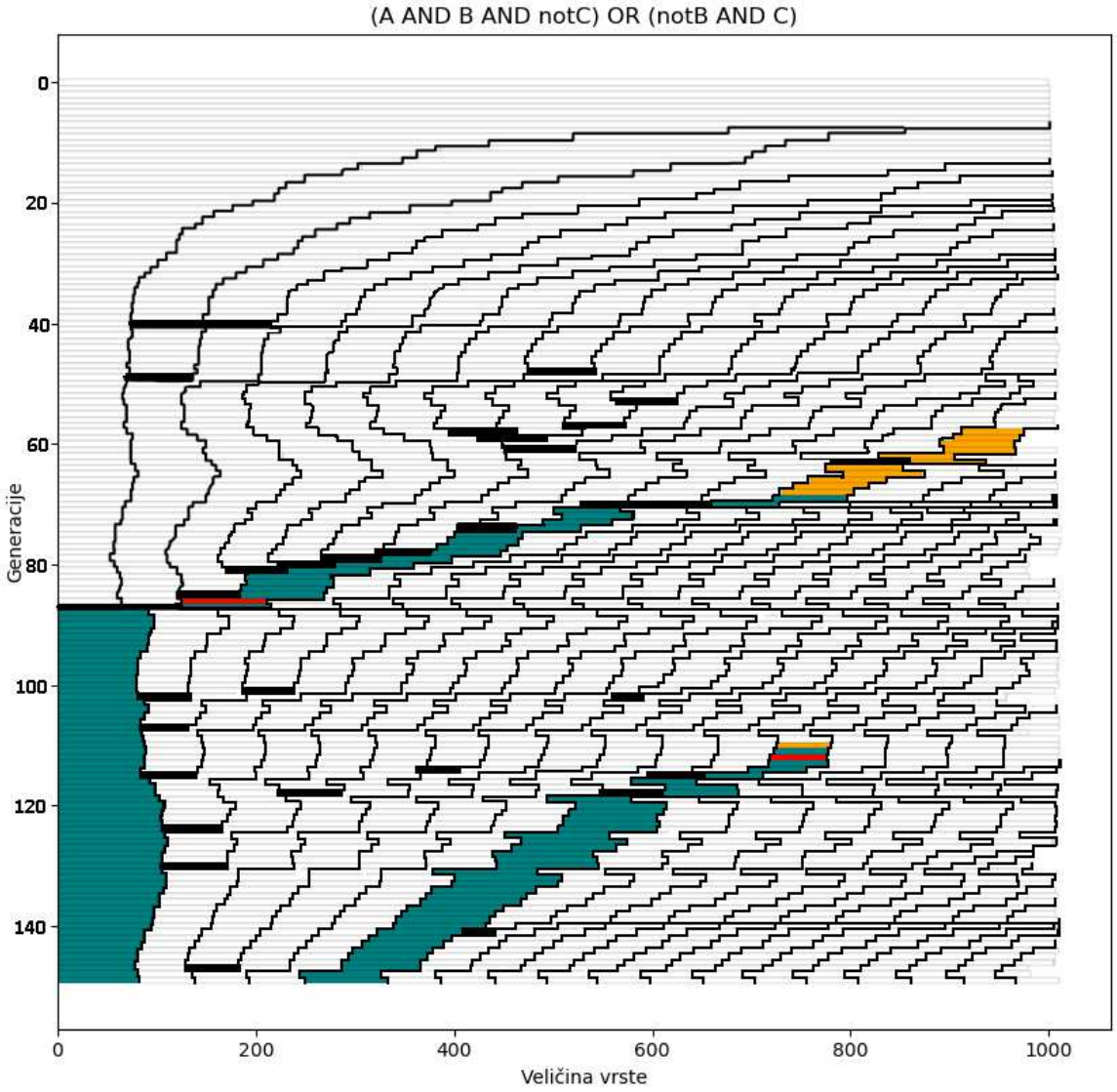
Četvrti problem na kojemu je odlučeno testirati algoritam je logička funkcija 4.4. (A AND B AND notC) OR (notB AND C) sa logičkom tablicom prikazanom u nastavku (Tablica 4.4).

Tablica 4.4 Ulazi i odgovarajući izlazi logičke funkcije četvrtog problema

A	B	C	(A AND B AND notC) OR (notB AND C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Ovaj problem zahtjeva 3 ulazna neurona i dodavanje barem 2 neurona uz dodatne bridove na njima, kako bi se dostigla jedinka s maksimalnom mogućom dobrotom. Zbog tih svojstva, ovaj problem korišten je kod testiranja varijacija algoritma. Algoritam rijetko pronalazi rješenje minimalne strukture iz prvog pokušaja, pa je kod rješavanje problema potrebno dozvoliti algoritmu velik broj generacija. U nastavku prikazan primjer (Slika 4.8), jedan je od bržih slučajeva rješavanja ovog problema, ali ne pokazuje slučaj u kojem je prva jedinka maksimalne dobrote, istovremeno i jedinka minimalne strukture. Zelenom (tamno sivom) bojom označene su vrste koje imaju dobrotu najbolje jedinke veću od medijana svih vrsta za barem dvije standardne devijacije. Crno obojane vrste, predstavljaju posljednju generaciju u kojoj je vrsta bila prisutna, kako bi se olakšalo snalaženje u grafu. Na grafu vrste nisu označene zbog manjka prostora. U nastavku, obojana vrsta koja se pojavljuje ranije, je nazvana V1, dok je druga obojana vrsta nazvana V2. Vrsta V1 razvila je jedinku s

maksimalnom mogućom dobrotom u generaciji 70, te je u generaciji 107 imala broj jedinki duplo veći od druge najveće vrste. Najbolja jedinka ove vrste predstavlja slijedeću logičku funkciju : not(not(A AND B AND notC) AND not(notB AND C) AND (notC OR not(A AND B AND notC))). Vrsta V2 je razvila svoju najbolju jedinku u 111. generaciji, te je u 135. generaciji brojem postala duplo veća o slijedeće manje vrste. Najbolja jedinka ove vrste predstavlja logičku funkciju: not(not(notB AND C) AND not(A AND notC AND B)). Iako ova funkcija nije minimalna moguća, njezina struktura neuronske mreže je minimalna. Prateći razvoj ove vrste, primijećeno je da se nije razvila iz vrste V1. Također je vidljivo da se vrsta razvila slijedom nekoliko micanja bridova. Ovo zapažanje je bitno, jer potvrđuje da moguće pronaći jednostavne strukture i dati im dovoljno vremena da razviju kvalitetne jedinke, čak i ako nisu nastale isključivo dodavanjem bridova i neurona.



Slika 4.8 Veličina vrsta po generacijama za četvrti problem

5. Varijacije algoritma

U nastavku se nalaze varijacije trenutnog algoritma koje su testirana kod razvoja. Kod svake promjene, ostatak algoritma se poklapa s trenutnim algoritmom kako bi se mogao izolirati učinak određene varijacije. Varijacije kod kojih nije bilo moguće generirati rješenje koje bi se moglo razmatrati kao prihvatljivo, imaju kod obrazloženja oznaku NE RADI.

Tablica 5.1 Popisa varijacija algoritama i njihova opažanja

Varijacija	**Opažanja**
VA001	
Promijenjena vrijednost absoluteTreshold 0.1	Kod korištenja ove vrijednosti, algoritmu je trebalo više generacija kako bi se unutar vrste s dobrom strukturom, razvila jedinka s maksimalnom dobrotom. Nakon pronalaska ove jedinke, povećanja broja jedinka u vrsti pokazalo brže, nego u trenutnom algoritmu.
VA002	
Promijenjena vrijednost absoluteTreshold 0.4	Kod korištenja ove vrijednosti, jedinke koje su bile blizu maksimalne dobrote davale su potomstvo koje se ponašalo potpuno drugačije.
VA003	
Promijenjena vrijednost absoluteTreshold 0	Kod korištenja ove vrijednosti, vrste s potencijalom za razvoj jedinki maksimalne dobrote su trebale puno generacija da napreduju, ali su imale podjednaku dobrotu kroz cijelu vrstu. Nažalost ova vrijednost je često dovodila do limita od 50 generacija bez napretka. Na ovaj način nije moguće bilo pronaći ni jedno rješenje koje unutar nekoliko operacija pojednostavljivanja bi bilo minimalno. NE RADI
VA004	
Ponavljanje testova 200 puta	Promjene u efikasnosti su minimalne. Primjećuje se bolje potomstvo kod križanja jedinki. Vrijeme evaluacije se udvostručilo, što stvara problem kod složenijih problema s 3 ulazna neurona.

VA005	
Računanje razlike između jedinki kao u originalnom algoritmu	Koristeći različite vrijednosti c_1 ,c_2 i c_3 ovaj princip razdvajanja jedinki u vrste je razdvajao jedinke iste strukture u različite vrste zbog razlika u težini. U rješavanju ovog problema, korisnije je gledati samo strukturu jedinki kako bi se lakše moglo upravljati dobrim jedinkama. NE RADI
VA006	
Suma različitih elemenata u neuronskoj mreži	Ovaj postupak je brojio bridove i neurone s inovacijskim brojevima koji se ne nalaze u drugoj jedinki. Razlika od trenutnog algoritma je u tome, da ova varijacija ne ignorira elemente koji ne mijenjaju mogućnost logičke funkcije koja se može prikazati ovom strukturom. Kod ove varijacije se broj vrsta povećavao puno većom brzinom, samim time se smanjivao broj jedinki u svakoj vrsti. Ovakvim razvrstavanjem ograničava se koliko mutacija svaka vrsta može napraviti po jednog generaciji i samim time usporava se proces učenja.
VA007	
Usporedba logičkog sklopa kojeg neuronska mreža predstavlja	Korištenje ovakvog načina razvrstavanja ne doprinosi ničemu. Vrste predstavljaju kopije jedinke koje imaju svoje vrijednosti promijenjene za manje od 0.5. Ova metoda se ničim ne razlikuje od toga da u opće ne koristimo razvrstavanje u vrste. NE RADI
VA008	
Dodatna provjera dobrote jedinke koja ulazi u elitizam	Korištenje ove metode ne mijenja rezultate izvan očekivanih razlika koje dolaze zbog velikog oslanjanja algoritma na generiranje slučajnih brojeva. Jedina promjena koja je zamijećena, a mogla bi utjecati na kompleksnije probleme, je ne dodavanje jedinki koje su imale bitnu promjenu u sebi u odnosu na ostatak vrste, ali nisu dovoljno tom promjenom digle svoju dobrotu. Kod pomnijeg promatranja algoritma usred učenja, primijećeno je da u nekoliko slučaja zaboravljeno rješenje koje se opet otkrilo nakon nekoliko generacija i zatim koristilo u konačnom rješenju.
VA009	
Bez elitizma	Vrste koje su tek došle do dobrote koja dozvoljava 50 iteracija bez promjene, u svojim jedinkama imaju samo jedinke lošije od svoje najbolje do sad pronađene. NE RADI
VA010	
Dodavanje najbolje jedinke iz svake vrste	Nakon odumiranja vrste, njihove najbolje jedinke zauzimaju mjesto koje je potrebno za brži razvoj ostalih vrsta. Algoritam i dalje pronalazi rješenje ali kod složenijih problema zahtjeva primjetno više generacija za učenje.

VA011	
Promijenjene šanse za mutiranje (0.991, 0.003, 0.003, 0.003)	Zbog prevelikih šansi za strukturne mutacije, broj vrsta se poveća prebrzo. Broj jedinki koje vrsta može imati se smanji prebrzo, i vrste ne mogu razviti svoje najbolje moguće jedinke. Algoritam funkcionira kod jednostavnijih problema, ali ne daje rješenje za XOR ili probleme s 3 ili više ulaza koji imaju više od jednog dodatnog neurona u očekivanom rješenju.
VA012	
Promijenjene šanse za mutiranje (0.91, 0.03, 0.03, 0.03)	Zbog prevelikih šansi za strukturne mutacije, broj vrsta se poveća prebrzo. Broj jedinki koje vrsta može imati se smanji prebrzo, i vrste ne mogu razviti svoje najbolje moguće jedinke.NE RADI
VA013	
Promjena vrijednosti *weigthShiftStrength (8)*	Ova vrijednost obrnuto je proporcionalna promjeni težine koja se može desiti kod mutacije Promjena težine. Jedina primjetna razlika kod ove promjene, je bila brzina kojom se povećavaju vrste nakon pronalaska dobrih rješenja, proces je usporio, ali zanemarivo . Kod težih problema koji imaju 3 ulazna neurona, algoritam rijetko pronalazi prihvatljivo rješenje.
VA014	
Promjena vrijednosti *weigthShiftStrength (2.7)*	Ova vrijednost obrnuto je proporcionalna promjeni težine koja se može desiti kod mutacije Promjena težine. Jedina primjetna razlika kod ove promjene, je bila brzina kojom se povećavaju vrste nakon pronalaska dobrih rješenja. Proces je jako usporio.
VA015	
Promjena vjerojatnosti za malu promjenu težine/novu vrijednost/bez promjene (0.5,0.5,0)	Kod mutiranja, jedinke kojima je potrebno bilo podesiti samo nekoliko težina, često su imale ostale bridove previše promijenjene, da bi napredak biti kontinuiran. Algoritam se ponašao slično kao kada su sve vrijednosti bile određene nasumice. NE RADI
VA016	
Promjena vjerojatnosti za malu promjenu težine/novu vrijednost/bez promjene (0.4,0.1,0)	Kod problema s 3 ulaza, vrste koje su sadržavale obećavajuće strukture su provodile jako puno vremena između stvaranja i postizanja dobrote koja je potrebna za dozvoljenih 50 generacija bez napretka. Algoritam je i dalje funkcionirao, ali je izrazito rijetko dolazio do optimalnih rješenja.

VA017	
Reaktivacija deaktiviranih bridova	Kod mutacije dodavanja bridova u jedinku, u odabir mogućih bridova za dodavanje, dodani su deaktivirani bridovi koji ne stvaraju cikluse u neuronskoj mreži. Većina bridova dodanih na ovaj način nisu stvarali nove, nego već postojeće strukture. S obzirom na dodatno vrijeme koje je potrebno provesti tražeći deaktivirane bridove i provjeravati njihovu mogućnost aktivacije, ne isplati se koristiti ovu varijaciju.
VA018	
Postavljanje početne težine novog brida na 0/1	Nema primjetne razlike ni kod kojeg problema
VA019	
Promjena parametra r (1)	Parametar r označava koliko manju šansu za stvaranje novog neurona ima sljedeći stvorenu brid u jedinki. Postavljajući ovu vrijednost na 1, svi bridovi imaju jednake šanse. Ovo dovodi do problema kada je potrebno dodati neuron na jedan od početnih bridova kako bi se dovršila relativno jednostavna jedinka. Umjesto toga, svako dodavanje novog neurona na neki drugi brid, smanjuje šanse za odabir pravog brida. Najsigurniji način za pronalazak minimalnih rješenja je pretraživanje struktura koje se razvijaju u širinu, ne dubinu. NE RADI
VA021	
Promjena parametra r (1.5)	Parametar r označava koliko manju šansu za stvaranje novog neurona ima sljedeći stvorenu brid u jedinki. Postavljajući ovu vrijednost na 1.5 stariji bridovi imaju veću šansu za odabir. Ova varijacija se pokaza bolja od varijacije VA019, ali i dalje dolazi do istih problema u rjeđim situacijama.
VA022	
Promjena parametra r (5)	Parametar r označava koliko manju šansu za stvaranje novog neurona ima sljedeći stvorenu brid u jedinki. Postavljajući ovu vrijednost na 5 stariji bridovi imaju preveliku šansu za odabir, kod kompliciranijih jedinki, gotovo uvijek se koristi najstariji brid za dodavanje novog neurona. NE RADI

VA023	
Križanje jedinki iz različitih vrsta	Kod križanja jedinki različitih struktura, potrebno je odabrati jednog roditelja i zatim na njegovu strukturu dodavati promjene iz drugog roditelja ako je to potrebno. Zbog potencijalno znatno različitijih struktura., potrebno je raditi puno rekurzivnih poziva nad oboje neuronske mreže kako bi se elementi dodali u prvog roditelja. Također za svaki element prije dodavanja potrebno je provjeriti smije li se dodati, kako ne bi narušio pravila strukture neuronskih mreži koje ovaj algoritam podržava. Varijacija algoritma pokazala se korisnom kod generiranja novih struktura u složenijim problemima. Nažalost u tim istim problemima je samo učenje trajalo predugo da bi se moglo kvalitetno testirati kao dio trenutnog algoritma.

Zaključak

Tema ovo završnog rada bila je implementacija NEAT (*NeuroEvolution of augmenting topologies*) algoritma i njegova primjena u izgradnji i pojednostavljivanju logičkih sklopova na temelju danih logičkih tablica. Nakon odabira načina na koji će logički sklopovi biti prikazani kao neuronske mreže, implementirana je osnovna verzija NEAT algoritma. Zbog specifične domene i kodomene problema, promijenjeni su klasični operatori koji se koriste u neuronskim mrežama. Kako bi se došlo do rješenja koja zadovoljavaju dane logičke tablice algoritam je modificiran u načinu na koji određuje vrste i načinu na koji mutira jedinke. Dokumentirane su različite varijacije algoritma i njihov učinak na dobivene rezultate. Rezultati se temelje na jedinka i vrstama koje su generirane za zadana 4 problema različitih težina. Pokazano je da algoritam može generirati minimalne logičke sklopove koji zadovoljavaju dane tablice. Algoritam teži najmanjim rješenjima, ali ih ne daje u svim slučajevima. Trenutnom algoritmu nedostaje način da u dobrotu jedinke uključi i broj logičkih vrata koja koristi logički sklop, predstavljen neuronskom mrežom. Također je potrebno istražiti način na koji bi se moglo maknuti nepotrebne bridove iz izračuna vrijednosti na kraju neuronske mreže, bez da se koristi strukturna mutacija, nego i sama mutacija težina.

Literatura

[1] Stanley, K. O., Miikkulainen, R. *Evolving Neural Networks through Augmenting Topologies.* Evolutionary Computation, *10*,2 (2002), 99–127.

[2] Goldberg, D. E., Richardson, J. *Genetic algorithms with sharing for multimodal function optimization*, Proceedings of the Second International Conference on Genetic Algorithms, California, (1987), 148–154.

Sažetak

PRIMJENA ALGORITMA NEAT U OPTIMIZIRANJU LOGIČKIH SKLOPOVA

Sažetak:

Cilj ovo rada bio je implementirati NEAT (*NeuroEvolution of Augmenting Topologie*s) algoritam i testirati njegovu efikasnost kod optimiziranja struktura neuronskih mreža. Odabrani problem je optimizacija logičkih sklopova. Prilagodbom NEAT algoritma potrebama problema, razvijen je algoritam koji može ograničeno optimizirati logičke sklopove prikazane kao neuronske mreže. Tijekom razvoja algoritma korištene su različite varijacije parametara i metoda, čiji utjecaj na uspješnost i ponašanje algoritma je dokumentiran.

Ključne riječi: NEAT algoritam, Optimizacija, Logički sklopovi, Logičke funkcije, Genetski algoritam

Summary

APPLICATION OF THE NEAT ALGORITHM IN OPTIMIZING LOGIC CIRCUITS

Summary:

The goal of this work was to implement the NEAT (NeuroEvolution of Augmenting Topologies) algorithm and test its effectiveness in optimizing neural network structures. The selected problem is the optimization of logic circuits. By adapting the NEAT algorithm to the needs of the problem, an algorithm was developed that can optimize logical circuits represented as neural networks in a limited way. During the development of the algorithm, different variations of parameters and methods were used, whose impact on the success and behavior of the algorithm was documented.

Keywords: NEAT algorithm, Optimization, Logic circuits, Logic functions, Genetic algorithm