SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1733

SUSTAV ZA RASPOZNAVANJE ZNAKOVA TRENIRAN GLOBALNIM PARALELNIM GENETSKIM ALGORITMOM

Marin Nevešćanin

Zagreb, srpanj 2008.

Sažetak

Ovaj diplomski rad opisuje paralelizaciju genetskih algoritama te primjenu globalno paralelnog genetskog algoritma na treniranje neuronske mreže za raspoznavanje rukom pisanih znakova. Praktični dio ovog rada je implementiran u programskom jeziku C++, te je prilagođen za rad na računalnom grozdu ili višejezgrenom računalu. U analizama rezultata najveći naglasak je stavljen na ovisnosti vremena izvođenja paralelnog genetskog algoritma u odnosu na broj procesora, te osjetljivost kvalitete rješenja u odnosu na parametre genetskog algoritma i neuronske mreže.

Abstract

This diploma thesis describes parallelization of genetic algorithms and the use of global parallel genetic algorithm for training neural network to recognize handwritten characters. Practical part of this thesis is implemented in C++ programming language and it is specialized to run on a computer cluster or on a multi-core computer. In the results analysis, the biggest emphasis is placed on the correlation between run time of parallel genetic algorithm and number of processors, and sensitivity of results quality in relation to parameters of genetic algorithm and neural networks.

Sadržaj

1. Uvod

Mnogi problemi koje čovjek s lakoćom rješava, znaju biti prilično teški kada se oni pokušaju riješiti na računalu. Čovjek kad jednom nauči čitati vrlo lako raspoznaje slova koja vidi oko sebe. No i čovjek dok je bio dijete nije znao čitati, već je to savladao postepeno. Svako novo slovo koje bi vidio proučavao je po čemu se razlikuje od ostalih, a ako ga je već prije vidio pokušavao bi se sjetiti koje je to slovo. Tek nakon dužeg procesa učenja uspio je zapamtiti sva slova i tako naučio čitati.

Ako se želi isti problem raspoznavanja slova riješiti uz pomoć računala, najbolje je simulirati taj isti proces. Ljudi su tako došli do ideje umjetnih neuronskih mreža koje oponašaju rad ljudskog mozga, tj. mozga živih bića.

Međutim, neuronsku mrežu treba naučiti raspoznavati slova, jer ni čovjek nije mogao sam znati što neko slovo predstavlja osim ako mu to neko drugi nije objasnio. Tu se javlja drugi problem: kako naučiti neuronsku mrežu, koja je u početku *tabula rasa*, da raspoznaje slova. Taj se problem može riješiti i pomoću konvencionalnih metoda na računalu, ali u ovom diplomskom radu je to riješeno uz pomoć genetskog algoritma. I taj algoritam nalazi inspiraciju u prirodi i svijetu koji nas okružuje. On simulira evoluciju, te pokušava evoluirati neuronsku mrežu da dođe do optimalne neuronske mreže koja raspoznaje rukom pisana slova. Prirodnom selekcijom bolje jedinke (pametnije neuronske mreže) opstaju a lošije izumiru.

Danas, iako računala obavljaju sve više i više operacija u sekundi, vremenski ciklus (engl. *clock cycle*) se približava fizikalnim ograničenjima (brzina svjetlosti, brzina elektrona). Stoga je jedno od rješenja koje se nameće udruživanje više računala (npr. u grozd računala) kako bi paralelno rješavali jedan problem. Pošto je i u prirodi evolucija trajala dugo (milijardama godina) tako je i na računalu genetski algoritam, koji ju oponaša, dugotrajan proces. Kako bi se smanjilo samo vrijeme izvođenja genetskog algoritma moguće ga je paralelizirati tako da se izvršava na više računala istodobno. Naglasak diplomskog rada jest na paralelnom genetskom algoritmu, dok je za neuronsku mrežu izabrana najjednostavnija moguća mreža.

Ovaj diplomski rad je sastavljen od 9 poglavlja. Nakon uvodnog poglavlja slijedi poglavlje o genetskim algoritmima. Ono započinje temom o prirodnoj evoluciji te onda govori o analogiji genetskog algoritma s prirodnom evolucijom. Nastavlja se detaljnim opisom genetskog algoritma i operatora koje taj algoritam koristi.

Treće poglavlje govori o paralelizaciji genetskog algoritma, koje sve vrste paralelnih genetskih algoritama imamo, njihove prednosti i mane. Na kraju poglavlja opisuje zahtjeve koje bi trebao zadovoljiti *idealni* paralelni genetski algoritam, te koje preinake treba napraviti jedna od navedenih metoda (globalni paralelni genetski algoritam koji je opisan u tom poglavlju) da zadovolji te zahtjeve.

Četvrto poglavlje daje opis neuronske mreže. Govori o analogiji umjetne neuronske mreže s mozgom živih bića, opisuje jednoslojnu i višeslojnu neuronsku mrežu, te postupak učenja i testiranja neuronske mreže. Poglavlje završava s primjenom neuronskih mreža u računarstvu.

Peto poglavlje ukratko objašnjava što je to grozd računala (klaster), te daje opis klastera Isabelle na Srcu (Sveučilišnom računarskom centru) na kojem je testirana implementacija paralelnog genetskog algoritma koji trenira neuronsku mrežu za raspoznavanje rukom pisanih znakova.

Šesto poglavlje opisuje praktični rad, odnosno implementaciju programa. Opisuje korištenu bazu ulaznih znakova, korištenu neuronsku mrežu, programsko ostvarenje paralelnog genetskog algoritma te upute za izvođenje programa na klasteru ili na osobnom računalu.

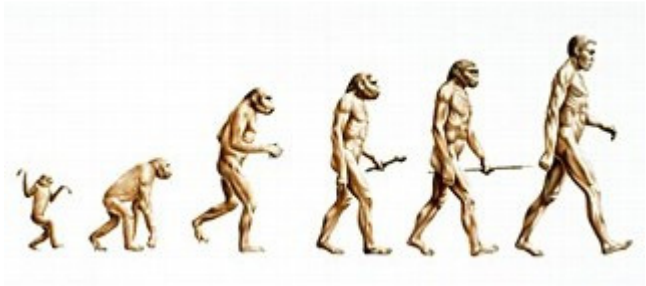
U sedmom poglavlju su analizirani dobiveni rezultati. Uspoređuje se vrijeme izvođenja paralelnog sa sekvencijalnim genetskim algoritmom, te koje ubrzanje postiže paralelni model. Prikazuje se osjetljivost kvalitete rješenja na parametre genetskog algoritma, i analiziraju se rezultati dobiveni korištenjem različitih neuronskih mreža. Na kraju poglavlja predstavljeni su rezultati raspoznavanja rukom pisanih slova hrvatske abecede.

2. Genetski algoritmi

2.1. Prirodna evolucija

Evolucija je neprekidan proces prilagođavanja živih bića na svoju okolinu, tj. na uvjete u kojima žive. Charles Darwin je prvi iznio teoriju biološke evolucije po kojoj populacija neke vrste napreduje tj. evoluira kroz generacije pomoću procesa prirodne selekcije. On je primijetio da živa bića reproduciraju više potomaka od cijele njihove populacije, no ipak kroz vrijeme broj jedinki u populaciji je, umjesto da eksponencijalno raste, stalan ili se jako sporo mijenja. Količina dostupne hrane je ograničena, ali je i ona većinu vremena konstantna. Iz toga proizlazi da je potrebna nemilosrdna borba za opstanak.

Kod vrsta koje se spolno razmnožavaju, ne postoje dvije iste jedinke. Sve su različite, a te različitosti utječu na sposobnost preživljavanja jedinke. Svaku jedinku se može okarakterizirati nizom svojstava (oblik zubi, boja kože, boja očiju, otpornost na neke bolesti, sposobnost trčanja itd.). Jedinke s boljim svojstvima će lakše preživjeti i dalje se reproducirati, za razliku od jedinki sa slabijom mogućnošću prilagodbe. Većina svojstava je nasljedna, pa se prenosi s oba roditelja na jedinke. To znači da će jedinke koje prežive vjerojatno ostaviti svoje značajke budućim generacijama u nasljeđe. Svaka sljedeća generacija pojedine vrste pamti dobra svojstva prethodne generacije, mijenja ta svojstva tako da ostanu dobra u novim uvjetima te pronalazi nova. Taj spori proces rezultira novom prilagođenijom vrstom, a akumuliranjem tih malih promjena nakon dugo vremena može doći i do potpuno nove vrste [3].



Slika 2.1. Evolucija čovjeka

Sva svojstva jedinki su zapisana u kromosomima. Kromosomi se sastoje od gena, a svaki gen sadrži skup informacija koje karakteriziraju jedno svojstvo. Kromosomi dolaze uvijek u paru. Jedan je od majke, a jedan od oca. U takvom paru geni mogu biti ili ravnopravni ili može jedan bit dominantan, a drugi recesivan. U neravnopravnom paru svojstvo koje nosi dominantan gen se manifestira na fenotipu (vidljive značajke jedinke – boja kose, očiju, visina itd.), dok se kod ravnopravnog para gena dobiva svojstvo koje je negdje između svojstava majke i oca. Svaka jedinka ima više parova kromosoma, a ne samo jedan. Čovjek ima 46 kromosoma, tj. 23 para kromosoma.

Prenošenje genetskog materijala s roditelja na djecu postiže se križanjem kromosoma. Time se prenose i svojstva roditelja na djecu. Svaka jedinka posjeduje genetski materijal oba roditelja. Neka je broj kromosoma koje ima jedinka *2n*. Slijedeća generacija jedinki

također treba imati *2n* kromosoma, što znači da svaki od roditelja treba dati po *n* kromosoma. To se postiže sa spolnom diobom kromosoma koja se zove mejoza.

Uz križanje, druga moguća promjena genetske poruke naziva se mutacija. Mutacija je slučajna promjena gena, koja se može dogoditi prilikom diobe stanica. Vjerojatnost pojave mutacije je relativno mala (kreće se od 10^{-4} do 10^{-5}). Posljedica mutacije može biti i dobra i loša. Upravo zahvaljujući pozitivnim mutacijama populacija može ubrzano napredovati i postati prilagođenija na okoliš (npr. mutiranjem jedinka postane otporna na neku bolest i to joj daje veću šansu da preživi ona i njeni potomci), dok kod negativne mutacije jedinka smanjuje sebi šansu za preživljavanje pa vjerojatno umire bez stvaranja potomaka.

2.2. Genetski algoritam kao imitacija evolucije

Genetski algoritam (GA) je heuristička metoda slučajnog i usmjerenog pretraživanja prostora rješenja koja simulira prirodni evolucijski proces. Sama evolucija se može gledat kao metoda optimiranja, tj. traženja najbolje i najprilagodljivije jedinke na okoliš i uvjete u prirodi.

Ta analogija evolucije kao prirodnog procesa i genetskog algoritma kao metode optimiranja, očituje se u procesu selekcije i genetskim operatorima. No genetski algoritam nije idealna preslika evolucije. Simuliranje evolucije na računalu svodi se na grube aproksimacije rješenja, jer smo ograničeni mogućnošću računala. Npr. za preslikavanje genotipa samo jedne jedinke žabe, koji sadrži približno $3.2*10^9$ nukleotida, potrebno nam je oko 1GB memorijskog prostor. To bi značilo da vjerna simulacija evolucije jedne vrste od milijun jedinki zahtjeva približno milijun gigabajta [1].

Prirodnom selekcijom preživljavaju jedinke koje su se najbolje prilagodile okolini i uvjetima u prirodi, te time imaju najveću vjerojatnost prenošenja svog genetskog materijala na svoje potomke. U genetskim algoritmima ključ selekcije je funkcija cilja, koja na odgovarajući način predstavlja problem koji se rješava. Slično kao što su okolina i uvjeti u prirodi ključ selekcije nad nekom vrstom živih bića, tako je i funkcija cilja ključ selekcije nad populacijom rješenja u genetskom algoritmu.

U prirodi pri svakoj reprodukciji dolazi do rekombinacije gena koja uzrokuje različitost među pojedincima iste vrste, ali i sličnost s roditeljima nove jedinke. U genetskim algoritmima izmjena gena koja nastaje pri reprodukciji se naziva križanje. Osim križanja, genetski algoritam ima još jedan genetski operator koji se zove mutacija. On, naravno, imitira mutaciju u prirodi, tj. slučajno mijenjanje genetskog materijala koje nastaje pod djelovanjem vanjskih uzroka.

U genetskom algoritmu svaka jedinka predstavlja potencijalno rješenje problema koji se obrađuje. Svaka jedinka je predstavljena jednakom podatkovnom strukturom, kao što su broj, niz, matrica, stablo ili neka posebna struktura. Te jedinke se nazivaju **kromosomi**. Svakoj jedinki se pridjeljuje određena mjera kvalitete koja se u literaturi obično naziva **dobrota**, dok se funkcija koja tu kvalitetu određuje naziva **funkcija cilja** ili **funkcija dobrote**. Genetski operatori se dijele na unarne, koji stvaraju novu jedinku mijenjajući manji dio genetskog materijala (mutacija) i operatore višeg reda, koji kreiraju nove

jedinke kombinirajući osobine dvije ili više jedinki (križanje). Nakon nekog broja izvršenih generacija čitav postupak se zaustavlja kada se zadovolji uvjet zaustavljanja, a najbolji član trenutne populacije predstavlja rješenje koje bi trebalo biti sasvim blizu optimuma [1].

```
Genetski_algoritam
{
    t = 0;
    generiraj početnu populaciju potencijalnih rješenja P(0);
    sve dok nije zadovoljen uvjet završetka evolucijskog procesa
    {
        t = t + 1;
        selektiraj P'(t) iz P(t-1);
        križaj jedinke iz P'(t) i djecu spremi u P(t);
        mutiraj jedinke iz P(t);
    }
    ispiši rješenje;
}
```

Slika 2.2. Pseudokod genetskog algoritma

Slika 2.2 pokazuje pseudokod genetskog algoritma. Iz njega možemo vidjeti da algoritam započinje generiranjem početne populacije potencijalnih rješenja. Obično se početna populacija generira slučajnim odabirom rješenja iz domene. Nakon inicijalizacije populacije, algoritam ulazi u petlju koja se ponavlja sve dok se ne zadovolji neki uvjet zaustavljanja (može biti zadan broj iteracija ili npr. da funkcija dobrote najbolje jedinke odstupa za manje od ε). Zatim se ispiše najbolje rješenje u rezultirajućoj populaciji.

Unutar same petlje nad populacijom se obavljaju genetske operacije. U svakoj iteraciji se prvo obavlja selekcija, tijekom koje loše jedinke odumiru, a bolje opstaju te se dalje križaju. Križanjem nastaju djeca na koja se prenose svojstva roditelja, dok se mutacijom mijenjaju svojstva jedinke slučajnom promjenom gena. Tim postupkom se postiže iz generacije u generaciju sve veća i veća prosječna dobrota populacije, ali i sve veća dobrota najbolje jedinke u populaciji koju tražimo.

2.3. Binarni prikaz jedinke

Izbor prikaza jedinke je vrlo značajan, jer može bitno utjecat na učinkovitost genetskog algoritma. U praksi se pokazalo da binarni prikaz daje najbolje rezultate u većini primjena gdje se može binarni prikaz iskoristiti, a i sama teorija genetskog algoritma je većinom vezana uz binarni prikaz.

Kromosom možemo prikazati kao niz bitova, koji predstavljaju kodiranu vrijednost rješenja $x \in [dg, gg]$, gdje dg predstavlja donju granicu rješenja, a gg gornju granicu. Ako je dužina kromosoma n, u njega je moguće zapisati 2^n različitih kombinacija nula i jedinica. To znači da je donja granica predstavljena kromosomom s vrijednošću 0, dok je gornja granica predstavljena s kromosomom s najvećom vrijednošću tj. s $2^n - 1$.

Općenito, ako je binarni broj $b \in [0,\ 2^n - 1]$ zapisan kao binarni vektor (niz bitova) $v(b) = [B_{n-1}B_{n-2} \dots B_1B_0]$, gdje je $B_i = \{0,1\}$, tada vrijedi [1]:

$$b = \sum_{i=0}^{n-1} B_i \cdot 2^i \tag{2.1}$$

$$x = dg + \frac{b}{(2^n - 1)} \cdot (gg - dg) \tag{2.2}$$

Jednadžba (2.2) je proces pretvaranja binarnog broja u potencijalno rješenje, a taj proces se naziva dekodiranje. Dakle, binarni vektor $v(b)$ predstavlja vrijednost x u intervalu $[dg, gg]$.

Suprotno dekodiranju postoji i kodiranje, tj. određivanje binarnog broja b za zadani realni broj x, koje se obavlja prema sljedećoj formuli:

$$b = \frac{x - dg}{gg - dg} \cdot (2^n - 1) \tag{2.3}$$

Preciznosti rješenja ovisi o dužini kromosoma. Što je veći niz bitova u kromosomu to se rješenje može prikazati na više decimala. Ako se uzme da je preciznost na p decimalnih mjesta, s tim da je $p \in N$, onda rješenje x može odstupati od točnog rješenja za maksimalno 10^{-p}. Uvjet koji mora zadovoljiti duljina kromosoma n zadan je sljedećim izrazom:

$$(gg - dg) \cdot 10^{-p} < 2^n - 1 \tag{2.4}$$

Iz tog uvjeta proizlazi da je minimalna duljina kromosoma potreban za postizanje preciznosti p:

$$n \geq \frac{\log[(gg - dg) \cdot 10^p + 1]}{log2} \tag{2.5}$$

2.4. Genetski operatori

2.4.1. Evaluacija

U fazi evaluacije se određuje dobrota svake jedinke. Kako bi se dobrota mogla izračunati, prvo je potrebno odrediti *funkciju dobrote* ili *funkciju cilja* (engl. *fitness function*). U najjednostavnijem slučaju funkcija dobrote je jednaka funkciji koju treba optimizirati:

$$dobrota(v) = f(x) \tag{2.6}$$

U jednadžbi (2.6) vektor v predstavlja binarni vektor, tj. $dobrota(v)$ jest dobrota kromosoma v. Što je dobrota veća, jedinka ima veću vjerojatnost preživljavanja i križanja. Funkcija dobrote je ključ za proces selekcije. Tijekom procesa evolucije iz generacije u generaciju GA generira populaciju čija je prosječna dobrota sve bolja i bolja.

Na početku genetskog algoritma se evaluiraju sve jedinke (izračunaju se njihove dobrote), jer je dobrota važna prilikom selekcije jedinki. Nakon što se stvori nova jedinka ili više novih jedinki, ovisno o algoritmu, dovoljno je evaluirati samo te nove jedinke, jer se dobrota već postojećih jedinki nije mijenjala.

2.4.2. Selekcija

Prirodna selekcija čuva i prenosi dobra svojstva na sljedeću generaciju. Ona odabire *dobre* jedinke koje će sudjelovati u sljedećem koraku, u reprodukciji, a *loše* jedinke odumiru. Pošto genetski algoritam imitira prirodnu evoluciju onda je i proces selekcije važan dio algoritma. Kad bi se izabralo za postupak selekcije kod genetskog algoritma izbaciti M najgorih jedinki, a preostale jedinke dalje reproducirati, vrlo brzo bi se došlo do konvergencije genetskog algoritma. No time bi se izgubio *dobar* genetski materijal koji mogu sadržavati *loše* jedinke, te bi genetski algoritam ostao u najbližem lokalnom optimumu. Zato je potrebno osigurati i lošim jedinkama da imaju neku (manju) vjerojatnost preživljavanja [1]. Naravno, boljim jedinkama je ta vjerojatnost mnogo veća, ali nikad nije jednaka jedinici. Iako to predstavlja rizik gubitka dobrog genetskog materijala, genetska raznolikost populacije je ipak važnija od egzistencije pojedinih jedinki [4].

Genetske algoritme s obzirom na selekciju dijelimo na *generacijske* i *eliminacijske*. Generacijskom selekcijom se direktno biraju bolje jedinke čiji će se genetski materijal prenijeti u sljedeću generaciju, dok kod eliminacijske selekcije se biraju lošije jedinke za eliminaciju, a bolje jedinke prežive postupak selekcije.

Jednostavna selekcija (*selekcija roditelja ruletom*, engl. r*oulette wheel parent selection*) generira novu populaciju koja ima isti broj jedinki kao i stara populacija. Ona pripada generacijskoj selekciji. Cilj te selekcije je odabir roditelja čija je vjerojatnost selekcije proporcionalna njihovoj dobroti. Postupak jednostavne selekcije je sljedeći [1]:

- dobrota jedinke mora biti pozitivan broj, a ako je negativan dodaje se konstanta *C*

$$dobrota(v) = f(x) + C, \text{ tako da je } dobrota(v) \geq 0,\ \forall x \epsilon [dg, gg] \tag{2.7}$$

- izračunaju se sve vrijednosti dobrote jedinki
- izračuna se ukupna dobrota populacije prema sljedećoj formuli (Neka je N broj jedinki u populaciji):

$$D = \sum_{i=1}^{N} dobrota(v_i) \tag{2.8}$$

- izračunaju se kumulativne dobrote q_k za svaki kromosom tako da vjerojatnost selekcije za svaki kromosom v_k iznosi p_k:

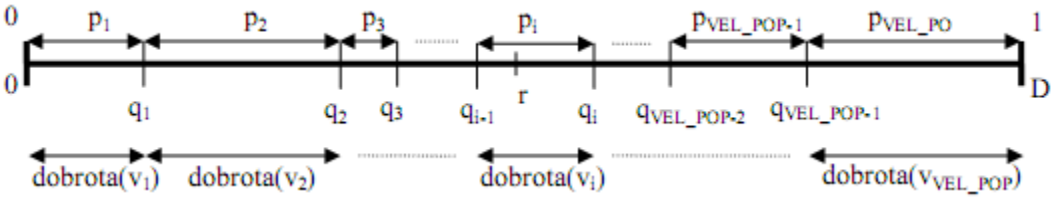
$$q_k = \sum_{i=1}^{k} dobrota(v_i), \text{ gdje je } k = 1,2,\dots,N \tag{2.9}$$

$$p_k = \frac{dobrota(v_k)}{D} \tag{2.10}$$

Ovim se dobiva da je vjerojatnost selekcije proporcionalna dobroti kromosoma:

$$p_k \approx dobrota(v_k) \tag{2.11}$$

- generira se slučajan realan broj r u intervalu $[0, D\rangle$ i potraži se i-ti kromosom za koji vrijedi da je $r \in [q_{i-1}, q_i\rangle$. Taj kromosom se prenosi u sljedeću populaciju



Slika 2.3. Kumulativna dobrota q_i i vjerojatnost selekcije p_i

Slika 2.3. grafički prikazuje mehanizam gore opisane selekcije. Ovaj postupak se ponavlja onoliko puta koliko ima jedinki u populaciji tj. *N* puta. Jedinke s većom vjerojatnošću selekcije će vjerojatno biti izvučene više puta.

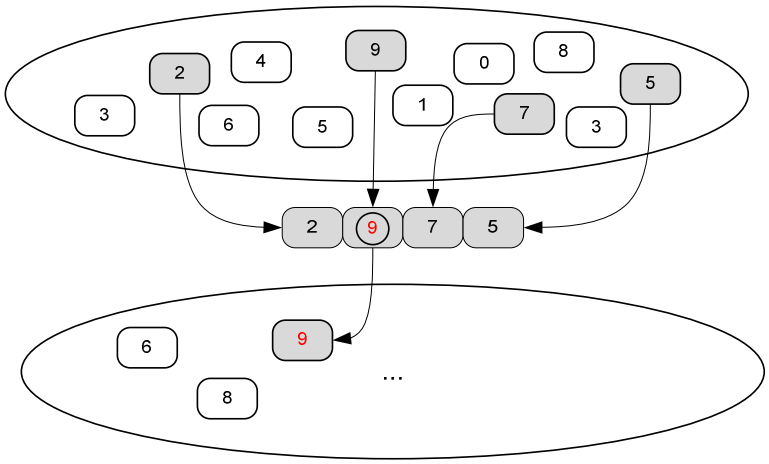
Eliminacijska selekcija (engl. *ready-state selection*) bira *loše* kromosome koje treba eliminirati iz populacije i reprodukcijom ih zamijeniti s novima (djecom nastalom reprodukcijom preživjelih roditelja). Ova selekcija ne koristi dvije populacije niti ne stvara duplikate, koji samo usporavaju algoritam, za razliku od jednostavne generacijske selekcije. Još jedna dobra stvar je što se kod eliminacijske selekcije najbolja jedinka ne može eliminirati, stoga ne treba dodatno raditi *elitizam* (mehanizam zaštite najbolje jedinke od eliminacije).

Kod eliminacijske selekcije, umjesto funkcije dobrote, treba definirati *funkciju kazne*. Vjerojatnost eliminacije proporcionalna funkciji kazne:

$$p_k = kazna(v_k), \quad k = 1,2,\dots,VEL\_POP \tag{2.12}$$

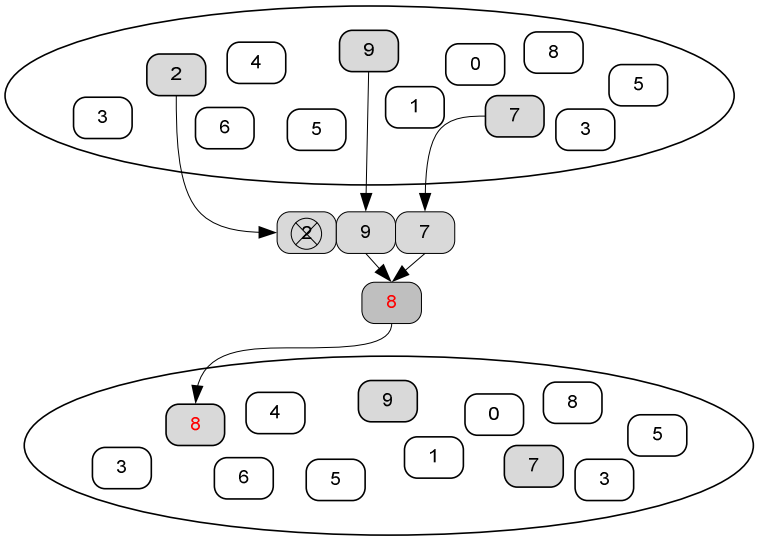
$$kazna(v_k) = \max_i\{dobrota(v_i)\} - dobrota(v_k) \quad \textbf{(2.13)}$$

k-turnirske selekcije *(k=2,3,4,…,N)* s jednakom vjerojatnošću odabire *k* jedinki između kojih selektira najbolju ili najlošiju jedinku. Generacijska turnirska selekcija N puta slučajno odabire k jedinki, između njih selektira najbolju te ju kopira u novu populaciju. Ta će jedinka sudjelovati dalje u reprodukciji.



Slika 2.4. Treći korak generacijske 4-turnirske selekcije

Slika 2.4. prikazuje primjer generacijske 4-turnirske selekcije, i to treći korak. Na ovom primjeru u prva dva koraka su izabrane jedinke s vrijednošću funkcije dobrote 6, odnosno 8. U trećem koraku prvo slučajnim odabirom algoritam izabire 4 jedinke, a onda najbolju od njih (jedinka s dobrotom 9) kopira u novu populaciju.



Slika 2.5. Eliminacijska 3-turnirska selekcija

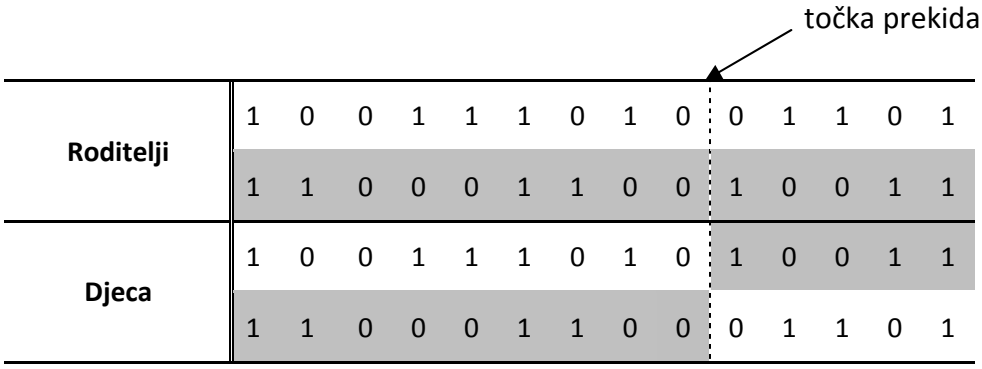
Eliminacijska turnirska selekcija ne stvara novu populaciju nego mijenja već postojeću. Prvo se izabere k jedinki, a onda se izbaci najlošija jedinka. Od preostalih izabranih jedinki slučajnim se odabirom odabiru dva roditelja čijim se križanjem dobiva nova jedinka

(dijete). Ta jedinka zamijeni izbačenu jedinku. Slika 2.5. prikazuje primjer eliminacijske 3-turnirske selekcije. Izabrane su jedinke 2, 9 i 7 tj. jedinke s tim vrijednostima dobrota. Jedinka 2 je najlošija te se ona eliminira, dok jedinke 9 i 7 križanjem daju jedinku s dobrotom 8 koja zamijeni izbačenu jedinku.

2.4.3. Križanje

Križanje (engl. *crossover* ili *crossing over*) je binarni operator u kojem sudjeluju dvije jedinke koje se nazivaju *roditelji*. Križanjem nastaju nove jedinke koje se zovu *djeca*. Djeca nasljeđuju svojstva od roditelja. Pošto su roditelji prošli proces selekcije znači da imaju i dobrih svojstava (imaju relativno dobru funkciju dobrote u odnosu na neke druge jedinke) te će i njihova djeca najvjerojatnije biti dobra, ako ne i bolja od roditelja.

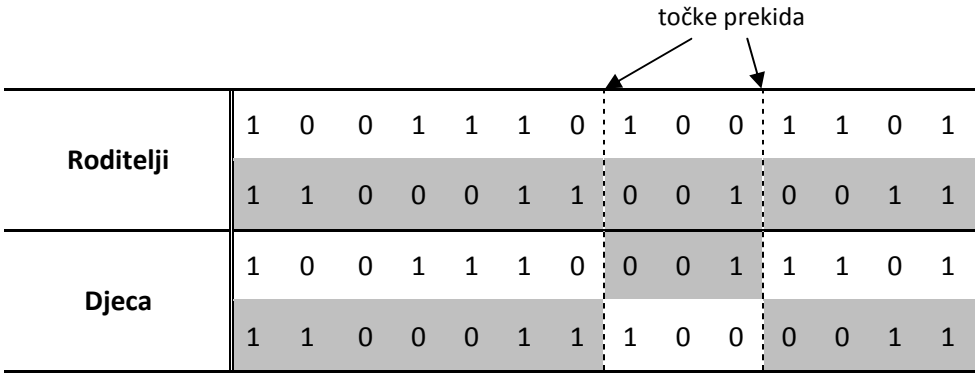
Kod genetskog algoritma ima više načina križanja i ode će biti spomenuti samo neki. Najjednostavniji način križanja je križanje s jednom točkom prekida, koji se vidi na Slika 2.6.



Slika 2.6. Križanje s jednom točkom prekida

Slučajno se odabere jedna točka prekida unutar kromosoma, te se sadržaji kromosoma razmijene. Tim križanjem se dobije dva djeteta. Kromosom prvog djeteta do točke prekida sadrži gene prvog roditelja, a od točke prekida gene drugog roditelja, dok kromosom drugog djeteta obrnuto.

Općenito križanje može biti definirano s proizvoljnim brojem prekidnih točaka, a ne samo s jednom. To se zove *križanjem s m točaka prekida*, s tim da je $m \in [1,2,\dots,n-1]$. Na Slika 2.6 možemo još vidjeti križanje s dvije točke prekida.

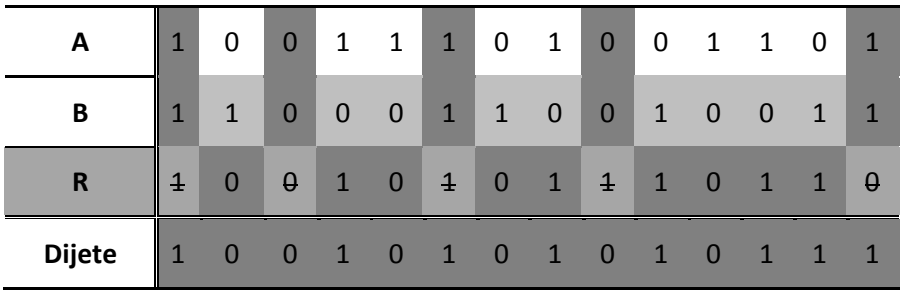


Slika 2.7. Križanje s $m = 2$ točke prekida

Križanje s $m = n - 1$ točaka prekida se naziva *uniformno križanje*. Kod takvog križanja vjerojatnost da dijete naslijedi svojstvo jednog roditelja je 0.5, odnosno jednaka je vjerojatnost nasljeđivanja svojstava za oba roditelja. Ako se ta vjerojatnost razlikuje za pojedine gene (npr. vjerojatnost da će jedan bit biti naslijeđen od jednog roditelja je p=0.3 tj. 30%, a od drugog 70%) tada se takvo uniformno križanje naziva *p-uniformno križanje* [1].

Uniformno križanje se može izvesti kao logička operacija na bitovima. Prvi korak je prenošenje bitova (gena) koji su jednaki, a ostale postaviti na 0. To se radi logičkom operacijom **I**: *A* **I** *B*. Sljedeći korak je slučajno generirati kromosom *R*. Tada logička operacija *R* **I** (*A* ***XILI*** *B*) daje masku slučajnih bitova upravo na mjestima gdje se roditelji razlikuju. Dijete se dobije tako da se na ta dva međurješenja djeluje s operacijom **ILI**:

$$dijete = A \cdot B + R(A \oplus B) \tag{2.14}$$

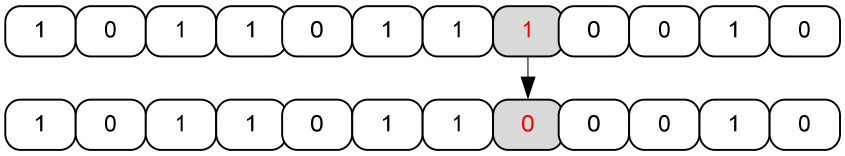


Slika 2.8. Uniformno križanje

2.4.4. Mutacija

Mutacija je slučajna promjena jednog ili više gena u kromosomu, te time dobivamo novu izmijenjenu jedinku. Pošto djeluje nad jednom jedinkom to je unarni operator. Ako genetski algoritam zaglavi u lokalnom optimumu, mutacija je ta koja ga može izbaciti iz tog područja. Ako se pomak pokaže pozitivnim cijela populacija će se brzo preseliti na to

novo područje, gdje se možda i nalazi globalni optimum. Ukoliko mutacija da loše rezultate, jedinka ubrzo odumire jer ne preživi selekciju.



Slika 2.9. Mutacija slučajno odabranog bita kromosoma

U genetskom algoritmu parametar p_m određuje vjerojatnost mutacije, što znači da treba biti između 0 i 1. Ako teži k jedinici onda se genetski algoritam pretvara u algoritam slučajne pretrage prostora rješenja. No, ako teži prema nula onda postupak vrlo lako može zaglaviti u nekom lokalnom optimumu.

Imamo više različitih vrsta mutacija u genetskom algoritmu. *Jednostavna mutacija* mijenja svaki bit s jednakom vjerojatnošću p_m. *Miješajuća mutacija* je vrsta mutacije koja slučajnim postupkom odabire kromosom (a ne samo jedan gen) za mutaciju, prvu i drugu granicu (ili uzorak) i tada ili izmiješa gene ili ih slučajno generira (*potpuna miješajuća mutacija*) ili invertira sve gene (*invertirajuća miješajuća mutacija*).

Ukoliko je kromosom ne koristi binarni prikaz, nego je složenije građe, onda umjesto vjerojatnosti mutacije jednog gena (p_m), koristimo vjerojatnost mutacije kromosoma (p_M). Ako je *n* duljina kromosoma onda vrijedi [1],[4]:

$$p_M = 1 - (1 - p_m)^n \tag{2.15}$$

3. Paralelni genetski algoritmi

3.1. Podjela paralelnih genetskih algoritama

Genetski algoritam služi za rješavanje težih optimizacijskih problema, za koje ne postoji egzaktna matematička metoda rješavanja ili su NP-teški pa se za veći broj nepoznanica ne mogu riješiti u zadanom vremenu.

Genetski algoritam je vremenski zahtjevan i troši najviše procesorskog vremena od bilo koje druge metode optimiranja. Poboljšavanjem mogućnosti računala nastoje se sve teži problemi riješiti u što je moguće kraćem vremenu. Proširivanjem radnog spremnika, dodavanjem procesora te umrežavanjem računala stvara se radno okruženje u kojem se posao može brže obaviti. Evolucija u prirodi se odvija potpuno paralelno. Stoga je za očekivati da je i genetski algoritam, koji je preslika prirodnog evolucijskog procesa, moguće lako paralelizirati. Osnovna ideja paralelizacije nekog algoritma je podijeliti posao na podzadatke koji se mogu izvoditi paralelno, a podzadatke podijeliti računalima, odnosno procesorima. Cilj paralelizacije je skratiti trajanje izvođenja složenih programa, a da se pri tom ne naruše njihova svojstva.

Dakle, za ostvarenje paralelnog genetskog algoritma (PGA) treba odrediti što će se odvijati paralelno. Genetski algoritam ciklički ponavlja jedan te isti posao. Iz iteracije u iteraciju izračunavaju se vrijednosti funkcije cilja i genetski operatori (selekcija, mutacija i križanje) djeluju nad populacijom jedinki. Nameću se dva pristupa paraleliziranja genetskih algoritama:

- ***standardni pristup*** – paralelizirati genetske operatore i izračunavati vrijednosti funkcije cilja paralelno ili
- ***dekompozicijski pristup*** – podijeliti populaciju na manje dijelove - subpopulacije i obavljati cijeli genetski algoritam nad subpopulacijama.

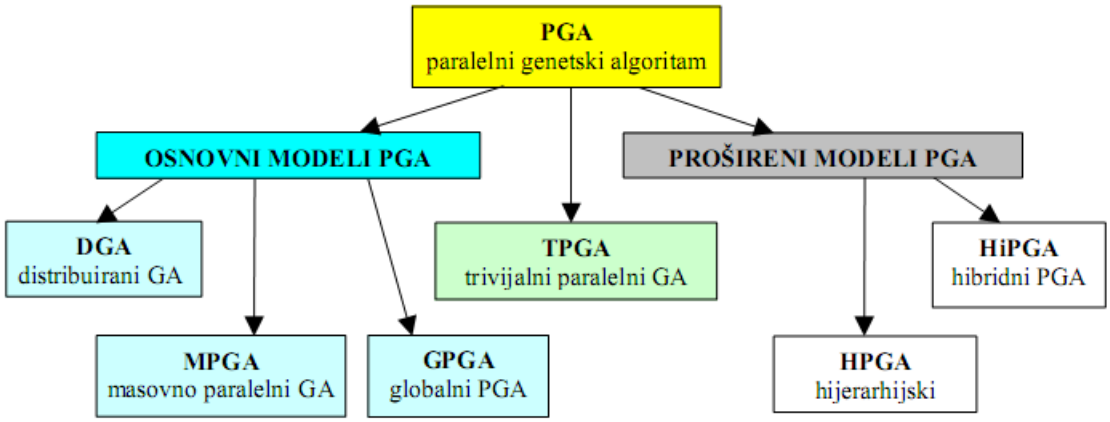
U prvom slučaju se paralelizira samo posao evaluacije (postupak izračunavanja vrijednosti funkcije cilja). Jednako kao i kod sekvencijalnog genetskog algoritma, genetski operatori djeluju samo nad jednom, zajedničkom populacijom pa se takav model naziva jednopopulacijski model. U drugom slučaju se populacija dijeli na nekoliko subpopulacija pa se takav model naziva višepopulacijski model, odnosno genetski algoritam se naziva multipopulacijskim paralelnim genetskim algoritmom. Očekuje se skraćenje trajanja izvođenja multipopulacijskog PGA jer subpopulacije broje manje jedinki od inicijalne populacije pri jednopopulacijskom modelu.

Postoje nekoliko mogućih razina paraleliziranja genetskih algoritama: paraleliziranje na razini populacije, na razini jedinki te na razini evaluacije. Prema razini paralelizacije, postoje tri osnovna načina podjele sekvencijalnog genetskog algoritma na podzadatke:

- **Krupnozrnata podjela** – podjela velike populacije na manje dijelove (subpopulacije).
- **Sitnozrnata podjela** – ekstremni oblik podjele velike populacije na subpopulacije veličine jedne jedinke. Svaki procesor obavlja genetske operatore nad njemu dodijeljenom jedinkom i nad susjednim jedinkama.

- **Podjela na gospodara i sluge** – u svakoj iteraciji gospodar sekvencijski izvodi genetske operatore nad zajedničkom populacijom, dok sluge paralelno izračunavaju vrijednosti funkcije cilja (obavljaju evaluaciju)

Kod prvih dviju podjela se radi o dekompozicijskom pristupu i riječ je o višepopulacijskom modelu genetskog algoritma, dok se u podjeli na gospodara i sluge radi o standardnom pristupu, odnosno o jednopopulacijskom modelu, jer gospodar obavlja genetski algoritam nad zajedničkom populacijom [2].



3.1. Podjela paralelnih genetskih algoritama

U ovisnosti o načinu podjele genetskog algoritma na podzadatke, tri su *osnovna modela* paralelnih genetskih algoritama[2]:

1) Distribuirani genetski algoritam (DGA)
2) Masovno paralelni genetski algoritam (MPGA)
3) Globalni paralelni genetski algoritam (GPGA)

Ta tri osnovna modela se mogu međusobno kombinirati ili nadograditi s nekom drugom metodom optimiranja. Prema tome postoje još dva *proširena modela* paralelnih genetskih algoritama:

4) Hijerarhijski paralelni genetski algoritam (HPGA)
5) Hibridni paralelni genetski algoritam (HyPGA)

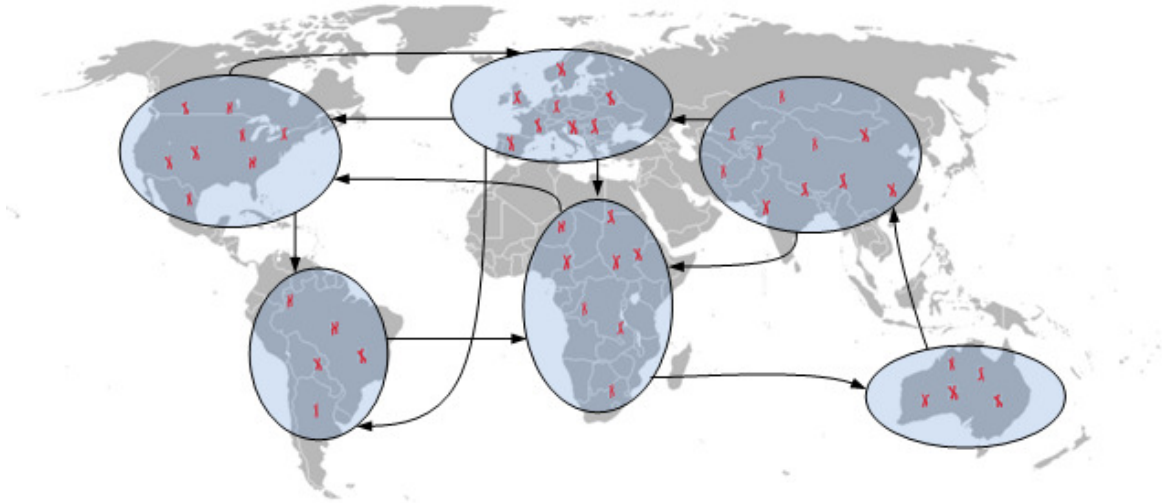
Posljednji, ali i najjednostavniji model paralelnih genetskih algoritama je:

6) Trivijalni paralelni genetski algoritam (TPGA)

3.2. Osnovni modeli PGA

3.2.1. Distribuirani genetski algoritam

Distribuirani genetski algoritam (engl. *distributed genetic algorithm*) ili raspodijeljeni genetski algoritam sastoji se, prema krupnozrnatoj podjeli, od nekoliko subpopulacija pa se naziva još i višepopulacijski genetski algoritam. Model DGA je nastao kako bi se umrežena računala iskoristila za paralelno obavljanje genetskih algoritama. Na svakom čvoru (računalu ili procesoru u višeprocesorskom sustavu) se obavlja sekvencijski genetski algoritam nad jednom subpopulacijom. Genetski algoritmi se mogu međusobno i razlikovati, ali zato imaju zajednički optimizacijski problem koji rješavaju paralelno. Subpopulacije su relativno izolirane kako bi genetski algoritam pretraživao različite dijelove prostora rješenja, no ipak postoji komunikacija među njima. Čvorovi međusobno komuniciraju i razmjenjuju jedinke preko komunikacijskog kanala u nadi da će novo pristigla jedinka u novoj okolini potaknuti pretraživanje još neistraženog područja prostora rješenja i na taj način postići još bolje rješenje. Ta razmjena se zove *migracija*.



Slika 3.2. Distribuirano paralelni genetski algoritam

DGA je najpopularniji model paralelnih genetskih algoritama, jer je najjednostavniji za implementaciju na umreženim računalima, ali i zbog analogije s prirodom. Ima mnogo primjera evolucije u prirodi gdje ista vrsta živi u potpuno razdvojenim okolinama na različitim kontinentima. Isto tako u prirodi ima stalnih migracija pojedinih vrsta na druge kontinente.

Prednost DGA u odnosu na sekvencijalni je ta što podjelom populacije na subpopulaciju genetski algoritam obavlja genetske operatore na manjim brojem jedinki, a i raspodijeljena populacija sprječava preranu konvergenciju prema lokalnom optimumu. Međutim vrijeme trajanja izvođenja programa nije kraće onoliko puta koliko je manja subpopulacija od populacije. Vrijeme trajanja razmjene jedinki (komunikacije) i vrijeme potrebno za sinkronizaciju znatno usporavaju genetski algoritam. No pravilnim odabirom

broja jedinki za migraciju i strategije odabira jedinki može se postići *superlinearno ubrzanje*[1].

Veliki broj novih parametara je veliki nedostatak DGA, jer je zbog toga postupak podešavanja algoritma za rješavanje specifičnog optimizacijskog problema otežan. Ima čak sedam dodatnih parametara, dva su potrebna za podjelu populacije:

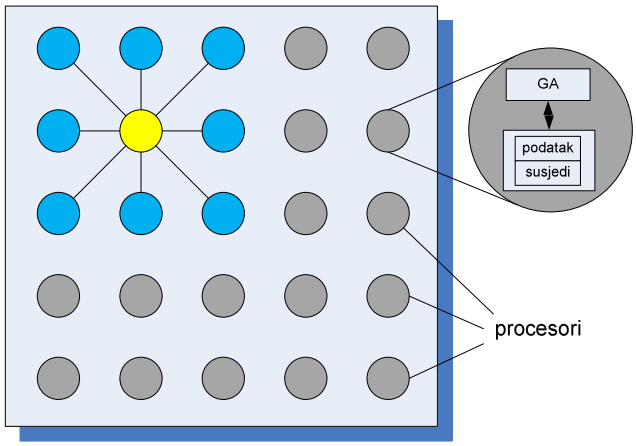
- *broj subpopulacija*
- *veličina pojedine subpopulacije*

te pet za podešavanje migracije:

- M_i - migracijski interval ili period izmjene jedinki između procesora,
- M_s - migracijska stopa ili broj jedinki koja se izmjenjuje,
- *strategija odabira boljih jedinki*,
- *strategija odabira jedinki za eliminaciju* i
- *topologija razmjene jedinki* [2]

3.2.2. Masovno paralelni genetski algoritam

Masovno paralelni genetski algoritam (engl. *massively parallel genetic algorithm*) se prema sitnozrnatoj podjeli sastoji od N_p procesora koji predstavljaju N_p jedinki. Dakle, veličina populacije je jednaka broju procesora ($N = N_p$). Svaki procesor obavlja genetske operatore nad svojom jedinkom i nad susjednim jedinkama.



Slika 3.3. ***Masovno paralelni genetski algoritam***

Evaluaciju i mutaciju obavljaju nad pripadajućom jedinkom koja se nalazi u internoj memoriji tog procesora, a križanje nad svojom jedinkom i na nekoj od susjeda. Selekcijom se odabire susjedna jedinka s kojom će se obavit križanje. Migracija kod MPGA se odvija samo između susjednih procesora. Za razliku od ostalih modela paralelnih genetskih algoritama, model masovno paralelnih genetskih algoritama zahtijeva višeprocesorsko računalo koje se sastoji od mnogo procesora.

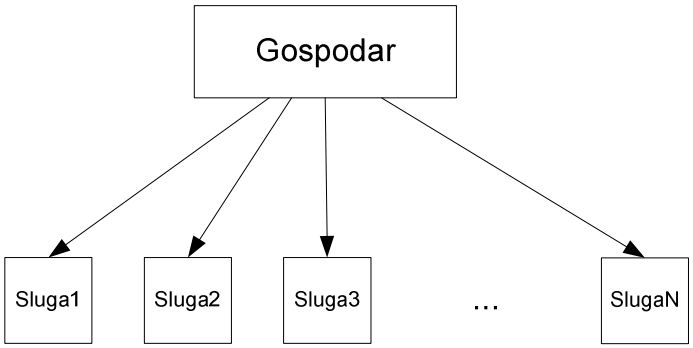
[1] *superlinearno ubrzanje* – u paralelnom programiranju to je ubrzanje koje je veće od broja procesora koji se koriste: $a > N_p$

Slično kao i kod DGA, populacija je na neki način podijeljena na subpopulacije. Svaki procesorski element je povezan sa svojim susjedima. Dakle, veličina subpopulacije je jednaka broju susjeda plus jedan (pripadajuća jedinka). Što je broj susjeda manji jedinke su izoliranije. Zbog preklapanja subpopulacija omogućeno je brzo širenje dobrih rješenja po cijeloj populaciji. S porastom broja susjeda algoritam dobiva sve lošija svojstva, jer se suboptimalna rješenja (lokalni optimumi) u početku evolucijskog algoritma prebrzo prošire po cijeloj populaciji. Neka je broj susjeda mali i neka je algoritam u početku pronašao lokalni optimum u jednoj od procesorskih elemenata. Taj krivi lokalni optimum se neće brzo proširiti cijelom populacijom, jer su subpopulacije udaljenih procesorskih elemenata međusobno izolirane. Za vrijeme dok se lokalni optimum sporo širi, zbog malog broja susjeda, genetski algoritam ima vremena pronaći neko bolje rješenje u drugim područjima prostora rješenja. Stoga je broj susjeda obično puno manji od ukupne veličine populacije.

Prednost MPGA je ta što se postiže gotovo linearno ubrzanje s porastom broja procesora, ali nedostaci su što je potrebno podešavati nova dva parametra (*topologiju* i *broj susjeda*), te zahtjev za računalom s velikim brojem procesora. U slučaju prevelikog broja susjeda može se javiti i problem da komunikacijski kanal postane usko grlo algoritma.

3.2.3. Globalni paralelni genetski algoritam

Globalni paralelni genetski algoritam je predstavnik podjele na gospodara i sluge (*master-slave genetic alghoritm* ili *micro-grained G – mgGA*). Tradicionalni GPGA se sastoji od jednog gospodara i više sluga koji obavljaju samo evaluaciju.



Slika 3.4. ***Globalni paralelni genetski algoritam***

Radi se o jednopopulacijskom modelu, jer dretva gospodar ima pohranjenu cijelu populaciju u svojem korisničkom segmentu podataka, nad kojom obavlja sve ostale genetske operatore osim evaluacije. Osim genetskih operatora gospodar raspodjeljuje posao slugama i po potrebi sinkronizira proces razmjene jedinki. Komunikacija između gospodara i sluga odvija se u dva navrata: kada gospodar slugama šalje genetski materijal za evaluaciju i kada sluge vraćaju gospodaru izračunate vrijednosti funkcije cilja. Iako sluga najčešće obavljaju samo evaluaciju, oni mogu obavljati i ostale genetske operatore, ali se tada izostavlja atribut tradicionalni.

Kod računalnog sustava s distribuiranim (raspodijeljenim) radnim spremnikom, samo gospodar ima pristup populaciji. Tu gospodar, nakon što obavi selekciju, križanje i mutaciju, mora poslati slugama cijeli genetski materijal novostvorenih jedinki na

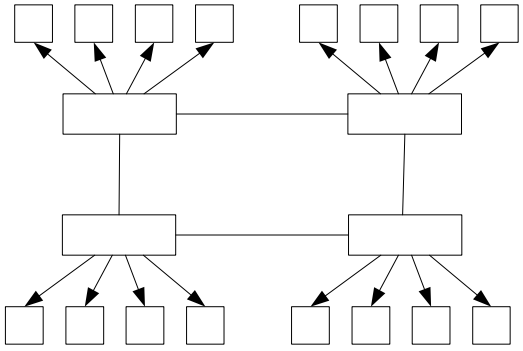
evaluaciju. Dok kod višeprocesorskih računala sa zajedničkim radnim spremnikom populacija je zajednička svim dretvama, stoga svaki sluga može pristupati i obavljati evaluaciju nad njemu dodijeljenim jedinkama. Gospodar dinamički dodjeljuje slugama jedinke tijekom izvođenja programa. Pošto gospodar i sluge koriste zajednički memorijski prostor, gospodar ne treba poslati slugama cijeli genetski materijal nego je dovoljno samo redni broj jedinke ili kazaljku na jedinku. Isto tako sluge ne trebaju natrag slati izračunate vrijednosti gospodaru nego ih mogu direktno upisati u memoriju gdje se nalazi dodijeljena jedinka. Time se znatno uštedi na komunikaciji, što značajno ubrza rad algoritma.

Komunikacija se obavlja na početku i na kraju faze evaluacije. Kako bi se postiglo što veće ubrzanje s porastom broja procesora vrijeme komunikacije treba biti puno manje od vremena potrebnog za izračunavanje funkcije cilja. U idealnom slučaju, kad bi vrijeme za komunikaciju bilo jednako nula, ubrzanje bi bilo proporcionalno broju procesora.

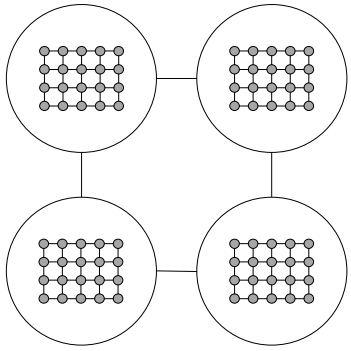
3.3. Prošireni modeli PGA

3.3.1. Hijerarhijski paralelni genetski algoritam

Hijerarhijski paralelni genetski algoritam (engl. *hierarchical parallel genetic algorithm* ili HPGA) se dobiva kombinacijom dva od tri osnovna modela PGA, pa čak i kombinacijom sva tri modela. Primjerice, to može biti distribuirani genetski algoritam na nekoliko međusobno povezanih računala, a na svakom od računala obavlja se globalni paralelni genetski algoritam nad subpopulacijama (Slika 3.5).



Slika 3.5. DGA/GPGA hijerarhijski PGA



Slika 3.6. DGA/MPGA hijerarhijski PGA

Hijerarhijski modeli značajno ubrzavaju izvođenje algoritma, jer je njihovo ukupno ubrzanje produkt ubrzanja pojedinih modela od kojih se sastoji. Npr. neka je globalno paralelni genetski algoritam a_g puta brži, a distribuirani genetski algoritam a_d puta brži od sekvencijskog genetskog algoritma. Tada je hijerarhijski model koji kombinira ta dva modela točno $a_g \cdot a_d$ puta brži od serijske verzije programa [2]. To je najveća prednost hijerarhijskog modela. Nedostatak tog modela je veliki zahtjevi nad sklopovljem, te veliki broj parametara za podešavanje (ukupan broj parametara jednak je sumi svih parametara pojedinih modela od kojih se sastoji HPGA).

Hijerarhijski model gotovo u pravilu ima na višem nivou DGA, ali mogao bi tu biti i masovno paralelni genetski algoritam. Tada bi na nižem nivou mogao biti sam distribuirani genetski algoritam ili DGA u kombinaciji s GPGA.

3.3.2. Hibridni paralelni genetski algoritam

Hibridni paralelni genetski algoritam (engl. *hybrid paralel genetic algorithm* ili HyPGA) je kombinacija jednog od prethodna četiri modela s nekim drugim algoritmom za lokalno pretraživanje. Najčešće se radi o nekoj od gradijentnih metoda (metoda najbržeg spusta, Newton-Raphsonova metoda), koje se primjenjuju nakon određenog broja iteracija i to samo nad nekim jedinkama. No, može se koristiti i neka druga optimizacijska metoda (traženje po koordinatnim osima, postupak po Hookeu i Jeevesu ili simpleks postupak po Nelderu i Meadu).

```
Hibridni_paralelni_genetski_algoritam(){
   generiraj_paralelno_populaciju_slučajnih_jedinki();
   dok (nije_zadovoljen_uvjet_završetka_evolucijskog_procesa){
      evaluiraj();          // evaluiraj paralelno svaku jedinku
      optimiraj_lokalno(); // za svaku jedinku paralelno obavi lokalno
                              pretraživanje uporabom neke gradijentne
                              metode
      selektiraj();         // selektiraj paralelno jedinku za
                              reprodukciju među susjedima
      reproduciraj();       // obavi paralelno reprodukciju među
                              odabranom jedinkom iz prethodnog koraka
                              i vlastitom jedinkom
      optimiraj_lokalno(); // dijete paralelno optimiraj lokalno

      ako (je dijete bolje od roditelja){
         nadomjesti();      // nadomjesti paralelno vlastitu jedinku
                              s boljom novodobivenom jedinkom
      }
   }
```

Slika 3.7. Primjer hibridnog paralelnog genetskog algoritma [2]

Hibridni PGA ima sva dobra svojstva sekvencijalnog genetskog algoritma. S tim da je poboljšano fino podešavanje rješenja. Još jedna prednost je ta što cijeli algoritam brže konvergira, no kod hibridnog PGA postoji veća vjerojatnost zaostajanja u lokalnom optimumu zbog manje raspršenosti rješenja.

3.4. Trivijalni paralelni genetski algoritam

Trivijalni paralelni genetski algoritam se izvorno na engleskom jeziku zove *embarassingly parallel genetic algorithm*. Radi se o više genetskih algoritama koji se paralelno obavljaju na više potpuno nezavisnih računala. Ovaj algoritam služi kako bi se statistički obradili eksperimentalno dobiveni rezultati ili kako bi se odredio optimalan skup parametara.

Ova trivijalna paralelna metoda je izuzetno korisna za statističku analizu algoritma. Isti algoritam se paralelno pokrene na nekoliko odvojenih računala s različitim početnim uvjetima i promatra se kvaliteta dobivenog rješenja. Potom se rezultati statistički obrađuju i međusobno se uspoređuju.

3.5. Idealni paralelni genetski algoritam

Zahtjevi koje bi trebao zadovoljiti *idealni* paralelni genetski algoritam su [2]:

- Paralelni genetski algoritam treba biti približno onoliko puta brži od sekvencijalnog koliko računalni sustav ima procesora
- Broj parametara bi trebao biti što manji kako bi se lakše obavilo podešavanje algoritma
- Rješenje koje se dobije uporabom PGA bi trebalo biti iste kvalitete kao i rješenje dobiveno sekvencijalnim GA

Distribuirani genetski algoritam je pogodan za izvođenje na svim paralelnim arhitekturama, što je ujedno i glavna prednost tog modela. Nedostatak DGA je problem velikog broja dodatnih parametara (kojih ima čak sedam [2]), stoga DGA ne zadovoljava drugi i treći uvjet *idealnog PGA*. Masovno paralelni genetski algoritam se može implementirati samo na računalima s velikim brojem procesora (50 i više), a osim toga i taj model ima dodatne parametre: veličinu i topologiju susjeda, stoga ni on ne zadovoljava drugi i treći uvjet. Tradicionalni GPGA je kao i DGA pogodan za sve paralelne arhitekture i njegova prednost je što nema novih parametara, dakle ima ista svojstva kao i sekvencijalni GA. Problem kod GPGA je što se paralelno odvija samo evaluacija, jer se pretpostavlja da ona traje značajno više od ostalih genetskih operatora. Tradicionalni GPGA ne zadovoljava prvi uvjet, tj. nije ubrzane proporcionalno broju procesora, ako evaluacija ne traje znatno duže od svih ostalih genetskih operacija koje se ne obavljaju zajedno.

Kako bi GPGA zadovoljio i prvi uvjet treba uvesti novi model GPGA koji bi trebao sadržavati sve karakteristike tradicionalnog modela, a da se pritom svi genetski operatori izvode paralelno [2]. Time bi se postiglo ubrzanje otprilike jednako broju procesora.

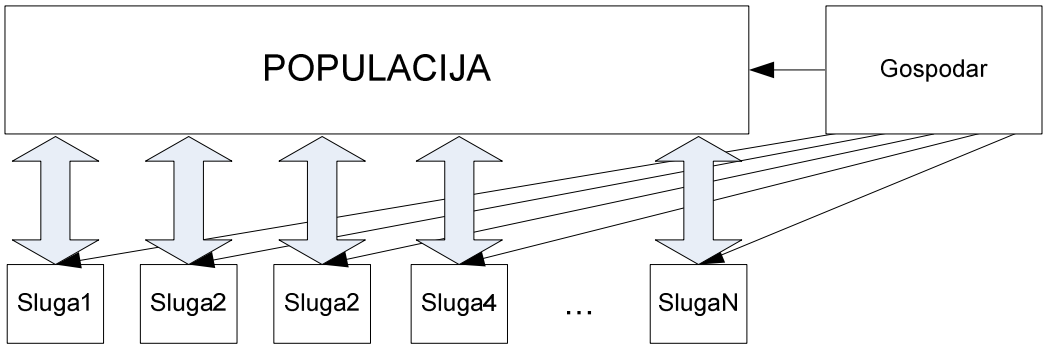
3.5.1. GPGA gdje sluge obavljaju sve operatore

Predloženi model GPGA gdje sluge obavljaju sve operatore koristi zajedničku populaciju, ali u pojedinim trenutcima je raspodjeljuje na subpopulacije. U svakoj iteraciji pojedina dretva slučajno odabere *k* jedinki koje čine subpopulaciju. Svaka dretva selekciju i reprodukciju obavlja sekvencijalno nad svojom subpopulacijom, slično kao i kod DGA.

Razlika između DGA i ovog modela GPGA je u tome što DGA ima podijeljenu populaciju stalno, tijekom cijelog evolucijskog postupka, dok GPGA dinamički mijenja jedinke koje čine subpopulaciju u svakoj iteraciji.

Eliminacijska turnirska selekcija omogućava obavljanje i selekcije i reprodukcije nad jednom jedinkom u istom koraku. U svakoj iteraciji genetski algoritam odabire *k* jedinki te najlošiju od njih zamijeni s djetetom preostalih jedinki. Kako u križanju sudjeluju dva roditelja od preostalih *k-1* jedinki, idealni slučaj je da su ta dva roditelja jedine preostale jedinke ($k - 1 = 2$, tj. $k = 3$). Osim toga roditelji trebaju biti različiti jer će u suprotnom križanjem nastati duplikat. To znači da je za ovaj model GPGA najpogodnija *3-turnirska eliminacijska selekcija bez duplikata*. Ovakav model je pogodan za paralelizaciju svih genetski operatora: selekcije, križanja, mutacije i evaluacije.

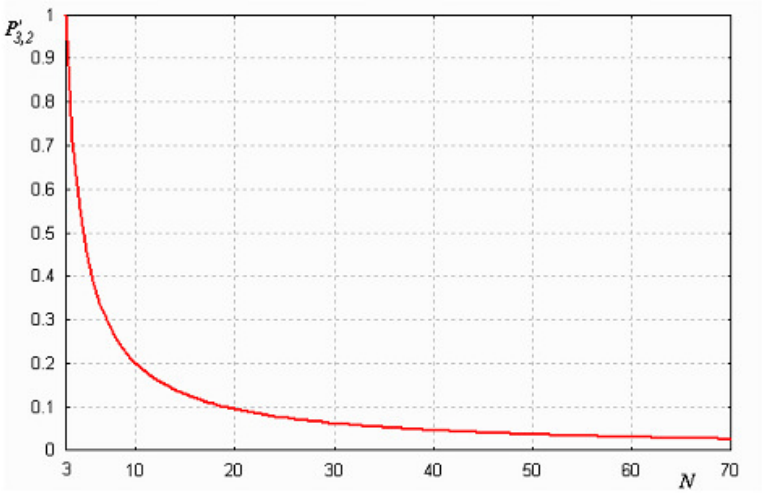
GPGA gdje sluge obavljaju sve operatore najsličniji je tradicionalnom GPGA, ali se razlikuje od njega u poslu koji gospodar i sluge obavljaju. U tradicionalnom sluge obavljaju paralelno evaluaciju, a gospodar sve ostale genetske operatore, dok u novom sluge obavljaju gotovo sav posao paralelno. Gospodar u novom modelu samo inicijalizira populaciju i pokrene dretve sluge, te još ispiše rezultate.



3.8. Model GPGA gdje sluge obavljaju sve operatore

Ukoliko se pri selekciji jedinki ne vodi računa je li neka druga dretva pristupa istim podacima u isto vrijeme može se, čitanjem i pisanjem po istim memorijskim lokacijama, narušiti jednoznačnost podataka. Samo jedna jedinka se mijenja pri križanju i mutaciji, i to najlošija jedinka. Ako je kod više dretvi ta ista jedinka najlošija i one u isto vrijeme pokušavaju zapisati svoju novodobivenu jedinku umjesto te, samo će jedna dretva uspjeti u tome (ona koja zadnja zapiše novu jedinku na memorijsku lokaciju stare jedinke). Sve ostale dretve su uzalud obavljale svoj posao. Posljedica višestruke selekcije za eliminaciju jedne te iste jedinke je smanjenje broja iteracija u odnosu na odgovarajući sekvencijalni algoritam (GPGA sa samo jednom dretvom). Povećavanjem broja iteracija može se postići da paralelni model obavlja isti broj *korisnih* iteracija kao što sekvencijalni ima običnih iteracija. Drugi način za rješavanje tog problema je sinkronizacija dretvi prilikom pristupa istim podacima. Svako čitanje i izmjena zajedničkih podataka treba se označiti kao kritični odsječak i zaštiti ga nekim od mehanizama za međusobno isključivanje (semafori, uvjetne varijable, varijable međusobnog isključivanja, itd.). Oba rješenja usporavaju algoritam.

Vjerojatnost da dretva obavlja posao uzalud je funkcija veličine turnira *k*, broja dretvi *D* i veličine populacije *N*. Slika 3.9. prikazuje vjerojatnost da dvije dretve selektiraju istu jedinku za eliminaciju ovisno o veličini populacije *N*, pri 3-turnirskoj selekciji [2].



Slika 3.9. Vjerojatnost dvostruke selekcije s dvije dretve i 3-turnirskom selekcijom

Osim što se svi genetski operatori izvode paralelno, ovaj model GPGA ima i mnoge druge prednosti. Jednostavan je za implementaciju. Najpogodniji je za izvršavanje na višeprocesorskom sustavu sa svega nekoliko procesora (2, 4 ili 8) i sa zajedničkim spremnikom. No isto tako se može, bez izmjene u izvornom kodu, izvoditi na proizvoljnom broju procesora. Zbog toga što koristi eliminacijsku 3-turnirsku selekciju nije potrebno dodavati *elitizam*, jer je on inherentno ugrađen u navedenoj selekciji. Jedna od velikih prednosti GPGA gdje sluge obavljaju sve operatore je ta što nema potrebe za međusobnom komunikacijom između dretvi, jer imaju zajedničku populaciju. Time se ne gubi vrijeme na komunikaciju koja kod paralelizacije zna biti veliki problem. Unatoč tome što su podaci zajednički, moguće je izvoditi dretve paralelno bez ikakve sinkronizacije. Trajanje izvođenja jedne iteracije ne ovisi o broju populacije, jer svaka dretva u jednoj iteraciji djeluje nad tri slučajno izabrane jedinke, a ne nad svih *N* jedinki.

GPGA gdje sluge obavljaju sve operatore postiže nad N_p-procesorskom računalu rješenje iste kvalitete kao i odgovarajući sekvencijalni algoritam na jednoprocesorskom računalu, ali gotovo N_p puta brže.

4. Neuronske mreže

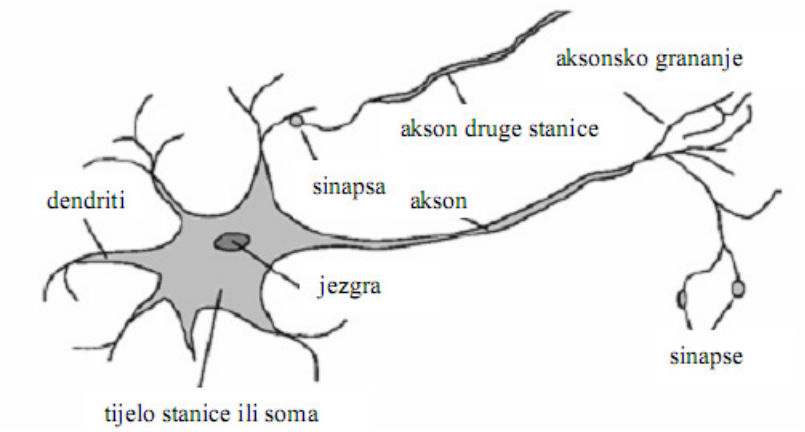
4.1. Uvod u neuronske mreže

Mnogi poslovi koji uključuju umjetnu inteligenciju ili raspoznavanje uzoraka znaju bit prilično teški za izvest na računalu konvencionalnim metodama, a tako jednostavno ih mogu izvoditi životinje i ljudi. Ptice grabljivice iz velikih visina lako primijete svoj plijen na tlu, tj. gledajući tlo raspoznaju što je njihov plijen a što nije. Proučavajući kako živa bića obavljaju poslove i simulirajući te procese, dokle je to moguće fizički izvesti na računalu, računarstvo je mnogo napredovalo u umjetnoj inteligenciji.

U računarstvu algoritam koji se zove umjetne neuronske mreže se temelji na biološkoj neuronskoj mreži, tj. mozgu živih bića. Dakle moglo bi se reći da je umjetna neuronska mreža replika ljudskog mozga kojom se nastoji simulirati postupak učenja i obrade podataka.

4.1.1. Mozak i živčana stanica

Mozak živih bića je dio živčanog sustava i sastoji se od velikog broja međusobno povezanih živčanih stanica tzv. neurona. Ljudski mozak se sastoji od oko 100 milijardi (10^{11}) neurona. Naravno da su umjetne neuronske mreže puno jednostavnije od ljudskog mozga, te sadrže puno manji broj neurona.



Slika 4.1. Osnovna struktura biološkog neurona

Biološki neuron se sastoji od *some* (staničnog tijela) i dviju vrsta izdanaka: *dendrita* i *aksona*.

- ***soma*** ili ***stanično tijelo*** - centralni dio neurona koji upravlja metaboličkim procesima u stanici (sintezom bjelančevina iz aminokiselina koje su neophodne za funkcioniranje).
- ***dendriti*** - više kratkih nastavaka na staničnom tijelu koji služe za primanje informacija tj. podražaja. Oni se granaju u veći broj manjih nastavaka, tvoreći tzv. dendritičko razgranjenje. Time povećavaju površinu dendrita, a time i broj kontakata koje jedan neuron može imati s drugim neuronima.

- ***akson*** - jedini nastavak na neuronu čija je funkcija prenijeti živčani signal od staničnog tijela prema drugim stanicama. Uglavnom neuroni imaju samo jedan akson, ali na kraju se on grana u niz tankih niti tako da može prenositi podražaj prema većem broju drugih neurona.

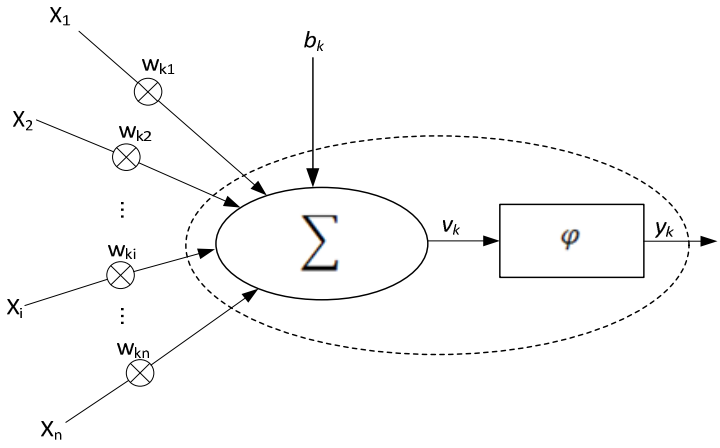
Između aksona jedne stanice i dendrita druge postoji mala pukotina koja se zove *sinapsa.* Preko sinapse se vrši komunikacija između neurona, tj. prenosi se podražaj s jednog neurona na drugi.

Pojedini neuron prima električne (živčane) signale od drugih neurona. Oni se zbrajaju i kad pređu kritičnu razinu taj neuron, koji je primao impulse, stvara električni impuls koji šalje duž aksona koji se grana na mnogo malih čvorića (aksonskih završetaka). Prijelaz električnog signala od jednog neurona do drugog, preko sinapse, obavljaju neurotransmiteri. Neurotransmiteri su kemijske tvari koje se izlučuju u sinaptičku pukotinu s aksonskih završetaka neurona. S druge strane sinapse se nalaze receptori, na dendritima susjednog neurona, na koje se vežu ti neurotransmiteri. U kolikoj će se mjeri taj signal prenijeti s jednog na drugi neuron ovisi o mnogo faktora, kao što su količina dostupnih neurotransmitera, broj i raspored receptora, količina neurotransmitera koji će se razgraditi itd.

Mozak uči tako da mijenja efikasnost pojedine veze između neurona, te stvaranjem novih, odnosno ukidanjem starih veza. Pošto mozak uči na temelju iskustva i efikasnost sinapse se mijenja kao rezultat iskustva. Jedan od načina promjene efikasnosti je promjena količine dostupnih neurotransmitera za izlučivanje, ali i mnoge druge promjene mogu nastati [6].

4.1.2. Neuron – osnovna jedinica neuronske mreže

U umjetnoj neuronskoj mreži obrada se informacija također izvodi u jedinicama koje zovemo neuronima ili elementima za obradu. Umjetni neuron je osnovna jedinica umjetne neuronske mreže. Slika 4.2. prikazuje model neurona koji je napravljen po uzoru na biološki neuron živih bića.



Slika 4.2. Umjetni neuron

Može se identificirati tri osnovna elementa neurona [7]:

- skup sinaptičkih težina - pojedini signal x_i, koji ulazi u neuron, se množi s odgovarajućom sinaptičkom težinom w_i. Te težine mogu imati i negativne i pozitivne vrijednosti
- zbrajalo - zbraja sve ulazne signale u neuron
- aktivacijska funkcija - ograničava amplitudu izlaznog signala neurona, što znači da je izlaz iz samog neurona funkcija aktivacijske funkcije

Dakle umjetni neuron ima više ulaza od kojih prima informacije, zbraja ih, te na tu sumu djeluje aktivacijskom funkcijom. Dobivena vrijednost, nakon aktivacijske funkcije, je izlaz iz neurona.

Analogija biološkog i umjetnog neurona:

- električni (živčani) signali –> numeričke vrijednosti
- tijelo stanice –> zbrajalo
- akson –> aktivacijska (prijenosna) funkcija φ
- jakost sinapse –> težinski faktor w

Neuron k se može matematički prikazati sljedećim jednadžbama:

$$v_k = w_{k1}x_1 + w_{k2}x_2 + \cdots + w_{ki}x_i + \cdots + w_{kn}x_n - b_k$$

Ako se uzme da je

$$w_0 = -b_k,$$

te neka je

$$x_0 = 1$$

gornju jednadžbu možemo zapisati kao:

$$v_k = w_{k0}x_0 + w_{k1}x_1 + w_{k2}x_2 + \cdots + w_{ki}x_i + \cdots + w_{kn}x_n$$

ili skraćeno:

$$v_k = \sum_{i=1}^{n} w_{ki}x_i \qquad \textbf{(4.1)}$$

Još ostaje izlaz iz neurona nakon aktivacijske funkcije, koji je jednak:

$$y_k = \varphi(v_k) = \varphi(\sum_{i=1}^{n} w_{ki}x_i) \qquad \textbf{(4.2)}$$

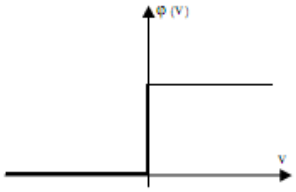
U gornjim jednadžbama x_1, x_2, ..., x_n su ulazni signali, w_{k1}, w_{k2}, ..., w_{kn} su sinaptičke težine neurona k, v_k je zbroj ulaznih signala pomnoženih njihovim težinama, b_k je prag, φ je aktivacijska funkcija, a y_k je izlazni signal neurona.

Prag b_k je eksterni parametar neurona i ima efekt afine transformacije[2] zbroja ulaznih vrijednosti dobivenog zbrajalom. Prag općenito omogućuje da decizijska ravnina, koja odvaja razrede, ne mora prolaziti kroz ishodište prostora rješenja.

Za aktivacijsku funkciju najčešće se koriste sljedeće funkcije:

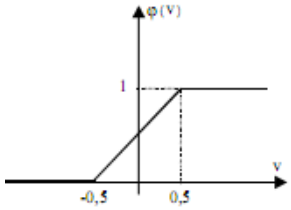
1) *Step funkcija*

$$\varphi(v) = \begin{cases} 1, & v \geq 0 \\ 0, & v < 0 \end{cases}$$



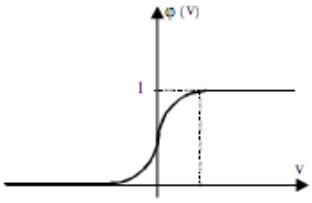
2) *Funkcija praga*

$$\varphi(v) = \begin{cases} 1, & v \geq 0.5 \\ v + 0.5, & -0.5 < v < 0.5 \\ 0, & v \leq -0.5 \end{cases}$$



3) *Sigmoidalna funkcija*

$$\varphi(v) = \frac{1}{1 + \exp(-av)}$$



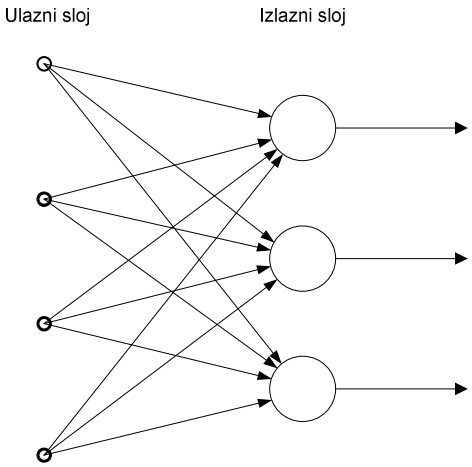
4.2. Jednoslojna neuronska mreža

Neuroni su spojeni u mrežu tako da izlaz svakog neurona predstavlja ulaz u jedan ili više drugih neurona. Prema smjeru, veza između neurona može biti jednosmjerna ili dvosmjerna. Neuroni su obično u neuronskoj mreži organizirani u slojeve u kojima se informacije paralelno obrađuju.

Jednoslojna neuronska mreža sastoji se od jednog sloja neurona tzv. *izlaznog sloja*. Ali osim izlaznog sloja neurona, jednoslojna mreža sadrži i *ulazni sloj*, koji sadrži ulazne podatke. On se ne broji kao sloj neurona, jer u njemu nema nikakvog računanja. Ulazi u mrežu su spojeni na ulaze neurona izlaznog sloja, a izlazi neurona predstavljaju i izlaz mreže. Nema povratnih veza s izlaza na ulaz.

Slika 4.3. prikazuje primjer jednoslojne neuronske mreže koja u ulaznom sloju ima četiri, a u izlaznom tri neurona.

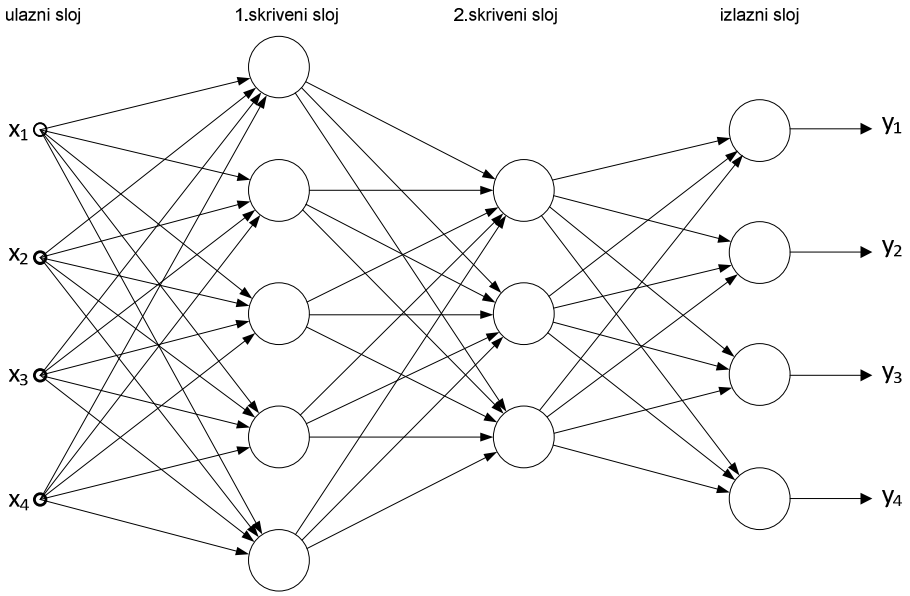
[2] Afina transformacija je transformacija vektorskih prostora koja se sastoji od linearne transformacije i translacije: $x \mapsto Ax + b$



Slika 4.3. Jednoslojna neuronska mreža

4.3. Višeslojna neuronska mreža s propagacijom unaprijed

Višeslojna neuronska mreža s propagacijom unaprijed (engl.*multilayer feedforward networks*) se razlikuju od jednoslojnih u tome što imaju jedan ili više *skrivenih slojeva* između ulaznog i izlaznog sloja. Upravo je *skriveni sloj* onaj u kojem se uče međuzavisnosti u modelu, informacije neurona se ovdje obrađuju i šalju u neurone izlaznog sloja. Izlazi neurona iz *n*-tog sloja predstavljaju ulaze u neurone iz *n+1*-og sloja. Kod mreže s propagacijom unaprijed nema povratnih veza među neuronima, tj. nema veza iz neurona jednog sloja (*n*-tog) prema prethodnim slojevima (*1, 2, …, n-1*). To kretanje veza samo prema sljedećim slojevima (prema izlazu mreže) zovemo propagacija unaprijed.



Slika 4.4. Potpuno povezana višeslojna neuronska mreža s propagacijom unaprijed

Slika 4.4. prikazuje višeslojnu neuronsku mrežu s 4 ulazna i 4 izlazna neurona (naravno, broj može biti i različit). Osim ulaznog i izlaznog sloja ona sadrži i dva skrivena sloja. U

prvom skrivenom sloju ima 5 neurona, a u drugom 3. Ovo je potpuno povezana mreža te ona nema povratnih veza među neuronima već samo sadrži propagaciju unaprijed.

Mreža je potpuno povezana kada je svaki neuron u svakom sloju povezan na sve neurone u sljedećem sloju, a ako neke veze nedostaju onda je mreža djelomično povezana.

4.4. Stvaranje neuronske mreže učenjem

Ni jedan čovjek kad se rodi ne zna odmah hodati, govorit, pisati i sl. Tek s vremenom to nauči. Proučava kako se njegova okolina ponaša i to pokušava oponašati. Npr. kod pisanja tek kad vidi u svojoj okolini slovo *A*, neko mu objasni (roditelj, učitelj) što to *A* predstavlja te onda pokuša napisati par puta to slovo, čovjek može naučiti pisati slovo *A*. Pošto neuronske mreže simuliraju ljudski mozak onda je logično da i njih prvo treba naučiti što bi trebale raditi kako bi mogle izvršavati svoju dužnost. Ta sposobnost učenja iz okoline je jedna od najvažnijih karakteristika neuronskih mreža.

Rad umjetne neuronske mreže odvija se u dvije osnovne faze: najprije se odvija faza učenja ili treniranja mreže, a zatim slijedi faza testiranja. Prije samog učenja potrebno je definirati model (ulazne i izlazne varijable), te prikupiti podatke iz prošlosti na kojima će se primijeniti mreža. Prikupljene podatke treba podijeliti u dva poduzorka (uzorak za treniranje i uzorak za testiranje).

4.4.1. Postupak učenja

Već je spomenuto da mozak uči tako da mijenja efikasnost pojedine veze ili izmjenom veza između neurona. Slično tome i neuronska mreža uči o okolini kroz iterativni proces podešavanja sinaptičkih težina i pragova. Idealno bi bilo da nakon svake iteracije učenja mreža ima više znanja o okolini.

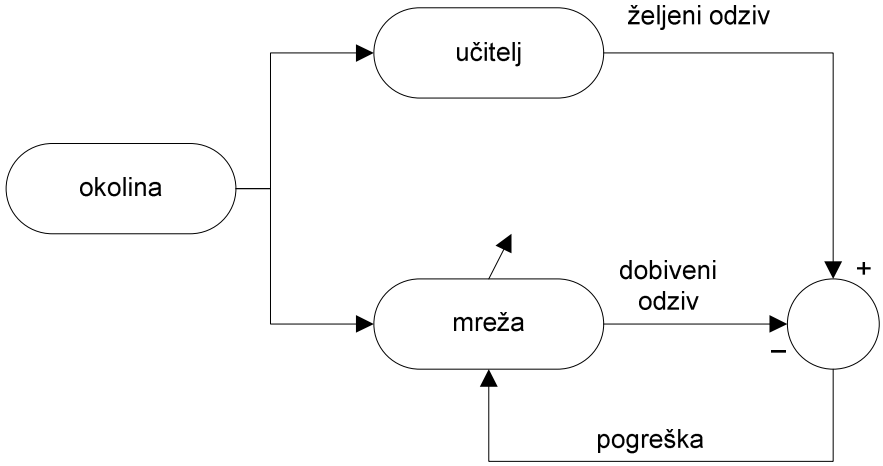
Učenje ili treniranje mreže se može izvoditi u dva načina rada:

1. *Off-line način učenja*
   Sustav prvo uči, a kad je učenje gotovo konfiguracija sustava se više ne mijenja. Mreža nakon što jednom nauči više ne nastavlja učiti na novim podacima, tj. mreža radi na statički način.
2. *On-line način učenja*
   Sustav za vrijeme rada uči i mijenja svoju konfiguraciju u skladu s novim znanjem. U ovom načinu mreža je dinamička.

Paradigme učenja određuju odnos neuronske mreže prema okolini. Postoje tri osnovne paradigme učenja [10], [12]:

1. **Učenje pod nadzorom** (engl. *supervised learning*)
   Glavna karakteristika učenja pod nadzorom je prisutnost vanjskog učitelja. Učitelj ima znanje o okolini u obliku parova ulaz-izlaz:
   $$(x, y), \quad x \in X, y \in Y$$
   Gdje je *X* skup ulaznih podataka *x*, a *Y* skup izlaznih podataka, tj. skup rješenja. Razlika između željenog (rješenje koje učitelj zna) i dobivenog odziva (rezultat koji je mreža dobila) na neki ulazni vektor jest *pogreška*. Parametri mreže se mijenjaju

pod utjecajem ulaznih vektora i signala pogreške. Proces se iterativno ponavlja sve dok mreža ne nauči imitirati učitelja, tj. dok algoritam ne pronađe funkciju koja će zadovoljiti sve parove (x, y). Nakon što je učenje završilo učitelj više nije potreban i mreža može raditi bez nadzora.



Slika 4.5 Učenje pod nadzorom učitelja

Primjeri algoritama za učenje pod nadzorom su:

- LMS ili algoritam najmanjeg kvadrata (engl. *least-mean-square*)
- BP ili algoritam s povratnom propagacijom rješenja (engl. *back-propagation*)

Mana učenja pod nadzorom je da bez učitelja mreža ne može naučiti nove strategije koje nisu pokrivene primjerima koji su korišteni za učenje.

2. **Učenje bez nadzora** (engl. *unsupervised learning*)
   Kod učenja bez nadzora ili samoorganiziranog učenja nema učitelja koji upravlja procesom učenja. Ovakvo učenje nema već pripremljene parove ulaznih podataka i traženog rješenja, već ima samo ulazne podatke te mjeru kvalitete znanja koju mreža mora naučiti. Ta mjera kvalitete znanja jest problem koji treba mreža riješiti tj. funkcija koju treba optimirati.
3. **Učenje podrškom** (engl. *reinforcement learning*)
   Kod učenja podrškom ne postoji učitelj koji određuje kolika je pogreška za određeni par ulaz-izlaz, nego sudac koji samo kaže koliko je određeni korak u učenju dobar (daje ocjenu ili podršku). Učenje podrškom je *on-line* karaktera. Sustav za učenje mora sam ustanoviti "smjer" gibanja kroz prostor učenja metodom pokušaja i pogrešaka, jer, za razliku od učenja pod nadzorom, ovdje nije poznata funkcija pogreške koju bi sustav mogao optimirati nekim gradijentnim metodama.

4.4.2. Testiranje neuronske mreže

Testiranje mreže je druga faza rada neuronske mreže, i ona je odlučujuća za ocjenjivanje mreže. Razlika između faze učenja i faze testiranja je u tome što u ovoj drugoj fazi mreža više ne uči, a to znači da su težine fiksne na vrijednostima koje su dobivene kao rezultat prethodne faze učenja. Takvoj mreži se predstavljaju novi ulazni vektori koji nisu

sudjelovali u procesu učenja, a od mreže se očekuje da za predstavljen novi ulazni vektor proizvede izlaz. Ocjenjivanje mreže obavlja se izračunavanjem greške ili neke druge mjere točnosti, na način da se izlaz mreže uspoređuje sa stvarnim izlazima.

Dobivena greška mreže na uzorku za validaciju je rezultat kojim se tumači uspješnost ili neuspješnost neuronske mreže i njezina korisnost u primjeni za predviđanje na budućim podacima.

Kod problema klasifikacije se u većini istraživanja koristi stopa klasifikacije kao mjerilo ocjenjivanja mreže. Stopa klasifikacije prikazuje postotak ili udio ispravno klasificiranih promatranja [8].

4.5. Primjena neuronskih mreža

Neuronske mreže se mogu primijeniti gotovo na svaki računalni zadatak koji simulira neki od ne mehaničkih zadataka koji živa bića mogu obavljati. Neki od primjera su [5]:

1) **Klasifikacija** (engl. *classification*)
   To je pridruživanje pojedinog objekta odgovarajućoj klasi ili razredu. Na temelju skupa uzoraka koji sadrži predstavnike iz svih razreda, zajedno s razredom kojem pripadaju (parovi ulaz-izlaz kod učenje pod nadzorom), mrežu istreniramo tako da nauči raspoznavati koji uzorak pripada kojem razredu. Takva istrenirana mreža bi trebala znati razvrstati i nove uzorke, na kojima nije trenirala, u odgovarajući razred.
   Neki od praktičnih klasifikacijskih zadataka:
   - Raspoznavanje rukom pisanih slova
   - Klasificiranje zahtjeva za kredit u one koji će biti odobreni i oni koji neće bit odobreni (provjera kreditne sposobnosti)
   - Analiziranje sonarskih i radarskih podataka za određivanje prirode izvora signala

2) **Uzorkovanje** (engl. *clustering*)
   Ono zahtjeva grupiranje zajedno objekata koji su slični jedni drugima. Jedino što je dostupno kod uzorkovanja jest skup uzoraka i njihovih svojstava po kojima bi ih neuronska mreža trebala razvrstati, te broj grupa u koje će uzorci biti razvrstani. Npr. cvijeće se može rasporediti u različite grupe na temelju boje i broja latica i sl.

3) **Vektorska kvantizacija** (engl. *vector quantization*)
   Ona je tehnika kompresije podataka bazirana a principu blokovskog kodiranja. Funkcija kvantizacije je da se veliki broj ulaznih razina ili kontinuirana funkcija svedu na manji konačni broj izlaznih razina. Vektorska kvantizacija se koristi za kompresiju slike,zvuka i videa.

4) **Asocijacija ili prepoznavanje uzoraka** (engl. *pattern association*)
   Prezentacija ulaznog uzorka neuronskoj mreži treba asocirati mrežu na neki od izlaznih uzoraka koji sadrži u svojoj memoriji. Postoji auto-asocijacija i hetero-asocijacija. Kod auto-asocijacije se pretpostavlja da je ulazni uzorak oštećen ili

nepotpun pa mreža na temelju pohranjenih uzoraka u svojoj memoriji treba prepoznati o kojem se uzorku radi. Dok kod hetero-asocijacije proizvoljne ulazne uzorke treba upariti s drugim proizvoljnim izlaznim uzorcima. Primjer auto-asocijativnog bi bio prikazati čistu sliku nečijeg lica na temelju oštećene verzije, a hetero-asocijativni primjer bi bio prikazati ime osobe na temelju njegove slike.

5) **Aproksimacija funkcija** (engl. *function approximation*)
Neka se zna vrijednost funkcije za sve ulazne podatke u skupu podataka za učenje, ali se pri tome ne zna matematička funkcija kojom su dobivene te vrijednosti. Aproksimacija funkcije je zadatak pronalaženja funkcije koja za iste ulazne podatke dobiva približno iste izlazne vrijednosti.

6) **Predviđanje** (engl. *forecasting*)
Mnogo je primjera u životu gdje budući događaji mogu biti predviđeni na temelju prošlih događanja (vremenska prognoza, predviđanje kretanja dionica na burzi…). Iako je savršeno predviđanje gotovo nemoguće, neuronske mreže se mogu istrenirati da daju relativno dobre rezultate predviđanja za mnoge slučajeve.

5. Klaster

Općenito gledajući **klaster** je skupina nezavisno djelujućih elemenata povezanih nekim medijem u cilju koordiniranog i kooperativnog ponašanja.

Računalni klaster (grozd) je skup usko povezanih samostalnih računala koji djeluju kao jedinstveno računalo. Obično su ta računala povezana korištenjem brze lokalne mreže (LAN) pomoću koje računala međusobno komuniciraju, ali ne i uvijek. Korištenje specifične programske podrške daje visok stupanj integracije računala, omogućava njihov koordinirani zajednički rad i pretvara ih efektivno (u mjeri u kojoj je to fizički moguće) u jedinstven višeprocesorski sustav. Neizbježna heterogenost arhitekture te korištenje distribuirane umjesto zajedničke memorije jedino je po čemu se klaster razlikuje od jedinstvenog višeprocesorskog sustava.

Svojstvo sustava sastavljenog od skupa komponenata da skriva složenost i predočava se kao jedinstveni sustav naziva se *Privid Jedinstvenog Sustava* (*Single System Image* ili *SSI*). Klasteri mogu ostvarivati svojstvo SSI na tri razine: na razini sklopovlja, na razini operacijskog sustava i na aplikacijskoj razini. SSI na razini sklopovlja ostvaruje se uz pomoć posebnog sklopovlja koji omogućava korisniku da vidi računala u klasteru kao jedinstveno računalo. SSI na razini OS-a sastoji se od posebnih operacijskih sustava ili OS-a s dodacima koji stvaraju privid jedinstvenog stroja. SSI na razini aplikacija se ostvaruje se skupom računalnih programa koji se nazivaju klasterski middleware.

Ciljevi povezivanja računala u klastere su razni, a najčešće se računala povezuju s ciljem osiguravanja veće pouzdanosti ili većih performansi u odnosu na pojedino računalo. Osim toga jeftinije je povezati više računala u jedinstveni sustav nego proizvest jedno super računalo istih karakteristika.

5.1. Podjela klastera

Postoji više podjela klastera. Jedna od mogućih podjela je sljedeća:

1) **Beowulf klaster** – Sastoji se od skupa računala koja ne sadrže periferne jedinice (tipkovnicu, ekran). U Beowulf klasteru jedan čvor se razlikuje od svih ostalih i naziva se *front-end*. *Front-end* čini središte klastera i na njemu su smješteni poslužitelji pojedinih sustava klasterskog middlewarea. Konkretno, na *front-endu* su smješteni datotečni sustav te poslužitelji sustava za upravljanje poslovima, sustava za nadzor klastera i sustava za automatsku instalaciju čvorova. *Front-end* i čvorovi su povezani u privatnu mrežu koja je fizički izolirana od javne mreže. Na taj način se ostvaruje učinkovitija komunikacija između čvorova. *Front-end* ima dva mrežna sučelja: jedno prema javnoj mreži i drugo prema privatnoj mreži. Korisnici pri korištenju Beowulf klastera rade isključivo na *front-endu* koji tako čini jedinstvenu točku pristupa.
2) **Mreže radnih stanica** (Networks of Workstations NOW) ili **klasteri radnih stanica** (Clusters Of Workstations COW) – Sastoji se od skupa računala koja korisnici svakodnevno koriste, a u razdobljima kada su neopterećena koriste se za izvršavanje klasterskih poslova. Klasterski middleware kod mreža radnih stanica

mora omogućavati praćenje opterećenja računala. Nadalje, sustav za upravljanje poslovima mora omogućiti premještanje poslova s čvora koji postane opterećen, tj. kada ga vlasnik krene koristiti, na neki neopterećeni čvor.

Beowulf klasteri ostvaruju bolje performanse od mreža radnih stanica. Čvorovi Beowulf klastera povezani su računalnom mrežom koja je fizički izolirana od javne mreže pa ne postoji pozadinsko opterećenje mreže. Računala koja čine čvorove se koriste isključivo za potrebe klastera pa ne postoji potrebe za praćenjem pozadinskog opterećenja računala. Kako se čvorovi koriste isključivo za potrebe klastera moguće je provesti prilagodbe jezgre OS-a čvorova tako da se ostvare bolja svojstva cijelog sustava. Činjenica da se računala nalaze u privatnoj lokalnoj mreži olakšava administraciju i povećava sigurnost.

Jedan oblik podjele klastera je prema vrsti računarstva kojoj su namijenjene, te ih dijelimo na:

1) **Klasteri visokih perfomansi** (engl. *High performance clusters* ili *HPC*)

   Služe prvenstveno za povećavanje performansi raspoređivanjem računalnih zadataka na više raznih čvorova u klasteru. Koriste se kod aplikacija koje zahtijevaju veliku računalnu moć. Tipične HPC aplikacije su paralelne aplikacije čiji su potprocesi usko povezani i razmjenjuju veliku količinu informacija. HPC aplikacije su pogodne za Beowulf klasu klastera.

2) **Klasteri velike propusnosti** (engl. *High throughput clusters* ili *HTC*)

   Namijenjeno je računarstvu velike propusnosti. HTC se odnosi na skup aplikacija koje se sastoje od velikog broja međusobno neovisnih zadataka. HTC aplikacije su pogodne za mreže radnih stanica.

3) **Klasteri s visokim stupnjem dostupnosti** (engl. *High availability (HA) clusters*)

   Koriste se prvenstveno kako bi poboljšali dostupnost usluga koje klaster pruža. Rade na principu ponavljajućih čvorova, koji onda pružaju usluge kada zakažu komponente. Na njima se izvršavaju aplikacije koje su virtualni dio nekog sustava i koje moraju biti neprestano aktivne.

4) **Klaster-i za raspoređivanje opterećenja** (engl. *Load balancing clusters*)

   Klaster-i za raspoređivanje opterećenja funkcioniraju tako da cijelo opterećenje dolazi kroz jedan ili više front end-ova, te se tada raspoređuje na više back-end servera. Iako se primarno koriste kako bi poboljšali performanse, obično također uključuju i veliku dostupnost (engl. *High-availability*) i sposobnosti koje proizlaze iz nje. Takav se klaster ponekad naziva serverska farma. Primjer aplikacija koje zahtijevaju load balancing klastere su web poslužitelji, mail poslužitelji i sl.

5) **Hibridni klasteri** (engl. *Hybrid clusters*)

   Hibridni klasteri su mješavina drugih vrsta klastera, npr. HPC+HA [11].

5.2. Klaster Isabella

5.2.1. Povijest klastera Isabelle

Prvi klaster pod nazivom Dgrid, s 8 jednoprocesorskih čvorova i klasterskom distribucijom Rocks, izgrađen je u Srcu u siječnju 2002. za potrebe projekta Ministarstva znanosti i tehnologije - DataGrid. Cilj je tog projekta bio priključivanje Hrvatske europskom projektu DataGrid. Ovaj je projekt pokrenut i vođen od strane CERN-a u cilju izgradnje data-intenzivne mreže resursa za analizu i obradu velike količine podataka dobivenih od znanstvenih istraživanja LHC eksperimenata.

Slika 5.1 - Prvi klaster u Srcu

Slika 5.2 - Današnji klaster Isabella

Isti je klaster, pod imenom Isabella u svibnju 2002. godine stavljen na raspolaganje i akademskoj zajednici kako bi se hrvatskim stručnjacima omogućilo sudjelovanje u vrhunskim znanstvenim projektima te ih potaknulo na razvoj klastering tehnologija.

U jesen iste godine od Ministarstva znanosti i tehnologije (danas Ministarstvo znanosti, obrazovanja i športa) je dobiveno 8 novih dvoprocesorskih računala koja preuzimaju funkciju klastera Isabella čime on postaje odvojen produkcijski klaster namijenjen samo akademskoj zajednici. Od računala koja su do tada korištena izgrađen je novi klaster Dgrid koji se s klasterskom distribucijom Mosix koristi samo u ispitivanjima CERN-ovog projekta ALICE (AliEn).

Početkom 2004. godine Isabella je sredstvima Srca proširena s nova 24 dvoprocesorska blade-servera (HP BL20p G2) i priključena na središnji diskovni prostor Srca.

Krajem 2004. godine Srce je nabavilo još 32 dvoprocesorska računalna čvora (Dell 1850) i novo pristupno računalo (Dell 2850). U mrežnu infrastrukturu za povezivanje računalnih čvorova, uz postojeći 1 Gb/s Ethernet, uvedena je 10 Gb/s Infiniband tehnologija.

Financirano sredstvima Ministarstva znanosti, obrazovanja i športa, računalni klaster je u svibnju 2005. godine proširen s 24 dvoprocesorska Opteron računala (Pyramid GX28), a

nabavkom proširivog Infiniband preklopnika (InfiniCon InfinIO 9100) unaprijeđena je i mrežna infrastruktura.

Početkom 2006. godine Srce je, financiranjem iz vlastitih sredstava, proširilo klaster Isabella s 8 novih HP ProLiant računala.

Isabella se danas sastoji od 100 računala i kao zajednički resurs svih znanstvenika u Hrvatskoj omogućava korištenje značajnih računalnih resursa pri zahtjevnim obradama podataka u sklopu znanstveno-istraživačkih projekata [11].

Računalni klaster Isabella spada u skupinu Beowulf HPC klastera.

5.2.2. Isabella u brojkama

- 100+2 računala
- 352 procesora na računalnim čvorovima
- 544 GB RAM-a
- 10 TB lokalnog diskovnog prostora
  + dodatni diskovni prostor sa središnjeg diskovnog prostora Srca (SAN - Storage Area Network) ukupnog kapaciteta do 8 TB

5.2.3. Tehničke specifikacije klastera Isabella

Pristupno računalo:

Dell 2850

1 x Intel Xeon 2.8 GHz, EM64T
1 GB RAM
2 x 72 GB SCSI HDD (HW RAID 1)
1 x FC adapter (priključak na SAN - 0.5 TB diskovnog prostora)

Računalni čvorovi:

1) **24 x** HP ProLiant BL20p G2 blade server
   2 x Intel Xeon 2.8 GHz
   2 GB RAM
   2 * 36 GB SCSI HDD (HW RAID 0)
   redundantni mrežni preklopnici i napajanje

2) **32 x** Dell 1850 1U server
   2 x Intel Xeon 3.4 GHz, EM64T
   2 GB RAM
   1 x 72 GB SCSI HDD
   InfiniServ 7000 Infiniband HCA (10 Gb/s)

3) **24 x** Pyramid GX28
   2 x AMD Opteron 248
   2 GB RAM
   1 x 72 GB SCSI HDD
   InfiniServ 7000 Infiniband HCA (10 Gb/s)

4) **16 x** HP ProLiant DL585
   4 x AMD Opteron 875 / AMD Opteron 8218 (dual core)
   16 GB RAM
   4 x 73 GB SAS (Serial Attached SCSI) HDD
   Infiniband HCA (10 Gb/s)

5) **4 x** Sun Fire x4600
   8 x AMD Opteron 8218 (dual core)
   32 GB RAM
   4 x 73 GB SAS (Serial Attached SCSI) HDD
   Infiniband HCA (10 Gb/s)

Mrežna infrastruktura:

- 2 x HP ProCurve 2848 1 Gb/s Ethernet preklopnik (ukupno 96 pristupa)
- 1 x InfiniCon InfinIO 3000 Infiniband 10 Gb/s preklopnik (32 pristupa)
- 1 x InfiniCon InfinIO 5000 Infiniband 10 Gb/s preklopnik (48 pristupa, proširivo na 144)

Programska podrška:

- Oscar (Open Source Cluster Application Resources) 5.0
- Sun Grid Engine 6.0
- Portland Group (PGI) Fortran i C++ prevoditelji
- Intel Fortran i C++ prevoditelji
- Intel programske biblioteke - Intel IPP (Integrated Performance Primitives),
- Intel Cluster MKL (Math Kernel Library), Intel MPI
- Gaussian98 i Gaussian03
- Fluent
- OpenFOAM [11]

Specifikacija na dan 7. svibnja 2007.

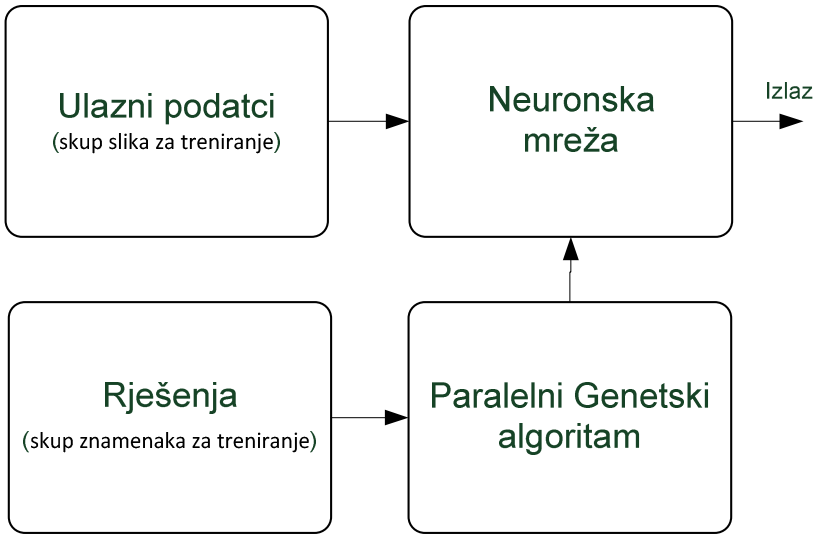
6. Praktični rad

6.1. Neuronska mreža za raspoznavanje znakova

Ideja ovog diplomskog rada je istrenirati neuronsku mrežu da prepoznaje rukom pisane znakove uz pomoć genetskog algoritma, te to treniranje paralelizirati kako bi se moglo obavljati na više računala (na klasteru Isabelli). Za primjer neuronske mreže je izabrana potpuno povezana višeslojna neuronska mreža s propagacijom unaprijed. Dakle svi neuroni ulaznog sloja su povezani sa svim neuronima srednjeg sloja, te svi neuroni srednjeg sloja s neuronima izlaznog sloja. Smjer izračunavanja ide od ulaznog sloja prema izlaznom, bez povratnih veza.

Neuronska mreža se trenira sa skupom slika koje predstavljaju rukom pisane brojeve. Nakon predprocesiranja slike (centriranja i smanjivanja slike) svaki piksel slike se pridruži jednom ulaznom neuronu neuronske mreže. Iako postoje i dodatne metode koje izvlače posebnosti i značajke znakova za raspoznavanje, one zahtijevaju komplicirano predprocesiranje slike. Pošto je naglasak diplomskog rada na paralelnom genetskog algoritmu uzeta je neuronska mreža koja je jednostavna za implementaciju.

Nakon što se neuronskoj mreži predstave ulazni podaci ona za svaki neuron zbraja sve ulaze u njega pomnožene s određenom težinom, te na kraju na tu sumu djeluje aktivacijskom funkcijom. Tu dobivenu vrijednost neuron šalje dalje neuronima prema izlazu iz mreže. U izlaznom sloju svaki neuron predstavlja jedan znak koji mreža treba prepoznat. Izlazni neuron koji dobije najveću vrijednost predstavlja rješenje mreže. Dakle nakon izračuna neuronska mreža odredi da ulazna slika pripada određenom znaku. Je li to točno ili nije određuje *učitelj*. Ako nije točno razvrstano težine između neurona se trebaju ispraviti. Jedan od načina treniranja neuronske mreže je uz pomoć genetskog algoritma.



Slika 6.1. Treniranje neuronske mreže uz pomoć PGA

6.2. MNIST baza podataka

Za ulazne podatke je korištena MNIST baza rukom pisanih znamenaka skinuta s interneta. Baza je besplatna i sadrži skup od 60000 podataka za treniranje, tj. za učenje, te skup od 10000 za testiranje. Postoji i MNIST baza rukom pisanih slova engleske abecede, ali ta baza nažalost nije besplatna.

Baza sadrži 4 datoteke[3]:

- *train-images-idx3-ubyte* – skup slika za treniranje
- *train-labels-idx1-ubyte* – skup znamenaka za treniranje
- *t10k-images-idx3-ubyte* – skup slika za testiranje
- *t10k-labels-idx1-ubyte* – skup znamenaka za testiranje

Skup slika predstavlja slike rukom pisanih znamenaka, dok skup znamenaka sadrži poredane znamenke tako da prva znamenka odgovara prvoj slici, druga drugoj itd. Podaci u korištenoj MNIST bazi podataka su zapisani u vrlo jednostavnom formatu dizajniranom za spremanje vektora i multidimenzionalnih matrica (*IDX format datoteke*).

```
magični broj
veličina dimenzije 0
veličina dimenzije 1
veličina dimenzije 2
.....
veličina dimenzije N
podaci
```

Slika 6.2. *IDX format datoteke*

Magični broj pobliže objašnjava koje podatke sadrži datoteka. On je zapisan u *high-endian* ili *big-endian* formatu (format podataka gdje je prvi dio najznačajniji). Prva dva bajta su uvijek 0 (0x00 – heksadekadski zapis). Treći bajt kodira tip podataka i to na sljedeći način:

```
0x08: unsigned bajt
0x09: signed bajt
0x0B: short (2 bajta)
0x0C: int (4 bajta)
0x0D: float (4 bajta)
0x0E: double (8 bytes)
```

Slika 6.3. Treći bajt magičnog broja

[3] Sve datoteke su dostupne na http://yann.lecun.com/exdb/mnist/

Četvrti bajt kodira broj dimenzija vektora (ako je jedan), odnosno matrice (ako je dva i više).

Veličina pojedine dimenzije je 32-bini integer, a podaci su zapisani u niz tako da se indeks zadnje dimenzije najsporije mijenja.

Za skup slika za treniranje datoteka izgleda ovako (samo stupac vrijednosti, jer ostali stupci samo opisuju datoteku):

[pomak]	[tip podatka]	[vrijednosti]	[opis]
0000	32 bit integer	0x00000803(2051)	magični broj
0004	32 bit integer	60000	broj slika
0008	32 bit integer	28	broj redova
0012	32 bit integer	28	broj stupaca
0016	unsigned byte	??	piksel
0017	unsigned byte	??	piksel
....			
xxxx	unsigned byte	??	piksel

Slika 6.4. Izgled datoteke skupa slika za treniranje

Iz magičnog broja se vidi da datoteka sadrži podatke tipa `unsigned byte` te da su podaci zapisani u trodimenzionalnim matricama. Prva dimenzija je broj slika i njih je 60000, druga je broj redaka svake slike, a treća broj stupaca. I redaka i stupaca ima 28. Pikseli su organizirani po redovima, što znači da prvih 28 piksela predstavlja prvi red prve slike, drugih 28 drugi red. Tek nakon što prođe svih 28 redova prve slike počinje druga slika itd. Svaki piksel može imat vrijednost od 0 do 255, s tim da 0 predstavlja bijelu boju (pozadina), a 255 crnu (sama znamenka).

Isti oblik datoteke ima i skup slika za testiranje, samo je broj slika umjesto 60000 jednak 10000.

Slika 6.5. MNIST baza podataka rukom pisanih brojeva

Datoteka skupa znamenaka za treniranje je sljedeća:

[pomak]	[tip podatka]	[vrijednosti]	[opis]
09	32 bit integer	0x00000801(2049)	magični broj
0004	32 bit integer	60000	broj znamenaka
0008	unsigned byte	??	znamenka
0009	unsigned byte	??	znamenka
….....			
xxxx	unsigned byte	??	znamenka

Slika 6.6. Datoteka skupa znamenaka za treniranje

Magični broj između ove datoteke i prethodne se razlikuje samo u zadnjem bajtu, tj. u broju dimenzija. Kod skupa znamenaka broj dimenzija je samo 1, što znači da su znamenke zapisane u jednodimenzionalnom vektoru. Broj znamenaka je 60000. A njihova vrijednosti su cijeli brojevi od 0 do 9.

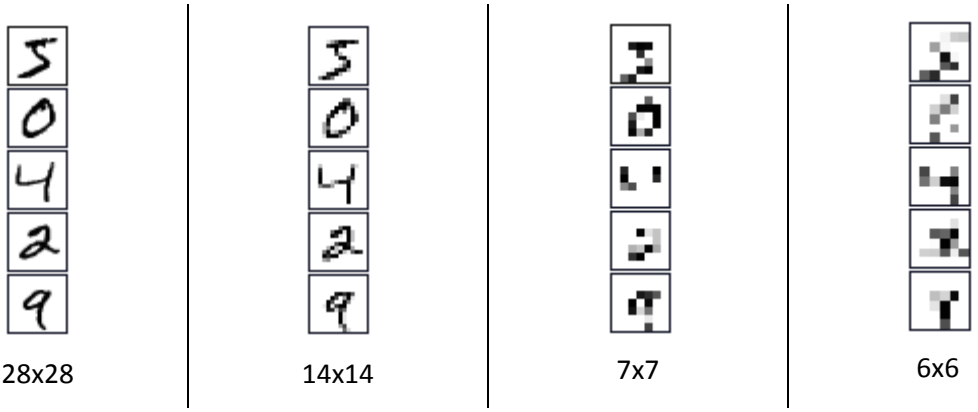
Datoteka sa skupom znamenaka za testiranje je slična kao i kod skupa znamenaka za treniranje. Jedina razlika, kao i kod skupa slika, je u broju znamenaka (10000 naspram 60000) [14].

6.3. Programsko ostvarenje

Program je napisan u programskom jeziku C++, zbog mogućnosti izvođenja na klasteru Isabelli, te zbog objektne paradigme koja olakšava pisanje složenijih programa. Program sadrži 5 klasa:

- `MNISTdbProcessor`
- `Neuron`
- `NeuralNetwork`
- `Chromosome`
- `GeneticAlgorithm`

Prva klasa služi samo za procesiranje MNIST baze podataka, tj. izvlačenje ulaznih podataka iz baze rukom pisanih brojeva. Svaka slika u bazi sadrži 28x28 crno-bijelih piksela (točaka).

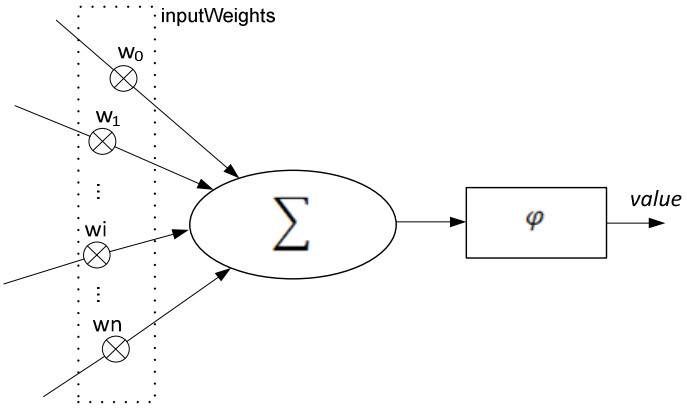


Slika 6.7.Prikaz originalne slike i umanjenih verzija

`MNISTdbProcessor` umanji slike na 7x7 te svaku sliku, piksel po piksel, spremi u skup ulaznih vektora (svaki vektor ima 49 elementa). Ispitivanje je provedeno i na slikama dimenzija 6x6 i 14x14 piksela radi usporedbe. Nakon smanjivanja slika na dimenzije 7x7 ili 6x6 može doći do neprepoznatljivih uzoraka, dok kod 14x14 uzorci ostaju prepoznatljivi (Slika 6.7).

Ostale klase imaju engleske nazive onog što predstavljaju. Tako klasa `Neuron` predstavlja umjetni neuron, koji je već ranije spomenut. Za svaki neuron navedeno je kojem sloju neuronske mreže pripada (ulazni, skriveni ili vanjski), koji je po redu neuron u tom sloju te najvažnije: vektor ulaznih težina i izlaznu vrijednost (Slika 6.8).

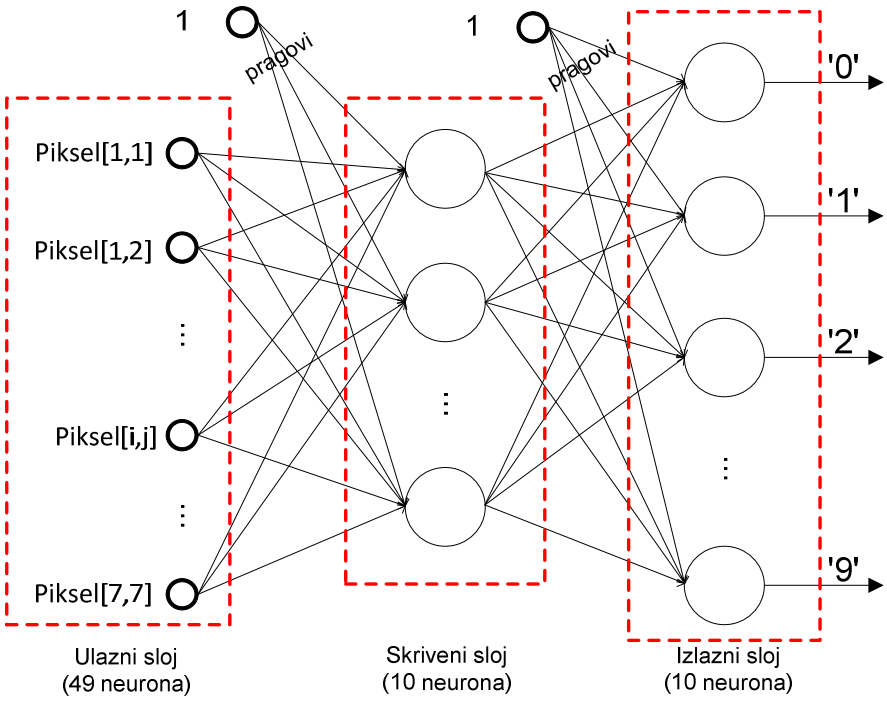
Ulazne težine (`inputWeights`) se postavljaju na trenutne vrijednosti težina s kojima će se množiti ulazni podaci u neuron, a izlazna vrijednost (`value`) se dobije tek nakon što se aktivacijskom funkcijom djeluje na sumu tih umnožaka.



Slika 6.8. Slikovni prikaz klase *Neuron*

Ako neuron pripada ulaznom sloju onda on neće imati vektor ulaznih težina te na njega neće djelovati aktivacijska funkcija. Ulazni sloj se zapravo sastoji od neurona koji samo sadrže ulaznih podataka, tako da se oni čak ni ne trebaju nazivati neuronima.

Klasa `NeuralNetwork` predstavlja naravno neuronsku mrežu. Svaki sloj neurona (ulazni, skriveni i vanjski) je predstavljen nizom, odnosno vektorom neurona (objekata instanciranih od klase `Neuron`).

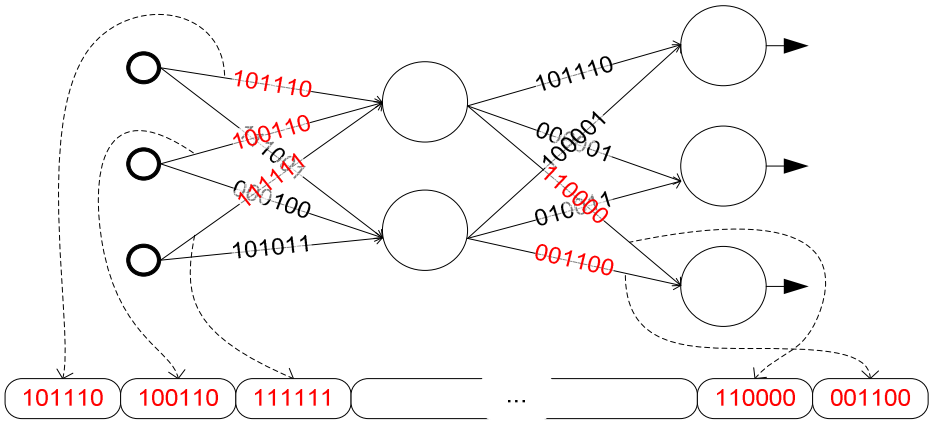


Slika 6.9. Slikovni prikaz programski ostvarene neuronske mreže

Neuronskoj mreži se postave na ulaz pikseli jedne slike te se podese vrijednosti težina veza među neuronima. Nakon toga ona izračuna izlazne vrijednosti pojedinog neurona. Svaki izlazni neuron predstavlja jedan znak koji neuronska mreža treba prepoznati. Mreža

se odluči za onaj znak koji predstavlja izlazni neuron s najvećom izlaznom vrijednošću. U ovom slučaju ti znakovi su brojevi od 0 do 9.

Kromosom je predstavljen kao niz težina zapisanih u binarnom zapisu. I pragovi su uključeni u taj niz težina. Njega naravno predstavlja klasa `Chromosome`. Svaka težina u neuronskoj mreži se može promatrati kao skup od *n* gena koji predstavljaju jedno svojstvo jedinke (kao što je na primjer boja očiju kod živih bića).



Slika 6.10. Slikovni prikaz programski ostvarenog kromosoma

Naglasak diplomskog rada je na genetskom algoritmu koji treba istrenirati navedenu neuronsku mrežu, stoga se može reći da je najvažnija klasa u ovom programu `GeneticAlgorithm`. Slijedi lista genetskih operatora te koje vrste pojedinih operatora su programski ostvareni:

1. **Evaluacija**
   Kao *funkcija dobrote* u ovom programu je izabran broj točno razvrstanih znakova. U svakom kromosomu je zapisana njegova dobrota (atribut `score`). Za jedan zadani znak neka je *pogodak* jednak 1 ako je točno razvrstan, a 0 ako je krivo razvrstan, te neka je *brPodataka* ukupni broj ulaznih podataka (znakova), onda za *dobrotu* vrijedi:

$$pogodak \in [0,1]$$

$$dobrota = \sum_{i=0}^{brPodataka} pogodak(i) \qquad \textbf{(6.1)}$$

2. **Selekcija**
   Korištena je *k-turnirska eliminacijska selekcija bez duplikata*. Bez duplikata znači da selekcija između *k* selektiranih jedinki ne može imati iste jedinke. Iako program omogućava promjenu parametra *k* (u programu je naveden kao parametar `chromosomesToBreed`), u svim primjerima je korištena *3-turnirska selekcija*.
3. **Križanje**
   Koristi se *uniformno križanje*. Pošto kromosom sadrži veći broj težina onda se svaka težina jednog roditelja križa s odgovarajućom težinom drugog roditelja.

4. **Mutacija**
   Jednostavna mutacija. Za svaki bit kromosoma vrijedi ista vjerojatnost hoće li mutirati ili ne. Tu vjerojatnost predstavlja parametar `mutationProbability` koji se može mijenjati.

Osim navedenih 5 klasa još postoji glavni program (`main.cpp`) koji povezuje sve te klase i paralelizira genetski algoritam. Program zauzme zajedničku memoriju u kojoj bude spremljena cijela populacija kromosoma. Na početku gospodar stvori populaciju te svakom slugi, ali i sebi, dodijeli pravedno podijeljeni dio populacije na kojem svaki od procesora izvršava evaluaciju. Nakon evaluacije svih jedinki gospodar ovaj put podijeli ukupni broj iteracija s brojem procesora te dobije broj iteracija koje svaki procesor mora izvršiti. I opet gospodar pošteno podijeli iteracije tako da i on sam izvršava jednaki posao kao i sluge s tim da gospodar još i ispisuje rezultate na ekran i u datoteku *log.txt*. Time se maksimalno iskorištava broj procesora koji su zaduženi za izvođenje paralelnog genetskog algoritma. Komunikacije među procesorima nema, nego svi rade nad istim zajedničkim jedinkama. Time se ne gubi vrijeme na nepotrebnu komunikaciju. Jedino kod obavljanja evaluacije početnih jedinki gospodar čeka sluge (ili oni njega) da završe svoj posao. Osim te sinkronizacije, procesori se sinkroniziraju i na kraju programa, prije nego što se ispišu završni rezultati. Mana ovog načina je ta da ako je neki procesor puno sporiji od drugih svi će morati čekati da i on završi svoj dio posla.

6.4. Priprema za izvođenje na klasteru Isabella

Za opisivanje poslova (engl. *job description*) koji će se izvršavati na klasteru Isabelli koristimo *Sun Grid Engine*-ov jezik za opisivanje poslova. Ovaj jezik je jednostavan i omogućava korisniku da opiše osnovna svojstva poslova kao što su ime posla, ulazne i izlazne datoteke, prostorne varijable i slično.

```
#$    - <parametar1>    <vrijednost1>
#$    - <parametar1>    <vrijednost1>
<naredba1>
<naredba2>
```

Slika 6.11. Format jezika za opisivanje poslova

Podnošenje poslova (engl. *job submission*) na klasteru vrši se korištenjem *qsub* naredbe. Prilikom podnošenja posla potrebno je unutar skripte kojom se pokreće posao ili korištenjem parametara u komandnoj liniji specificirati zahtjeve koje posao ima prema klasteru i računalnim resursima (tip posla, okolina, broj procesora, korištenje memorije, zahtijevano vrijeme obrade itd.).

```
$    qsub [opcije]   <skripta>
```

opcije – opcije koje se koriste za opisivanje posla

skripta – datoteka koja služi za opis posla

Slika 6.12. Opis naredbe *qsub*

Točna specifikacija posla utjecat će na mogućnost i brzinu JMS sustava (engl. *Job Management System*) da poslu dodjeli odgovarajuće resurse i pokrene ga. Stoga je što detaljnije opisivanje parametara posla nužno i korisno[4].

Podnošenje zahtjeva za treniranje neuronske mreže pomoću genetskog algoritma na klasteru, koji je opisan datotekom *skripta.sge* (Slika 6.13), se postiže sljedećom naredbom:

```
$    qsub skripta.sge
```

U toj skripti su navedeni neki potrebni parametri za posao. Parametrom *-N* se zadaje ime posla, u ovom slučaju to je *NNuPGA*. Parametrom *-cwd* se trenutni direktorij (onaj u kojem se skripta nalazi) bira za radni direktorij. Nakon *-o* navodi se putanja izlazne datoteke, a s *-j y* korisnik navodi da izlaz i greške budu u istoj datoteci. Pomoću parametra *-l* korisnik može upravljati resursima. U ovoj skripti je specificirano da se koristi računalo compute-4-1.local (jedno od računala s 16 jezgri) te je ograničena arhitektura računala. Još dva parametra su korištena za primanje obavijesti o promjeni statusa posla na *e-mail*. To su parametar *-M* s kojim se navodi na koju e-mail adresu se šalju obavijesti, te s *-m ae* odabiremo koje će obavijesti biti poslane (a – posao je pokrenut, e – posao je obavljen).

```
#$ -N NNuPGA
#$ -cwd
#$ -o output/NNuGA.out
#$ -j y

#$ -l hostname=compute-4-1.local
#$ -l arch=lx26-amd64

#$ -M marin.nevescanin@fer.hr
#$ -m ae

/home/mnevesca/dipl/NNuPGA
```

Slika 6.13. skripta za podnošenje posla na klasteru Isabelli

Nakon opisa posla navedena je naredba koja će biti pokrenuta, odnosno sam posao koji se treba obaviti. U ovom slučaju se pokreče program *NNuPGA*[5] koji se nalazi u direktoriju */home/mnevesca/dipl* na pristupnom računalu klastera Isabelle.

[4] Za detaljnije upute o podnošenju i opisivanju poslova pogledajte skriptu http://www.cro-ngi.hr/fileadmin/cro-ngi/dokumenti/Koristenje_racunalnih_klastera.pdf

[5] NNuPGA – skraćenica od engleskog Neural Network using Parallel Genetic Algorithm

6.5. Upute za pokretanje programa

Program je prilagođen radu na operacijskim sustavima Unix ili Linux, zbog čega ga je potrebno prevesti pomoću skripte *compileNNuPGA*. To se postiže naredbom:

```
$ ./compileNNuPGA
```

Nakon prevođenja programa, on se pokreče naredbom:

```
$ ./NNuPGA
```

Svi parametri potrebni za rad programa su navedeni u zaglavlju *parameters.h*:

- *nProc* – broj procesora

parametri kromosoma:

- *minValue* – minimalna vrijednost koju težine unutar kromosoma mogu poprimiti
- *maxValue* – maksimalne vrijednost koju težine unutar kromosoma mogu poprimiti
- *precision* – preciznost (broj decimala) težina unutar kromosoma

parametri neuronske mreže

- *inputNeuronsNum* – broj neurona u ulaznom sloju
- *hiddenNeuronsNum* – broj neurona u srednjem sloju
- *outputNeuronsNum* – broj neurona u izlaznom sloju

parametri genetskog algoritma

- iterationsNum – broj iteracija
- populationSize – veličina populacije
- chromosomesToBreed – veličina turnira (parametar *k* kod *k*-turnirske selekcije)
- mutationProbability – vjerojatnost mutacije jednog gena kromosoma

Jedino se broj ulaznih i izlaznih neurona ne preporuča mijenjati, jer program neće raditi, osim ako se u kodu ne promjeni ulazni podaci ili ako se program ne proširi da može prepoznavati i neke druge znakove.

Ako se koji od parametara promjeni potrebno je ponovno prevesti program pomoću skripte *compileNNuPGA*, pa tek onda pokrenuti program.

Pri izvršavanju na ekranu se ispisuje napredovanje genetskog algoritma. Za svaku stotu iteraciju se ispisuje broj iteracije, dobrota najbolje jedinke i prosječna dobrota. Pošto treniranje može bit i dugotrajan proces svi ti podaci koji se ispisuju na ekran se ujedno i upisuju u datoteku *log.txt*. Ta datoteka je prikladna za prikazivanje uz pomoć tabličnog kalkulatora u obliku grafikona.

Nakon što genetski algoritam dođe do kraja evolucije u datoteci *trainedNN.txt* zapiše najbolji kromosom, tj. sve njegove težine. On nam služi za testiranje neuronske mreže, jer se njegove težine uzmu za težine neuronske mreže, pa se mreža testira na uzorcima za testiranje.

7. Analiza rezultata

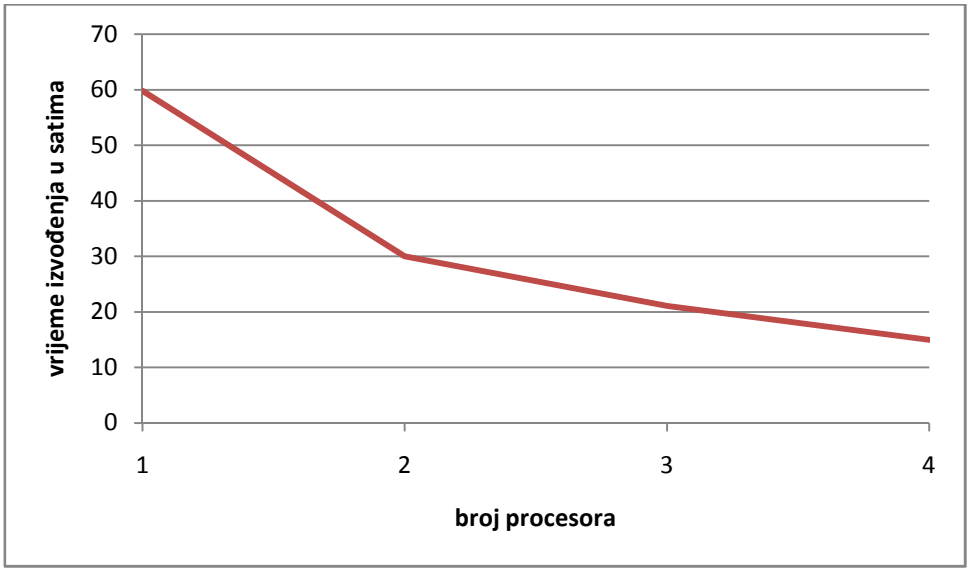
7.1. Ovisnost vremena izvođenja PGA u odnosu na broj procesora

Cilj paralelnog genetskog algoritma je ubrzati genetski algoritam, a da pri tome kvaliteta rješenja ostane ista, što znači da dobrota najbolje jedinke bude približno ista. Tablica 7.1. prikazuje trajanje algoritma za različiti broj procesora te dobrotu najbolje jedinke (za 120 000 iteracija i 60 000 ulaznih podataka).

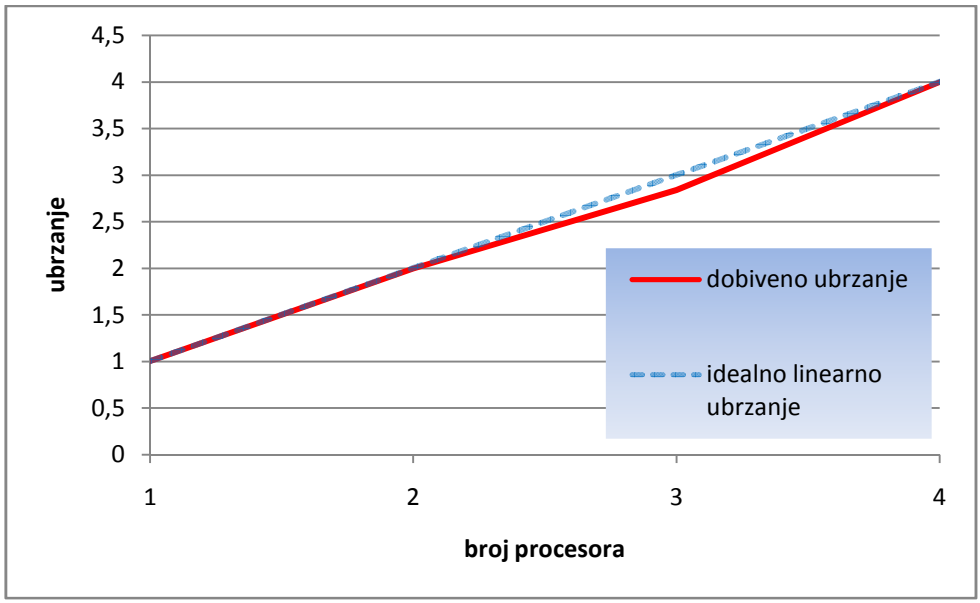
broj procesora	dobrota najbolje jedinke	trajanje algoritma [s]
1	74.7833%	215264
2	72.2433%	108010
3	70.7017%	75846
4	75.0283%	53859

Tablica 7.1. Rezultati algoritma za različiti broj procesora

Vidimo da je kvaliteta rješenja za različiti broj procesora približno jednaka (od 70% do 75%). Dakle kvaliteta rješenja ne ovisi o broju procesora, ali zato se vrijeme izvođenja smanjuje znatno. Koliko je procesora obavljalo algoritam toliko je puta smanjeno vrijeme izvođenja samog algoritma. Naravno da bi se pri velikom broju procesora to smanjenje vremena, tj. ubrzavanje algoritma, polako počelo usporavati. Koliki broj procesora se može koristiti da algoritam i dalje ubrzava izvođenje ovisi o broju iteracija koje se koriste, jer svaki procesor obavlja svoj dio iteracija. No točan broj procesora nije se mogao odrediti zbog fizičke ograničenosti računala (samo 4 procesora).



Slika 7.1. Vrijeme izvođenja algoritma u ovisnosti o broju procesora



Slika 7.2. Ubrzanje izvođenja PGA u odnosu na sekvencijalni GA

Paraleliziranjem genetskog algoritma u idealnom slučaju bi trebalo dobiti linearno ubrzanje samog algoritma. Iz grafa (Slika 7.2) se može primijetiti da ovaj model paralelnog genetskog algoritam ima gotovo linearno ubrzanja. To je odlična karakteristika ovog modela GPGA. Ne treba čuditi zašto je ubrzanje linearno, jer se praktički uopće na gubi vrijeme na komunikaciju među procesorima. Kod sinkronizacije dođe jedino kod prve evaluacije svih elemenata jer gospodar čeka da svi procesori (uključujući i njega) izračunaju dobrote jedinki za koje su zaduženi, te na samom kraju se pričeka da svi obave svoje iteracije prije nego i se ispiše rezultat. U slučaju da jedan procesor sporiji od ostalih ili je zauzet još nekim poslom onda cijeli algoritam traje koliko tom najsporijem procesoru treba da obavi svoj dio iteracija.

7.2. Testiranje istrenirane neuronske mreže

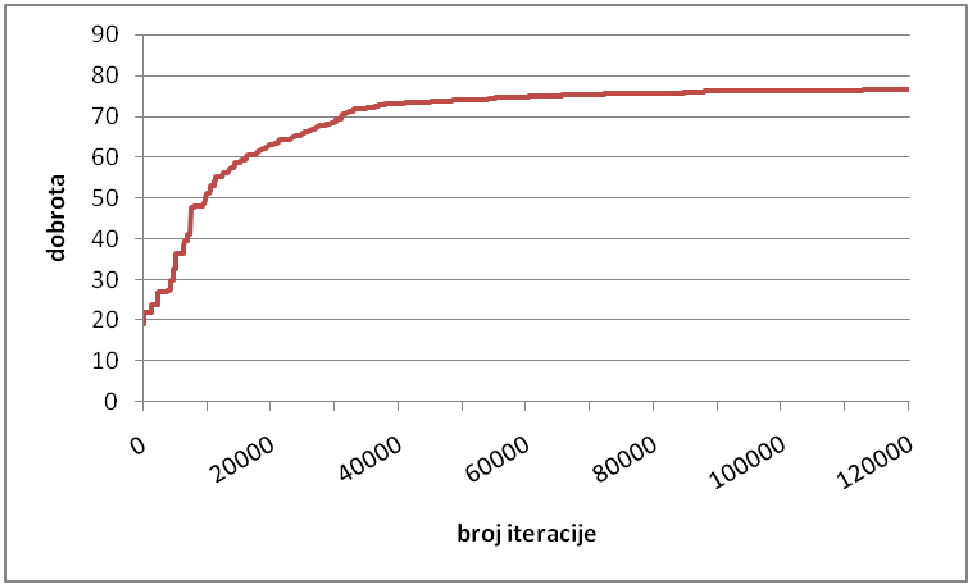
Neuronska mreža je trenirana s već spomenutim skupom od 60000 znakova (MNIST baza podataka), te su korišteni sljedeći parametri:

Kromosom:	
- maksimalna vrijednost	-1
- minimalna vrijednost	1
- preciznost	3
Neuronska mreža:	
- broj ulaznih neurona	49
- broj neurona u srednjem sloju	10
- broj izlaznih neurona	10

Genetski algoritam:	
- broj iteracija	120000
- veličina populacije	120
- k-turnirska selekcije	3-turnirska selekcija
- vjerojatnost mutacije	0,003

Tablica 7.2. Parametri korišteni za testiranje neuronske mreže

Zadanu neuronsku mrežu genetski algoritam je uspio istrenirati sa 76,5% točnošću prepoznavanja znakova. Vidimo da i nakon 120000 iteracija genetski algoritam lagano napreduje stoga bi trebalo još povećati broj iteracija. Kad se provede testiranje na skupu uzoraka za testiranje (10000 uzoraka) neuronska mreža točno klasificira 77,21% znakova.



Slika 7.3.Napredak najbolje jedinke genetskog algoritma

Dobiveni rezultat nažalost nije dobar rezultat, ali greška nije u genetskom algoritmu već u jednostavnosti neuronske mreže. Ipak da bi neuronska mreža mogla bolje raspoznavati uzorke trebao bi se dati veći naglasak i na predprocesiranju uzoraka (ulazne slike znakova) i izvlačenja svojstava iz slika koja su karakteristična za pojedina slova (zakrivljenost slova, zatvorene krivulje, ravne crte itd.).

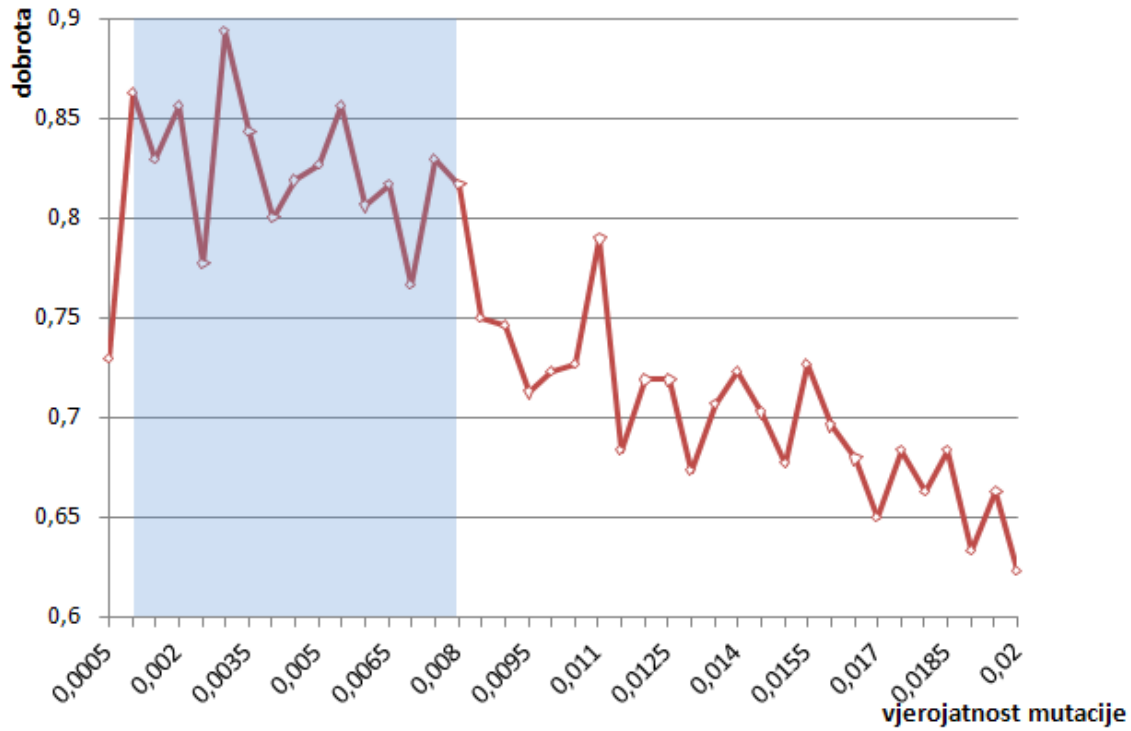
7.3. Osjetljivost na parametre genetskog algoritma

Prilikom gledanja osjetljivosti kvalitete rješenja na parametre genetskog algoritma testirana je neuronska mreža s jednim srednjim slojem koji sadrži 10 neurona. Naravno broj ulaznih neurona je 49, a izlaznih 10. Duljina kromosoma odgovara broju težina među neuronima pomnoženih s brojem bitova potrebnih za svaku težinu. Težina ima 610, a vrijednosti težina se kreću od -1 do 1 s preciznošću na 3 decimale,. Prema formuli (2.5) potrebno je 11 bitova za svaku težinu, što znači da je kromosom dug 6710 bitova, odnosno sadrži isto toliko gena. Za ispitivanje je korišten skup od samo 100 podataka, jer

je optimizacija nad malim skupom puno brža te se time omogućava ispitivanje puno većeg raspona pojedinog parametra. Iako na prvi pogled rezultati izgledaju mnogo bolje nego za veći broj ulaznih parametara, to je samo zato jer se mreža bolje istrenira za prepoznavanje tih 100 podataka. Zato puno lošije rezultate tako istrenirana mreža daje kod testiranja nad testnim podacima, no u ovom poglavlju se proučava ponašanje samog genetskog algoritma za što je dovoljan i manji skup ulaznih podataka. Svaki test se provede na 100000 iteracija.

7.3.1. Ovisnost kvalitete rješenja o vjerojatnosti mutacije

Za genetski operator mutacije je izabrana jednostavna mutacija, što znači da je za svaki gen kromosoma jednaka vjerojatnost promjene gena. Vjerojatnost mutacije ima vrijednost između 0 i 1, odnosno između 0% i 100%. Ako ona teži k jedinici onda se genetski algoritam pretvara u algoritam slučajne pretrage prostora rješenja. No, ako teži prema nula onda postupak vrlo lako može zaglaviti u nekom lokalnom optimumu. Ovo je najvažniji parametar evolucijskog programa i kvaliteta konačnog rješenja je najosjetljivija na njegovu vrijednost. Stoga je neuronska mreža testirana s različitim vrijednostima tog parametra.

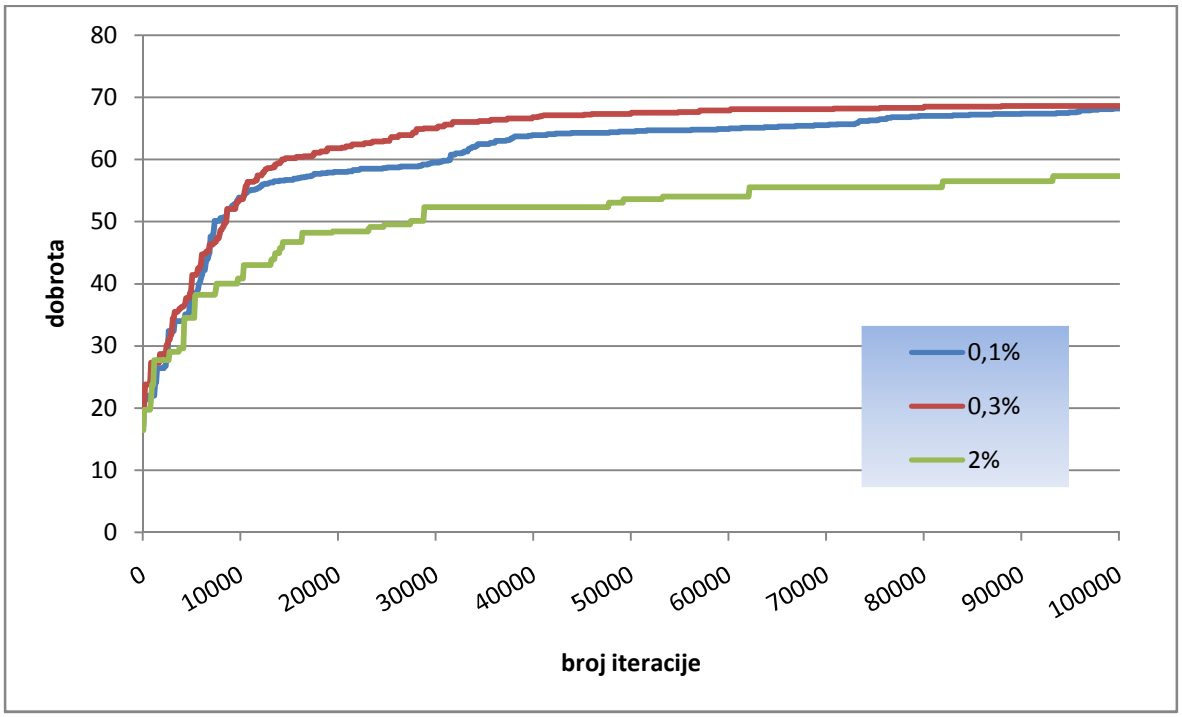


Slika 7.4. Dobrota u ovisnosti o vjerojatnosti mutacije

Slika 7.4 predstavlja graf ovisnosti dobrote rješenja o vjerojatnosti mutacije. Za veličinu populacije uzeto je 120 jedinki. Za svaku vrijednost vjerojatnosti mutacije ponovljena su tri mjerenja, te je uzeta srednja vrijednost sva tri mjerenja. Iz grafa se vidi da je genetski algoritam pri treniranju neuronske mreže postizao najbolje rezultate za vjerojatnost mutacije između 0,1% i 0,8%. Algoritam zbog svoje stohastičke prirode nije došao do boljeg rješenja za vjerojatnosti 0,25% i 0,7%, što ne znači da ne može. Najbolja prosječna

dobrota od 89.33% (dobiveno je 93%, 83% i 92% točno klasificiranih znakova) je pri vjerojatnosti mutacije od 0,3%. To znači da se genetski algoritam najbolje ponaša kad od 1000 gena u kromosomu mutira samo 3 gena. S obzirom da se u ovom primjeru kromosom sastoji od 610 težina, a svaka težina ima 11 bitova, svaki se kromosom sastoji od 6710 bitova, odnosno gena. Dakle prosječno mutira 20 gena u svakom novom kromosomu koji je dobiven križanjem.

Vidi se da kako vjerojatnost mutacije raste tako se dobivaju sve slabija rješenja. To je zato jer se genetski algoritam počinje ponašati kao obični algoritam slučajne pretrage prostora rješenja (ovisi o potpuno slučajnom odabiru hoće li doći do optimuma). Slično tome može se dogoditi da pri manjim vjerojatnostima mutacije (0,05%, a slični rezultati su i za manje vrijednosti) algoritam zapne u lokalnom optimumu.

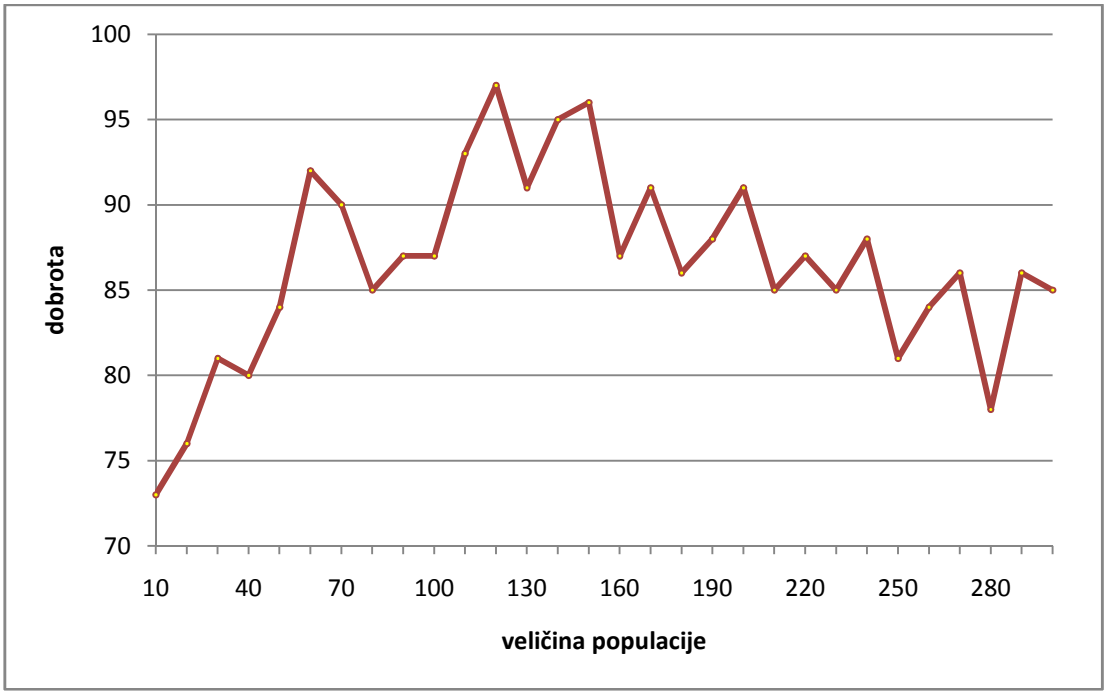


Slika 7.5. Usporedba napredovanja najbolje jedinke za različite vjerojatnosti mutacije

Slika 7.5. prikazuje napredovanje najbolje jedinke za tri različita iznosa vjerojatnosti mutacije, za 0,1%, 0,3% i 2%. Korišteno je 1000 ulaznih podataka. Za 0,3% je dobivena gotovo idealna krivulja napretka najbolje jedinke. U početku se jedinka iznimno brzo poboljšava, zatim dolazi do *koljena* nakon kojeg se značajno uspori rast krivulje, gotovo da ga i nema. Takav izgled grafa označava da je genetski algoritam obavio temeljitu pretragu prostora rješenja, te imamo sigurnije kretanje prema globalnom optimumu. U idealnom slučaju, ovaj graf bi se trebao poklapati s grafom eksponencijalne funkcije i asimptotski težiti globalnom optimumu. Za 0,1% graf malo više odstupa od idealnog grafa. Iako je krajnji rezultat za 0,1% i 0,3% gotovo identičan (68,2% točno klasificiranih uzoraka naspram 68,6%) vidi se da je za 0,1% algoritam gotovo zapeo u lokalnom optimumu kod 30000. iteracije. Za vjerojatnost mutacije od 2% izgled krivulje najviše odstupa od idealne. Takav izgled je naznaka prevelike vjerojatnosti mutacije koja genetski algoritam približava algoritmu slučajnog pretraživanja prostora rješenja. To što je dobiveno lošije rješenje je

zapravo potpuno slučajno, jer je isto tako algoritam mogao doći i do boljeg rješenja. No genetski algoritam bi trebao što je moguće manje ovisiti o slučajnosti, tako da je bolja manja vjerojatnost mutacije.

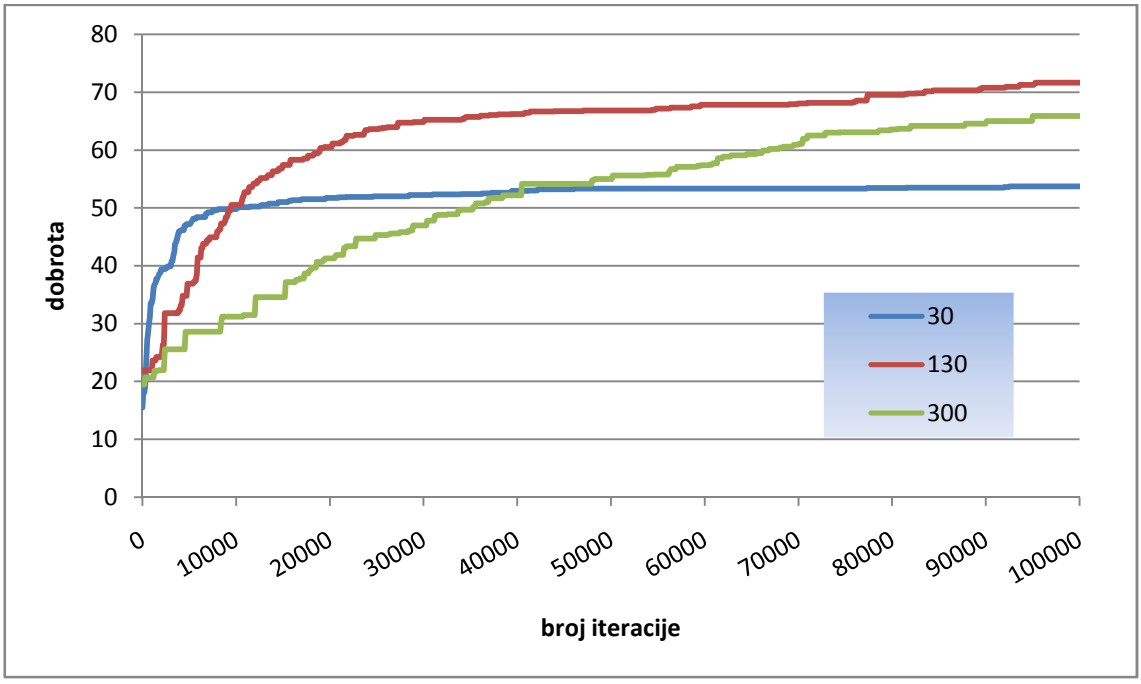
7.3.2. Ovisnost kvalitete rješenja o veličini populacije



Slika 7.6. Dobrota u ovisnosti o veličini populacije

Slika 7.6. prikazuje osjetljivost kvalitete konačnog rješenja o veličini populacije. Za sve veličine populacije od 10 do 300 (s korakom 10) je provedeno tri mjerenja te je uzet najbolji rezultat. Za vjerojatnost mutacije je uzeto 0,3%. Za manje populacije se vidi da algoritam ne dolazi do najboljeg mogućeg rješenja, jer nema velike genetske raznolikosti, pa populacija lako zapne u lokalnim optimumima. Pri većoj populaciji dobiva se dovoljna raznolikost, pa algoritam dolazi do boljeg rješenja. Najbolja rješenja kod 100000 iteracija su za interval od 110 do 150 jedinki u populaciji. Kod populacije od 120 jedinki je postignut maksimum od 97 točno razvrstanih od 100 znamenaka. Daljnjim porastom populacije algoritma kvaliteta rješenja ima trend laganog opadanja, ali to ne znači da je algoritam lošiji što ima veću populaciju. Razlog opadanja je taj što za velike populacije algoritam sporije teži prema globalnom optimumu i stoga je potrebno veći broj iteracija kako bi se došlo do optimuma.

Graf za 300 jedinki i nakon 100000 iteracija još nije došao do koljena (Slika 7.7.), nego još uvijek napreduje gotovo jednoliko. Da bi dobio izgled krivulje karakteristične za genetski algoritam potrebno mu je još mnogo iteracija. Za 130 jedinki graf više podsjeća na traženu eksponencijalnu krivulju koja ima usporen rast kako teži k optimumu. No i za male populacije kao što je 30 jedinki vidimo rast i to u ovom slučaju čak najsličniji krivulji karakterističnoj za genetski algoritam. No problem kod malih populacija, kao što se na slici vidi, jest taj da je algoritam zapeo u lokalnom optimumu.

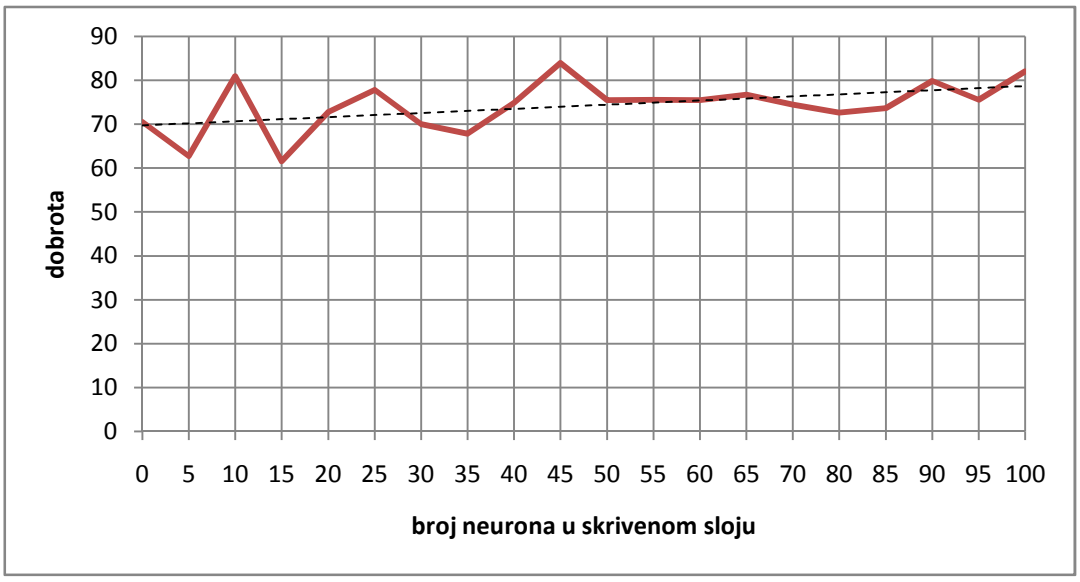


Slika 7.7. Usporedba napredovanja najbolje jedinke za različite veličine populacije

7.4. Analiza rezultata za različite neuronske mreže

Za ispitivanje genetskog algoritma u ovisnosti o broju neurona u skrivenom sloju, te o broju skrivenih slojeva korišteno je 120 jedinki u populaciji, te vjerojatnost mutacije od 0,1%. Svako ispitivanje je provedeno na 1000 podataka i na 100000 iteracija.

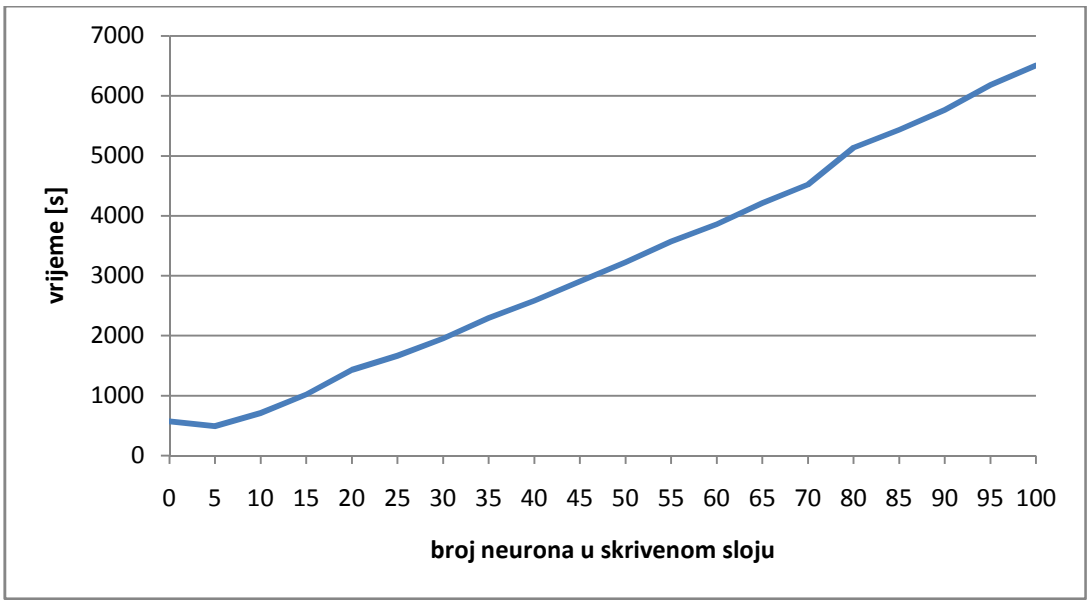
7.4.1. Analiza neuronskih mreža s različitim brojem neurona u skrivenom sloju



Slika 7.8. Dobrota u ovisnosti o broju neurona u skrivenom sloju

Na Slika 7.8 se vidi da pri porastu broja neurona u skrivenom sloju lagano raste i kvaliteta rješenja (dobrota najbolje jedinke) dobivena genetskim algoritmom. Međutim, najbolje rješenje ipak se dobiva za 45 neurona u skrivenom sloju (83,9%), dok jedino još za 10 i za 100 neurona je neuronska mreža uspjela točno klasificirati više od 80% znamenaka.

Podatak od 0 neurona u skrivenom sloju predstavlja neuronsku mrežu bez skrivenog sloja, dok svi ostali podaci predstavljaju neuronske mreže s jednim skrivenim slojem.



Slika 7.9. Vrijeme izvođenja algoritma u ovisnosti o broju neurona u skrivenom sloju

Neuronska mreža s više neurona u skrivenom sloju ima i više težina među neuronima. S obzirom da se kromosom genetskog algoritma sastoji od svih težina neuronske mreže znači da s povećanjem neurona u skrivenom sloju raste i veličina kromosoma.

Neka je N_{ul} broj neurona u ulaznom sloju, N_{sr} broj neurona u srednjem, odnosno skrivenom sloju, te neka je N_{izl} broj neurona u izlaznom sloju. Broj težina kod neuronske mreže sa jednim skrivenim slojem je jednak:

$$broj\_težina = N_{ul} * N_{sr} + N_{sr} * N_{izl} + N_{sr} + N_{izl} \quad \textbf{(7.1)}$$

Dok je broj težina kod neuronske mreže bez srednjeg sloja jednak:

$$broj\_težina = N_{ul} * N_{izl} + N_{izl} \quad \textbf{(7.2)}$$

Slika 7.9. prikazuje vrijeme izvođenja genetskog algoritma u ovisnosti o broju neurona u skrivenom sloju. S porastom broja neurona vrijeme izvođenja gotovo linearno raste. Jedino je na početku vrijeme izvođenja veće za mrežu bez skrivenog sloja nego za mrežu koja ima 5 neurona u skrivenom sloju. Razlog tome je ta što po jednadžbi (7.1), odnosno (7.2), Neuronska mreža sa 5 neurona ima manje težina (310) nego mreža bez srednjeg sloja (500). Dakle vrijeme izvođenja algoritma ovisi o broju težina, odnosno o duljini kromosoma, i to linearno raste s porastom broja težina.

Ako se uzme za primjer 3 najbolja dobivena rješenja može se doći do zaključka da je najbolja mreža sa samo 10 neurona u skrivenom sloju. Do gotovo istih rezultata se dolazi u puno kraćem vremenu.

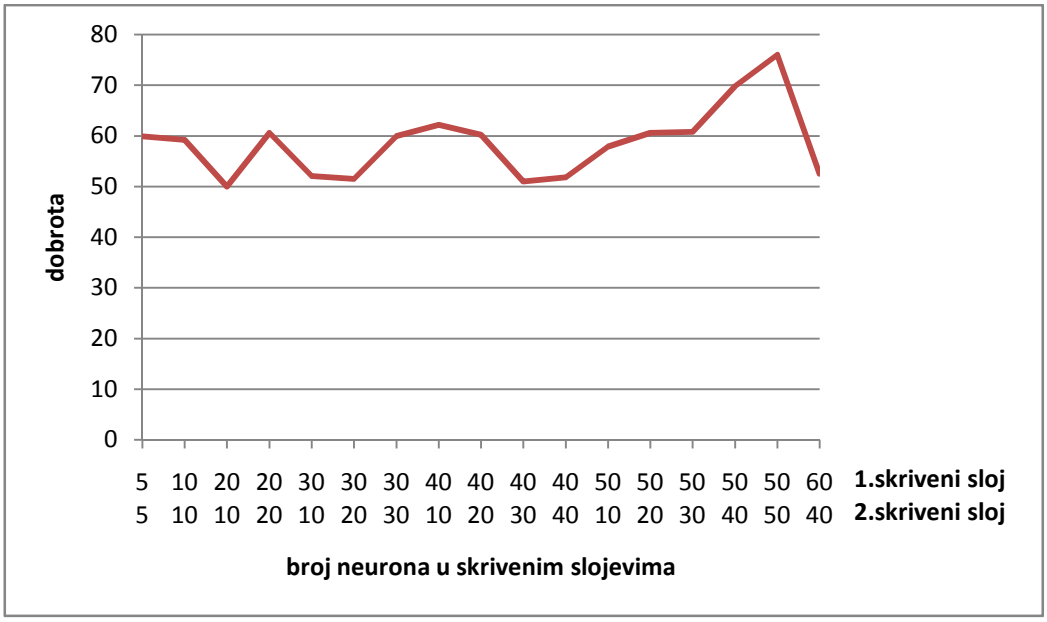
Broj neurona u srednjem sloju	Broj težina u neuronskoj mreži	dobrota	Vrijeme izvođenja algoritma
10	610	80,9%	710 s
45	2710	83,9%	2900 s
100	6010	82%	6505 s

Tablica 7.3. Usporedba tri neuronske mreže s najboljim dobivenim rješenjima

Neuronska mreža sa 45 neurona u skrivenom sloju ima 4,44 puta više neurona od mreže sa 10 neurona te se zbog toga algoritam 4,08 puta sporije izvršava. Isto tako i neuronska mreža sa 100 neurona ima 9,85 puta više težina, te se izvršava 9,16 puta sporije od mreže sa 10 neurona.

7.4.2. Analiza neuronskih mreža s dva skrivena sloja

Kod neuronskih mreža s dva skrivena sloja dobivaju se puno lošiji rezultati nego kod mreže s jednim slojem. Jedino rješenje koje odstupa od ostalih mreža sa dva skrivena sloja je kod mreže koja sadrži 50 neurona u prvom i 50 neurona u drugom skrivenom sloju (76%). Međutim, i to rješenje je lošije od rješenja dobivenih kod mreža s jednim skrivenim slojem.

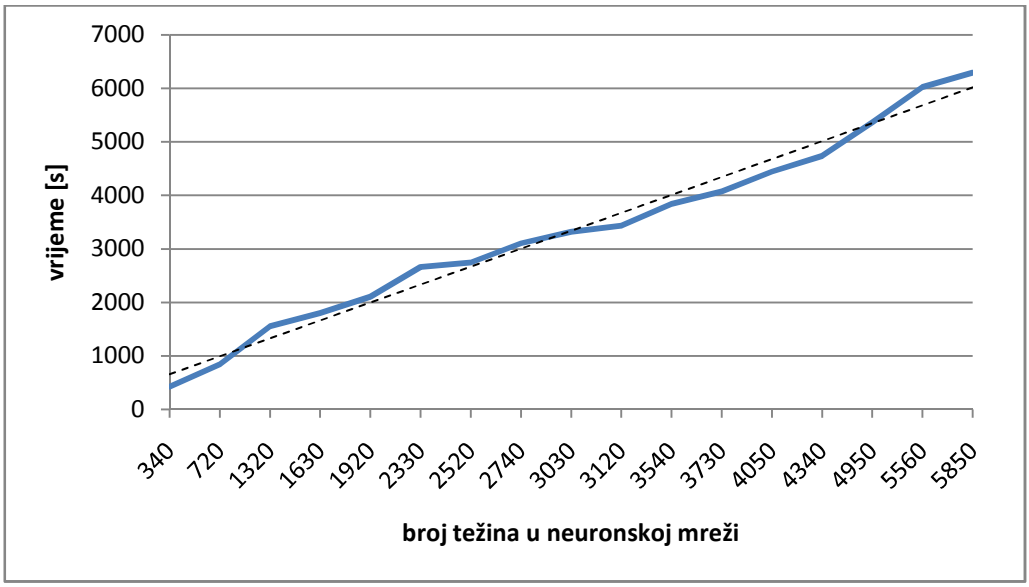


Slika 7.10. Dobrota u ovisnosti o broju neurona u dva skrivena sloja

Slika 7.11. prikazuje rast vremena izvođenja u ovisnosti o broju težina u neuronskoj mreži sa dva skrivena sloja. Kod takvih mreža broj težina se računa prema jednadžbi:

$$broj\_težina = N_{ul} * N_{sr1} + N_{sr1} * N_{sr2} * N_{izl} + N_{sr1} + N_{sr2} + N_{izl} \quad \textbf{(7.3)}$$

Kao i u prethodnom poglavlju dolazi se do istog zaključka: s porastom broja težina gotovo linearno raste i vrijeme izvođenja algoritma.

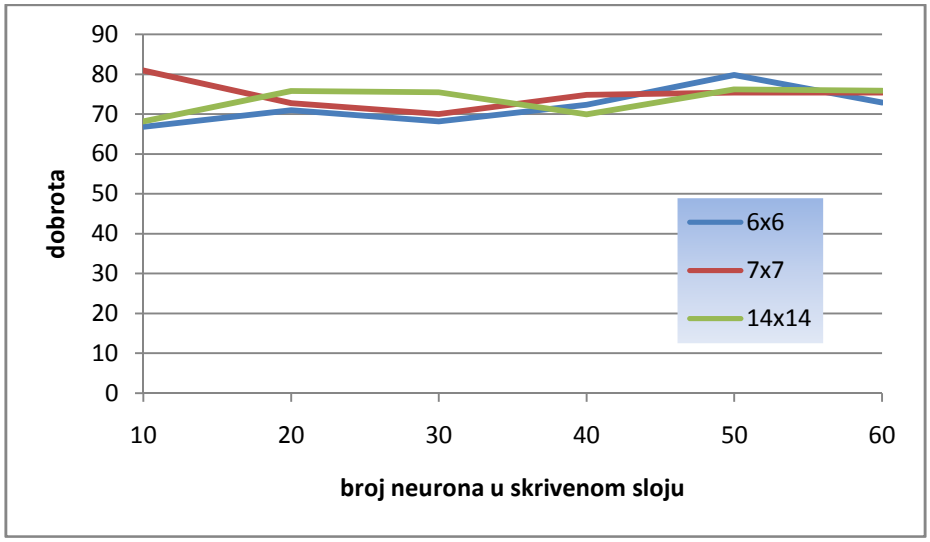


Slika 7.11. Vrijeme izvođenja algoritma u ovisnosti o broju težina u neuronskoj mreži

7.5. Ovisnost kvalitete rješenja o ulaznim podacima

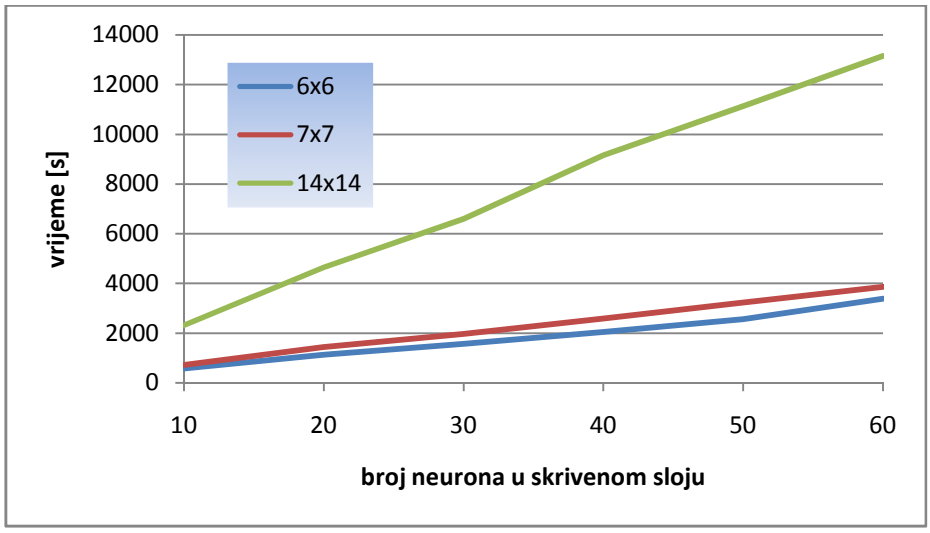
U svim dosadašnjim analizama rezultata za ulazne podatke su korišteni uzorci dimenzija 7x7. Radi usporedbe, na Slika 7.12, su prikazana dobivena rješenja za tri različita izbora dimenzija ulaznih uzoraka u ovisnosti o broju neurona u skrivenom sloju neuronskih mreža. Crvenom bojom je su označeni rezultati dobiveni pri korištenju ulaznih uzoraka dimenzije 7x7, plavom bojom 6x6, a zelenom 14x14 piksela.

Može se primijetiti da su rješenja u većini slučajeva malo bolja za raster 7x7 nego za 6x6, osim kod slučaja neuronske mreže s 50 neurona u srednjem sloju. Ako se usporedi rješenja dobivena pri korištenju rastera 7x7 s onima pri korištenju 14x14 vidi se da nema velikog poboljšanja pri korištenju većeg rastera, dapače kod nekih slučajeva su rezultati bolji za 7x7 piksela.



Slika 7.12. Usporedba dobivenih rješenja za ulazne uzorke različitih dimenzija

S obzirom da su dobiveni rezultati približno isti, iz Slika 7.13. se može zaključiti da je nepotrebno koristiti raster 14x14, jer se znatno uspori vrijeme izvođenja. Kod rastera 7x7 se dobivaju malo bolji rezultati nego kod 6x6, ali se zato algoritam malo sporije izvršava. S obzirom da se algoritam ne usporava značajno, ipak je bolje koristiti raster 7x7.



Slika 7.13. Usporedba vremena izvođenja algoritma za ulazne uzorke različitih dimenzija

7.6. Raspoznavanje rukom pisanih slova hrvatske abecede

Za ispitivanje rada neuronske mreže na znakovima, a ne samo znamenkama od 0 do 9, stvorena je baza od 1024 znaka hrvatske abecede. Jedina slova hrvatske abecede koja nisu uzeta u obzir su ona koja se sastoje od dva znaka, kao što su dž, nj i lj. Dakle baza sadrži 27 različitih znakova.

Baza je slična MNIST bazi podataka te sadrži dvije datoteke:

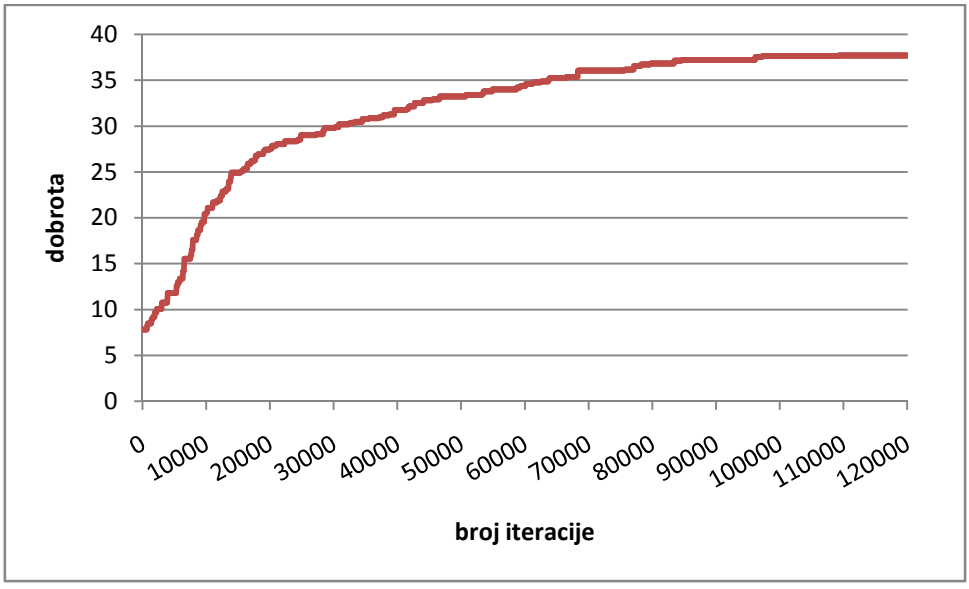
- *bitmaps.txt* – sadrži slike skeniranih znakova za treniranje dimenzija 28x28 piksela
- *labels.txt* – sadrži popis znakova koji predstavljaju skenirane slike

Broj izlaznih neurona u neuronskoj mreži je promijenjena sa 10 na 27, jer nova mreža ne prepoznaje samo 10 znamenaka, već 27 slova hrvatske abecede. Svaki izlazni neuron predstavlja jedan znak abecede. Sve ostalo je isto kao i kod treniranja neuronske mreže za prepoznavanje znamenaka.

Neuronska mreža:	
- broj ulaznih neurona	49
- broj neurona u srednjem sloju	10
- broj izlaznih neurona	27

Tablica 7.4. Parametri neuronske mreže za prepoznavanje slova hrvatske abecede

Na Slika 7.14 se vidi da treniranje takve mreže za prepoznavanje znakova hrvatske abecede ne daje dobre rezultate. Algoritam je nakon 120000 iteracija uspio točno klasificirati samo 37,7% znakova. Povećanjem broja iteracije mogu se postići malo bolji rezultati, ali ne značajno.



Slika 7.14. Napredovanje najbolje jedinke genetskog algoritma

8. Zaključak

Genetski algoritmi su vremenski zahtjevni algoritmi optimiranja i pošto se temelje na evoluciji, koja se odvijala paralelno, sasvim je logično da se mogu znatno ubrzati ako se paraleliziraju. U idealnom slučaju paralelizacijom genetskog algoritma bi se trebalo dobiti linearno ubrzanje izvođenja algoritma, što je u ovom diplomskom radu i postignuto sa modelom GPGA gdje svi obavljaju sve genetske operatore (i gospodar i sluge). Linearno ubrzanje je postignuto zbog rada svih procesora nad zajedničkom populacijom bez gubljenja vremena na međusobnu komunikaciju. Za to je potrebno da svi procesori koriste zajedničku memoriju. Cijeli algoritam će trajati koliko treba najsporijem procesoru da obavi svoj posao do kraja.

Promjenom ulaznih parametara genetskog algoritma može se značajno popraviti konačno rješenje. Ispitivanjem rada genetskog algoritma sa različitim parametrima najbolji rezultati za 100 000 iteracija pri korištenju 3-turnirske eliminacijske selekcije su postignuti sa 120 jedinki u populaciji i vjerojatnosti mutacije od 0,3%.

Korištenje genetskog algoritma za treniranje neuronske mreže zna biti dugotrajan proces čije vrijeme trajanja najviše ovisi o veličini neuronske mreže, tj. o broju veza među neuronima. Zbog jednostavnosti korištene neuronske mreže nisu dobiveni odlični rezultati. Stoga bi budući rad trebao više pozornosti posvetiti neuronskoj mreži te predprocesiranju ulaznih podataka, odnosno obradi slika ulaznih znakova i izvlačenja svojstava koja su karakteristična za pojedine znakove (zakrivljenost slova, zatvorene krivulje, ravne crte itd.).

9. Literatura

[1] Golub, M: Genetski algoritam – prvi dio, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta1.pdf , 2004.

[2] Golub, M: Genetski algoritam – drugi dio, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/ga_skripta2.pdf , 2004.

[3] Wikipedia.org: On the Origin of Species, s Interneta, http://en.wikipedia.org/wiki/On_the_Origin_of_Species , 26. lipnja 2008.

[4] Tonči Damjanić: Primjena evolucijskih algoritama za rješavanje aproksimacijskog problema, diplomski rad, FER, Zagreb, Hrvatska, s Interneta, http://www.zemris.fer.hr/~golub/ga/studenti/damjanic/Diplomski.html , 2008.

[5] Kishan Mehorotra, Chilukuri K.Mohan, Sanjay Ranka: Elements od Artificial Neural Networks, MIT Press, 1996.

[6] Willamette University: Neural Network, s Interneta http://www.willamette.edu/~gorr/classes/cs449/intro.html , 26. lipnja 2008.

[7] Simon Haykin: Neural Network - A Comprehensive Foundation, Pearson Education, 2005.

[8] Fakultet organizacije i informatike: Edukacijski Repozitorij za Inteligentne Sustave, s Interneta, http://eris.foi.hr/11neuronske/nn-predavanje.html , 26. lipnja 2008.

[9] Umjetne neuronske mreže, nastavni materijal, FER-a, s Interneta http://www.zemris.fer.hr/predmeti/nenr/nastava/ANNv441part123.pdf , 26. lipnja 2008.

[10] Neuronske mreže / Predavanja, FER, s Interneta http://nm.zesoi.fer.hr/predavanja.html , 26. lipnja 2008.

[11] Sveučilišni računarski centar (SRCE): Klaster Isabella, s Interneta, http://www.srce.hr/isabella/ , 26.lipnja 2008.

[12] Wikipedia.org: Neural Network, s Interneta, http://en.wikipedia.org/wiki/Neural_network , 26. lipnja 2008.

[13] Wikipedia.org: Genetic Algorithm, s Interneta, http://en.wikipedia.org/wiki/Genetic_algorithm , 26. lipnja 2008.

[14] Yann LeCun, Corinna Cortes: The MNIST Database of handwritten digits, s Interneta, http://yann.lecun.com/exdb/mnist/ , 26. lipnja 2008.

[15] Christopher M.Bishop: Neural Networks For Pattern Recognition, Clarendon Press – Oxford, 1995.