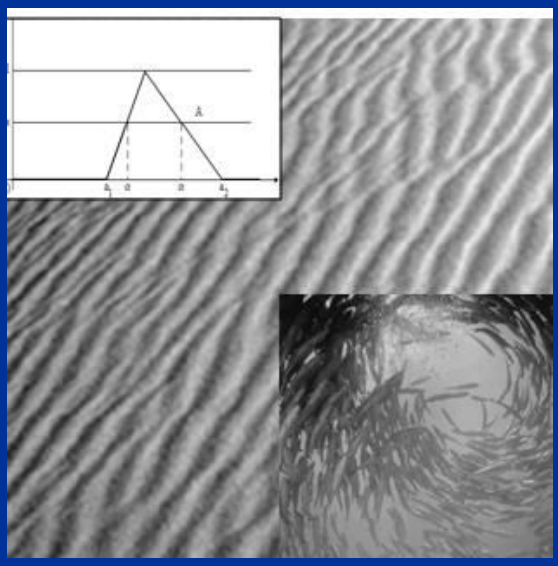
UVOD U VEŠTAČKU INTELIGENCIJU



Vesna Šešum Čavić

UVOD U VEŠTAČKU INTELIGENCIJU

Vesna Šešum- Čavić

Recenzenti:

Prof. Dr. Slobodanka Mitrović, dipl. mat

Prof. Dr. Goran Lazović, dipl. mat

v. Prof. Dr. Milan Kilibarda, dipl. inž. geod.

Izdavač

Univerzitet u Beogradu, Građevinski fakultet u Beogradu

ISBN:

978-86-7518-235-1

Ilustracija na koricama: kombinacija slika iz knjige (1.4, 2.4 i 5.7)

PREDGOVOR

Tema kojom se bavi ova knjiga je računarska ili kompjutaciona inteligencija. Iako je bliska veštackoj inteligenciji, te smatrana kao podoblast iste (Bezdek, 1998), i dalje ne postoji zvanično prihvaćena opšta definicija ove naučne (pod)oblasti. Veštačka i kompjutaciona inteligencija se mogu posmatrati kao dva tipa mašinske inteligencije: veštačka inteligencija bazirana na tkzv. "hard computing" tehnikama i kompjutaciona inteligencija bazirana na tkzv. "soft computing" metodama koje omogućavaju visoku adaptibilnost u mnogim situacijama.

Veštačka inteligencija se nalazi u fokusu istraživanja već dugi niz godina. Zasnovana je kao akademska disciplina 1956 godine. Od tada se razvijala u fazama, raznolikim intenzitetom. Naročit podsticaj u razvoju dešava se tokom 1980-tih godina, dok još ubrzaniji razvoj i grananje podoblasti istraživanja javljaju se krajem prošlog veka i početkom 2000 godine. U poslednjih 20 godina, veštačka inteligencija sve više dobija na značaju zahvaljujući inovativnim tehnikama i metodama, te brojnih uspešnim aplikacijama.

Kompjutaciona inteligencija se diferencira početkom 1990-tih godina i predstavlja skup kompjutacionih metoda inspirisanih procesima iz prirode (engl. nature-inspired, bio-inspired), koji se primenjuju na probleme iz stvarnog života karakterisane visokom kompleksnošću i stoga, nemogućnošću da budu rešeni tradicionalnim matematičkim modeliranjem. Naime, za takvu vrstu kompleksnih problema je karakteristično: 1) kompleksnost može biti veoma izražena što onemogućava tradicionalno matematičko rezonovanje; 2) mogu sadržavati neizvesnost, tj. tada govorimo o neizvesnim procesima gde očekujemo nedovoljno poznate ili nepotpune informacije; to uslovljava potrebu za anticipiranjem budućih događaja i njihovih ishoda; neizvesnost se najčešće javlja u stohastičkim okruženjima; 3) proces može biti stohastički po svojoj prirodi. U tako kompleksnim situacijama gde raspolazemo sa nekompletnom informacijom, kompjutaciona inteligencija nudi rešenje.

Kompjutaciona inteligencija objedinjuje sledeće komplementarne tehnike za koje možemo da kažemo da su osnovni stubovi kompjutacione inteligencije[1]: rasplinuta logika (engl. fuzzy logic), neuronske mreže (engl. neural networks), evolutivnu kompjutaciju (engl. evolutionary computation), i probabilističke metode.

U ovoj knjizi obrađuju se upravo navedene podoblasti kompjutacione inteligencije i odgovarajuće tehnike koje se mogu primenjivati u inženjerskim aplikacijama. Algoritmi razmatrani u okviru ove knjige obezbedili su inovativan pristup brojnim kompleksnim problemima iz prakse sa obećavajucim rezultatima. Zato je cilj ove knjige da predstavi na jednostavan način neke od najvažnijih podoblasti kompjutacione inteligencije. Knjiga se sastoji iz tri dela sa ukupno 8 poglavlja i dodatka (na kraju knjige).

Prvi deo je uvodni deo.
Poglavlje 1: U ovom uvodnom poglavlju, dat je kratak istorijski pregled nastanka kompjutacione inteligencije, predstavljeni su ciljevi i motivacija za upotrebu metoda kompjutacione inteligencije i opisani tipovi problema na koje se ove metode najuspesnije primenjuju. Takođe, ovo poglavlje sadrži informacije za koje bismo mogli reći da su preduslov za bolje razumevanje navedenih metoda kao što su osnovni pojmovi o optimizaciji, o metaheuristikama i samo-organizovanju.

[1] https://cis.ieee.org/about/what-is-ci

Drugi deo se sastoji od 4 poglavlja i razmatra osnovne metode kompjutacione inteligencije.
Poglavlje 2: U ovom poglavlju navedeni su neki od najznačajnijih metaheuristika zasnovanih na inteligenciji roja (engl. swarm based metaheuristics).
Poglavlje 3: U drugoj skupini nalaze se evolutivni algoritmi (izabrani su genetski algoritmi kao najčešće korišćeni iz ove kategorije).
Poglavlje 4: U ovom poglavlju fokus je na osnovnim metodama i algoritmima veštačkih neuronskih mreža.
Poglavlje 5: U poslednjem poglavlju drugog dela razmatrani su osnovni postulati rasplinutih sistema.
Svako poglavlje je nezavisno jedno od drugog i može se pratiti i čitati u redosledu koji je za čitaoca interesantan ili bitan.

Treći deo sadrži metodologiju i načine podesavanja parametara, neke od primena navedenih metoda u geoinformatici, kao i zaključak.
Poglavlje 6: Algoritmi navedeni u knjizi obuhvataju veliki broj parametara. Iako postoje preporučene vrednosti parametara, za svaki problem se treba odrediti njihova optimalna vrednost. Ručno podešavanje vrednosti parametara je neprihvatljivo. Stoga su razvijene metode za automatsko podešavanje vrednosti parametara.
Poglavlje 7: Navedne metode imaju primenu u aplikacionim scenarijima geoinformatike. U ovom poglavlju su razmatrani neki od njih.
Poglavlje 8: Poslednje poglavlje jest ezaključno i sumarizuje osnovne poruke prezentovane u knjizi.

Dodatak
Radi ilustracije, navedeni algoritama iz drugog dela predstavljeni su odgovarajućim primenom na nekom od poznatih problema i implementacijom korišćenjem Matlab-a.

Napomena: U knjizi je korišćena i terminologija na engleskom jeziku.

Kome je knjiga namenjena?
Knjiga je prvenstveno namenjena studentima master akademskih studija Geoinformatike na Građevinskom fakultetu Univerziteta u Beogradu koji slušaju istoimeni predmet Uvodu u veštačku inteligenciju u drugom semestru. Treba naglasiti da su u knjizi date samo kratke napomene iz oblasti mašinskog učenja, tj. mašinsko učenje nije obrađeno kao zasebna tema/poglavlje iz razloga što studenti master akademskih studija Geoinformatike već slušaju predmet Mašinsko učenje u prvom semestru.

Knjiga takođe može poslužiti i studentima koji se bave različitim inženjerskim problemima, tj. problemima iz različitih i/ili multidisciplinarnih inženjerskih oblasti (informatike, geoinformatike, primenjene matematike, elektrotehnike, mašinstva, industrijskog inženjerstva, itd).

Podrazumeva se da studenti već poseduju osnovna znanja iz algoritmike, kompleksnosti algoritama i strukture podataka.

Autor, Boegrad 2023.

O AUTORU

Vesna Šešum-Čavić je vanredni profesor na Građevinskom fakultetu, Univerzitet u Beogradu. Diplomirala je i magistrirala na Matematičkom fakultetu, Univerzitet u Beogradu. Doktorirala je na Fakultetu za Informatiku, Tehnički Univerzitet Beč, gde je provela 15 godina u nastavno-naučnom radu.
Oblasti istraživanja pokrivaju teoriju i dizajn algoritama, kombinatornu optimizaciju, kompjutacionu inteligenciju, swarm intelligence, kompleksne sisteme, self-organization, multi-agent sisteme, optimizaciju mreža, i skalabilne distribuirane sisteme. Autor je ili koautor preko 50 naučnih radova. Učesnik je na više međunardnih i domaćih istraživačkih projekata. Član je Uređivačkih odbora nekoliko naučnih časopisa - Editorial Board member (Associate Editor) kao i organizacionih odbora naučnih skupova. Takođe, član je u profesionalnom udruženju međunarodnog nivoa (IEEE CIS, IEEE Women in Engineering Committee), a jedno vreme (tokom 2019) je bila i predsedavajuća subkomiteta IEEE Women in Computational Intelligence i član IEEE CIS Member Activities Committee. Udata, ima kćerku Lanu.

SADRŽAJ

PRVI DEO
UVOD

§1 UVODNE INFORMACIJE

Veštačka inteligencija (engl. Artificial Intelligence, AI) je široka oblast koja u današnje vreme uključuje brojne metode i bavi se rešavanjm raznolikih problema iz realnog života, obično komplikovanih i teško rešivih drugim pristupima. Kao što samo ime govori, "veštačka", misli se na "inteligenciju" koja je "ostvarena" preko računara, a zapravo preslikana iz prirodnih procesa.

Mnogi procesi u prirodi, označeni kao inteligenti, još uvek su nepotpuno objašnjeni čak i sa biološke strane. Sama definicija inteligencije i reprezentacija inteligencije je u oblasti aktivnog naučnog istaživanja. Prema jednoj od definicija (Engelbrecht, 2022; Siddique & Adeli, 2013), inteligencija je sposobnost razumevanja, rezonovanja i učenja. To implicira da je inteligentni sistem sposoban da razume okruženje i/ili procese u okruženju, rezonuje i identifikuje razne promenljive vezane za proces ili okruženje, njihove veze i uticaj na okruženje. Dalje, inteligentni sistem treba biti sposoban da uči iz okruženja, novonastalih situacija i procesa (Engelbrecht, 2022; Siddique & Adeli, 2013).

Veštačka inteligencija pokušava da simulira inteligentno ponasšanje u sistemu zahtevajući tačnu i kompletnu informaciju i prikaz znanja. U realnosti, mnogi sistemi ne mogu biti opisani na takav način. Često se kod izrazito kompleksnih sistema javljaju zahtevi obrade situacija koje uključuju nepredvidive, nagle i neočekivane promene u okruženju. Kao posledica suočavamo se sa ograničenim mogućnostima konvencionalnog modeliranja, naročito kada se radi kompleksnim sistemima sa izrazito dinamičkim procesima.

Kompjutaciona inteligencija (engl. Computational Intelligence, CI) podrazumeva upotrebu inteligentnih tehnika i algoritamskih modela koje se primenjuju na procese karakterisane nekompletnom definicijom, nelinearnošću, vremenskim variranjem i stohastikom. Stoga inteligentni sistemi koriste tehnike koje uspešno deluju bez *a priori* znanja o okruženju i koje su fleksibilne, adaptivne i robustne. U opštem slucaju, inteligentni algoritmi uključuju *evolucionu kompjutaciju* (engl. evolutionary computing), *inteligenciju roja* (engl. swarm intelligence), *veštacke neuronske mreze* (engl. artificial neural networks) i *rasplinute skupove* (engl. fuzzy sets).

Ova knjiga fokusira se na upravo gore navedene oblasti CI koje proučavaju adaptivne mehanizme za omogućavanje inteligentnog ponašanja u kompleksnom i promenljivom okruženju. Navedene tehnike su uspešno primenjene na brojne aplikacije i probleme iz prakse. U poslednje vreme trend razvoja hibridnih tehnika je izražen imajući u vidu da nijedna tehnika nije generalno superiorna u svim situacijama.

1.1. Kratak istorijski pregled

Počeci primene veštačke inteligencije datiraju iz 1956. godine. U tom periodu govorilo se o mašinskoj inteligenciji, i inicirana su dva dominantna pristupa: 1) korišćenje računara za kreiranje simboličke reprezentacije sistema, i veoma blizak pristup heurističke pretrage, koji podrazumeva istraživanje prostora pretrage, tj., prostora mogućnosti za nalaženje rešenja; 2) pristup koji povezuje inteligenciju sa "učenjem" (engl. connectionist approach). Dakle, na ovom polju, početna istraživanja su se u pojedinim podoblastima odvijala paralelno. Tako je u periodu od 1956 do 1969 obavljen deo istraživanja vezano za modeliranje bioloških neurona, a takođe početkom 1950-tih godina pojavljuju se i inicijalne ideje evolucione kompjutacije. John Holland je već početkom 1960-tih godina razvio ideju genetskih algoritama, koja se je dalje usavršavala tokom 1970-tih godina. Istraživanja vezana za veštačke

neuronske mreže su naročito došla u fokus krajem 1980-tih sa brojnim aplikativnim scenarijima. Takođe sredinom 1960-tih godina, Lofti Zadeh je razvio teoriju rasplinutih skupova i veoma doprineo razvoju rasplinute logike.

Veštačka inteligencija se je rapidno razvijala, preko ekspertskih sistema i brojih drugih oblasti tkzv. "soft computing" metoda kao sto su neuronske mreže, rasplinuti (fuzzy) sistemi, evolutivna kompjutacija, kao i brojnih drugih pristupa izvedenih iz statistike i matematičke optimizacije.

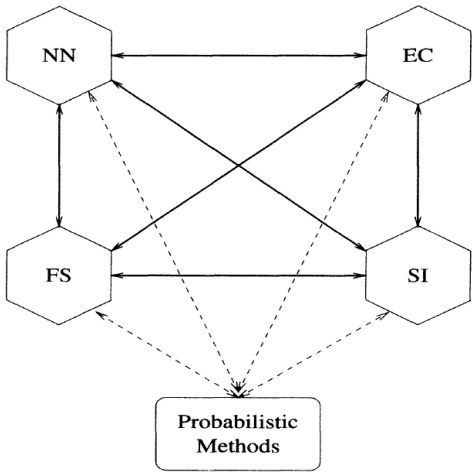
Krajem prethodnog veka i početkom novog, dolazi do novog uspona veštačke inteligencije s obzirom na pronalaženje specifičnih rešenja za neke određene probleme, poboljšanja teorijske zasnovanosti metoda u smislu saradnje sa oblastima matematike i statistike, a samim tim i verifikacijom dobijenih rezultata.

Početkom 1990-tih, pojavljuje se istraživanje vezano za razvoj metoda i algoritama baziranih na inteligenciji roja (engl. swarm-based intelligence). Marco Dorigo je modelirao kolonije mrava formirajući odgovarajuće algoritme.

U poslednjih 20-tak godina, brojni faktori utiču na dominaciju veštačke i kompjutacione inteligencije, npr. brzi računari, brojna algoritamska poboljšanja, pristup velikoj količini podataka. U poslednjih 10 godina beleži se brzi razvoj i korišćenje metoda mašinskog učenja i tkzv. tehnike dubokog učenja (engl. deep learning).

1.2. Paradigme i pristupi kompjutacione inteligencije

Kao sto je već napomenuto u uvodu, ova knjiga razmatra četiri osnovne paradigme kompjutacione inteligencije: evolucionu kompjutaciju, inteligenciju roja, veštačke neuronske mreže i rasplinutu logiku (Engelbrecht, 2022). Slika 1.1 je šematski prikaz ovih paradigmi sa njihovim interkonekcijama koje ukazuju na mogućnost formiranja hibridnih tehnika. Na slici je navedena još jedna komponenta – probabilističke metode koje se često koriste zajedno sa CI tehnikama i/ili su inkorporirani u same tehnike.



Slika 1.1. CI paradigme: neuronske mreže (NN), evoluciona kompjutacija (EC), inteligencija bazirana na roju (SI), rasplinuti skupovi (FS) i njihova veza sa probabilističkim metodama (Engelbrecht, 2022).

Svaka od navedenih paradigmi je inspirisana mehanizmima iz bioloških sistema. Karakteristično za CI jeste da tretira procese koji nisu podlozni tradicionalnom, matematičkom modeliranju iz razloga da su suviše kompleksni, nelinearni, distribuirani, nekompletni, te da su po prirodi stohastički (Engelbrecht, 2022; Siddique & Adeli, 2013). Kompleksni sistemi imaju sposobnost samo-organizovanja, mogućnost da uče i rade sa novim i nepoznatim situacijama, kao i da anticipiraju i/ili donose odluke u vezi budućih događaja. CI objedinjuje mnoge naučne oblasti i potpuno je multidisciplinarna.

U ovoj knjizi, četiri gore navedene metodologije su razmatrane. Ostale metodologije su referisane kao metode podrške u CI.

Rasplinuti skupovi i rasplinuta logika omogućavaju tkzv. približno rezonovanje (engl. approximate reasoning). To znači da neki element pripada skupu sa određenim stepenom izvesnosti. Rasplinuta logika dozvoljava rezonovanje ovim "neizvesnim" činjenicama da bi se zaključile nove, sa odgovarajucim stepenom sigurnosti koji je povezan sa svakom činjenicom. Neizvesnost u rasplinutim sistemima se naziva nestatistička neizvesnost/nepouzdanost i ne treba je mešati sa statističkom neizvesnoscu/nepouzdanosti (Engelbrecht, 2022).

Veštačke neuronske mreže prate analogiju bioloških neuronskih mreža. One predstavljaju biološki inspirisane, paralelne i distribuirane informaciono-procesirane sisteme. Proces učenja u ovim mrežama može se odvijati u razlicitim formama, kao sto je nadgledano, nenadgledano, kompetitivno i podržano (engl. reinforcement) učenje (Siddique & Adeli, 2013).

Evolutivna kompjutacija sledi procese prirodne selekcije iz Darvinove teorije evolucije. Po tome, organizmi u prirodi poseduju odgovarajuće karakteristike koje utiču na njihovu sposobnost preživljavanja u nepredvidivom okruženju i koje prenose na sopstveno potomstvo u "poboljšanoj" formi, tako da potomstvo postaje jos bolje prilagođeno uslovima okruženja. Genetske informacije se kodiraju u hromozome, koje predstavljaju ove karakteristike. Jedinke ulaze u proces reprodukcije – formiranja potomaka, tj. sledeće generacije koja bi trebala da poseduje još bolje karakteristike u smislu prilagođavanja okruženju. Ovaj proces prirodne selekcije implicira da bolje prilagođene jedinke imaju vise šansi za dobijanjem većeg broja potomaka sa sličnim i/ili boljim karakteristikama (tj., sa sličnom ili boljom fitness funkcijom) (Siddique & Adeli, 2013).

Mehanizmi bazirani na inteligenciji roja koriste biološke mehanizme socijalnog ponašanja insekata u kolonijama (npr., mravi, pčele) ili mehanizme grupisanja riba i/ili jata ptica. Tokom njihovog proučavanja pokazalo se je da su ove kolonije veoma organizovane ka rešavanju zajedničkog zadatka – globalnog cilja, pri čemu se svaka jedinka ponaša potpuno autonomno, distribuirano i lokalno donosi odluke. Princip samo-organizovanja u ovim kolonijama je fascinantan, jer ne postoji nijedan centralni koordinator niti bilo kakav vid supervizije. Lokalna interakcija između jedinki vodi ka pojavljivanju globalnog ponašanja i posledično formiranju paterna (rešenja problema) (Siddique & Adeli, 2013).

1.3. Opšte informacije

Izbor odgovarajuće metode je u direktnoj vezi sa tipom problema koji se rešava. U drugom delu, predstavljeni su metodi upravo prema prevalentnom tipu problema. U poglavljima 2 i 3 opisane se razlicite metode i algoritmi koji se uspešno primenjuju kod problema optimizacione prirode, naročito kada su ti problemi visoke kompleksnosti (tkvz. NP-hard[2]) i kada je potrebno naći približno-optimalno rešenje (engl. near-optimal solution). U poglavlju 4 opisane su tehnike koje se koriste kod problema klasifikacije, grupisanja i predvidjanja.

Pre samog opisa nekih od najpoznatijih metoda i algoritama kompjutacione inteligencije, uvedeni su optimizacioni problemi i njihova klasifikacija (sekcija 1.3.1), objašnjene su osnovne karakteristike

[2]Nedeterministički u polinomijalnom vremenu teški

algoritama (sekcija 1.3.2), principi na kojima počivaju kompleksni sistemi i samo-organizovanje (engl. self-organization), kao i inteligentni agenti (sekcije 1.3.3 i 1.3.5).

1.3.1 Tipovi problema

Uopšteno govoreći, brojne su primene vestačke inteligencije: napredni pretraživaci (web search engines), razumevanje ljudskog govora, automatsko donosenje odluka, automatska vozila (dronovi i self-driving automobili), automatski prevodioci (npr. Google translate), označavanje slika i filterovanje spamova. Takođe, brojne su primene vestačke inteligencije u industriji, npr. medicinska dijagnostika, vojna logistika, upravljanje lancem snabdevanja, itd.
Što se tiče tipova problema na koje se metode vestačke inteligencije mogu primeniti, tu u obzir najčešće dolaze:

- optimizacija
- klasifikacija,
- grupisanje/klasterovanje
- predviđanje,
- prepoznavanje oblika.

Metode bazirani na evolutivnom programiranju i metodi tkzv. swarm-based inteligencije uspešno rešavaju brojne optimizacione probleme. Metode mašinskog učenja pokrivaju ostale nabrojane tipove problema.

1.3.2 Optimizacioni Problemi

Optimizacioni problem je problem pronalaženja najboljeg rešenja od svih mogućih rešenja iz oblasti dopustivih rešenja u kojem je minimalna (ili maksimalna) vrednost objektivne funkcije (Šešum–Čavić, 2020). Zadati problem je najčešće matematički modeliran i funkcija putem koje se opisuje ili predstavlja problem se uzima i obzir u smislu minimizacije ili maksimizacije (zavisno od tipa problema i šta se zapravo traži). Jedan od početnih koraka jeste klasifikacija optimizacionog modela da bi se pravovremeno i na najoptimalniji način uzeli u obzir odgovarajući algoritmi koji bi bili kandidati za rešavanje problema. Naime, algoritmi za rešavanje optimizacionih problema su najčešće predisponirani za odgovarajuci tip problema (npr. neke vrste algoritama konstruisane su tako da rade sa diskretnim podacima; za uspešan rad sa kontinualnim podacima, bilo bi potrebno adaptirati takav algoritam u najboljem slučaju i tom prilikom „adaptacija“ često može voditi do značajnih promena tako da više ne govorimo o novoj verziji datog algoritma, već u nekim slučajevima i o novom algoritmu).
Jedna moguća klasifikacija optimizacionih modela je predstavljen na sledeći nacin[3]:

1. Diskretna optimizacija i kontinualna optimizacija
    - Diskretni optimizacioni problemi: modeli sa diskretnim promenljivim (koje uzimaju vrednosti iz diskretnog skupa, npr. skupa celih brojeva);
    - Kontinualni optimizacioni problemi: modeli sa kontinualnim promenljivim (npr. iz skupa realnih brojeva);
2. Optimizacija sa ograničenjem i optimizacija bez ograničenja
    - Optimizacija sa ograničenjem: postoje inicijalna ograničenja u okviru promenjivih i nastali su iz aplikacija, gde postoje eksplicitna ograničenja na promenljive;

[3] https://neos-guide.org

- Optimizacija bez ograničenja: problemi su direktno proizašli iz raznih praktičnih aplikacija kao reformulacija (uopštenje) optimizationih problema sa ograničenjima, često tamo gde su ograničenja zamenjena sa penalima u objektivnoj funkciji;

3. Nepostojeća, jedinstvena ili multi – optimizaciona funkcija
   Najčešći slučaj kod optimizacionih problema jeste postojanje jedne objektivne funkcije. Međutim, ima slučajeva kada optimizacioni problem ima višestruke objektivne funkcije ili čak nema nijednu. Ako se desi da postoji nekoliko konfliktnih ciljeva, tada se pribegava korišćenju multi-objektivne optimizacije, jer optimalna odluka treba da se donese u prisustvu balansa između tih konfliktnih ciljeva.
4. Deterministička optimizacija i stohastička optimizacija
   - Deterministička optimizacija: iako bi trebalo da su podaci za dati problem tačno poznati, u praksi, međutim, to nije uvek moguće iz raznih razloga.
   - Stohastička optimizacija: faktor neizvesnosti je inkorporiran u sam model.

1.3.3 O metaheuristikama

Algoritam se može definisati kao skup uređenih, precizno definisanih koraka (instrukcija) za rešavanje određenog problema (Blass & Gurevich, 2003). Instrukcije opisuju izračunavanje koje počinje od inicijalnog stanja, nastavlja se uređenim nizom sukcesivnih stanja i konačno se završava. Međutim, tranzicija od jednog stanja do drugog nije uvek deterministička, tj. instrukcije u tim slučajevima ne konstituišu konačan niz. Posebna klasa algoritama sa takvim osobinama su nedeterministički algoritmi (Floyd, 1967) koji imaju jednu ili više tačaka izbora u kojima su mnogobrojni, različiti nastavci mogući, bez specifikacije unapred koji put će biti izabran. Uglavnom takvi algoritmi imaju inkorporirane stohastičke elemente slučajnosti.

Postoje razni načini za klasifikaciju algoritama (Cormen et al., 2009; Goodrich, 2001; Skiena, 2008). Tako algoritmi mogu biti klasifikovani prema:

- implementaciji (rekurzivni vs. iterativni, serijski vs. paralelni, deterministički vs. nedeterministički, tačni vs. aproksimativni),
- dizajnu (iscrpna pretraga - engl. brute force, podeli pa vladaj - engl. divide and conquer, linearno programiranje, dinamičko programiranje, pretraga i optimizacija),
- oblasti/naučnoj disciplini (npr. kombinatorna optimizacija, kompjutacione nauke – engl. computational sciences, procesiranje signala, softversko inženjerstvo, itd.),
- kompleksnosti,
- izračunavajucoj moći (engl. computing power).

Pošto smo već uveli pojam optimizacije, u daljem će fokus biti upravo na algoritmima pretrage i optimizacije.

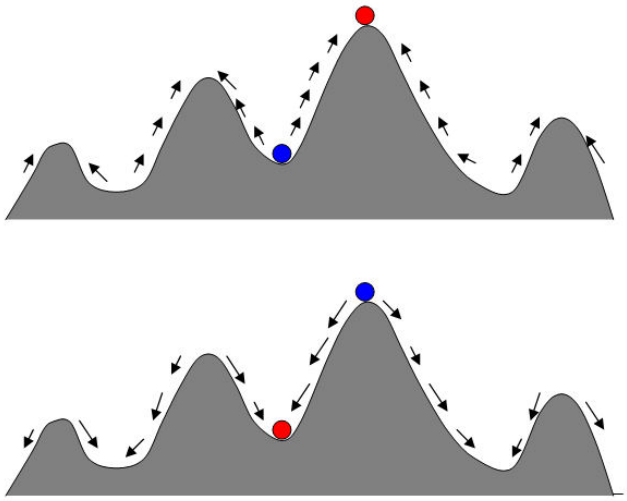
Mnogi problemi kombinatorne optimizacije su tkzv. NP teški problemi (engl. NP-hardness**,** non-deterministic polynomial-time hardness) (Ladner, 1975; Kleinberg &Tardos, 2006). To znači da ne mogu biti (optimalno) rešeni u okviru polinomski ograničenog vremena izračunavanja, naročito u slučajevima kada se izračunavaju velike instance problema. Dakle, u situacijama kada su vreme i/ili resursi ograničeni, aproksimativni algoritmi koji pokušavaju da pronađu aproksimativno rešenje su od pomoći i mogu se koristiti. Ovi algoritmi se obično nazivaju heuristike. Njihova prednost jeste da izračunavaju tkzv. približno-optimalno rešenje (engl. near optimal solution) u relativno kratkom vremenu (Yang, 2008). Dalje su razvijene ekstenzija heuristika, skup algoritamskih koncepata, koji se mogu koristiti za definiciju heurističkih metoda primenljivih na široki skup različitih problema, tkzv. metaheuristike. One povećavaju sposobnost nalaženja kvalitetnih rešenja NP-hard problema u razumnom vremenu (Yang, 2008). Metaheuristika je formalno definisana kao iterativan process koji “upravlja” podređenom

heuristikom inteligentno kombinujući koncepte istraživanja i eksploatacije (engl. exploring & exploiting) prostora pretrage, kao i strategija "učenja" da bi kao krajnji rezultat dobilo približno-optimalno rešenje (Osman & Laporte, 1996). Osnovne osobine metaheuristika su (Blum & Roli, 2003):

- "vode" proces pretrage;
- efikasno istražuju prostor pretrage da bi pronašle približno-optimalno rešenje;
- nalaze se u rasponu od jednostavnih lokalnih pretraga do kompleksnih obrazaca inteligentnog učenja;
- približne su i nedeterminističke;
- osnovni koncepti dozvoljavaju apstraktan nivo opisa;
- nisu problemski specifične;
- mogu koristiti domenski-specifično znanje u formi heuristika koje su kontrolisane od strane neke strategije višeg nivoa.

Evolutivni algoritmi i algoritmi bazirani na inteligenciji roja spadaju u ovu grupu algoritama. Oni su adaptivni – menjaju ponašanje na osnovu raspoloživih resursa, i inteligentni – bazirani su na određenoj formi već postojeće inteligencije iz prirode, te je preslikavaju formirajući veštačku inteligenciju.
1) Obično manipulišu sa populacijom jedinki, i rešenje se pronalazi kroz mnogobrojne iteracije razmatrane populacije. Preslikavanje koristi odgovarajući matematički model.
2) Korisnost i prilagođenost odgovarajuće jedinke u populaciji se "proverava" pomoću tkzv. funkcije prilagođenosti (engl. fitness function), koja predstavlja matematički opis ciljeva algoritma i dinamike sistema. Funkcija prilagođenosti se definiše kao funkcija f definisana na prostoru pretrage, f: $S \rightarrow \mathbf{R}$, $f = f(s)$. Svaka tačka prostora pretrage ima odgovarajuću "težinu" koja odgovara njenoj vrednosti fitnes funkcije. Trajektorija sistema prolazi kroz različita stanja (pomera se od datog stanja s ka susednom stanju) za koje je f optimalna[4]. Fitnes funkcija predstavlja stepen "poželjnosti" odgovarajućeg stanja i transformiše prostor stanja (pretrage) u tkzv. *fitness landscape* sa "vrhovima" and "dolinama" (slika 1.2). Strelice označavaju smer u kojem se sistem kreće i indiciraju preferirani tok populacije. Tačke A i C su lokalne optimalne vrednosti, dok je B globalni optimum. Crvena loptica indicira pomeranje od niskih vrednosti fitnes funkcije ka vrhu.
3) Tip algoritma određuje kompromis između eksploracije i eksploatacije u prostoru stanja. Eksploracija (istraživanje) je traženje nepoznatih regiona, a ekspoatacija je (ponovno) korišćenje prethodnog znanja radi pronalaženja boljih tačaka kao kandidata za rešenje. Balans izmedju ove dve strategije je obično kontrolisan putem vrednosti odgovarajućih parametara. Ispravan balans je od presudne vaznosti, jer prevalecija jedne kategorije predstavlje devijaciju algoritma: preterana eksploracija vodi ka čistoj slučajnoj pretrazi, dok preterana eksploatacija vodi ka metodi pretraživanja usponom (engl. hill-climbing).

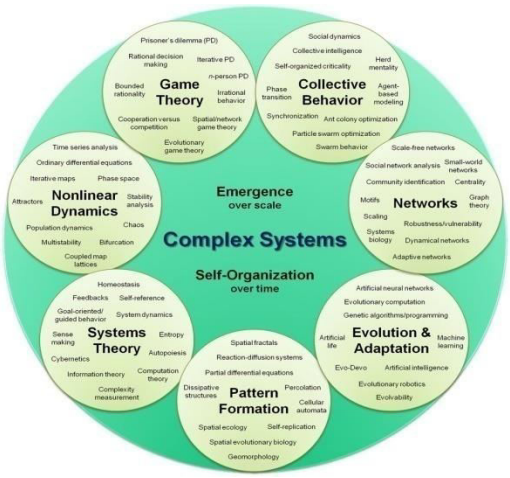
[4]Odnosi se na mimimalnu ili maksimalnu.



Slika 1.2. Prikaz fitness landscape: Strelice indiciraju preferirani tok populacije.

1.3.4 Samo-organizovanje (self-organization)

Okruženi smo raznim oblicima samo-organizovanja koja nudi naizgled jednostavne odgovore na kompleksna pitanja i rešavaju probleme preko spontane organizacije sistema bez bilo kakve akcije kontrolisane od strane centralnog koordinatora ili nekog eksternog sistema. Može se posmatrati kao prirodan evolutivan process kroz koji kompleksni sistemi pronalaze kvalitativno bolje paterne u cilju redukcije sopstvene kompleksnosti. Najpopularniji mehanizmi samo-organizovanja su upravo oni koji već postoje u prirodi (detektovani i opisani u biologiji, fizici, hemiji). S druge strane, ovakvi mehanizmi se mogu pronaći i u humanističkim i ekonomskim naukama (slika 1.3). Naučna studija samo-organizovanja sistema pokusava da otkrije generalna pravila pojavljivanja paterna samo-organizovanja i formi koje mogu posledično nastati.



Slika 1.3. Kompleksni sistemi (Sayama, 2010)

Ovi mehanizmi su izuzetno interesantni i primenljivi u različitim kompleksnim sistemima, naročito u IT-sistemima. Današnji razvoj IT industrije je karakterisan rastućom globalizacijom enterprajza. U cilju kompetitivnosti, kompanije zahtevaju softverske sisteme koji su su stanju da kominiciraju i "sarađuju" koristeći softverske komponente koji obezbeđuju podatke i usluge od mnogih različitih, distribuiranih i heterogenih sajtova. Takvi softverski sistemi se rapidno menjaju i te promene su prouzrokovane promenama tržisnih potreba kao i tehnološkom evolucijom. Tvorci distribuiranih softverskih sistema ulazu značajan napor da bi držali ove sisteme u stanju "up-to-date" sa svim današnjim standardima i, zbog toga, moraju da se bore sa enormnim rastom softverske kompleksnosti. Osnovni faktori koji određuju softversku kompleksnost su: veliki broj distribuiranih komponenti koje moraju da uzajamno dejstvuju ka ostvarivanju globalnog rešenja, veličina/dimenzije problema kao što su broj kompjutera, klijenata, zahteva, veličina upita, itd., heterogenost, autonomija organizacije i dinamičke promene u okruženju. Ove promene su fundamentalne, te raniji uobičajeni pristupi koji su uključivali kontrolu distribuiranih komponenti putem centralnog koordinatora su dostigli tehnički i koncepcijski limit. Teško je dizajnirati jedinstvenu kontroling komponentu koja je à priori "svesna" svih mogućih promena i nedostataka okruženja. Komponente i njihov kapacitet povećavaju se eksponencijalno, a sveukupna kompleksnost raste super-eksponencijalno (Heylighen, 2001). Zato su neophodni novi, napredni pristupiu bazirani na mehanizmima samo-organizovanja.

Slika 1.4. Primeri formiranja samo-organizujućih paterna: a) pruge peščanih dina; b) Belusov-Zhabotinsky hemijska reakcija; c) naizmenične zebrine pruge (Camazine et al., 2003).

Generalno govoreći, kompleksni sistemi su sistemi koje karakteriše veliki broj komponenata koje deluju u međusobnoj interakciji, čije interno stanje je definisano ogromnim brojem parametara. Ovi sistemi mogu biti u bilo kojem od ogromnog broja stanja u bilo kojem momentu. Kompleksni sistemi se ponašaju nepredvidivo, nepouzdano i često sa neobjašnjivim preokretima (Camazine et al., 2003). Njih takodje karakterišu nelinearnost, asimetrija i aperiodičnost.

Takve sisteme je teško "dekomponirati" i analizirati zbog velikog broja heterogenih elemenata i relacija izmedju tih elemenata, a uz to donošenje odluka u takvim sistemima je potpuno decentralizovano. Stoga se opravdano postavlja pitanje – kako da uopšte radimo sa njima?

Definitivno nije moguće predvideti šta se sve može desiti. Šta se može uraditi jeste biti spreman na adaptiranje neočekivanim promenama što je moguće bolje i anticipirati ih što je moguće više. Kompleksni sistemi nisu proizvoljno uređeni. Naprotiv, oni su uređeni na veoma organizovan način. Međutim, ta organizacija nije ugrađena u sistem od njegovog postanka, već je nastala kroz niz samo-organizujućih procesa koji predstavljaju tranziciju u nova stanja više organizacijske kompleksnosti i formiranje paterna (slika 1.4). Spontano ne znači "to se tek tako desilo" bez nekog naročitog razloga. Izazov i potreba je naći osnovne principe ovih sistema i objasniti ih. Za neke od poznatih metoda za rad sa kompleksnošću (apstrakcija, dekompozicija, klasifikacija, itd.) je dokazano da su podesne za upotrebu i obrađivanje specifičnih problema.

Samo-organizujući sistemi (Heylighen, 2001) se ne mogu prilagoditi svim mogućim događajima, ali je dokazano da poseduju dobru perspektivu za rad sa kompleksnošću. Cilj elemenata sistema je da se samo-organizuju, bez intervencije inženjera ili menadžera. Osnovne prednosti naspram tradicionalnih sistema su robusnost, fleksibilnost, mogućnost da funkcionišu autonomno zahtevajući nimalo ili minimum nadgledanja, kao i spontani razvoj kompleksnih adaptacija bez potrebe za detaljnim planiranjem.

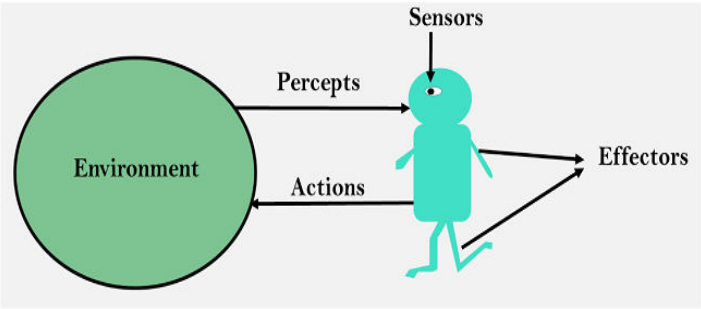
Kako se postiže samo-organizacija? U svakom slučaju putem komunikacije koja je u ovom aspektu najvažnija. Neki od tipova komunikacije u prirodnim sistemima su (Camazine et al., 2003):

- *point-to-point*: npr. razmenom hrane ili tečnosti, direktnim vizuelnim kontaktom, hemijskim kontaktom, itd.;
- broadcast-like: signal se širi u određenoj ograničenoj meri kroz okruženje i/ili je dostupan na prilično kratko vreme;
- *indirect*: dve individue indirektno deluju kada jedan od njih modifikuje okruženje, a drugi reaguje na novo okruženje kasnije (tkzv. stigmergija – engl. stigmergy)

1.3.5 Inteligentni agenti

U agent-baziranim sistemima (Shoham & Leyton-Brown, 2009), agent je entitet (npr. softverski modul) koji deluje ili ima moć i autoritet da deluje i prouzrokuje promene. Njegova autonomnost implicira da njegove akcije nisu kontrolisane od strane drugih niti su pod bilo kakvim spoljašnjim uticajem. Agent je nezvisan u odlučivanju i daljem vođenju procedure (enlg. self-directed, self-governing). On prikuplja podatke iz okruženja i samostalno odlučuje kako da poveže eksterne stimulanse sa sopstvenim ponašanjem da bi postigao odgovarajući cilj. Odgovarajući na različite stimulanse iz okruženja, agent bira i ispoljava različite paterne ponašanja. Oni mogu biti predefinisani ili dinamički stečeni kroz fazu učenja i adaptacije agenta. Važno je i koje "sposobnosti" agent poseduje. U okruženju gde postoji samo jedan agent (engl. single-agent setting), taj agent mora da "razume" ciljeve visokog nivoa i da poseduje znanje o njegovim sposobnostima (koje akcije može da preduzme, koje dejstvo te akcije proizvode, itd.). U okruženju gde postoji više agenata (engl. multi-agentsetting), agent treba da bude "svestan" ostalih agenata sa kojima sarađuje, komunicira i deli znanje (Shoham & Leyton-Brown, 2009).

Snaga autonomnih agenata je u njihovoj sposobnosti da rade sa nepredvidivim, dinamičkim, promenljivim, heterogenim okruženjem. Zbog toga, inteligentni algoritmi koriste autonomne agente, koji u specifičnim algoritmima preuzimaju specifične uloge (npr. ulogu mrava, pčele, itd). Oni imaju sposobnost samoodgovornosti (engl. self-responsible) za pokretanje, implementaciju odgovarajuceg ponašanja, a takođe mogu dinamički da se "prikuljuče" ili "isključe".



Slika 1.5. Komponente inteligentnog agenta[5]

[5] https://www.analyticssteps.com/blogs/5-types-intelligent-agents-artificial-intelligence

Kao sto je prikazano na slici 1.5, agenti su u interakciji sa okruženjem preko senzora i efektora/aktuatora. Razlikuju se percepcioni ulazi agenta dobijeni preko senzora u bilo kojem trenutku i percepciona sekvenca agenta koja je potpuna istorija svega što je agent ikada opazio. Agentov izbor akcije može zavisiti od čitave percepcione sekvence uočene do tog trenutka.

Ponašanje agenta opisano funkcijom agenta koja preslikava bilo koju percepcionu sekvencu agenta u njegovu akciju $f : S \rightarrow A$. Po pitanju implementacije, agenti je specijalizovani softverski program koji je visoko adaptibilni, autonomni, poseduju sposobnost opažanja okruženja i usvajanja drugačijih ciljeva, tj. prilagođavanja. Dakle, program agenta je realizacija funkcije agenta koja se izvršava se računarskoj mašini sa sopstvenom arhitekturom. Dakle mozemo da definišemo

agent = arhitektura + program

Mera performanse agenta služi kao kriterijum uspeha ponašanja agenta. Racionalni agent bira onu akciju koja maksimizuje očekivanu vrednost mere performanse za do tog trenutka datu sekvencu opažaja. Sta podrazumevamo pod pojmom *racionalni* agent? Racionalnost zavisi od mere performanse, agentovog prethodnog znanja o okruženju, akcija koje agent može izvršavati i agentove opažajne sekvence do posmatranog trenutka. Stoga, projektovanje racionalnog agenta uključuje specifikaciju okruženja, mere performanse, aktuatora i senzora.

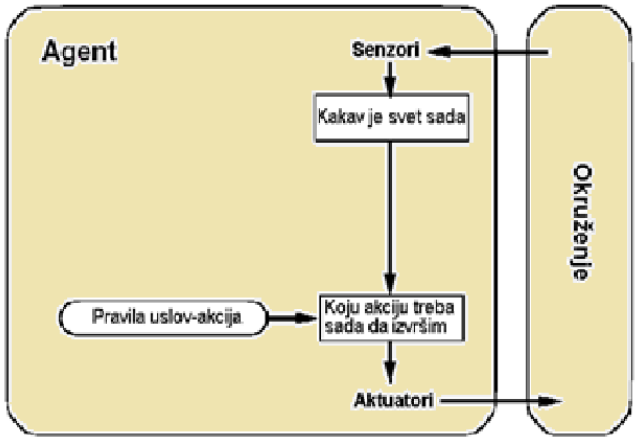
Autonoman agant je onaj agent čije ponašanje je determinirano sopstvenim iskustvom i koji ima sposobnost učenja i adaptacije.

Postoji pet osnovnih tipova agenata:

- Jednostavni refleksni agenti
- Refleksni agenti sa stanjem
- Agenti zasnovani na cilju
- Agenti zasnovani na korisnosti
- Agenti sa obučavanjem

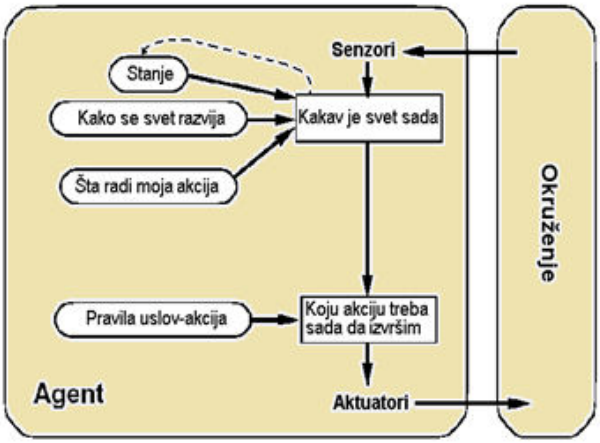
Svi ovi tipovi agenata imaju svoju varijantu agenata sa obučavanjem.

Jednostavni refleksni agenti su najjednostavniji agenti sa osnovnim refleksima (slika 1.6). Ovi agenti donose odluke i čine akcije ne uzimajući u obzir istoriju percepcije, te zasnivaju svoje sudove samo na svojim sadašnjim percepcijama. Oni dobro funkcionišu samo u okruženjima koja su potpuno vidljiva/observabilna. Pravilo “uslov-akcija” pod kojim radi ovakav agent omogućava mu da prevede trenutno stanje u akciju.



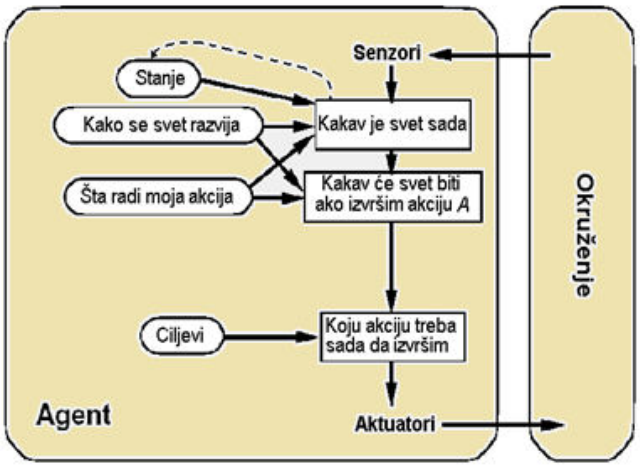
Slika 1.6. Šematski dijagram jednostavnog refleksnog agenta[5]

Refleksni agenti zasnovani na modelu (slika 1.7) mogu pratiti situaciju i delovati u delimično observabilnom okruženju. Delimična observabilnost se može uspešno tretirati tako što će agent pratiti promene onog dela okruženja koje mu je nedostupno (npr. uvođenje unutrašnjih stanja agenta, koja zavise od istorije opažanja). Da bi se informacije o unutrašnjim stanjima uspešno ažurirale, potrebno je pratiti: 1) kako se okruženje razvija i kako funkcioniše nezavisno od agenta (tkzv. model sveta), 2) kako delovanje agenta utiče na okruzenje. Ovaj agent prati tekuće stanje sveta koristeći unutrašnji model, a potom bira akciju na isti način kao i refleksni agent.
Dakle, razlikuju se dve vazne komponente kod ove vrste agenta: 1) model sveta, 2) interno stanje - ovaj prikaz sadašnjeg stanja zasnovan je na perceptivnoj istoriji. On funkcioniše tako što identifikuje pravilo čije stanje odgovara sadašnjoj okolnosti. Korišćenjem modela sveta, agent zasnovan na modelu može upravljati delimično vidljivim podešavanjima. Agent mora da prati svoje unutrašnje stanje, koje se menja svakom percepcijom i na koje utiču njegove prethodne percepcije.



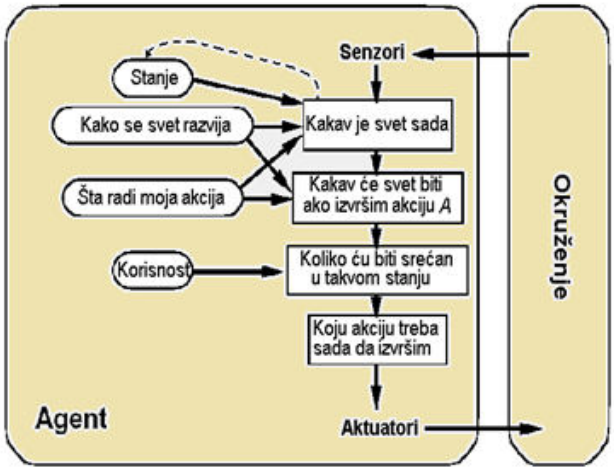
Slika 1.7. Agent zasnovan na modelu, sa refleksom[5]

Agenti zasnovani na cilju odlučuju šta će učiniti da bi postigli svoje ciljeve (slika 1.8). Posedovanje znanja o trenutnom stanju okruženja nije uvek dovoljno da se odluči šta da se radi. Dakle, agentu je potrebna neka vrsta ciljne informacije kojom se opisuju situacije koje su poželjne. Posedujući "ciljno" znanje, agenti zasnovani na ciljevima poboljšavaju sposobnosti agenta zasnovanog na modelu. Ovo daje agentu mogućnost da bira između različitih opcija, birajući onu koja vodi do željenog stanja. Ovi agenti su prilagodljiviji, jer se znanje koje je u osnovi njihovih odluka direktno izražava i može se modifikovati.
Oni odlučuju o pravcu delovanja kako bi ispunili svoj cilj. U nekim slučajevima je izbor delovanja zasnovanog na cilju neposredan (npr. neposredno iz pojedine akcije). Pre nego što utvrde da li je cilj postignut ili ne, ovi agenti će možda morati da uzmu u obzir dugačku listu potencijalnih aktivnosti. Takve analize mnogih scenarija nazivaju se traženjem i planiranjem, što omogućava agentu da preuzme inicijativu. Postoji mnogo suprotstavljenih ciljeva, ali samo određeni broj se može postići.
Agent zasnovan na cilju prati stanja okruženja, a bira akciju koja će voditi ka ostvarivanju njegovih ciljeva.



Slika 1.8. Agent zasnovan na cilju[5]

Agenti zasnovani na korisnosti su tipovi agenata koji pored ciljeva uzimaju u obzir i funkciju korisnosti (slika 1.9). Naime, sami ciljevi nisu dovoljni za generisanje visokokvalitetnog ponašanja u većini okruženja. Najpovoljnija situacija je kada su unutrašnja funkcija korisnosti i spoljna mera performansi u saglasnosti. U tom slučaju, agent će birati akcije maksimizirajući svoju korisnost, a takođe ce biti racionalan prema spoljnoj meri performansi. Dakle, racionalni agent zasnovan na korisnosti bira akciju koja maksimizuje očekivanu korist (imajuci u vidu delimično observabilno i stohastično okruženje).



Slika 1.9. Agent zasnovan na korisnosti[5]

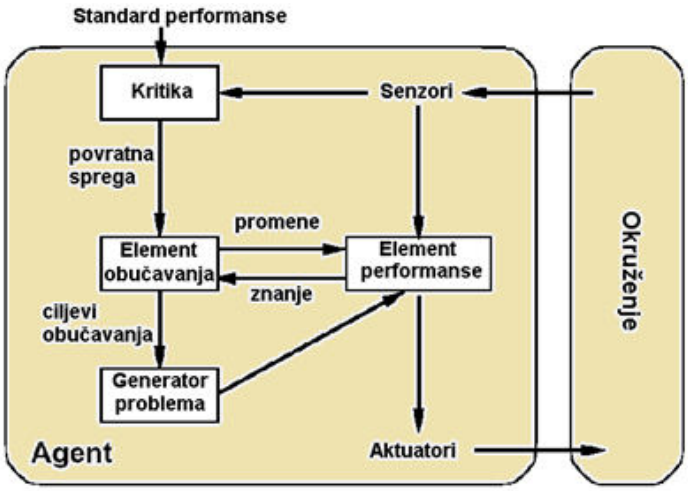
Agenti sa obučavanjem su agenti koji imaju mogućnosti učenja ili sposobnosti da uče iz svojih iskustava iz prošlosti (engl. learning agents). Počevši od osnovnih informacija, onda mogu delovati i prilagođavati se samostalno kroz učenje (slika 1.10).

Opšti model agenata sa obučavanjem:

1) Element učenja - odgovoran za generisanje poboljšanja;

2) Element performanse - odgovoran za delovanje na osnovu opažanja (celokupni agent u dosadašnjim razmatranjima).

Komponenta učenja koristi kritiku da proceni koliko dobro agent radi u odnosu na unapred određeno merilo učinka. Element učenja koristi povratnu informaciju kritike u vezi sa delovanjem agenta, što pomaze modifikovanju elementa performance u smislu poboljšanja buduceg rada agenta.
3) Element izvođenja – odredjuje izbor spoljne akcije.
4) Generator problema - odgovoran je za predlaganje delovanja koja će voditi ka novim i informativnim iskustvima (akcije istraživanja) Ovaj element je zadužen za davanje preporuka za aktivnosti koje će rezultirati novim i obrazovnim iskustvima.



Slika 1.10.Agenti sa obucavanjem[5]

Inteligentni agenti imaju brojne primene u raznolikim prakticnim i naučnim scenarijima kao sto su npr., pretraživanje informacija, navigacija, autonomna vozila, itd.

DRUGI DEO
METODE

§2 ALGORITMI BAZIRANI NA INTELIGENCIJI ROJA

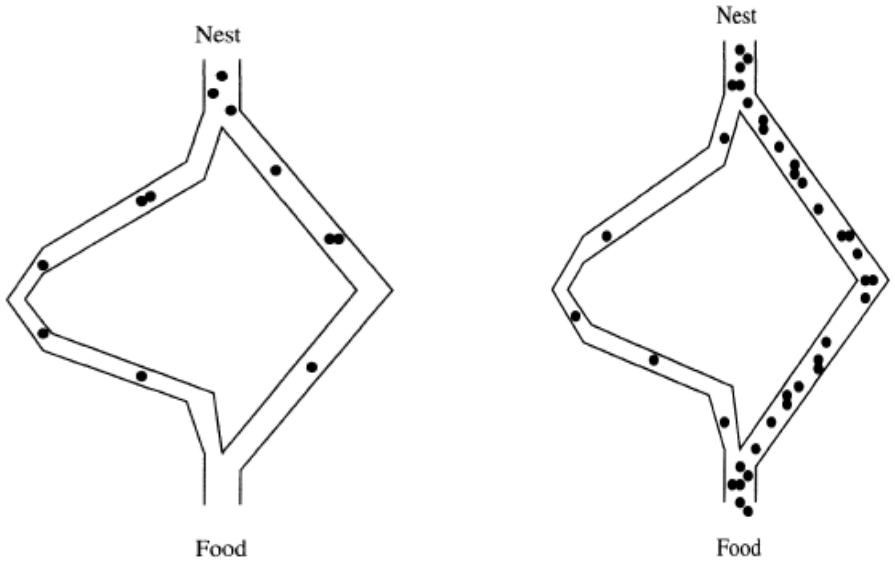
Roj (engl. swarm) se može definisati kao samo-organizujuci biološki sistem, tj. strukturirana kolekcija jedinki koje međjusobno koordiniraju i preduzimaju odgovarajuce akcije. Njihovo kolektivno ponašanje je u interakciji sa okruženjem, što prouzrokuje stvaranje koherentnih funkcionalnih paterna. Inteligenciju roja (swarm-intelligence) karakterišu distributivne i autonomne osobine. Naime, svaka jedinka u roju/populaciji aktivno radi, "donosi" lokalne odluke i uzajamno koordinira sa ostalim jedinkama u pravcu globalnog cilja bez uticaja bilo kakve centralne kontrole (supervizora) i razmenjuje informacije (npr. mravi preko feromona, pčele preko specifičnih pokreta iznad košnice,…). Glavna osobina inteligencije roja je bazirana na kolektivnom socijalnom ponašanju organizama (Kennedy & Eberhart, 2001). Agenti lokalno istrazuju prostor pretrage, preuzimaju potpuno lokalne i samostalne odluke, i rade na potpuno distribuiran način. Tako formiraju samo-organizovan model ponašanja koji vodi ka globalnoj konvergenciji. Prisutan stepen slučajnosti u ovakvom procesu povećava raznovrsnost rešenja na globalnoj skali (Blum & Roli, 2003; Xing & Gao, 2014). Mnogi ovakvi mehanizmi iz prirode su iskorišćeni, uspesno preslikani i modelirani u algoritme (engl. swarm-based algorithms).

2.1. Algoritmi bazirani na inteligenciji mrava (Ant Algorithms)

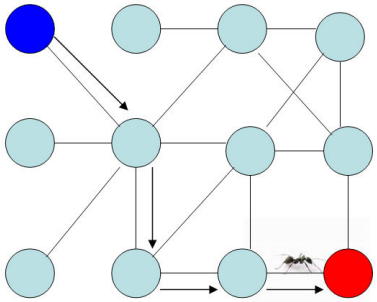
Kolonije mrava imaju raznolike manifestacije kolektivnog ponašanja: građenje i odrzavanje gnezda (mravinjaka) (Theraulaz et al., 2011), podela zadataka i adaptibilna dodela zadataka, formiranje struktura (npr. u radu sa preprekama), kooperativni transport (npr. hrane), pronalaženje najkraćeg puta od gnezda/mravinjaka do izvora hrane, grupisanje i sortiranje (npr., jaja) (Gaubert et al., 2007), regrutovanje za traganje za hranom (engl. foraging) (Mlot et al., 2011), itd. Svi nabrojani mehanizmi spadaju u oblike samo-organizovanja, od kojih su neki već preslikani, modelirani i iskorišćeni u optimizacione svrhe i ostale primenjive inženjerske probleme. Ipak, najčešće korišćeni mehanizam jeste *pronalaženje najkraćeg puta* od gnezda/mravinjaka do izvora hrane. Mravi međusobno komuniciraju na indirektan način putem hemijske supstance koju luče, feromona i to: što je jači intenzitet feromona (traga feromona) na određenom putu, to je veća verovatnoća da će jedinka pratiti taj put. Često se taj koncept posebno naziva/označava kao stigmergija (engl. stigmergy) i odnosi se na indirektnu komunkaciju između jedinki sistema tokom modifikovanja lokalnom okruženja. Dakle, feromon je stigmergijska informacija koja služi za komunikaciju između mrava. Dakle, kako se sam proces odvija? Mrav obavlja određeni zadatak - traži hranu, te ide od jedne do druge tačke u okruženju. Ovu fazu najčešće nazivamo probabilističko kretanje napred i konstrukcija rešenja (engl. probabilistic forward ants and solution construction). Nakon završenog zadatka, mrav se vraća istim putem – deterministički put nazad i ažuriranje feromona – i ostavlja feromon na tom putu koji služi kao povratna informacija ostalim jedinkama u populaciji u vezi s tim da li je - i ako jeste kakav kvalitet i količina hrane je pronađena, kao i kolika je udaljenost izvora hrane od gnezda/mravinjaka (Šešum–Čavić, 2020). Feromon postavljen na put postepeno tokom vremena isparava. U slučaju da više jedinki koristi taj isti put, na njihovom povratku

količina feromona će se uvećavati (engl. self-reinforcing process) što dovodi do formiranja puteva sa visokom koncentracijom feromona (Šešum–Čavić, 2020).

Ovaj prirodni proces je slikovito opisan eksperimentom dvostrukog mosta koji oslikava ponašanje jedne kolonije mrava na zadatku pronalaska najkraćeg puta od gnezda od izvora hrane (Slika 2.1). Na početku, jedinke biraju puteve potpuno slučajno (levi deo slike 2.1). Sa protokom vremena dolazi do akumulacije sve veće koncentracije feromona na (naj)kraćem putu, jer izbor jedinki upravo konvergira ka tom putu (desni deo slike 2.1). Dakle, kolonija mrava konvergira ka odgovarajućem putu na osnovu jednostavnog mehanizma od dve osnovne faze – faza putovanja napred i faza povratka (eng. forward phase i backward phase). Količina feromona na drugim, slabije posećenim putevima vremenom će da ispari.



Slika 2.1. Eksperiment dvostrukog mosta (Dorigo & Stützle, 2004).



Slika 2.2. Generalizovani model: šematski prikaz grafa (mreže) sa čvorovima i lukovima (linkovima) (Šešum–Čavić, 2020).

Na slici 2.2 je generalizovani model, koji predstavlja povezani graf $G = (N, A)$, gde je N skup od $n = |N|$ čvorova i A je skup neusmerenih lukova koji povezuju dva čvora. Dalje, reći ćemo da su čvorovi $i,j \in N$ susedni, ako postoji luk $(i,j) \in A$.

2.1.1. Opis algoritma

Prvi algoritam mrava, Ant System (AS), razvijen je od strane (Dorigo et al., 1996) i inicijalno primenjen na Traveling Salesman Problem. Nešto kasnije je nastao Ant Colony Optimization (ACO) (Dorigo & Stützle, 2004) sa ciljem da generalizuje celokupnu ideju i metod za rešavanje problema kombinatorne optimizacije korišćenjem biološkog, generičkog mehanizma mrava u prirodi.

Ant Colony Optimisation se sastoji od dve glavne faze: konstruisanje rešenja i ažuriranje feromona (engl. construct solution i pheromone update). Mravi su jedinke u populaciji koje se kreću od jednog do drugog čvora na grafu, birajući čvor na osnovu postojeće jačine koncentracije feromona na putu između dva susedna čvora. Kod rešavanja odgovarajućeg problema, put koji mrav napravi predstavlja potencijalno rešenje problema, tj. to je "kandidat" za rešenje. Nakon što je mrav "načinio put", vraća se nazad ka gnezdu i postavlja feromon na povratku srazmeran kvalitetu rešenja. Ta količina feromona utice na ponašanje ostalih mrava iz kolonije (Šešum–Čavić, 2020).

Na samom početku, potrebno je inicijalizirati tragove feromona. U tom pogledu, postavljaju se na vrednost nesto višu od očekivane vrednosti feromona ostavljenu od strane mrava u jednoj iteraciji:

$$\forall(i,j),\ \tau_{ij} = \tau_0 = m/C^{nn} \tag{2.1}$$

gde su τ_{ij} vrednosti feromona na linkovima (lukovima između čvorova), τ_0 je inicijalna vrednost feromona, $m$je broj mrava, C^{nn}je dužina ture generisane putem neke heuristike (npr. nearest-neighbor metod), n je broj čvorova.

U slučaju da su inicijalne vrednosti feromona suviše niske, pretraga je bazirana na osnovu prve generisane ture. U suprotnom, ako su inicijalne vrednosti feromona suviše visoke, dosta vremena se gubi na iteracije tokom kojih dolazi do evaporacije feromona i redukcije vrednosti feromona.

U prvoj fazi, konstruisanje rešenja, mravi konstruišu rešenje idući od početnog čvora do destinacije, korak po korak (čvor po čvor) promenjujući stohastičku polisu odluke:

$$p_{ij}^{k} = \frac{\left[\tau_{ij}\right]^{\alpha}\left[\eta_{ij}\right]^{\beta}}{\sum_{l\in N_i^k}\left[\tau_{il}\right]^{\alpha}\left[\eta_{il}\right]^{\beta}} \tag{2.2}$$

gde je τ_{ij} trag feromona na (i,j)-luku, η_{ij} heuristička vrednost raspoloživa a priori, α i β su parametri koji odlučuju o uticaju traga feromona i heurističke informacije, N_i^kje okruženje/susedstvo mrava k (koje ovaj mrav jos nije posetio) dok je još na čvoru i.

U drugoj fazi, najpre se vrednost feromona na svim lukovima umanjuje za konstantan faktor (evaporacija traga feromona):

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{2.3}$$

gde je $0 < \rho \leq 1$ je rata evaporacije feromona.

Nakon evaporacije, dodatna količina feromona se deponuje na lukove koje su mravi koristili tokom konstruisanja rešenja. To je ažuriranje feromona i služi da bi se uvećale vrednosti feromona pridružene kvalitetnim rešenjima (engl. pheromone trail reinforcement) i umanjile one koje su pridružene lošim rešenjima:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m}\Delta\tau_{ij}^{k} \tag{2.4}$$

gde je $\Delta\tau_{ij}^{k}$ količina feromona koji mrav *k* deponuje na luk koji je posetio.

Pojednostavljen, apstraktan prikaz ovog algoritma je predstavljen u obliku pseudo-koda:

```
procedure ACO_MetaHeuristic
    while(not_termination)
        constructSolutions()
        pheromoneUpdate()
        daemonActions()
    end while
end procedure
```

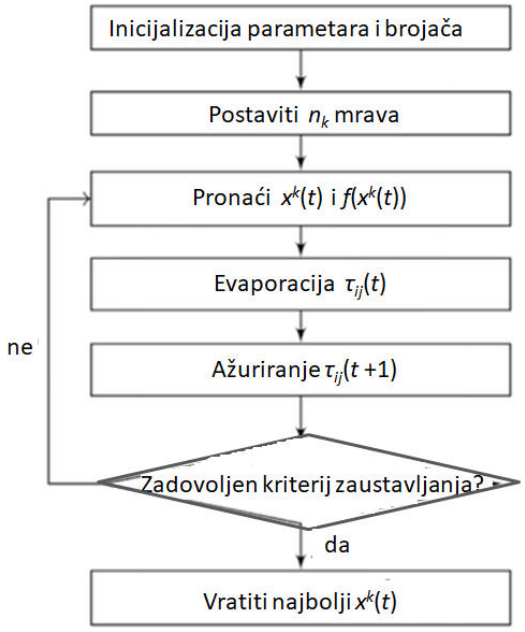
Alg.2.1. ACO pseudo-kod (Dorigo & Stützle, 2004).

Preporučene vrednosti parametara za korišćenje su sledeće:
α = 1, β uzima vrednosti između 2 i 5, ρ = 0.5, $\tau_0 = m/C^{nn}$.

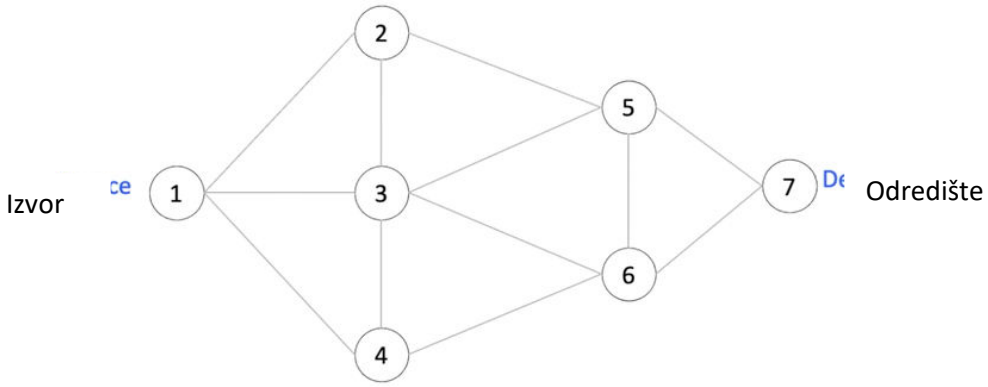
Primer

Pretpostavimo da želimo predstaviti koncept rada najjednostavnijeg oblika ACO na problemu trgovačkog putnika (Travelling Salesman Problem, TSP). U ovom poznatom NP-teskom problemu kombinatorne optimizacije, za dati spisak gradova i daljina između svakog para gradova, potrebno je naći najkraći mogući put da se poseti svaki grad tačno jednom i vrati se u početni grad.

Mi ćemo ilustrovati korake rada ACO za početak putem sledećeg flowchart-a:



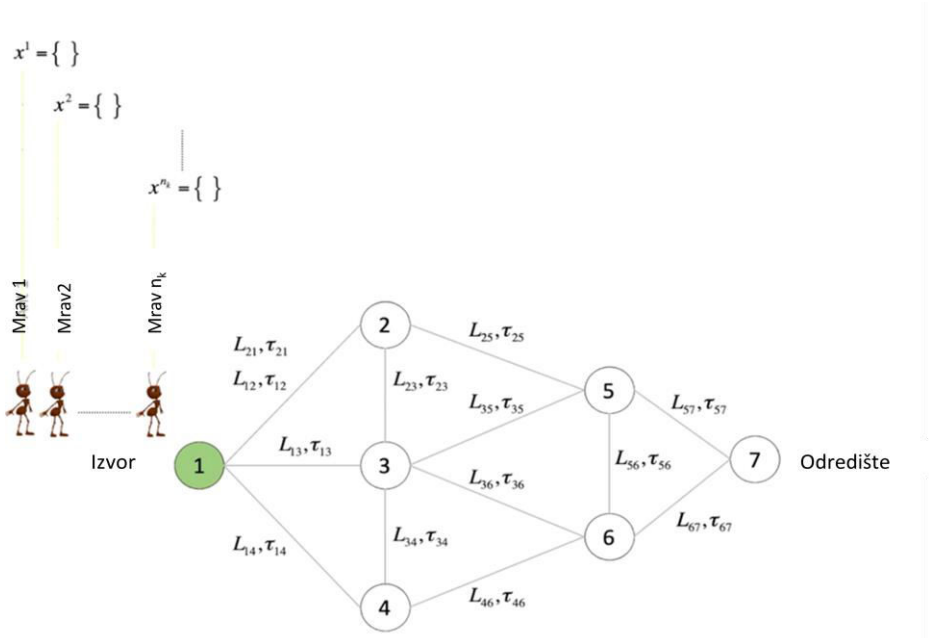
Uslovi terminacije (engl. stopping criteria) mogu biti: dostignut maksimalan broj iteracija, pronađeno prihvatljivo rešenje ili situacija da je stepen konvergencije zadovoljavajući – svi (ili većina) jedinki slede isti put.
Pretpostavimo da se TSP može preslikati na sledeći jednostavan graf sa 7 čvorova:



U početku određujemo inicijalnu veličinu populacije, n_k. Dakle, n_k jedinki (mrava) započinju da prave svoj put. U ovom jednostavnom primeru, svi mravi počinju od čvora 1 (source), ali bez umanjenja opštosti u komplikovanijim situacijama, mravi mogu biti raspoređeni na slučajan nacin, tj. tako da svaki počinje od slučajno izabranog čvora.

Inicijalno su zadate su odgovarajucće vrednosti funkcije troška (označene na lukovima grafa sa L_{ij}, a mogu se interpretirati i kao "udaljenosti" koje su u gornjim formulama označene sa d_{ij}) i koncentracije feromona (označene na lukovima grafa sa τ_{ij}).

Svaki mrav počinje da gradi svoj put. Počinjući od mrava 1, određuje se na osnovu formule 2.2 koji čvor ce biti sledeći na putu mrava 1. Na osnovu toga dobijamo tabelu tkvz. stohastičke polise odluke (engl. transition probability) za odgovarajuci čvor.



$$\text{Node 1: } \mathcal{N}_1^1(t) = \{2,3,4\}$$

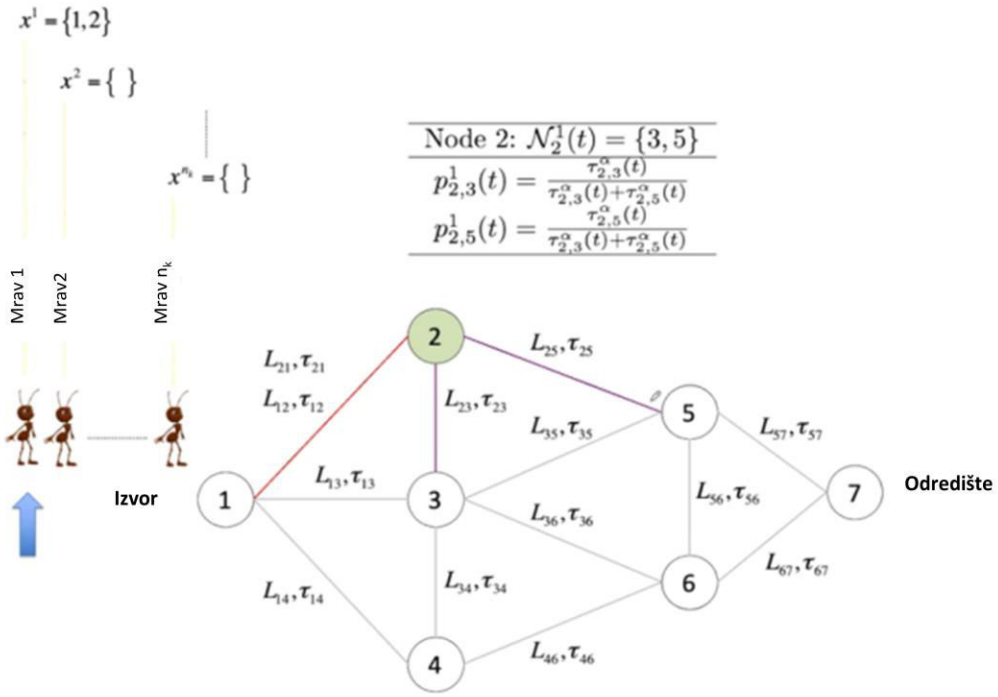
$$p_{1,2}^1(t) = \frac{\tau_{1,2}^\alpha(t)}{\tau_{1,2}^\alpha(t)+\tau_{1,3}^\alpha(t)+\tau_{1,4}^\alpha(t)}$$

$$p_{1,3}^1(t) = \frac{\tau_{1,3}^\alpha(t)}{\tau_{1,2}^\alpha(t)+\tau_{1,3}^\alpha(t)+\tau_{1,4}^\alpha(t)}$$

$$p_{1,4}^1(t) = \frac{\tau_{1,4}^\alpha(t)}{\tau_{1,2}^\alpha(t)+\tau_{1,3}^\alpha(t)+\tau_{1,4}^\alpha(t)}$$

Stohastička polisa odluke (čvor, engl. node)

Pretpostavimo da je izabran čvor 2 za dalji obilazak. Dakle, i za taj čvor se napravi tabela stohastičke polise odluke imajući u vidu da mrav ne moze da se vraća na cvor koji je već jednom obisao (čvor 1).

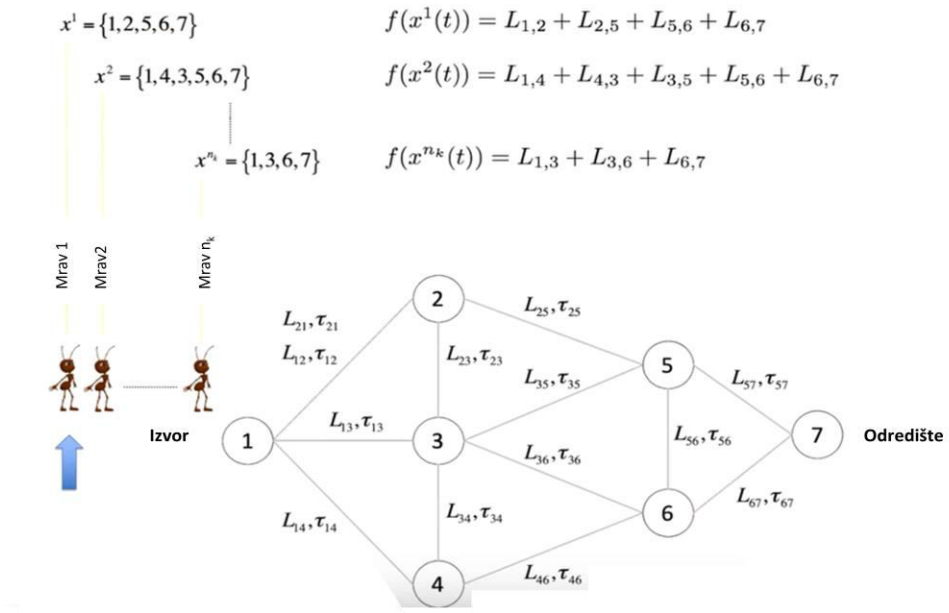


$$\text{Node 2: } \mathcal{N}_2^1(t) = \{3,5\}$$

$$p_{2,3}^1(t) = \frac{\tau_{2,3}^\alpha(t)}{\tau_{2,3}^\alpha(t)+\tau_{2,5}^\alpha(t)}$$

$$p_{2,5}^1(t) = \frac{\tau_{2,5}^\alpha(t)}{\tau_{2,3}^\alpha(t)+\tau_{2,5}^\alpha(t)}$$

Proces se dalje nastavlja po opisanom postupku. Pretpostavimo da je mrav 1 izgradio “rešenje” tj., pronasao svoj put do destinacije preko čvorova {1,2,5,6,7}. Takodje pretpostavimo da su svi mravi izgradili svoja rešenja / puteve, te da su ukupni troškovi sračunati u vidu $f(x^n(t))$:



U sledećem koraku, vrednosti feromona se umanjuju na svim lukovima prema formuli 2.3:

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij}(t)$$

Na primer, postavimo izabranu vrednost ρ = 0.2.

Nakon toga, ažurira se vrednost feromona:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^k(t)$$

Uzmimo na primer, luk izmedju čvorova 2 i 5. Ažuriranje će se vrsiti prema sledećim koracima:

$$\Delta\tau_{2,5}^1(t) = \frac{Q}{f(x^1(t))}$$
$$\Delta\tau_{2,5}^2(t) = 0$$
$$\vdots$$
$$\Delta\tau_{2,5}^{n_k}(t) = 0$$

$$\tau_{2,5}(t+1) = \tau_{2,5}(t) + \left\{ \underbrace{\frac{Q}{f(x^1(t))}}_{\text{Ant 1}} + \underbrace{0}_{\text{Ant 2}} + \cdots + \underbrace{0}_{\text{Ant } n_k} \right\}$$

Kao drugi primer, uzmimo luk izmedju čvorova 5 i 6. Ažuriranje će se vrsiti prema sledećim koracima:

$$\Delta\tau_{5,6}^{1}(t) = \frac{Q}{f(x^1(t))}$$
$$\Delta\tau_{5,6}^{2}(t) = \frac{Q}{f(x^2(t))}$$
$$\vdots$$

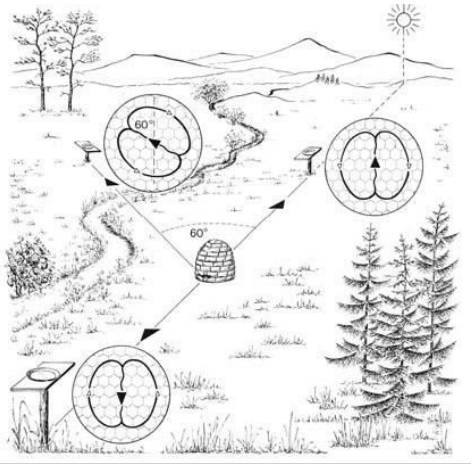
$$\tau_{5,6}(t+1) = \tau_{5,6}(t) + \left\{ \underbrace{\frac{Q}{f(x^1(t))}}_{\text{Ant 1}} + \underbrace{\frac{Q}{f(x^2(t))}}_{\text{Ant 2}} + \cdots + \underbrace{0}_{\text{Ant } n_k} \right\}$$

Nadalje se proces ponavlja do kriterijuma zaustavljanja.

2.2. Algoritmi bazirani na inteligenciji pčela (Bee Algorithms)

Ponašanje pčela u prirodi karakteriše visoki stepen autonomije jedinki, distribuirano funkcionisanje i samo – organizovanje (Camazine et al., 2003). Ono je inspirisalo tvorce algoritama na primenu i preslikavanje njihovih različtih bioloskih mehanizama. Ipak, i u ovom slučaju najpopularniji korišćeni mehanizam je pronalaženje najkraćeg puta od košnice do izvora hrane. Raznoliki manje-više slični algoritamski pristupi koriste upravo ovaj mehanizam kao inspiraciju: Artificial Bee Colony (ABC) (Karaboga & Basturk, 2007), Virtual Bee algorithm (Yang, 2005), Bee Colony Optimization (BCO) (Teodorovic et al., 2006), BeeHive algorithm (Wedde et al., 2004), Bee Swarm Optimization (Drias et al., 2005), Bees Algorithm (Pham et al., 2006). U daljem tekstu predstavljen je jedan od najpopularnijih algoritama baziranih na inteligenciji pčela, BCO.

U jednoj koloniji pčela jedinke imaju različite uloge (Camazine et al., 2003): izviđači (engl. foragers, scouts), pratioci (engl. followers) i radilice/primaoci posla (engl. receivers). Organizacija kolonije počiva na dva osnovna vida ponašanja: navigacija i regrutovanje (slika 2.3). Tokom navigacije, pcela izviđač traga u nepoznatoj oblasti za hranom, tj. cvetom koji ima najprivlačniji nektar imajuci u vidu i kvalitet i kvantitet. Za orijentaciju koristi se tkzv. integracija puta (Camazine et al., 2003). Na osnovu svog prethodnog traga pčela izviđač "izračunava" svoju trenutnu lokaciju. Stoga, integracija puta predstavlja znanje koje pčela poseduje u vezi udaljenosti nektara i pravca u kojem se cvet sa nektarom nalazi. Kada pčela izviđač pronađe i sakupi zadovoljavajući nektar, vraća se u košnicu da bi ga isporučila, a potom izvodi regrutovanje ostalih članova kolonije. Sta to zapravo znači? Tokom ovog procesa pčele komuniciraju informacije o putu koji su prošle u fazi izviđanja (npr. dužina puta) kao i kvalitetu hrane koju su pronašle na tom putu (u vezi posećenih cvetova) (Camazine et al., 2003), tj. "reklamiraju" one cvetove koje su posetile. Pčele komuniciraju putem specijalanog sistema pokreta/plesa (engl. waggle dance) i na taj način obaveštavaju ostale pčele iz košnice o pravcu, udaljenosti i kvalitetu pronađene hrane. Što je bolji kvalitet pronađenog nektara i/ili kraća udaljenost do hrane, pčela duže izvodi pokrete plesa (Camazine et al., 2003). Ostale pčele (tkzv. pčele pratioci) osmatraju signale pčela izvođača, slučajno biraju jednog izviđača čiji put će da preuzmu i prate bez izvođenja sopstvene pretrage oblasti. Uloge pčela mogu biti zamenjene, npr. izviđač može odlučiti da postane pratilac. Pčele radilice primaju nektar, uvek su u košnici i prerađuju nektar (Šešum–Čavić, 2020).



Slika 2.3. Funkcionisanje jedne kolonije pčela u prirodi (Barth, 1982).

2.2.1. Opis algoritma

Algoritam sledi dve osnovne vrste ponašanja u kolonije pčela: navigaciju i regrutovanje (Šešum–Čavić, 2020). U prvoj fazi - *navigaciji*, svaka pčela izviđač konstruiše rešenje (construct solution) obilazeći cvetove tj., čvorove grafa i odlučujući koji čvor će posetiti kao sledeći. To je realizovano probabilistički preko pravila promene stanja (Wong et al., 2008):

$$P_{ij}(t) = \frac{[\rho_{ij}(t)]^{\alpha} \cdot [1/d_{ij}]^{\beta}}{\sum_{j \in A_i(t)} [\rho_{ij}(t)]^{\alpha} \cdot [1/d_{ij}]^{\beta}} \tag{2.5}$$

gde je $\rho_{ij}(t)$ lučna fitnes funkcija od čvora *i* do čvora *j* u momentu *t* i d_{ij} je heuristička udaljenost između *i* i *j*, α je binarna promenljiva koja "uključuje"/aktivira i "isključuje"/deaktivira lučni fitnes uticaj, dok je β parametar koji kontroliše značaj heurističke udaljenosti. Tokom izračunavanja vrednosti lučne fitnes funkcije razlikuju se sledeće situacije:

(1) **Izviđač:** Pčela se ponaša prema pravilu prelaza stanja i $\rho_{ij} = 1/k$, gde je *k* broj susednih čvorova čvora *i*. Izviđač može odlučiti da postane pratilac u sledećem krugu navigacije.

(2) **Pratilac:** Pre napuštanja košnice, pčela opaža pokrete plesa ostalih pčela i slučajno bira čiju informaciju ce da prati/preuzme. Ta informacija sadrži skup pokreta za navođenje koji opisuju ceo put/turu od košnice do cveta/destinacije prethodno istraženu od druge pčele izviđača – tkzv. preferirani put (Wong et al., 2008). Kada je pčela na čvoru *i* u momentu *t*, dva skupa sledeće-posećujućih čvorova se mogu izvesti: skup dozvoljenih sledećih čvorova, $A_i(t)$ i skup favorizovanih čvorova, $F_i(t)$. $A_i(t)$ sadrži skup susednih čvorova čvora *i*, dok $F_i(t)$ sadrži jedan čvor favorizovan da bude posećen posle čvora *i* na osnovu preferiranog puta. Lučni fitnes je stoga definisan:

$$\rho_{ij}(t) = \begin{cases} \lambda & ako\ j \in F_i(t) \\ \dfrac{1-\lambda \cdot |A_i(t) \cap F_i(t)|}{|A_i(t)| - |A_i(t) \cap F_i(t)|} & ako\ j \notin F_i(t) \end{cases} \qquad \forall j \in A_i(t),\ 0 \le \lambda \le 1 \tag{2.6}$$

gdeje $|S|$ oznaka za kardinalnost skupa *S*. Dakle, $|A_i(t) \cap F_i(t)|$ može biti 0 ili 1, pošto $A_i(t)$ i $F_i(t)$ mogu imati ili nijedan element ili samo jedan element u preseku.

Druga faza – *regrutovanje* podrazumeva da pčele komuniciraju svoje stečeno iskustvo u vezi udaljenosti i kvaliteta pronađenog rešenja. Na osnovu toga, fitnes funkcija je:

$$f_i = \frac{1}{H_i}\delta \tag{2.7}$$

za pčelu *i*, gde je H_i broj skokova od čvora do čvora na turi obilaska, δ je funkcija prilagođenosti[6], pa je fitnes funkcija cele kolonije prosečna vrednost vrednosti fitnes funkcija svih pčela:

$$f_{colony} = \frac{1}{n}\sum_{i=1}^{n} f_i \tag{2.8}$$

gde je *n* broj pčela. Posle toga, pčela određuje "koliko dobro" rešenje je pronađeno upoređujuci svoje rezultate sa fitnes funkcijom kolonije i na osnovu toga odlučuje o svojoj sledećoj ulozi (izviđač ili pratilac) (Nakrani & Tovey, 2004).

Profitability Scores	P_{follow}
$f_i < 0.5 \cdot f_{colony}$	0.60
$0.5 \cdot f_{colony} \leq f_i < 0.65 \cdot f_{colony}$	0.20
$0.65 \cdot f_{colony} \leq f_i < 0.85 \cdot f_{colony}$	0.02
$0.85 \cdot f_{colony} \leq f_i$	0.00

Tabela 2.1.. Odluka o sledećoj ulozi na osnovu poređenja sopstvenih rezultata sa rezultatima kolonije.

BCO algoritam je predstavljen u apstraktoj formi psedo-koda (Alg.2.2.). Ceo proces se može opisati preko procedura: *observeWaggleDance*, *constructSolution*, i *performWaggleDance*.

```
procedure BCO_MetaHeuristic
 while(not_termination)
     observeWaggleDance()
     constructSolution()
     performWaggleDance()
 end while
end procedure
```

Alg.22. BCO pseudo-kod (Teodorovic et al., 2006).

BCO parameteri	preporučene vrednosti
α	0,1
β	od 8 do 12 sa korakom 2
λ	0.99

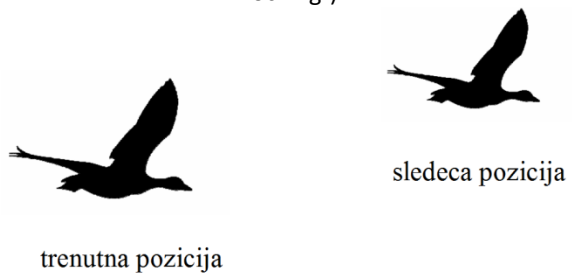
Zadatak. Na osnovu primera u prethodnoj podsekciji koja se odnosi na ACO, skicirati rešenje TSP pomoću BCO.

[6]Fitnes funkcija može dalje biti modifikovana uzimajući u obzir i razne druge parametre, npr. kvalitet linkova.

2.3. Optimizacija roja česticama (Particle Swarm Optimization)

U biološkim zajednicama gde ima mnogo jedinki, obavljanje odgovarajucih zadataka vaznih za funkcionisanje celine se odvija koordinisano. Na primer, kod kretanja jata ptica ili ponašanja skupine riba, iako se populacija sastoji od mnoštva jedinki, ponaša se kao jedna celina u obavljanju raznih zadataka, npr., traženja hrane, zaštite od predatora, putovanja od jedne lokacije do druge, itd. Način funkcionisanja ovakvih bioloških zajednica je služio kao inspiracija za nastanak posebne grupe algoritama, tkzv. optimizacija roja česticama ili partiklima (Particle Swarm Optimization, PSO) (Slika 2.4).

Slika 2.4. PSO je inspirisana socijalnim ponašanjem grupe riba (engl. fisch schooling[7]) ili jata ptica (engl. bird flocking[8]).



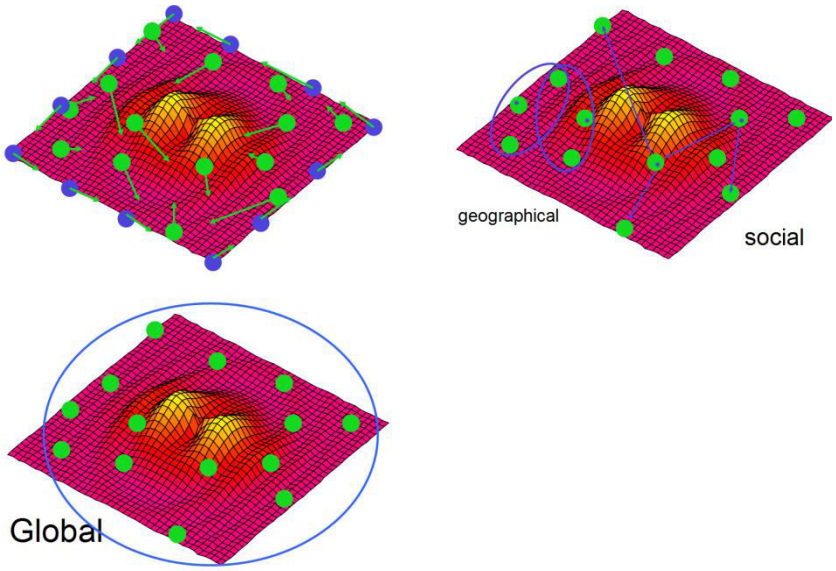
Slika 2.5. Kretanje prema poziciji vodeće susedne jedinke.

Pravila ponašanja koja univerzalno vrede su sledeća: izbegavanje gomilanja između susednih jedinki, ravnanje i upravljanje sopstvenog kretanja prema poziciji "vodeće" susedne jedinke, kao i prema prosečnoj poziciji svih susednih jedinki (Slika 2.5).

Svaka čestica je potencijalno rešenje zadatog problema i ima odgovarajuću vrednost fitnes funkcije. Takođe, svaka čestica se kreće i ima brzinu kretanja, zadržava svoju individualnu najbolju performansu (engl. personal best), ali zna koja je najbolja performansa cele grupe (engl. global best), tako da podešava svoju brzinu kretanja uzimajući u obzir vrednosti obe performanse. Čestice između sebe sarađuju i razmenjuju informacije u vezi toga šta su otkrile na mestima koje su posetile (Šešum–Čavić, 2020). Četiri vektora definišu više-dimenzionalni prostor pretrage za svaku česticu: trenutna pozicija čestice, najbolja pozicija pronađena do sada, najbolja pozicija pronađena u okruženju od čestice, i brzina čestice (Šešum–Čavić, 2020).

[7] https://blog.wakatobi.com

[8] https://en.m.wikipedia.org



Slika 2.6. a) Inicijalizacija pozicija i brzina u prostoru pretrage: čestice mogu biti pozicionirane slučajno ili na unapred određen način, brzina za svaku čestica je obično definisana slučajno (brzina označava pokret, jer je vreme diskretno); b) Okruženje/susedstvo: za svaku česticu, definiše se njeno okruženje/susedstvo – koji može biti npr., geografski određeno (izračunate su udaljenosti i uzete su najbliže čestice) ili "socijalno" određeno (lista susednih čestica koja može biti određena po bilo kojem kriterijumu, nezavisno od toga gde se te čestice nalaze); ako proces konvergira, svako "socijalno" određeno okruženje/susedstvo tendira da bude takođe i geografski određeno; c) Globalno okruženje (Šešum–Čavić, 2020).

Dakle, šta čestica radi (slika 2.6)?
U svakom momentu, čestica se konstantno pomera na novu poziciju u prostoru pretrage, što je urađeno stalnim podešavanjem brzine. Ukupna brzina je definisana kao suma trenutne brzine, odmerenog proizvoljnog dela u pravcu "personal best" same čestice i odmerenog proizvoljnog dela u pravcu najboljeg u okruženju. Nova pozicija je definisana kao tekuća pozicija plus nova brzina (Šešum–Čavić, 2020).
Ceo proces pretrage se može opisati jednostavnim jednačinama (Kennedy & Eberhart, 2001):

$v[\,] = v[\,] + c_1 * \text{rand}(\,) * (pbest[\,] - present[\,]) + c_2 * \text{rand}(\,) * (gbest[\,] - present[\,])$

$present[\,] = present[\,] + v[\,]$

gde je $v[\,]$ brzina čestice, $present[\,]$ je tekuća čestica (tekuće rešenje), $pbest[\,]$ je najbolja pozicija čestice koju je dosegla, $gbest[\,]$ ja najbolja pozicija dostignuta u okruženju te čestice, rand() je slučajan broj iz (0,1), c_1, c_2 su faktori "učenja" (learinig factors).

2.3.1. Opis algoritma

Particle Swarm Optimization (PSO) (Kennedy & Eberhart, 2001) je inspirisan socijalnim ponašanjem ptica u potrazi za hranom. Pseudo-kod opisuje implementaciju gore navedenih jednačina (Alg.2.3).

```
for each particle
  initialize particle
```

```
end
repeat
  for each particle
    calculate fitness value
    if fitness value is better than pBest
    set current value as the new pBest
   end
    choose the particle with the best fitness value of all particles as gBest
     for each particle
       update particle velocity
       update particle position
      end
until maximum iterations or minimum error criteria is attained
```

Alg.2.3. PSO pseudo-kod (Hu et al., 2004).

Kod podešavanja parametara, najčešće se uzimaju u obzir sledeće preporuke:

- broj čestica
  - 10 - 50 je obično dovoljno
- veličina okruženja/susedstva: 3 – 5
  - okruženje/susedstvo veličine 3 za probleme velikog ranga je pokazao dobre rezultate
- c_1 (važnost sopstveno najboljeg "personal best"), c_2 (važnost najboljeg u okruženju"neighborhood best"): obično se uzima $c_1 + C_2 = 4$ (empirijski rezultat)
- v_{max} → ako je preniska vrednost, suviše spor; ako je previsoka vrednost, suviše nestabilan
- ω (inertia term) → [0.9,1.2] (opcionalno)

Primer

Minimizirati funkciju $f(x_1, x_2, x_3) = 10(x_1 - 1)^2 + 20(x_2 - 2)^2 + 30(x_3 - 3)^2$

Korak 1: Inicijalizacija

Uzimamo inicijalne vrednosti za broj promenljivih m, veličinu populacije n, "inertia" vrednosti w_{max} i w_{min}, faktor ubrzanja c_1 i c_2, i maksimalan broj iteracija $maxt$:
m =3, n = 5, w_{max} = 0.9 i w_{min} =0.4, c_1 = 2 i c_2 = 2, $maxt$ = 50.
Nakon toga na slučajan nacin izaberemo jedinke populacije. Drugim rečima, treba nam 15 slučajnih inicijalnih vrednosti, za svaku od 5 jedinki (čestica, partikli) sa 3 vrednosti (za x_1, x_2, x_3). Na primer:

	x1	x2	x3
1. partikl	8	9	1
2. partikl	9	6	1
3. partikl	3	5	10
4. partikl	10	2	10
5. partikl	10	5	8

Inicijalna populacija

Dalje, inicijalizira se na slučajan nacin brzina (engl. velocity) v za svaki partikl:

$$v = 0.1 \cdot x_0(i, j)$$

To znači, za prvi partikl:
$v(x_1) = 0.1 \cdot 8 = 0.8$
$v(x_2) = 0.1 \cdot 9 = 0.9$
$v(x_3) = 0.1 \cdot 1 = 0.1$

	x1	x2	x3
1. partikl	0.8	0.9	0.1
2. partikl	0.9	0.6	0.1
3. partikl	0.3	0.5	1
4. partikl	1	0.2	1
5. partikl	1	0.5	0.8

Inicijalna brzina

Zatim, inicijalizira se na slučajan način pozicija x_i za svaki partikl:

trenutna pozicija = prethodna pozicija + brzina

Na primer, za prvi partikl:
$x_1 = 8 + 0.8 = 8.8$
$x_2 = 9 + 0.9 = 9.9$
$x_3 = 1 + 0.1 = 1.1$

	x1	x2	x3
1. partikl	8.8	9.9	1.1
2. partikl	9.9	6.6	1.1
3. partikl	3.3	5.5	11
4. partikl	11	2.2	11
5. partikl	11	5.5	8.8

Inicijalna pozicija

Korak 2: Evaluirati fitness funkciju, tj. izračunati vrednost fitness funkcije za svaki partikl

$$f(x_1^0) = 10 \cdot (8.8 - 1)^2 + 20 \cdot (9.9 - 2)^2 + 30 \cdot (1.1 - 3)^2 = 1.9649$$
$$f(x_2^0) = 10 \cdot (9.9 - 1)^2 + 20 \cdot (6.6 - 2)^2 + 30 \cdot (1.1 - 3)^2 = 1.3236$$
$$f(x_3^0) = 10 \cdot (3.3 - 1)^2 + 20 \cdot (5.5 - 2)^2 + 30 \cdot (11 - 3)^2 = 2.2179$$
$$f(x_4^0) = 10 \cdot (11 - 1)^2 + 20 \cdot (2.2 - 2)^2 + 30 \cdot (11 - 3)^2 = 2.9208$$
$$f(x_5^0) = 10 \cdot (11 - 1)^2 + 20 \cdot (5.5 - 2)^2 + 30 \cdot (8.8 - 3)^2 = 2.2542$$

Zatim biramo partikl najboljom vrednošću fitnes funkcije, označeno kao *gBest*:

Fitnes funkcija (t=0)
1.9649
1.3236
2.2179
2.9208
2.2542

Korak 3: Za svaki partikl izračuna se brzina i pozicija na sledeći način

pozicija $x_i^{t+1} = x_i^t + v_i^t$

brzina $v_i^{t+1} = wv_i^t + c_1 r_1 \left(xBest_i^t - x_i^t\right) + c_2 r_2 \left(gBest_i^t - x_i^t\right)$

Na primer, ako se uzmu sledeće vrednosti parametara: w = 0.9, c_1, c_2 = 2, dobija se sledeće izračunavanje.

Ažuriramo brzinu za prvi partikl u iteraciji t = 0:

$$v_1^{0+1} = wv_1^0 + c_1 r_1 (xBest_1^0 - x_1^0) + c_2 r_2 (gBest_1^0 - x_1^0)$$

$v_1^1 = 0.9 \cdot 0.8 + 2 \cdot rand(\ )(8.8 - 8.8) + 2 \cdot rand(\ )(9.9 - 8.8) = 1.0667$ (za x_1)

Na sličan način izračunavaju se i vrednosti: $v_1^1 = -4.4719$ (za x_2) i $v_1^1 = 0.0900$ (za x_3)

	x1	x2	x3
1. partikl	1.0667	-4.4719	0.0900
2. partikl	0.8100	0.5400	0.0900
3. partikl	7.4888	2.5728	-18.3177
4. partikl	-0.1678	1.4286	1.4286
5. partikl	-1.2109	0.5286	-13.6635

Ažurirana brzina

Ažuriramo poziciju partikla: $x_i^{t+1} = x_i^t + v_i^t$

Na primer, ažuriranje pozicije za prvi partikl: $x_1^{0+1} = x_1^0 + v_1^0$

$x_1^1 = 8.8 + 1.0667 = 9.8667$
$x_1^1 = 9.9 + (-4.4719) = 5.4281$
$x_1^1 = 1.1 + 0.900 = 2$

Na sličan način se izračunavaju vrednosti za svaki partikl i dobijaju se njihove ažurirane pozicije:

	x1	x2	x3
1. partikl	9.8667	5.4281	2
2. partikl	10.71	7.14	1.19
3. partikl	10.78	8.07	-7.317
4. partikl	10. 83	3.628	12.42
5. partikl	9.78	6.028	-4.8635

Ažurirane pozicije za svaku česticu

Korak 4: Evaluirati fitness funkciju, $f(x_i^t)$

Dakle, pronaći trenutni *gBest*

Nova vrednost fitnes funkcije
1.0512
1.5695
4.8866
3.6716
2.9504

Pri tome uzimamo u obzir sledeće:

if $f(x_1^1)$ < *f*(gBest) **then** gBest = x_i^t

$f(x_1^1) < f(gBest)$

$1.0512 < 1.3236$

gBest = 1.0512

$x^0_{1,2,3,4,5}$
Vrednost fitnes funkcije (t=0)
1.9649
1.3236
2.2179
2.9208
2.2542

$x^1_{1,2,3,4,5}$
Nova vrednost fitnes funkcije
1.0512
1.5695
4.8866
3.6716
2.9504

$f(x_2^1) < f(gBest)$

$1.5695 < 1.3236$

$f(x_3^1) < f(gBest)$

$4.8866 < 1.3236$

$f(x_4^1) < f(gBest)$

$3.6716 < 1.3236$

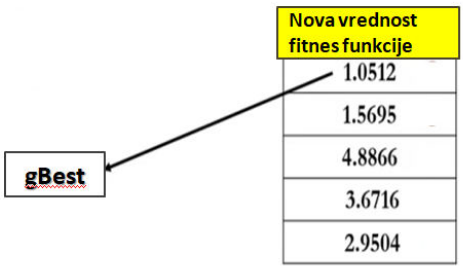
$f(x_5^1) < f(gBest)$

$2.9504 < 1.3236$

$f(x_1^1) < f(gBest)$

$1.0512 < 1.3236$

gBest = 1.0512



Zatim, izabrati partikl sa najboljom pozicijom (Best Known Position, *pBest*).

IF $f(x_i^t) < f(pBest)$ then

$pBest = x_i^t$

$f(x_1^1) < f(pBest)$

$1.0512 < 1.9649$

$pBest = 1.0512$

$f(x_2^1) < f(pBest)$

$1.5695 < 1.3236$

$f(x_3^1) < f(pBest)$

$4.8866 > 2.2179$

	x1	x2	x3
1. partikl	9.8667	5.4281	2
2. partikl	9.9	6.6	1.1
3. partikl	3.3	5.5	11
4. partikl	11	2.2	11
5. partikl	11	5.5	8.8

Ažurirane pozicije za svaku česticu

Nova pBest vrednost
1.0512
1.3236
7.2179
2.9208
2.2542

Korak 5: ažurirati brojac, $t = t+1$

Korak 6: Izlazne vrednosti (output) su *gBest* i x_i^t

1. partikl	9.8667	5.4281	2	1.0512

Nadalje se proces ponavlja sve dok uslov terminacije ne bude zadovoljen.

§3 EVOLUTIVNI ALGORITMI

Evolutivni algoritmi (EA) predstavljaju podskup jedne šire oblasti, evolutivne kompjutacije (engl. evolutionary computation), i odnose se na generičke populaciono-bazirane metahueristike (Eiben & Smith, 2015). Idejni koncept potiče od principa Darvinove evolucije. EAs su se pokazali kao odlična tehnika kod rešavanja optimizacionih problema. Ova vrsta adaptivnih algoritama može u razumnom vremenu da nađe približno-optimalno aproksimativno rešenje, što je veliki uspeh kada se uzmu u obzir NP-hard problemi.

Generalna šema rada bi se mogla ukratko opisati na sledeći način:

Populaciono-bazirani algoritmi – kao što ime govori – rade sa populacijom jedinki, od kojih je svaka potencijalno rešenje problema. Obično je populacija na startu slučajno generisana. Za svaku jedinku je određeno koliko je ona potencijalno dobro rešenje problema (upotrebom funkcije prilagođenosti). Najprilagođenije jedinke su izabrane za dalju proceduru, koja obično podrazumeva fazu reprodukcije. Nakon dobijanja novih jedinki, ciklus se ponavlja, a najslabije prilagođene jedinke se eliminišu iz populacije. Postoje razne varijante operatora koji se koriste. Uglavnom oni preslikavaju mehanizme selekcije, reprodukcije, mutacije i rekombinacije. Svaka sledeća generacija populacije dobijena kroz iterativna ponavljanja operatora trabala bi da sadrži bolji "genetski materijal", tj. potencijalno bolje jedinke (tačke) koje predstavljaju kandidate za rešenje razmatranog problema.

Kao što je rečeno, evolutivna kompjutacija objedinjuje slične tehnike koje se uglavnom razlikuju u detaljima implementacije (Eiben & Smith, 2015):

- Genetski algoritam (GA) – ova vrsta algoritama je nadalje detaljno opisana u podsekciji 3.1; ujedno predstavlja i najpopularniji tip EAs; reprezentacija jedinki u populaciji je od velikog značaja, i jedinke su obično (ne uvek) kodirane binarno u smislu da odražavaju nešto o rešenju problema;
- Genetičko programiranje - sama rešenja su predstavljena programima, dok funkcija prilagođenosti oslikava njihovu sposobnost da reše razmatrani problem;
- Evolutivno programiranje – metoda slična genetickom programiranju, s tim sto numerički parametri mogu da evoluiraju;
- Ostale srodne tehnike (evoluciona strategija, diferencijalna evolucija neuroevolucija, itd).

3.1. Genetski Algoritmi (GAs)

GAs su zasnovani na prirodnim, genetičkim zakonima i procesu evolucije (Holland, 1975; Goldberg, 1989). Oni zapravo simuliraju evolutivni proces i kojem je jedna od glavnih ideja – opstanak najprilagođenijih jedniki (tkzv. prirodna selekcija). U skladu s tim je njihov pristup rešavanju datog problema i sam tok rešavanja. Razmatra se oblast unutar koje se nalazi rešenje problema, tkzv. "prostor pretrage". Na slučajan način (ili nekim odeređenim postupkom) iz prostora pretrage izdvoji se konačan broj tačaka – kandidata za rešenje. Po analogiji sa prirodnim procesima formira se početna populacija čiji elementi (jedinke populacije) su izdvojene tačke prostora pretrage. Svaka jedinka u populaciji je jedinstveno određena sveukupnošću svog genetskog materijala predstavljenog preko hromozoma koji je preslikan u obliku nekog stringa (Alam et al., 2020). Hromozom je dugački DNK molekul sa genetskim materijalom jedinke gde su pohranjene nasledne karakteristike. Za svaku jedinku treba odrediti njenu prilagođenost u skladu sa činjenicom koliko je ona dobro rešenje problema. Zato se svakoj jedinki pridružuje odgovarajuća vrednost funkcije prilagođenosti (engl. fitness function). Pomoću genetskog

operatora selekcije (engl. selection) se - na osnovu vrednosti funkcije prilagođenosti - biraju "najbolje prilagođene" jedinke. Promene nastaju prilikom ukrštanja dve jednike kada dolazi do mešanja genetskog materijala oba roditelja. Ponekada se javlja i mutacija koja takođe dovodi do izmene karakteristika potomka.

Prema zakonima genetike i evolucije, postoji veća verovatnoća da nova populacija raspolaže boljim genetskim materijalom. Ponavljajući ovaj postupak, iz generacije u generaciju dolazi do poboljšanja genetskog materijala populacije i približavanja optimalnom rešenju datog problema. Postupak se završava nakon dostignutog broja generacija ili ispunjavanjem nekog kriterijuma konvergencije.

Moć genetskih algoritama proizlazi iz robustnosti ove tehnike: ne pretražuje se samo jedna tačka prostora pretrage, već cela populacija tačaka. GAs se razlikuju od drugih metoda i po tome što ne koriste pomoćne, dodatne informacije vezane za problem koji se rešava, niti parametrre problema, već njihove reprezentacije. Pripadaju grupi nedeterminističkih algoritama koji koriste probabilistička pravila prelaska (Whitley, 1994).

3.1.1. Prost GA

Princip rada kanonskog prostog genetskog algorima (engl. simple genetic algorithm, SGA) (Vose, 1999) može se opisati na sledeći način (Algoritam 3.1). Osnovni koraci su generisanje populacije rešenja, pronalaženje vrednosti fitnes funkcije i primena genetskih operatora.

```
begin
generate starting population
calculate fitness function for every individual in the population
t = false
while not t do
       begin
       repeat
         choose two individuals for crossover
         perform crossover and mutation to make two offsprings
         calculate fitness function of offsprings
         insert offsprings into new generation
       until new population is formed
       if new population converges then t:= true
       end
end
```

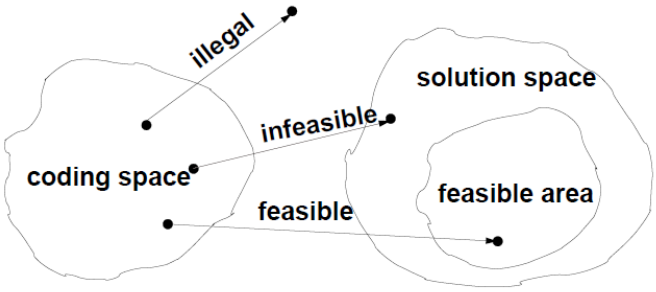
Alg 3.1. Prost GA.

Najvažnije komponente SGA su mehanizam kodiranja, funkcija prilagođenosti, genetski operatori i parametri genetskog algoritma.
Implementacija GA počinje formiranjem inicijalne populacije jedinki (najčešće na slučajan način). Nakon toga "evaluiraju" se jedinke na osnovu funkcije prilagođenosti u smislu da one jedinke koje predstavljaju bolje rešenje za zadati problem dobijaju veću šansu da se reprodukuju od jedinki koje predstavljaju lošije rešenje. Koliko je dobro/odgovarajuće neko rešenje je u kontekstu trenutne populacije.

Jedna od veoma važnih karakteristika GA jeste kodiranje (slika 3.1), tj. način kodiranja jedinki (Beasley et al., 1993). Jedinke kod SGA kodiraju se binarnim stringovima. Način kodiranja zavisi od konkretnog zadatka, tj. od prirode parametara i promenljivih u datom problemu. Funkcija prilagođenosti je zavisna od konkretnog zadatka koji se rešava pomoću GAs. Pridružena svakoj jedinki populacije, ona odražava validnost izbora odgovarajuće jedinke za rešenje problema. Naime, za dati problem se formira

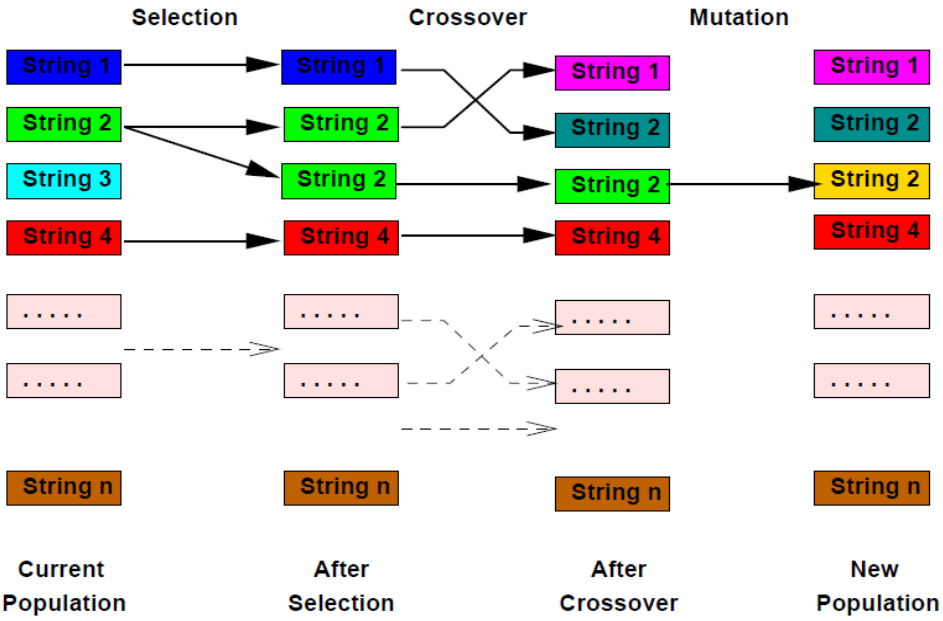
vrednosna funkcija (engl. objective function) koja treba da se optimizuje. Obično se vrednosna funkcija normalizuje tako da uzima vrednost izmedju 0 i 1 (radi obezbeđivanja uniformnosti) i upravo to predstavlja skup vrednosti funkcije prilagođenosti[9].

Ako zadati problem ima više varijabli, više-varijabilno kodiranje se konstruiše spajanjem onoliko jedno-varijabilnih kodiranja koliko ima varijabli u problemu.



Slika 3.1. Kodiranje u GA[9]

GAs procesiraju simultano više rešenja. U prvom koraku, kreira se populacija preko pseudo slučajih generatora čije jedinke predstavljaju izvodljivo/dopustivo rešenje (engl. feasible solution). U svakom slučaju, to predstavlja početno, inicijalno rešenje. Na taj način osigurava se robustna i "nepristrasna" pretraga, jer počinje od širokog ranga tačaka u prostoru pretrage. U sledećem koraku, jedinke populacije se procenjuju/evaluiraju putem objektivne funkcije. Objektivna funkcija je potom preslikana u fitnes funkciju koja računa prilagođenost svake jedinke populacije. Nakon toga se primenjuju GA operatori.



Slika 3.2. Osnovni GA operatori i formiranje nove populacije[9].

[9] https://datajobs.com/data-science-repo/Genetic-Algorithm-Guide-%5BTom-Mathew%5D.pdf

3.1.2. Fitness Funkcija

GAs imitiraju prirodni mehanizam preživljavanja najbolje adaptiranih jedinki. Funkcija prilagođenosti *F*(*i*) se u tu svrhu koristi (izvodi se na osnovu objektivne funkcije) i predstavlja meru prilagođenosti svake jedinke. To znači da će jedinke sa boljom funkcijom prilagođenosti imati veću šansu da budu izabrane za formiranje potomaka i posledično, njihov genetski materijal će ući u formiranje nove populacije.

Kod problema maksimizacije, funkcija prilagođenosti se obično uzima da bude jednaka objektivnoj funkciji *F*(*i*)=*O*(*i*). Kod problema minimizacije, potrebno je generisati nenegativne vrednosti u svim slučajevima i na odgovarajući način preslikati objektivnu funkciju na funkciju prilagođenosti. Najčešći metodi su (Beasley et al., 1993):

1) konverzija minimizacionog problema u ekvivalentan problem maksimizacije

$$\mathcal{F}(x) = \frac{1}{1 + f(x)} \tag{3.1}$$

2) transformacija objektivne funkcije u funkciju prilagođenosti *F*(*i*)

$$\mathcal{F}(i) = V - \frac{O(i)P}{\sum_{i=1}^{P} O(i)}, \tag{3.2}$$

gde je *O*(*i*) objektivna funkcija *i*-te jedinke, *P* je veličina populacije i *V* je dovoljno velika vrednost da bi se obezbedile nenegativne vrednosti funkcije prilagođenosti.

Nijedna navedena transformacija ne menja lokaciju rešenja.

3.1.3. GA operatori

Trenutna populacija se podvrgava GA operatorima u svrhu dobijanje nove populacije bolje prilagođenih jedinki. Osnovna tri GA operatora su: selekcija, ukrštanje i mutacija. Nova populacija je nadalje evaluriana i proces se ponavlja dok se ne ispuni uslov terminacije. Jedan ciklus primene ovih operatora i dobijanje nove populacije jedinki se često naziva i formiranje nove generacije.

Selekcija

Selekcijom se biraju najbolje prilagođene jedinke populacije u cilju dobijanja što većeg broja njihovih potomka u sledećoj generaciji. Verovatnoća izbora jedinke i veličina njene prilagođenosti u populaciji su direktno proporcionalne. Ovo je obicno prvi operator primenjen na populaciju. Dakle, operator selekcije bira “dobre” jedinke populacije i formira skup jedinki koje će se nadalje koristiti kod ukrštanja. Postoje brojni operatori selekcije u GA literaturi (Beasley et al., 1993), ali osnovna ideja kod svakog od njih jeste da korišćenjem principa verovatnoće jednike koje su prilagođene “iznad proseka” budu izabrane/izdvojene iz trenutne populacije i njihove višestruke kopije budu ubačene u skup jedinki koje će se nadalje koristiti kod ukrštanja. Jedan od najpoznatijih operatora selekcije jeste *rulet selekcija*.

Rulet selekcija se moze opisati na sledeći način:

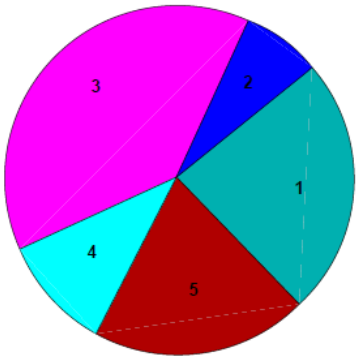
Neka je točak ruleta podeljen na kružne isečke, pri čemu je veličina *i*-tog isečka proporcionalna veličini $f_i/\sum f_i$, gde je f_i prilagođenost *i*-te jedinke (koja se nalazi u *i*-tom isečku), a $\sum f_i$ je prilagođenost cele populacije. Ona jedinka, na čijem se isečku zaustavi točak, bice izabrana. Pošto je veličina populacije

obično nepromenljiva vrednost (kod jednostavnog GA), suma verovatnoća svih izabranih jedinki mora biti 1. Stoga, verovatnoća izbora *i*-te jedinke je

$$p_i = \frac{\mathcal{F}_i}{\sum_{i=1}^{n} \mathcal{F}_i} \tag{3.3}$$

gde je *n* veličina populacije. Jedan od načina da se primeni ova šema izbora je da zamislite točak ruleta čiji je obim obeležen za svaku jedinku proporcionalan vrednosti prilagodjenosti te jedinke. Točak ruleta se okreće *n* puta, svaki put birajući instancu jedinke izabranu pokazivačem točka na ruletu. Pošto je obim točka označen prema vrednosti prilagođenosti jedinke, očekuje se da će ovaj mehanizam napraviti F_i=F kopije *i*-te jedinke u skup jedinki koje će se nadalje koristiti kod ukrštanja. Prosečna vrednost funkcije prilagođenosti se izračunava kao

$$\overline{\mathcal{F}} = \sum_{i=1}^{n} \mathcal{F}_i \tag{3.4}$$



Population	Fitness
1	25.0
2	5.0
3	40.0
4	10.0
5	20.0

Slika 3.3. Točak ruleta označen za pet jedinki prema njihovoj vrednosti funkcije prilagođenosti[9].

Na slici 3.3 prikazan je točak ruleta za svaku jedinku koji ima različite vrednosti fitnes funkcije. Pošto treća jedinka ima veću vrednost fitnes funkcije od bilo koje druge, očekuje se da će selekcija ruleta birati treću jedinku više od bilo koje druge jedinke. Koristeći vrednost fitnes funkcije *Fi* svih jedinki, može se izračunati verovatnoća p_i izbora *s* jedinke. Nakon toga, kumulativna verovatnoća P_i svake jedinke (koja se kopira) može se izračunati dodavanjem pojedinačnih verovatnoća sa vrha liste. Dakle, najniže rangirana jedinka u populaciji treba da ima kumulativne vrednosti verovatnoće od P_i-1 do P. Prva jedinka predstavlja kumulativne vrednosti od nule do P_1. Dakle, kumulativna verovatnoća bilo koje jedinke leži između 0 i 1[9].

Da bi se izabralo *n* jedinki, kreira se *n* slučajnih brojeva između nula i jedan. Jedinka sa višom vrednošću fitnes funkcije ce predstavljati veći opseg u kumulativnim vrednostima verovatnoće i stoga će imati veću mogućnost kopiranja u skup jedinki koje ce se koristiti kod ukrštanja. S druge strane, niz sa manjom vrednošću fitnes funkcije predstavlja manji opseg kumulativnih vrednosti verovatnoće i ima manju verovatnoću da bude kopiran u skup jedinki koje će se koristiti kod ukrštanja.

Tokom izvrsavanja GA, iz generacije u generaciju, dolazi do poboljšanja kvaliteta genetskog materijala populacije. U određenom momentu mogu se pojaviti jedinke koje poseduju relativno dobar genetski potencijal. To naravno ne mora da znači da se radi o traženim jedinkama koje bi uslovljavale konvergenciju procesa ka globalnom optimum. Moguće je (vrlo često primenom rulet selekcije) da se te jedinke rašire u populaciji postajuci nadmoćne, dovedu do preranog završetka procesa i njegove konvergencije ka lokalnom optimumu.

Selekcija bazirana na rangu (Beasley et al., 1993) prevazilazi problem prevremene konvergencije. Jedinke se sortiraju prema njihovoj funkciji prilagođenosti, a zatim se svakoj jedinki jednoznačno pridružuje vrednost ranga te jedinke tj. svakom stringu koji reprezentuje jedinku dodeljuje se jedinstveni rang. Nadalje, vrednost ranga jedinke je njena funkcija prilagođenosti.

```
begin
for i:=1 to number_of_individuals do
       assign to i^th individual its rank f_i
sum:= 0
for i:=1 to number_of_individuals do
        sum:=sum+ f_i
for i:=1 to number_of_individuals do
       begin
       s:=0
       r:= random ([0,sum])
       k:= 1
       while (k<= number_of_individuals and s<=r)
              begin
              s:=s+f_k
              k:=k+1
              end
       p_i:= p_k-1
       end
end
```

Alg 3.2. Selekcija bazirana na rangu.

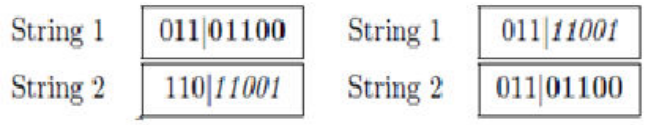
Ukrštanje

Ukrštanje je operacija od izuzetnog značaja za GA, pokretačka snaga kroz prostor pretrage. Primenjuje se posle selekcije u cilju generisanja potomaka. Jednopoziciono ukrštanje, prikazano na slici 3.4, uzima dve jedinke populacije i razmenjuje njihov genetski materijal počevši od neke pozicije u stringovima. Pozicija ukrštanje se određuje na slučajan način.

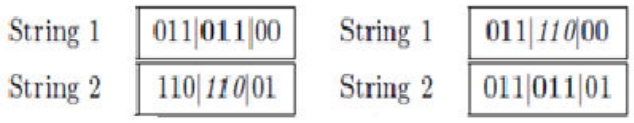
Operator ukrštanja se koristi za rekombinovanje dva niza da bi se dobio bolji string (Beasley et al., 1993). U ukrštanju, proces rekombinacije stvara različite jedinke u uzastopnim generacijama kombinovanjem materijala od dve jedinke prethodne generacije. U reprodukciji, dobrim nizovima u populaciji se dodeljuje veći broj kopija i formira se grupa za parenje (tkzv. mating pool). Važno je napomenuti da se u fazi reprodukcije ne formiraju nove jedinke. U operatoru ukrštanja, novi nizovi (jedinke) se kreiraju razmjenom informacija između jedinki skupa za sparivanje. Dve jedinke koja učestvuju u operaciji ukrštanja poznate su kao roditeljske jedinke, a rezultujuće jedinke poznate su kao jedinke potomci. Intuitivno je iz ove konstrukcije da se dobri podnizovi iz roditeljskih nizova mogu kombinovati da bi se formirao bolji niz potomka, ako se pogodno odabere odgovarajuća pozicija ukrštanja. Sa nasumično izabranom pozicijom, proizvedeni nizovi potomaka mogu ili ne moraju imati kombinaciju dobrih pod-stringova iz roditeljskih nizova, u zavisnosti od toga da li je pozicija ukrštanja odgovarajuća. Ovo nije stvar za ozbiljnu zabrinutost, jer ako se dobri nizovi kreiraju ukrštanjem, biće više njihovih kopija u sledećem skupu parenja generisanog ukrštanjem. Iz ove rasprave je jasno da efekat ukrštanja može biti štetan ili koristan. Dakle, da bi se sačuvale neki od dobrih stringova koje su već prisutne u grupi za parenje, svi stringovi u grupi za parenje se ne koriste u ukrstanju[9].

Kada se koristi verovatnoća ukrštanja, p_c, samo $100p_c$% stringova u populaciji se koristi u operaciji ukrštanja i $100(1-p_c)$% populacije ostaje kao što je u trenutnoj populaciji. Operator ukrštanja je uglavnom odgovoran za pretragu novih stringova iako se operator mutacije takođe retko koristi za ovu svrhu. U literaturi GA postoji mnogo operatora ukrštanja. Jednopoziciono ukrštanje i dvopoziciono ukrštanje su najčešće primenjeni. Kod većine operatora ukrštanja, dva stringa se nasumično biraju iz skupa za

spajanje i neki njihov deo se uzajamno razmenjuje. Operacija ukrštanja se vrši slučajnim odabirom dva stringa. Jednopozicioni operator ukrštanja se izvodi slučajnim odabirom pozicije ukrštanja u stringu i razmenom svih bitova na desnoj strani mesta ukrštanja kao što je prikazano na slici 3.4.



Slika 3.4 Jednopoziciono ukrštanje: levo – situacija pre primene ukrštanja, desno - situacija posle primene ukrštanja9.



Slika 3.5 Dvopoziciono ukrštanje: levo – situacija pre primene ukrštanja, desno - situacija posle primene ukrštanja9.

U jednopozicionom ukrštanju, mesto/ pozicija za ukrštanje se bira nasumično (prikazano kao vertikalne linije). Deo desno od izabranog mesta ova dva stringa/jedinke se razmenjuju da bi se formirao novi par jedinki. Nove jedinke su stoga kombinacija starih jedinki. Dvopoziciono ukrštanje je varijacija jednopozicionog ukrštanja, osim što su izabrane dve pozicije ukrštanja i bitovi između pozicija se razmenjuju kao što je prikazano na slici 3.5. Jednopoziciono ukrštanje je pogodnije kada je dužina stringa mala, dok je dvopoziciono ukrštanje pogodno za velike stringove. Osnovni cilj ukrštanja je razmena informacija između jedinki da bi se dobila jedinka/string koji je možda bolji od roditelja.

Kao sto je već definisano, jednopoziciono ukrštanje uzima dve jedinke populacije i razmenjuje njihov genetski materijal počevši od neke pozicije u stringovima. Pozicija ukrštanja se određuje na slučajan način.

```
begin
if (flip(pcross))
then crossover_position:=random(chromosome_length)
else crossover_position:=chromosome_length
 for i:=1 to chromosome_length do
       if (i<= pos) or (pos = 0)
       then
              offspring1.string[i]:= parent1.string[i]
              offspring2.string[i]:= parent2.string[i]
       else
              offspring1.string[i]:= parent2.string[i]
              offspring2.string[i]:= parent1.string[i]
 end
end
```

Alg 3.3. Jednopoziciono ukrštanje.

Mutacija

Mutacija kod SGA ima sekundaran značaj i služi za regenerisanje genetskog materijala. Primenjuje se na svaku jedinku posle ukrštanja. Prosta mutacija vrši izmenu izabranog gena (odnosno bita: 1 u 0, i obratno) sa malom verovatnoćom (Beasley et al., 1993).

Mutacija dodaje nove informacije na slučajan način u proces genetske pretrage i pomaže da se izbegne zaustavljanje procesa pronalaženjem na lokalnog optimuma. Ovaj operator unosi raznolikost u populaciju kad god populacija teži da postane homogena zbog ponovljene upotrebe operatora selekcije i ukrštanja. Mutacija može prouzrokovati da se hromozomi jedinki razlikuju od hromozoma njihovih roditelja. Operator mutacije radi na nivou bita: kada se bitovi kopiraju iz trenutnog stringa u novi string, postoji verovatnoća da svaki bit može postati mutiran. Ova verovatnoća je obično prilično mala vrednost, i naziva se verovatnoća mutacije p_m. Koristi se mehanizam za bacanje novčića; ako je slučajni broj između nule i jedan manji od verovatnoće mutacije, tada se bit invertuje, tako da nula postaje jedan, a jedan postaje nula. S jedne strane, ovo unosi raznolikost u populaciju, ali s druge strane, može proizvesti slabu jedinku koja nikada neće biti izabrana za dalje operacije[9].

Potreba za mutacijom je da se stvori tačka u okolini trenutne tačke, čime se postiže lokalna pretraga oko trenutnog rešenja. Mutacija se takođe koristi za održavanje raznolikosti u populaciji. Na primer, uzmimo u obzir sledeću populaciju koja ima četiri niza od osam bita:

01101011
00111101
00010110
01111100.

Može se primetiti da sva četiri niza imaju 0 na krajnjoj levoj poziciji bita. Ako pravo optimalno rešenje zahteva 1 u toj poziciji, onda ni operator reprodukcije ni ukrštanja neće moći da kreiraju 1 u toj poziciji. Uključivanje mutacije uvodi verovatnoću p_m pretvaranja 0 u 1.

Dakle, operator selekcije bira dobre stringove/jedinke, a operator ukrštanja rekombinuje dobre pod-stringove iz dobrih stringova zajedno da bi stvorile bolje jedinke. Operator mutacije lokalno menja string očekujući bolji string. Iako nijedna od ovih tvrdnji nije zagarantovana i/ili testirana tokom kreiranja stringa, očekuje se - ako se naprave loši stringovi - oni budu eliminisani od strane operatera selekcije u sledećoj generaciji, a ako se naprave dobri nizovi - oni budu sve više naglašavani. Primena ovih operatera na trenutnoj populaciji stvara novu populaciju. Ova nova populacija se koristi za generisanje sledećih populacija i tako dalje, dajući rešenja koja su bliža optimalnom rešenju. Vrednosti objektivne funkcije nove populacije ponovo se određuju dekodiranjem stringova. Ove vrednosti izražavaju podobnost rešenja novih generacija. Ovim se završava jedan ciklus genetskog algoritma koji se zove generacija. U svakoj generaciji, ako je rešenje poboljšano, ono se čuva kao najbolje rešenje. Ovo se ponavlja do konvergencije.

Performanse operatora mutacije poboljšane su korišćenjem normalne raspodele. Realizacija je brza, jer se obrađuju samo mutirani geni. Broj mutacija je generisan slucajnim izborom u $N(m,\sigma)$, gde

$$m = (npop - nelite)\cdot p_{mut}$$
$$\sigma^2 = (npop - nelite)\cdot p_{mut}\cdot(1- p_{mut})$$

m je prosečan broj mutacija u populaciji, σ je standardna devijacija, *npop* je veličina devijacije, a *nelite* je broj elitnih jedinki. Nakon toga, pozicije mutacije su slucajno generisane. Odgovarajući nivo mutacije omogućava dobru ravnotežu između dva kontradiktorna zahteva: eksploracije i eksploatacije koje bi morao objediniti svaki efikasan optimizacioni algoritam u cilju nalaženja globalnog optimuma.

Eksploracija omogucava prodiranje u neispitane, nove regione unutar prostora pretrage. Kod operatora ukrštanja i mutacije, populacija se obogaćuje svežim genetskim materijalom, novim jedinkama koje predstavljaju nove mogućnosti za rešenje. Eksploatacija podrazumeva upotrebu stečenih informacija o već ispitanim regionima prostora pretrage. Operator selekcije eksploatiše saznanja o prilagođenosti jedinki sa ciljem formiranja kvalitetnije populacije, tj. pronalaženja kvalitetnijih tačaka u prostoru pretrage. Ravnoteža eksploracije i eksploatacije eliminiše mogućnost pojave neželjenih posledica: prevremena konvergencija, koju bi uslovila preterano izražena eksploatacija, i degenerisanja genetskog algoritma u slučajnu pretragu, do koje bi doslo usled preterane eksploracije.

Pravilnim izborom genetskih operatora mogu se izbeći negativni efekti kao sto su: prevemena konvergencija (engl. premature convergence), spora konvergencija (engl. slow convergence), gubitak genetskog materijala, itd. koji dovode do neuspešnog ili manje uspešnog rešavanja problema. Zato je potrebno odrediti koja vrsta operatora selekcije, ukrstanja i mutacije najbolje odgovara za dati problem.

3.1.4. Parametri GA

Ravnoteža eksploracije i eksploatacije određena je upravo pravilnim izborom vrednosti parametara. U (Beasley et al., 1993) izloženi su opsti zaključci o posledicama uvećanja vrednosti parametara:

- Uvećanje verovatnoće ukrštanja
  Dobra strana: jača rekombinaciju gradivnih blokova;
  Nedostatak: uvećava raskidanje dobrih stringova (i nestajanje potencijalno dobrih jedinki);
- Uvećanje verovatnoće mutacije
  Dobra strana: pomaže regenerisanju izgubljenog genetskog materijala i uvođenju novog;
  Nedostatak: može da genetski algoritam degeneriše u slučajnu pretragu;
- Uvećanje velicine populacije
  Dobra strana: poboljšava raznolikost genetskog materijala;
  Nedostatak: uvećava vreme potrebno da populacija konvergira ka optimalnim regionima prostora pretrage;

U (Beasley et al., 1993) predložene su neke smernice u određivanju vrednosti parametara koji garantuju dobre performanse na pažljivo izabranim test primerima. Mogu se izdvojiti dva osnovna skupa parametara:

1. Mala veličina populacije (npr. 30) i relativno veliki nivo ukrštanja (0.9) i mutacije (0.01);
2. Veća populacija (npr. 100), a manje verovatnoće ukršanja (0.6) i mutacije (0.001).

3.1.5. Napredne tehnike genetskih algoritama

U cilju poboljšanja SGA i povećanja njihove efikasnosti, razvijeni su napredni operatori i tehnike genetskih algoritam (Beasley et al., 1993).
Neka poboljsanja genetskih operatora obuhvataju:

- Selekcija

Osim rulet selekcije, koriste se i drugi, raznovrsni operatori selekcije, formirani u cilju poboljšanja rešenja problema na koje se je primenio GA. Kod ovog operatora, predloga za njegovo poboljšanje ima mnogo. Neki su opšte primenljivi, dok su drugi više vezani za određene problem, gde su dali efikasne rezultate. Najznačajniji su: selekcija zasnovana na rangu (engl. rank-based selection), turnirska selekcija (engl. tournament selection) i selekcija sa primenom ostataka (engl. stochastic remainder selection), ali se koriste i ostali operatori selekcije: selekcija isecanjem (engl. truncation selection), linearno i

eksponencijalno rangiranje (engl. linear ranking, exponential ranking), univerzalna selekcija (engl. stochastic universal sampling) (Beasley et al., 1993).

- Ukrštanje

Osim jednopoziocionog ukrštanja, koriste se i druge varijante ukrštanja: dvopoziciono ukrštanje (engl. two-point crossover), visepoziciono ukrštanje (engl. multi-point crossover), uniformno ukrštanje (engl. uniform crossover)(Beasley et al., 1993).

- Mutacija

Najčešće se koristi prosta mutacija, iako su i ovde predloženi operatori mutacije specifični za određeni problem.

Načini kodiranja

Klasičan način kodiranja kod GA je binarno kodiranje koje je lako za implementaciju. Kod ovakvog kodiranja može se desiti da se genetski kodovi susednih jedinki razlikuju u velikom broju bitovnih mesta (engl. Hamming distance). Ovaj nedostatak moguće je prevazići korišćenjem Grey-ovih kodova (Hamming distance je 1). Postoje i kodiranja alfabetom kardinalnosti veće od 2 (Beasley et al., 1993).

Načini formiranja stringa koji reprezentuje jedinku

Uobičajen GA se zasniva na korišćenju haploidnih hromozoma (po jedan gen za svaku osobinu) prvenstveno zbog jednostavnosti. Genetička istraživanja viših oblika života u kojima figurišu diploidni hromozomi (po dva gena, dominantni i recesivni, za svaku osobinu) reflektovali su se i u oblst GAs (Beasley et al., 1993).

Politika zamene generacija

Prelaskom iz jedne generacije u drugu menja se i struktura populacije U GAs primenjuju se sledeće realizacije te zamene (Beasley et al., 1993):

- Generacijski oblik GA (engl. generational GA), gde se smenjuje cela populacija;
- Stacionarni GA (engl. steady-state GA), gde se smenjuju deo populacije;

Promena vrednosti kontrolnih parametara

Dinamičko određivanje vrednosti kontrolnih parametara tokom izvršavanja programa moze dovesti do poboljšanih rezultata i performansi GA u odnosu na rezultate dobijene uobičajenim korišcenjem statickih vrednosti parametara (Beasley et al., 1993).

Podešavanje parametara za operatore ukrštanja i mutacije

Kako implementirati verovatnoće ukrštanja i mutacije, p_c i p_m? Na primer, ako je $p_m = 0.08$ i $p_c = 0.78$. To znači da 0.08% između 100 hromozoma šansa za mutacijom je 0,08. Isto je u slučaju ukrštanja.

Kako korektno upotrebiti verovatnoću ukrštanja i mutacije? Jednostavan način jeste generisati slučajan broj R uniformno raspodeljen. Tada ako je $R \leq p_c$, operator ukrštanja je primenjen. Isti proces se primenjuje i za slučaj mutacije.

Primer

Maksimizirati funkciju $f(x) = x^2$ gde $x \in [0, 31]$.

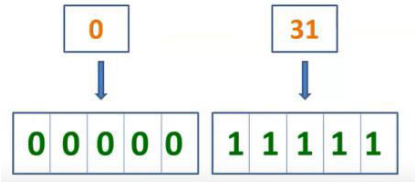
U tu svrhu prolazimo kroz sledeće korake GA:

1. Izabrati tip kodiranja
2. Izabrati veličinu populacije
3. Formirati populaciju slučajnim izborom
4. Izvršiti operator selekcije (izbor parova roditeljskih hromozoma)

5. Izvršiti operatore ukrštanja i mutacije
6. Evaluacija potomaka
7. Ako nije zadovoljen uslov terminacije, vratiti se na korak 4.

Korak 1: Izabrati tip kodiranja

Biramo konvencionalni pristup binarnog kodiranja:



Korak 2: Izabrati veličinu populacije

Uzmimo kao primer veoma jednostavnu postavku – da je veličina populacija svega 4 jedinke, $n = 4$.

Korak 3: Formirati populaciju slučajnim izborom

Na primer: 13, 24, 8, 19

Korak 4: Izvršiti operator selekcije (izbor parova roditeljskih hromozoma)

U tu svrhu, najjednostavniji operator selekcije je tkzv. rulet selekcija.

Broj jedinke	Inicijalna populacija	x	F(x)	Verovatnoća	Očekivani broj	Stvarni broj
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1
Total			1170	1	4	
Average			293			

Korak 5: Izvršiti operatore ukrštanja i mutacije

Biramo par hromozoma za ukrštanje (u ovom slučaju uzeto je jednopoziciono ukrstanje) i na slucajan način određujemo poziciju ukrštanja, što je za prvi par između 4-te i 5-te pozicije, dok je za drugi par između 2-ge i 3-ce pozicije.

String 2	11000	Kombinacija 1
String 1	01101	
String 2	11000	Kombinacija 2
String 4	10011	

String 2	11000	11001
String 1	01101	01100
String 2	11000	11011
String 4	10011	10000

Korak 6: Evaluacija potomaka

Dakle, ponovo izračunavamo vrednost fitnes funkcije sa novoformiranim potomcima. U ovom slučaju, kao sto se može videti, potomak (hromozom) 3 ima najbolju vrednost fitnes funkcije.

Broj jedinke	Potomak	x	F(x)
1	01100	12	144
2	11001	25	625
3	11011	27	729
4	10000	10	256

Dalje se postupak vraća na korak 4 i nastavlja sve dok se ne ispuni uslov terminacije (npr. dok se ne dostigne određeni broj iteracija ili stepen konvergencije procesa).

§4 NEURONSKE MREŽE

Iako je ovo poglavlje posvećeno veštačkim neuronskim mrežama, u prvoj sekciji 4.1. uvode se opšti pojmovi vezani za mašinsko učenje. Podsekcija 4.1.1 definiše osnovne pojmove nadgledanog mašinskog učenja, dok je podsekcija 4.1.2 posvećena uvodnim pojmovima nenadgledanog mašinskog učenja. Podsekcija 4.1.3 u kratkim crtama predstavlja učenje uz podsticaje.

4.1. Uvodni pojmovi – mašinsko učenje

Posebna grupa metoda veštačke inteligencije su metodi i algoritmi mašinskog učenja. Mašinsko učenje "omogućava" sistemu da uči kroz upotrebu podataka umesto primene eksplicitnog programiranja. U kategoriju mašinskog učenja ulazi veliki broj algoritama koji iterativno "uče" kroz podatke da bi poboljšali, opisali podatke i predvideli ishod.

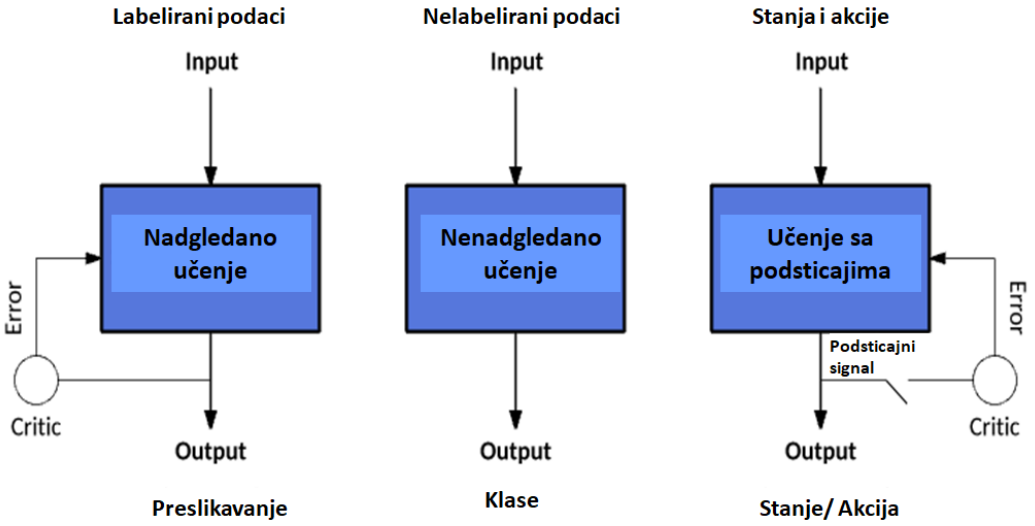
Na osnovu podataka koji se koriste u fazi "treniranja" (engl. training data), moguće je proizvesti preciznije modele bazirane na tim podacima. Na primer, prediktivni algoritam kreira prediktivni model. Kada se obezbede odgovarajući podaci za takav model, dobija se predikcija bazirana na tim podacima. Mašinsko učenje je esencijalno za kreiranje analitičkih modela. Uopšte govoreći, mašinsko učenje se odnosi na sposobnost softverskog sistema da: 1) učini generalizaciju na osnovu prethodnog "iskustva", tj. skupa podataka o pojavama ili entitetima koji su predmet učenja, 2) koristi kreirane generalizacije kako bi pružio odgovore na pitanja koja se tiču entiteta ili pojava koje pre nije sretao.

Osnovni oblici mašinskog učenja su (slika 4.1):

- Nadgledano učenje (engl. supervised learning)
- Nenadgledano učenje (engl. unsupervised learning)
- Učenje uz podsticaje (engl. reinforcement learning)

Možemo reći da su zajednički koraci za sve oblike procesa mašinskog učenja (Mitchell, 1997):

- Prikupljanje podataka potrebnih za formiranje skupova podataka za obuku, validaciju i testiranje modela mašinskog učenja;
- Priprema podataka (tkzv. "čišćenje" i transformacija podataka);
- Analiza rezultujućih skupova podataka, i njihovo, eventualno, dalje unapređenje kroz selekciju/transformaciju atributa;
- Izbor jednog ili više metoda mašinskog učenja;
- Obuka, konfiguracija i evaluacija kreiranih modela;
- Izbor modela koji će se koristiti (na osnovu rezultata koraka 5) i njegovo testiranje.



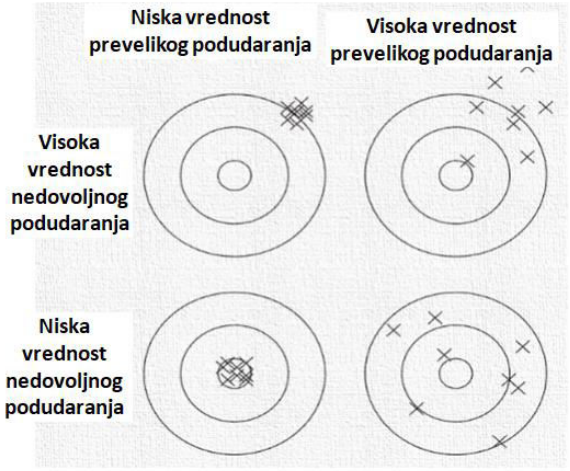
Slika 4.1. Osnovni oblici mašinskog učenja

Podaci su potrebni u tri različite faze: trening, validaciju i testiranje modela. Stoga, podaci se obično dele prema sledećoj semi: 60% za trening, 20% za validaciju i 20% za testiranje. Izbor uzoraka za trening, validaciju i testiranje radi se na slučajan način. U slucaju nadgledanog učenja, moramo imati "obeležene" (labelirane) podatke (na primer, obeležiti slike koje sadrže odgovarajuci patern, e-mail adrese koje su lažne, itd). Uočavanjem osobina ili izostankom nekih osobina, kao i uočavanjem odnosa između raznih osobina detektujemo pojave/enitete. To upravo uslovljava koji atributi (engl. features) će biti odabrani kao najbolji za opis odgovarajućeg entiteta ili pojave, kao i za njihovo razlikovanje. Odabir metode mašinskog učenja zavisi od vrste konkretnog problema koji se rešava, kao i od karakteristika atributa (tip atributa, stepena homogenosti tipova i opsega vrednosti atributa, stepen međuzavisnosti atributa) i obima podataka koji su na raspolaganju (Mitchell, 1997).

U fazi testiranja, radi se procena uspešnosti modela. Tada se koriste podaci koji se nisu koristili u fazi učenja (otprilike izdvaja se 20-30% ukupnih podataka). Uspešnost modela se utvrđuje različitim metrikama kao sto su tačnost, preciznost, odziv, itd. U fazi validacije bira se najbolji model između više kandidata, odredjuje se optimalna konfiguracija parametara modela sa ciljem izbegavanja problema prevelikog podudaranja (engl. over-fitting) i problema nedovoljnog podudaranja (engl. under-fitting).

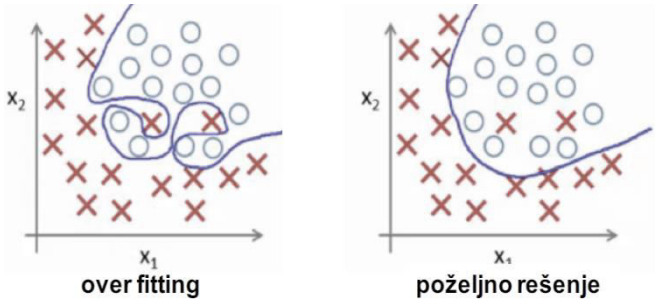
Kao što je već izloženo, kad se radi validacija, ukupan skup podataka se obično deli u odnosu 60/20/20 na podatake za trening, validaciju i testiranje. Šta je smisao validacije i za šta se koriste podaci za validaciju? Oni nam prvenstveno trebaju za poređenje performansi različitih modela, ali i izabranog modela sa različitim vrednostima parametara u cilju izbora optimalnog modela za dati problem. Poznat pristup za efikasno korišćenje raspoloživih podataka je tkzv. kros-validacija (engl. cross validation). Skup podataka za trening se deli na n podskupova, pri čemu se najčešće uzima n = 10 (engl. fold cross validation). Za to je potrebno da se izvede n iteracija treninga i validacije modela na sledeći način: u svakoj iteraciji, jedan deo podataka se uzima za potrebe validacije, a ostatak (n-1 delova) se koristi za učenje. Treba napomenti da se bira uvek različiti podskup koji će se koristiti za validaciju. Tokom svake iteracije računaju se performanse modela, a na kraju prosečna uspešnost na nivou svih n iteracija. Tako izračunate mere uspešnosti daju pouzdaniju sliku o performansama modela. Reći ćemo da je procena uspešnosti modela pouzdana, ako su rezultati u svih n iteracija vrlo slični (Mitchell, 1997).

Već su spomenuti problem prevelikog podudaranja (engl. over-fitting, variance) i problem nedovoljnog podudaranja (engl. under-fitting, bias). Veoma je važno naći njihov "kompromis" pri kreiranju modela mašinskog učenja.



Slika 4.2. Odnos problema prevelikog podudaranja i problema nedovoljnog podudaranja[10].

Problem prevelikog podudaranja opisuje situaciju u kojoj model odlično vrši predikciju za instance iz skupa za treniranje, ali u slučaju da se instance nesto razlikuju od naučenih, model ima veoma slabu sposobnost predikcije. Problem over-fitting-a je usko povezan sa visokom varijansom (engl. variance) korišćene metode mašinskog učenja.

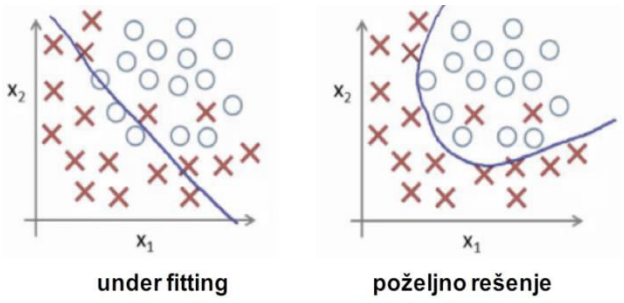


Slika 4.3. Problem prevelikog podudaranja[10]

Varijansa izrazava meru koliko bi se kreirani model mašinskog učenja promenio ukoliko bi došlo do promene podataka u korišćenom skupu za trening. U svakom slučaju, zavisna je od stepena kompleksnosti samog metoda mašinskog učenja: što je metoda mašinskog učenja složenija/fleksibilnija, to će njena varijansa biti veća.

Problem nedovoljnog podudaranja opisuje situaciju u kojoj model ne uspeva da aproksimira podatke za trening, pa posledicno ima slabe performanse čak i na trening setu. Ovaj problem je usko povezan sa visokim pomerajem (engl. bias) korišćene metode mašinskog učenja.

[10] https://www.coursera.org/learn/machine-learning



Slika 4.4. Problem nedovoljnog podudaranja[10]

Pomeraj reflektuje grešku koja se javlja u slučaju korišćenja jednostavnog modela za potrebe rešavanja složenog, realnog problema. Generalno, što je metoda mašinskog učenja složenija/fleksibilnija, to će pomeraj biti manji.

4.1.1. Nadgledano učenje

Ova vrsta mašinskog učenja obuhvata skup problema i tehnika za njihovo rešavanje u kojima program koji uči dobija skup ulaznih podataka (x_1, x_2, …, x_n) i skup ciljnih vrednosti. Za svaki ulazni podatak x_i, postoji ciljni izlaz y_i. Zadatak programa je da "nauči" kako da novom, neobeleženom ulaznom podatku dodeli tačnu izlaznu vrednost. Metode nadgledanog učenja počinju uspostavljanjem skupa podataka i načina kako su ti podaci klasificirani. Smisao ovih metoda je da nađu paterne u podacima koji bi se upotrebili u analitičkim procesima. Uočavaju i označavaju se (labeliraju) odgovarajuće osobine koje definišu značenje podataka. Ako je labela neprekidna/ kontinualna, govorimo o regresiji. U slučaju da podaci dolaze iz konačnog skupa vrednosti, onda govorimo o klasifikaciji (Mitchell, 1997).

Algoritmi su trenirani korišćenjem preprocesovanih primera, a performansa algoritma je evaluirana pomoću test podataka. Ponekada se desi da paterni koji su identifikovani u podskupu podataka ne mogu biti detektovani u većoj populaciji podataka. Kod nadgledanog učenja, svaki primer je par koji se sastoji od ulaznog objekta (obično vektorskog tipa) i željene izlazne vrednosti. Odgovarajući algoritam analizira podatke za trening i kreira funkciju koja može biti upotrebljena za preslikavanje novih primera.
Da bi se rešio problem nadgledanog učenja, neophodno je proći kroz sledeće korake (Mitchell, 1997):

- Odrediti tip podataka za kreiranje: to znači da korisnik kao prvo treba da odredi koja vrsta podataka će biti korišćena kao podaci za treniranje;
- Prikupiti podatke za treniranje: ovaj skup treba biti reprezentativan u odnosu na stvarne podatke i aplikacije; u tu svrhu se koristi prikupljanje ili od strane eksperata ili na osnovu merenja;
- Odrediti predstavljanje ulaznog objekta, tj. atributa: tačnost funkcije "učenja" (engl. learned function) zavisi od toga kako je ulazni objekat predstavljen; obično se ulazni objekat transformiše u vektorski tip koji sadrži veliki broj osobina koje opisuju objekat; u svakom slučaju, broj osobina ne treba da bude preveliki (zbog dimenzionalnosti), ali je potrebno da sadrži ipak dovoljno informacija da bi tačno predvideo izlaz (output);
- Odrediti strukturu funkcije "učenja" i odgovarajuceg algoritma (npr. decision trees);
- Kompletirati dizajn i pokrenuti algoritam na prikupljenom skupu podataka za treniranje: treba imati u vidu da i kod ovih algoritama postoje kontrolni parametri, tako da se određuje skup najboljih vrednosti tih parametara. Ovi parametri mogu biti podeseni optimizujući performansu

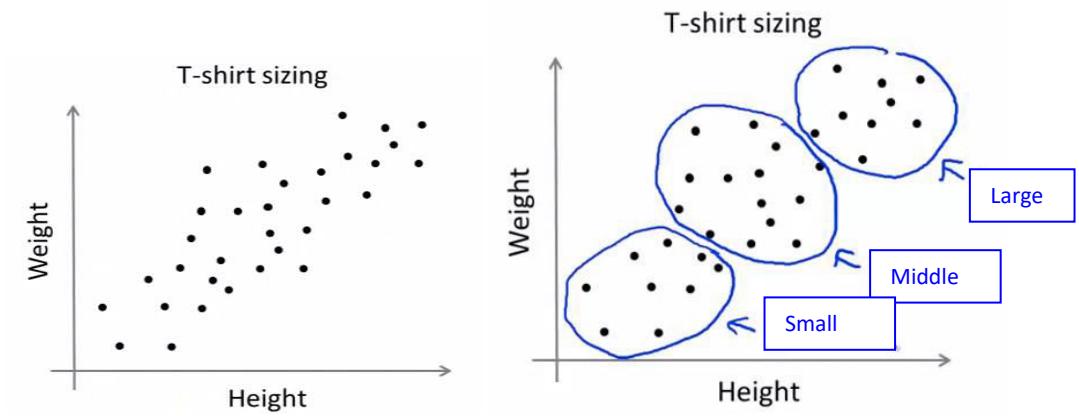
na podskupu (tkzv. validacionom skupu) skupa podataka za treniranje, ili preko kros-validacije (engl. cross-validation);

- Oceniti tačnost funkcije "učenja": posle podešavanja vrednosti parametara i "učenja", performansa funkcije treba da se oceni na skupu koji nije skup podataka za treniranje.

Veliki broj algoritama nadgledanog učenja je dostupan, od kojih svaki ima dobre strane i potencijalne nedostatke. Tako da ne možemo reći da postoji univerzalno najbolji algoritam nadgledanog učenja za sve problem nadgledanog učenja. Primer jedne od najpoznatijih metoda su stabla odlučivanja (engl. decision trees) (Rokach & Maimon, 2005).

4.1.2. Nenadgledano učenje

Metode i algoritmi nenadgledanog učenja segmentiraju podatke u grupe klastera (engl. clusters) ili grupe atributa (engl. features). Proces radi sa neoznačenim podacima i dodaje oznake (labelira podatke) tako da postaje nadgledano učenje. Ove metode mogu se primeniti u slučaju masivne količine podataka. U tom slučaju, kontekst podataka koji se analiziraju nije poznat, te labeliranje nije moguće u toj fazi (Mitchell, 1997). Dakle, nenadgledano učenje može da se koristi kao prvi korak pre predaje podataka nekom procesu nadgledanog učenja.



Slika 4.5. Primer formiranja klastera u nenadgledanom učenju[11]

Slicno kao kod nadgledanog učenja, ovi algoritmi traže paterne u podacima. Međutim, razlika jeste da ovde podaci jos nisu poznati/razumljivi. Dakle, kod ove vrste učenja, nemamo informacija o željenoj izlaznoj vrednosti. Program dobija samo skup ulaznih podataka (x_1, x_2, …, x_n). Zadatak programa je da otkrije paterne, tj. skrivene strukture i/ili zakonitosti u podacima (na primer, određivanje konfekcijskih veličina na osnovu visine i težine ljudi, slika 4.5). Na taj način, algoritmi nenadgledanog učenja pomažu u određivanju ishoda brze od algoritama nadgledanog učenja. Oni "uče"paterne iz neoznačenih podataka. Za razliku od metoda nadgledanog učenja gde su podaci označeni od strane eksperta, metodi nenadgledanog učenja ispoljavaju osobine samo-organizovanja (engl. self-organization). Takođe, postoji veliki broj algoritama nenadgledanog učenja. Jedna od popularnih metoda nenadgledanog učenja je hijerarhijsko klasterovanje (engl. hierarchical clustering). (Teichgraeber & Brandt, 2018).

[11] https://www.coursera.org/learn/machine-learning

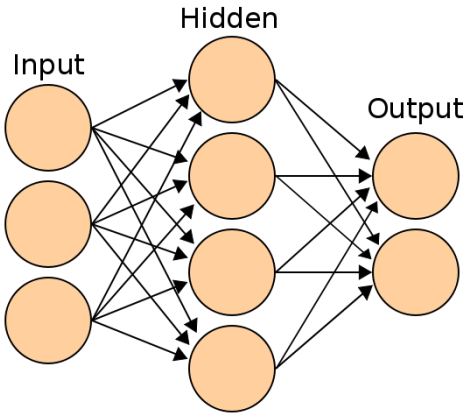
4.1.3. Učenje uz podsticaj

Ovaj oblik učenja podrazumeva da program (agent) deluje na okruženje izvršavanjem niza akcija. Ove akcije utiču na stanje okruženja, koje povratno utiče na agenta pružajući mu povratne informacije koje mogu biti "nagrade" ili "kazne". Cilj agenta je da nauči kako da deluje u datom okruženju tako da vremenom maksimizuje nagrade (ili minimizuje kazne). Primer: kompjuterske igre, autonomna vozila, itd. Učenje uz podsticaj je bihejvioralni model učenja. Algoritam dobija povratnu informaciju na osnovu analize podataka, tako da je korisnik "vođen" ka najboljem ishodu. Razlikuje od drugih metoda, jer sistem nije treniran sa uzorkom skupa podataka, već uči kroz prokušaje i greške. Na taj način, niz uspešnih odluka rezultira da proces biva "reinforced" jer najbolje rešava zadati problem (Mitchell, 1997).

4.2. Neuronske mreže

Kada se govori o neuronskim mrežama misli se na veštačke neuronske mreže (engl. artificial neural networks, ANNs), inspirisane prirodnim neuronskim mrežama koje karakteriše adaptivnost i mogućnost učenja novog znanja. Neuronske mreže se mogu posmatrati kao uprošćeni matematički modeli sistema slični mozgu i oni funkcionišu kao paralelne distribuirane računarske mreže. Međutim, nasuprot konvencionalnim računarima, koji su programirani da obavljaju specificirani zadatak, najveći broj neuronskih mreža se mora obučavati. Ove mreže se mogu obučiti da prepoznaju nove asocijacije, nove funkcionalne zavisnosti i nove oblike - svojstvo generalizacije (Bašić et al., 2008). Procedura kojom se obavlja obučavanje je algoritam obučavanja. Kroz ovu se proceduru se na algoritamski način menjaju sinaptičke težine u cilju dostizanja željenih performansi mreže. Osnovnu računarsku snagu neuronskih mreža čini masivni paralelizam, sposobnost obučavanja i generalizacija. Generalizacija predstavlja sposobnost produkovanja zadovoljavajućeg izlaza neuronske mreže i za ulaze koji nisu bili prisutni u toku obučavanja (Duval, 2018). Neuronske mreže simuliraju način rada ljudskog mozga pri obavljanju datog zadatka ili neke funkcije. Neuronska mreža je masovno paralelizovan distribuirani procesor sa prirodnom sposobnošću memorisanja iskustvenog znanja i obezbedivanja njegovog korišćenja.

Veštačke neuronske mreže zasnovane su na kolekciji veštačkih neurona koji predstavljaju međusobno povezane čvorove i preslikavaju funkcije stvarnih, bioloških neurona. Sinapse u biološkom mozgu prenose signale od jednog do drugog neurona. Ovaj mehanizam je preslikan i na veštačke neurone, tako da veštački neuron prima signale preko međusobnih veza i procesuira ih. U ovom slučaju, signal je obično predstavljen realnim brojem i izlaz svakog neurona je predstavljen kao rezultat/izračunavanje neke nelinearne funkcije (Da Silva et al., 2016). Veze su nekada naznačene kao ivice (engl. edges) koje nose odgovarajuće težinske vrednosti (engl. weight), čiji je cilj da se prilagođavaju i koriguju kako proces učenja napreduje. Tako težinske vrednosti povećavaju ili smanjuju jačinu signala na konekciji. Neuroni mogu imati i prag, tako da je signal poslat samo ako nagomilani signali prelaze odgovarajuci prag (Bašić et al., 2008). Obično su neuroni agregirani u slojeve (slika 4.6). Različiti slojevi mogu izvoditi razne transformacije ulaznih informacija. Signali putuju od prvog sloja (tkzv. input sloj) do poslednjeg sloja (tkzv. output sloj) prelazeći međuslojeve više puta (Duval, 2018).



Slika 4.6. Primer veštačke neuronske mreže pa povezanim čvorovima u različitim slojevima[12].

Neuronska mreža uči (obično kažemo da je trenirana) procesiranjem primera, od kojih svaki sadrži poznate "ulazne podatke" tj. "input" i "rezultat", formirajući asocijacije ponderisane verovatnoćom između ulaza i rezultata. Treniranje neuronske mreže na osnovu datog primera se obično izvodi određivanjem razlike između procesiranog output-a (predikcije) i ciljnog output-a. Ova razlika predstavlja grešku. Mreža tada podešava svoje asocijacije bazirane na težinskim vrednostima prema pravilu učenja i koristeći vrednost greške. Uzastopna podešavanja prouzrokuju da neuronska mreža proizvede output koji je veoma sličan ciljnom output-u. Nakon dovoljnog broja ovih podešavanja, treniranje se završava na osnovu odgovarajućeg kriterijuma. Ovako opisani proces je poznat kao nadgledano učenje (engl. supervised learning). Treba imati u vidu da algoritam koji odredjuje "učenje" (tkzv. learning algoritam) neuronske mreže može biti nadgledani ili nenadgledani (engl. unsupervised). Kao što je već diskutovano u podsekciji 4.1, reći ćemo da je proces učenja neuronske mreže nadgledan, ako je željena izlazna vrednost/informacija unapred poznata. U suprotnom slučaju, ako nam nije poznata željena/ciljana izlazna vrednost, reći ćemo da je proces učenja neuronske mreže nenadgledan.

4.2.1. Organizacija i arhitektura mreže

Neuroni su obicno organizovani u višestruke slojeve. Neuroni jednog sloja su povezani isključivo sa neuronima slojeva koji neposredno prethodi i neposredno sledi tom sloju. Sloj koji prima ulazne podatke je tkzv. input sloj, dok je sloj koji proizvodi konačne rezultate output sloj. Između ova dva sloja može se naći nula ili više skrivenih slojeva. U slučaju visšslojne mreže, između dva sloja raznoliki i višestruki paterni veze su mogući. Na primer, oni mogu biti zasnovani na potpunoj povezanosti tipa svaki sa svakim (engl. fully connected), gde je svaki neuron u jednom sloju povezan sa svakim neuronom u sledećem, susednom sloju. Takođe, mogući patern je tkzv. "pooling", gde je grupa neurona u jednom sloju povezana sa samo jednim neuronom u sledećem sloju. Neuroni sa takvom vrstom konekcije formiraju direktan aciklički graf i poznati su kao acikličke mreže. S druge strane, mreže koje dozvoljavaju konekcije između neurona u istom ili prethodnom sloju su poznate kao rekurentne mrezže (Duval, 2018).

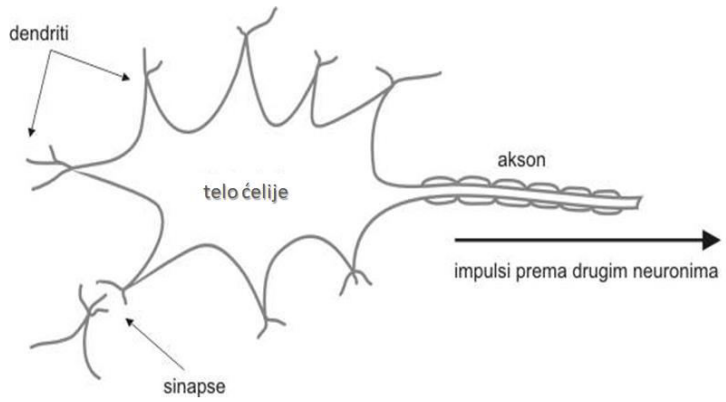
Osnovni elementi obrade u neuralnim mrežama su *neuroni* ili *čvorovi.* Raspored čvorova i veza između čvorova definiše arhitekturu mreže (Bašić et al., 2008). Svaki čvor (procesorska jedinica) okarakterisana je: 1) nekim nivoom aktivnosti koji predstavlja stanje polarizacije neurona; 2) izlaznom vrednošću koja predstavlja aktivaciju neurona; 3) skupom ulaznih veza koje predstavljaju sinapse koje

[12] https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg

vode ka ćeliji i njenim dendritima; 4) vrednošću pomeraja koji predstavlja unutrašnji nivo mirovanja neurona; 5) skupom izlaznih veza koji predstavlja neuronove aksonske projekcije. Svaki od ovih aspekata jedinice predstavljen je matematički realnim brojevima. Na ovaj način, svakoj vezi je pridružena težina (snaga sinapse), koja određuje uticaj dolazećeg ulaza na aktivacioni nivo jedinice. Težine mogu biti pozitivne (eksitacione) i negativne (inhibicione) (Duval, 2018).

4.2.2. Biološki i veštački neuron

Ljudski mozak sastoji se od ogromnog broja neurona (oko 10^{11}) od kojih je svaki povezan sa 10^4 drugih neurona. Neuron se sastoji od ćelijskog tela (soma), skupa dendrita (ogranaka), aksona (veze koje prenose električne poruke) i niza završnih delova (slika 4.6). Funkcionisanje mozga je zasnovano na radu neurona međusobno povezanih prenosnim vezama, aksonima, i prijemnim vezama, dendritima. Ćelijsko telo sadrži informaciju predstavljenu električkim potencijalom (Bašić et al., 2008). Dendriti su pokriveni sinapsama – spojnim sredstvom preko koga se primaju informacije od drugih neurona. Te informacije se u obliku tkzv. post-sinaptičkog potencijala i uticu na potencijal ćelije. Ukoliko ukupni napon – dobijen sumiranjem brojnih post-sinaptičkih potencijala od ogromnog broja susednih neurona - pređe određeni prag, neuron generiše tzv. akcijski potencijal. Dalje, informacija se prenosi akcijskim potencijalom do završnih delova, koji otpuštaju neurotransmitere. Poput talasa (propagacija impulsa je jednosmerna) to inicira gore navedene akcije u daljim neuronima (Duval, 2018; Bašić et al., 2008).

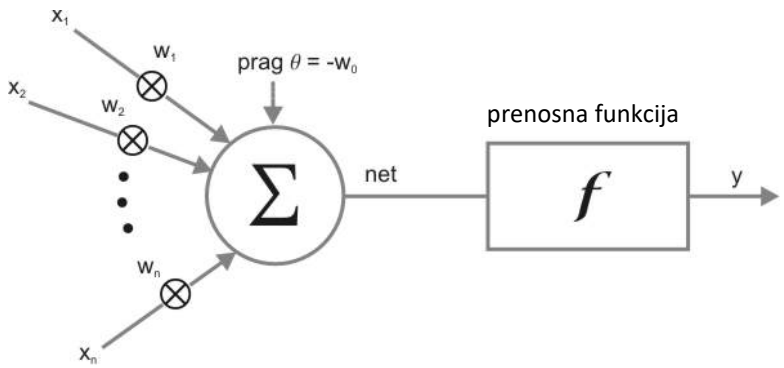


Slika 4.6. Građa neurona (Bašić et al., 2008)

McCulloch-Pitts model veštačkog neurona, tzv. *Threshold Logic Unit* (TLU) imitira funkciju biološkog neurona. U njihovom modelu, svaki neuron je "uključen" ili "isključen". Uključenje neurona nastaje kao odziv kada ga stimuliše dovoljan broj njemu susednih neurona. Stanje neurona se može posmatrati kao ekvivalent iskazu koji tvrdi da je taj neuron dovoljno stimulisan. U neuralnim mrežama ulazi u neuron se kombinuju na linearan način i sa raznim težinama. Rezultat ove kombinacije vodi se na nelinearnu jedinicu aktivacije (funkcija aktivacije), koja u najjednostavnijem obliku može biti pragovska jedinica (engl. treshold unit). Takođe, u opštem slucaju, umesto funkcije praga, vestacki neuron moze imati i neku drugu aktivacijsku funkciju. U terminologiji se veštački neuron često naziva procesna jedinica (Bašić et al., 2008). Obučavanje je zasnovano na dva mehanizma: kreaciji novih veza i modifikaciji veza. Svaki neuron ima nivo aktivacije, koji je u opsegu između nekog minimuma i maksimuma.

Neuralne mreže se mogu koristiti za poboljšavanje i optimizaciju rasplinutih sistema (sekcija 5), dajući im mogućnost obučavanja. Obučavanje rasplinutih sistema se ostvaruje dovodeći razne primere

iz skupa obučavanja na mrežu i koristeći algoritam obučavanja, kojim se menjaju težine u mreži ili parametri funkcije aktivacije tako da mreža daje korektan izlaz za korektne odgovarajuće ulazne vrednosti. Javlja se problem garancije uopštavanja i dovoljne obučenosti mreže. Željeni izlaz neuronske mreže dobija se izborom strukture čvorova mreže i adaptacijom težinskih faktora međusobnih veza na osnovu poznavanja karakteristika odabranog skupa izlaza (Duval, 2018).



Slika 4.7. Veštački neuron (Bašić et al., 2008)

Neka su ulazni signali $x_1,x_2,...x_n$, gde je $x_i \in$ R, i $x_i \in$ [-1,1], [0,1] ili samo elementi iz {0,1}, kada govorimo o Booleovom ulazu. Dalje neka su težine $w_1,w_2,...w_n$. Definisemo težinsku sumu *net* na sledeći način (Bašić et al., 2008):

$$net = w_1x_1 + w_2x_2 + ... + w_nx_n - \theta$$

Dalje, bez umanjenja opštosti pretpostavimo da je vrednost praga $\theta = -w_0$ i da je ulazni signal $x_0 = 1$[13]. Na osnovu toga dobijamo težinsku sumu i prenosnu funkciju:

$$net = w_0x_0 + w_1x_1 + w_2x_2 + ... + w_nx_n = \sum_{i=0}^{n} w_i x_i$$

$$y = f(\sum_{i=0}^{n} w_i x_i) = f(net)$$

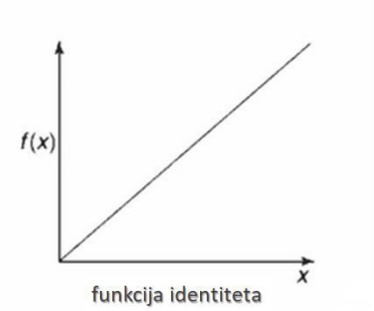
4.2.3. Aktivacijske funkcije

Kategorizacija veštačkih neurona se obavlja prema ugrađenoj prenosnoj funkciji (Duval, 2018; Bašić et al., 2008):

1) Jednostavan oblik aktivacijske funkcije je linearna funkcija identiteta

$$f(x) = x, \forall x$$

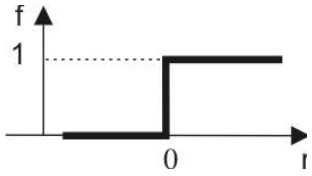
[13] Pomeraj je konstantna vrednost u mreži, čiji uticaj se može videti u izračunavanju *net input*. Naime, uključen je dodajući komponentu $x_0 = 1$ ulaznom vektoru *X*. Pomeraj igra odlučujuću ulogu u određivanju izlazne vrednosti (output) iz mreže. Dalje, pomeraj može biti pozitivan (tada pomaže povećanju net input mreže) ili negativan (tada pomaze smanjenju net input mreže).



funkcija identiteta

2) Drugi oblik aktivacijske funkcije je tkzv. funkcija skoka ili binarna step-funkcija (engl. threshold function, hard-limiter) i tu tom slučaju procesna jedinica daje Booleov izlaz:

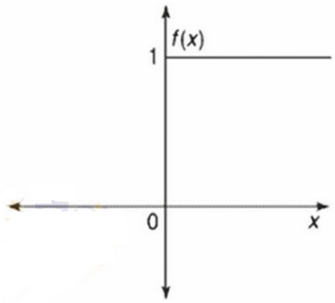
$$f(x) = \begin{cases} 0 & za\ x < 0 \\ 1 & inače \end{cases}$$



Generalizacija prethodnog oblika se moze opisati na sledeći način:

$$f(x) = \begin{cases} 1 & if\ x \geq \Theta \\ 0 & if\ x < \Theta \end{cases}$$

gde je *Θ* vrednost praga (engl. threshold value).

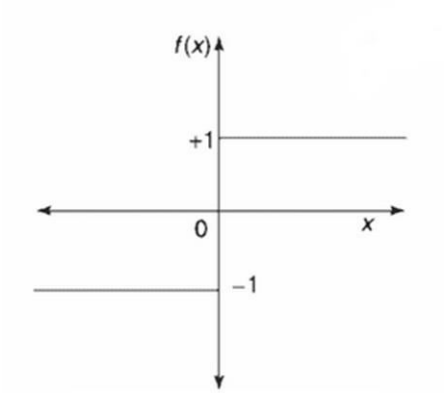


binarna step funkcija

3) Sledeći oblik aktivacijske funkcije je tkzv. bipolarna step-funkcija definisana sa:

$$f(x) = \begin{cases} 1 & if\ x \geq \Theta \\ -1 & if\ x < \Theta \end{cases}$$

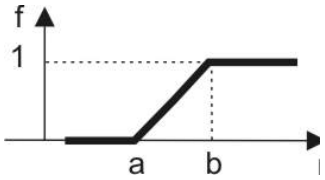
gde je *Θ* vrednost praga (engl. threshold value).



bipolarna step funkcija

4) Sledeći oblik aktivacijske funkcije podrazumeva separatno definisanje linearnih formi:

$$f(x) = \begin{cases} 0 \;\; za\, x \leq a \\ net \;\; za\, a < x < b \\ 1 \;\; za\, x \geq b \end{cases}$$

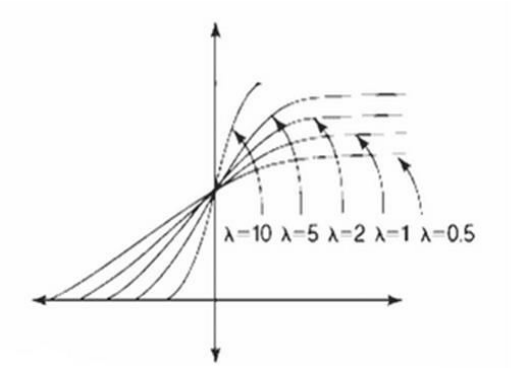


4) Takođe za aktivacijsku funkciju moze se koristiti i forma sigmoidalne funkcije. Razlikujemo dva tipa sigmoidalnih funkcija:

a) Binarna sigmoidalna funkcija (koja se preslikava na vrednosti 0 ili 1, kao što joj i naziv kazuje):

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

gde je λ parameter kosine (engl. steepness).

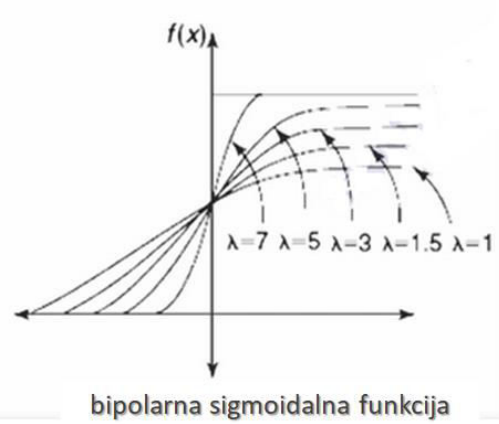


binarna sigmoidalna funkcija

b) Bipolarna sigmoidalna funkcija (njen skup vrednosti je izmedju -1 i 1).

$$f(x) = \frac{2}{1 + e^{-\lambda x}} - 1 = \frac{1 - e^{-\lambda x}}{1 + e^{-\lambda x}}$$

gde je λ parameter kosine.



bipolarna sigmoidalna funkcija

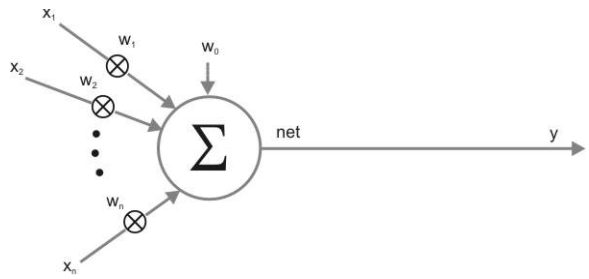
4.2.4. Postupak učenja mreže

Učenje je adaptacija mreže i uključuje podešavanje težinskih vrednosti (i opcionalno vrednosti praga) mreže u cilju poboljšanja tačnosti rezultata minimiziranjem uočenih grešaka (Duval, 2018; Bašić et al., 2008). Zavisno od toga da li nam je u postupku učenja á priori poznat izlaz iz mreže, pa ga pri učenju mreže koristimo uz svaki ulazni primer, ili nam je tačan izlaz nepoznat, razlikujemo dva načina učenja: nadgledano i nenadgledano. Treba imati u vidu da i nakog faze učenja red veličine greške (engl. error rate) uglavnom nije jednak 0. U svakom slučaju, ako je posle učenja stopa greške suviše visoka, mreža bi trebala da se redizajnira. To podrazumeva i (re)definisanje funkcije troška (engl. cost function) koja se evaluira periodično tokom faze učenja. Najčešće funkcija troška uključuje stohastičke komponente, čije vrednosti mogu biti aproksimirane. Izlazne vrednosti su brojevi, tako da kada je greška mala, razlika između izlaza (output) i korektnog rezultata je mala (Duval, 2018; Bašić et al., 2008). Rata učenja (engl. learning rate) definiše veličinu korektivnih koraka koje model uzima da bi se prilagodio i smanjio greške kod svake obzervacije. Visoka rata učenja skraćuje vreme treniranja, ali sa nižom tačnošću, dok niska rata učenja produžava proces, ali sa potencijalno većom tačnošću. Da bi se izbegle oscilacije unutar mreže i da bi se poboljšala rata konvergencije, koristi se adaptivna rata učenja koja se povećava ili umanjuje po potrebi. Težine veza između neurona implicitno sadrže znanje o obradi podataka i one se kroz postupak učenja mreže postepeno adaptiraju. Ta adaptacija traje sve dok izlaz iz mreže ne bude proveren na skupu podataka za testiranje i ocenjen kao zadovoljavajući.

Takođe u fazi učenja razlikujemo pojmove iteracije i epohe. Iteracija je korak u algoritmu postupka za učenje u kojem se odvija podešavanje težinskih faktora, dok je epoha jedno predstavljanje celokupnog skupa za učenje (Bašić et al., 2008).

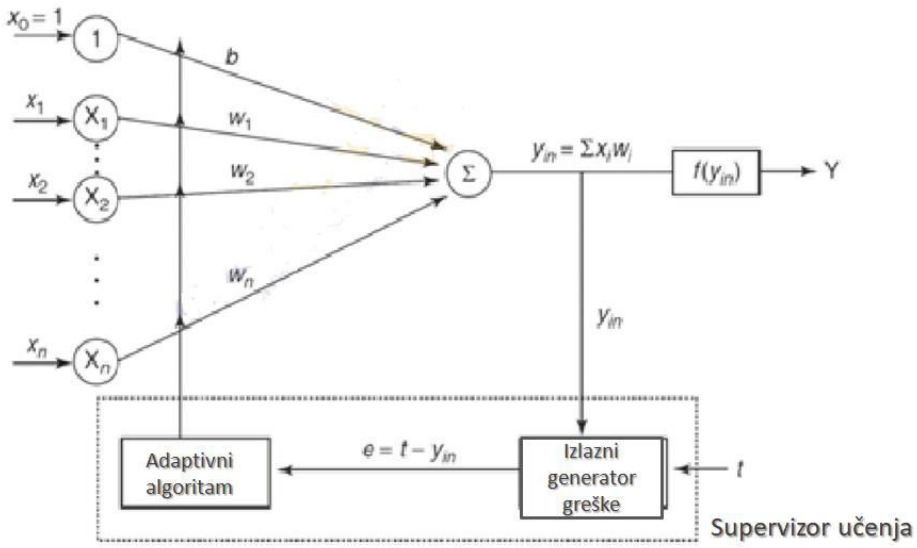
4.2.5. Vrste neurona

Najjednostavniji oblik aktivacijske funkcije implementira tkzv. ADALINE (Adaptive Linear Neuron), gde je *y=net* gde postoji jedan jedini linearni član (slika 4.7).



Slika 4.7. ADALINE procesna jedinica – osnovna struktura (Bašić et al., 2008).

Kako izgledaju osnovne metode učenja veštačkih neurona? Zapravo, podešavanjem težinskih faktora w_i možemo sami odrediti kakav izlaz želimo uz određeni ulaz. ADALINE koristi bipolarnu aktivacionu funkciju. Adaline neuron se moze trenirati koristeci jedno od sledecih pravila: 1) Delta pravilo, 2) Least Mean Square (LMS), ili 3) Widrow-Hoff pravilo (Duval, 2018; Bašić et al., 2008).
Net input se poredi sa ciljnom vrednošću da bi se izračunao signal greške. Dalje na osnovu adaptivnog algoritma za treniranje, težine su podešene. Nešto detaljniji prikaz Adaline procesne jedinice je prikazan na slici 4.8. Može se uočiti da je struktura Adaline slična perceptronu sa jednom dodatnom petljom sa povratnim informacijama pomoću koje je stvarni, realni output upoređen sa željenim output-om. Nakon tog poređenja, na osnovu trenirajuceg algoritma, težine i pomeraj su ažurirani.



Slika 4.8. ADALINE[14]

[14] https://studyglance.in/nn/display.php?tno=5&topic=Adaptive-Linear-Neuron

inicijalizirati težine i pomeraj koji se postavljaju na nenulte slucajne vrednosti
inicijalizirati stopu učenja α

1 raditi korake 2-6 sve dok nije zadovoljen kriterijum zaustavljanja
2 raditi korake 3-5 za svaki bipolarni par treniranja s:t.
3 aktivirati svaku ulaznu jednicu na sledeći način:

$$x_i = s_i \, (i = 1 \, to \, n)$$

4 izračunati net input:

$$y_{in} = \sum_{i}^{n} x_i . w_i + \mathrm{b}$$

gde je b pomeraj i n je ukupan broj ulaznih neurona

5 dok ne bude dobijen najmanji srednji kvadrat $(t - y_{in})$
podesiti težine i pomeraj:

$$w_i(new) = w_i(old) + \alpha(t - y_{in})x_i$$

$$b(new) = b(old) + \alpha(t - y_{in})$$

izračunati grešku: $E = (t - y_{in})^2$

6 testirati kriterijum zaustavljanja: ako je generisana greška manja ili jednaka specificiranoj toleranciji,onda zavrsiti process.

Algoritam 4.1 ADALINE algoritam učenja (Duval, 2018).

4.2.6. TLU perceptron

TLU (Threshold Logic Unit) je procesni element koji koristi step-funkciju (kao nelinearnost) (Duval, 2018). Razlikujemo dva tipa perceptrona: jednoslojni i višeslojni. Jednoslojni perceptron može učiti samo linearno odvojive paterne. Višeslojni perceptron sa dva ili više slojeva ima veću procesnu moć. Njihov rad je predstavljen preko odgovarajućih algoritama koji slede u nastavku.

Inicijalizirati težine i pomeraj (npr., njihove vrednosti mogu biti inicijalno postavljene na 0)
Inicijalizirati ratu učenja α (npr., jednostvnosti radi, vrednost parametra α moze biti postavljena na 1)
1: izvoditi korake 2 - 6 dok kriterijum zaustavljanja ne bude dostignut
2: izvoditi korake 3 - 5 za svaki par treniranja s:t
3: ulazni sloj koji sadrži ulazne jedinice primenjuje aktivacionu funkciju identiteta: $x_i = s_i$
4: izračunati izlaz iz mreže, tj. prvo izračunati net input:

$$y_{in} = \sum_{i}^{n} x_i . w_i + \mathrm{b}$$

gde je n broj ulaznih neurona u ulaznom sloju. Zatim primeniti aktivaciju preko net input da bi se dobila izlazna vrednost

$$f(y_{in}) = \begin{cases} 1 & if \, y_{in} > \theta \\ 0 & if \, -\theta \leqslant y_{in} \leqslant \theta \\ -1 & if \, y_{in} < -\theta \end{cases}$$

5: podešavanja težine i pomeraja: uporediti trenutnu (izračunatu) izlaznu vrednost i željenu (ciljnu) izlaznu vrednost

```
if y ≠ t, then
      w_i(new) = w_i(old)+ αtx_i
      b(new) = b(old)+ αt
else we have
      w_i(new) = w_i(old)
      b(new) = b(old)
```

6: trenirati mrežu sve dok ne bude više promene težinskih vrednosti. Ovo predstavlja kriterujm zaustavljanja. Ako ovaj kriterijum nije dostignut, vratiti se na korak 2.

Algoritam 4.2. Algoritam jednoslojnog perceptrona.

inicijalizirati težine, pomeraj i ratu učenja
1: proveriti kriterijum zaustavljanja; ako nije zadovoljen, izvoditi korake 2–6
2: izvoditi korake 3–5 za svaki bipolarni ili binarni par treniranja s:t
3: za svaku ulaznu jedinicu (i = 1 do n) postaviti aktivacionu funkciju identiteta: $x_i = s_i$
4: izracunati izlaznu vrednost za svaku izlaznu jedinicu (j = 1 do m):
Prvo, net input se izračunava prema formuli:

$$y_{inj} = \sum_{i}^{n} x_i . w_{ij} + b_j$$

Zatim aktivacije su primenjene preko net input za izračunavanje izlaznog odgovora:

$$f(y_{in}) = \begin{cases} 1 & if\ y_{in} > \theta \\ 0 & if\ -\theta \leqslant y_{in} \leqslant \theta \\ -1 & if\ y_{in} < -\theta \end{cases}$$

5: izvršiti podešavanja težine, pomeraj za j = 1 do m i i = 1 do n

```
if y_j ≠ t_j, then
      w_ij(new) = w_ij(old)+ αt_jx_i
      b_j(new) = b_j(old)+ αt_j
else we have
      w_ij(new) = w_ij(old)
      b_j(new) = b_j(old)
```

6: testirati kriterijum zaustavljanja; ako nema više promene težinskih vrednosti, tada zaustaviti process treniranja, u suprotnom vratiti se na korak 2.

Algoritam 4.3. Algoritam višeslojnog perceptrona.

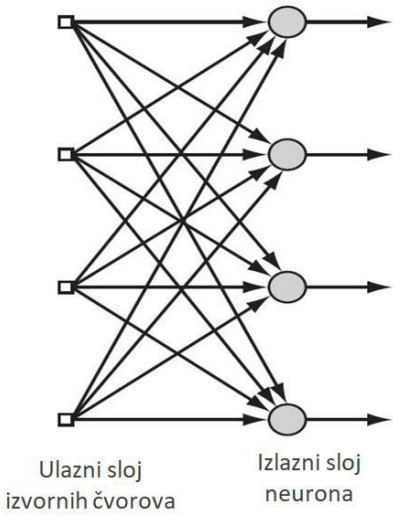
4.2.7. Višeslojne neuronske mreže

Arhitektura mreže je određena načinom na koji su neuroni međusobno povezani i organizovani u mreži. Postoje četiri osnovne arhitekture (Duval, 2018; Bašić et al., 2008):

- Aciklična (engl. feedforward net) mreža,
- Mreža sa povratnom vezom (engl. recurrent net),
- Lateralno povezana mreža (rešetkasta),
- Hibridne mreže.

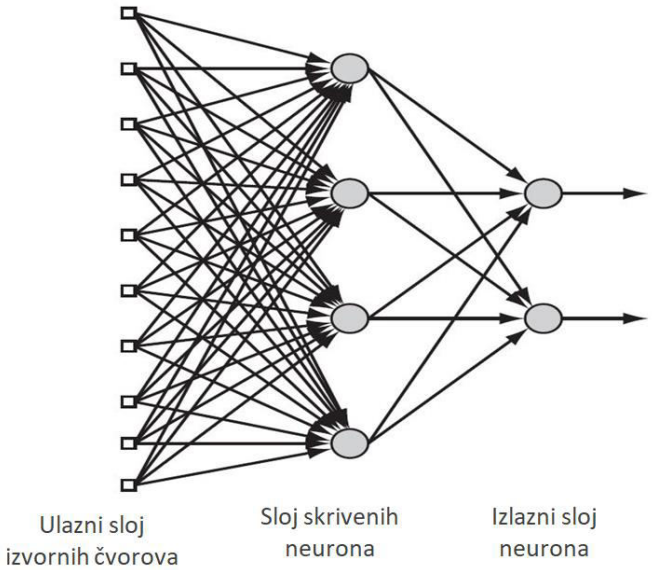
Kod a*ciklične mreže* propagacija signala je jednosmerna, tj. nema povratnih veza između neurona. U tom slucaju, signali koji polaze od ulaznih neurona nakon određenog broja prelaza dolaze do izlaza

mreže. Kod ovakve vrste mreža razlikujemo ulazni sloj neurona, izlazni sloj i skriveni sloj. Neuroni ulaznog sloja nemaju ulaznih signala. Dakle, tu su uglavnom podaci organizovani u vektor konkretnih vrednosti. Struktura mreže obično se zadaje kao *n*-torka $n_1 \times n_2 \times .. \times n_n$ kojom se označava mreža od *n* slojeva kod koje n_1 neurona čini ulazni sloj, n_2 neurona prvi skriveni sloj itd. (Bašić et al., 2008). Kod ove vrste mreže imamo dalje podele na jednostruke aciklične mreže (engl. single-feedforward network) što se može videti na slici 4.9, i višeslojne aciklične mrezu (engl. multilayer feedforward networks) prikazanu na slici 4.10.



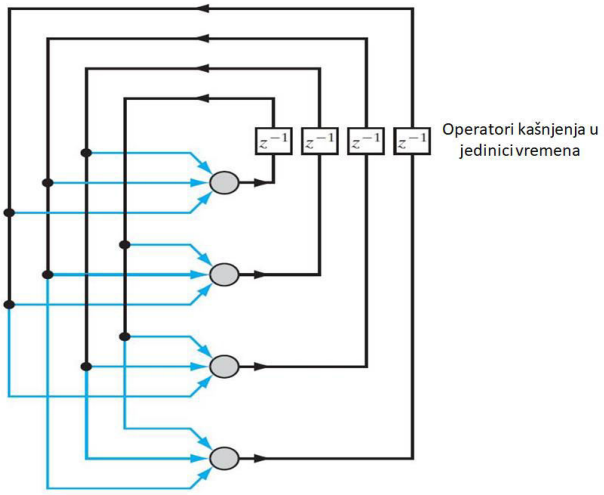
Slika 4.9. Jednostruka aciklična mreža[14].

Kod ovog tipa mreže, postoji ulazni i izlazni sloj. U ulaznom sloju se ne izvodi nikakvo izračunavanje. Izlazni sloj se formira kada su različite težine primenjene na ulazne čvorove i uzet je kumulativni efekat po čvoru. Nakog toga, neuroni zajednički „formiraju“ izlazni sloj da bi izračunali izlazne signale.



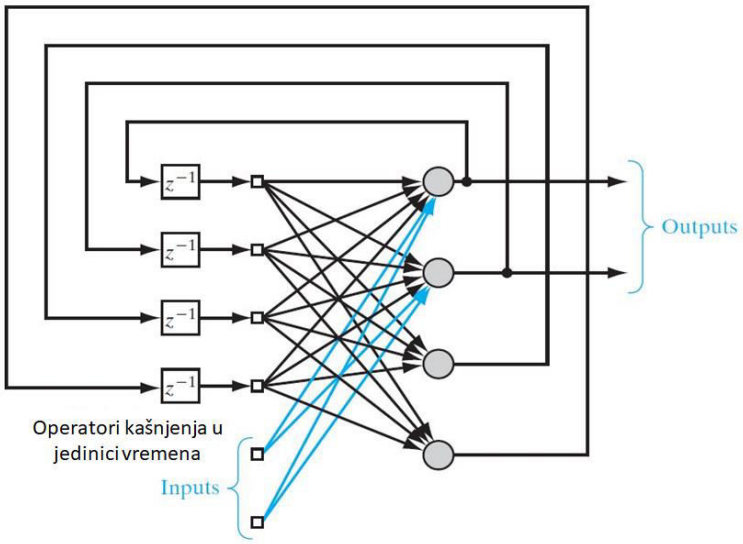
Slika 4.10. Višeslojna aciklična mreža[14].

Ova vrsta mreže ima jedan ili više skrivenih slojeva. Pod pojmom skriveni sloj, podrazumevamo da se ovaj deo neuronske mreze ne “vidi“ direktno niti iz ulaznog dela niti iz izlaznog dela mreze. Funkcija i smisao postojanja “skrivenih neurona“ jeste da intervenišu između ekstenog ulaza i izlaza mreze. Postojanje skrivenih slojeva pojačava kompjutacionu moć mreže.
Neuronske mreže s povratnom vezom sadrže u svojoj strukturi barem jednu povratnu vezu. Kod ovakvih mreza razlikujemo vidljive čvorove, koji interagiraju sa okolinom, i skrivene čvorove.



Slika 4.11. Jednoslojna rekurenta mreža[14].

Rekurentne neuronske mreže se razlikuju of acikličnih po tome što poseduju bar jednu petlju sa povratnom informacijom. Ove mreže su obicno jednoslojne (slika 4.11) sa „feedback“ vezama u kojima izlaz procesnog elementa može biti upućen nazad samom elementu ili prosleđen nekom drugom elementu ili oboje. Kod ovog tipa mreže konekcije između čvorova prave direktnu grafovsku srtukturu.
Za razliku od acikličnih mreža, rekurentne mreže mogu koristiti njihovo interno stanje/memoriju za procesiranje niza ulaznih informacija.

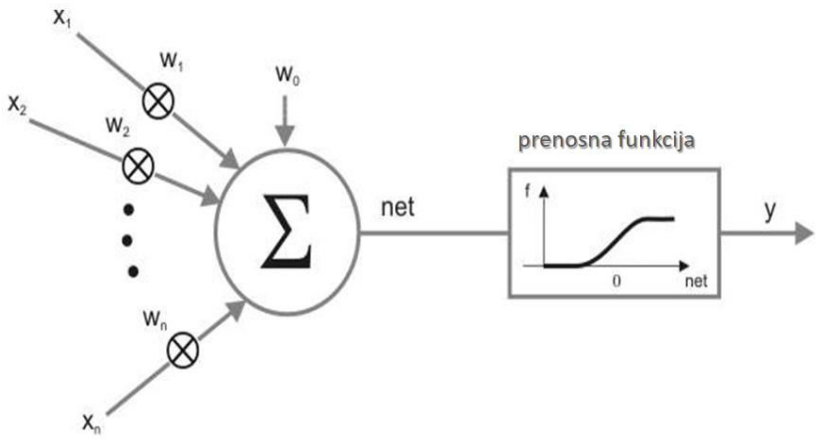


Slika 4.12. Viseslojna rekurenta mreža[14].

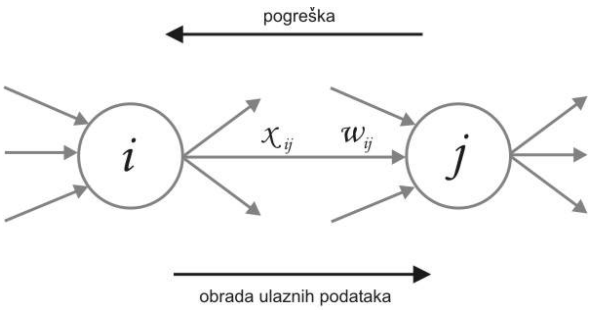
U ovom tipu mreže, izlaz procesnog elementa se može proslediti procesnom elementu istog sloja kao i sloja koji prethodi formirajući višeslojnu rekurentnu mrežu. Kod ovog tipa mreže, obavlja se isti zadatak za svaki element niza, pri čemu izlaz zavisi od prethodnih izračunavanja. Ulazne vrednosti nisu potrebne u svakom vremenskom koraku.

4.2.8. Backpropagation algoritam

Backpropagation, propagiranje unazad, je metod koji se koristi da bi se podesile težinske vrednosti na konekcijama u cilju kompenziranja greške tokom učenja. Veličina greške je efektivno podeljena između veza. Back Propagation Neural (BPN) je prvi put predstavljen 1960-tih godina. Odnosi se na višeslojnu neuronsku mrežu koja se sastoji od ulaznog sloja, najmanje jednog skrivenog sloja i izlaznog sloja. Greška izračunata na izlaznom nivou, poređenjem ciljnog/željenog izlaza sa stvarnim, trenutnim izlazom, biće propagirana unazad ka ulaznom sloju. Ovo je efikasan metod koji izračunava ažurirane težinske vrednosti da bi poboljšao samu mrežu i njeno funkcionisanje sve dok nije u mogućnosti da izvrši zadatak za koji je bila trenirana (Duval, 2018).



Slika 4.13. Sigmoidalna jedinica.



Slika 4.14. Povezani neuroni (Bašić et al., 2008)

Back Propagation Neural koristi binarnu sigmoidalnu aktivacijsku funkciju (slike 4.13 i 4.14). Treniranje ima sledece tri faze:

- faza – feed forward faza
- faza – propagacija unazad greške
- faza – azuriranje težina

Nadalje je predstavljen algoritam Backpropagation.

inicijalizirati težine i pomeraj (zbog jednostavnosti i lakšeg računanja, mogu se uzeti male, slučajne, nenulte vrednosti)
inicijalizirati ratu učenja α
1 izvoditi korake 2-10 dok kriterij zaustavljanja nije zadovoljen
2 nastaviti korake 3-9 za svaki trenirajuci par

1. faza

3 svaka ulazna jedinica prima ulazni signal x_i i šalje ga skrivenoj jedinici za svaki $i = 1$ do n
4 izračunati net input na skrivenoj jedinici na sledeći način:

$$Q_{inj} = \sum_{i=1}^{n} x_i v_{ij} + b_j \quad j = 1 \, to \, p$$

Zatim izračunati net output primenom sledeće sigmoidalne aktivacione funkcije:

$$Q_j = f(Q_{inj})$$

5 izračunati net input na jedinici izlaznog nivoa na sledeći način:

$$y_{ink} = \sum_{j=1}^{p} Q_j \, w_{jk} + b_k \quad k = 1 \, to \, m$$

Zatim izračunati net output primenom sledeće sigmoidalne aktivacione funkcije:

$$y_k = f(y_{ink})$$

2. faza

6 izračunati "korektivnu" gresku (engl. error correcting term) u odnosu na ciljni patern na svakoj izlaznoj jedinici:

$$\delta_k = (t_k - y_k) f'(y_{ink})$$

Izvod $f'(y_{ink})$ moze se izračunati kao:

za slučaj binarne sigmiodalne funkcije:

$$f'(y_{ink}) = f(y_{ink})(1 - f(y_{ink}))$$

za slučaj bipolarne sigmiodalne funkcije:

$$f'(y_{ink}) = (1 + f(y_{ink}))(1 - f(y_{ink}))$$

Zatim, poslati δ_k nazad ka skrivenom sloju
7 svaka skrivena jedinica biće suma njenih delta inputa od izlaznih jedinica:

$$\delta_{inj} = \sum_{k=1}^{m} \delta_k \, w_{jk}$$

Greška se izračunava na sledeći način:

$$\delta_j = \delta_{inj} f'(Q_{inj})$$

3. faza

8 svaka izlazna jedinica y_k (k = 1 do m) ažurira težine i pomeraj na sledeći način:

$$w_{jk}(new) = w_{jk}(old) + \Delta w_{jk}$$
$$b_k(new) = b_k(old) + \Delta b_k$$

$$\Delta w_{jk} = \alpha \delta_k \, Q_j$$
$$\Delta b_{0k} = \alpha \delta_k$$

9 svaka skrivena jedinica q_j (j = 1 do p) ažurira težine i pomeraj na sledeći način

$$v_{ij}(new) = v_{ij}(old) + \Delta v_{ij}$$
$$b_j(new) = b_j(old) + \Delta b_j$$

$$\Delta v_{ij} = \alpha \delta_j x_i$$
$$\Delta b_j = \alpha \delta_j$$

10 proveriti kriterijum zaustavljanja (koji moze biti da je broj epoha dostignut ili ciljni izlaz se poklapa/podudara sa aktuelnim izlazom).

Algoritam 4.4. Backpropagation algoritam

Primer (jednoslojni perceptron)

Output AND (izlaz) je 1 samo ako su oba inputa (u ovom slucaju, x_1 i x_2) vrednosti 1.
Tablica istinisoti za AND funkciju sa bipolarnim ulaznim vrednostima i ciljnim vrednostima (target) je data:

x1	x2	Target
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

1. red

inicijaliziramo w_1, w_2, i b na 0, α=1, i ϑ=0.
dobijamo: $x_1(0)+x_2(0)+0$
prolazeci 1. red AND tablice (x_1=1, x_2=1), dobijamo:
1*0+1*0+0 = 0
y_{in} = 0
$Y = f(y_{in})$ => f(0) => 0
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: 0 ≠ 1, dakle promene u težinama su zahtevane.

w_i(new) = w_i(old)+ $\alpha t x_i$
b(new) = b(old)+ αt

w_1(new) = w_1(old)+ $\alpha t x_1$
= 0+ 1*1*1 = 1
w_2(new) = w_2(old)+ $\alpha t x_2$
= 0+ 1*1*1 = 1
b(new) = 0+ 1*1 = 1

nove ažurirane težine su w_1=w_2=b=1
y_{in} = 1*1 + 1*1 + 1 = 3
$Y = f(y_{in})$ => f(3) => 1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: 1 = 1, dakle promene u težinama nisu zahtevane.

2. red

w_1, w_2, i b su 1, α=1, i ϑ=0.
dobijamo: $x_1(1)+x_2(1)+1$
prolazeci 2.red AND tablice (x_1=1, x_2=-1), dobijamo:
1*1+-1*1+1 = 1
y_{in} = 1
$Y = f(y_{in})$ => f(1) => 1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: 1 ≠ -1, dakle promene u težinama su zahtevane.

w_i(new) = w_i(old)+ $\alpha t x_i$
b(new) = b(old)+ αt

w_1(new) = w_1(old)+ $\alpha t x_1$
= 1+ 1*-1*1 = 0
w_2(new) = w_2(old)+ $\alpha t x_2$
= 1+ 1*-1*-1 = 2
b(new) = 1+ 1*-1 = 0

nove ažurirane težine su w_1=0, w_2= 2, b=0
y_{in} = 1*0 + -1*2 + 0 = -2
$Y = f(y_{in})$ => f(-2) => -1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: -1 = -1, dakle promene u težinama nisu zahtevane.

3. red

w_1=0, w_2= 2, b=0, α=1, i ϑ=0.
dobijamo: $x_1(0)+x_2(2)+0$
prolazeci 3.red AND tablice (x1=-1, x2=1), dobijamo:
-1*0+1*2+0 = 2
y_{in} = 2
$Y = f(y_{in})$ => f(2) => 1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: 1 ≠ -1, dakle promene u težinama su zahtevane.

w_i(new) = w_i(old)+ $\alpha t x_i$
b(new) = b(old)+ αt

w_1(new) = w_1(old)+ $\alpha t x_1$
= 0+ 1*-1*-1 = 1
w_2(new) = w_2(old)+ $\alpha t x_2$
= 2+ 1*-1*1 = 1
b(new) = 0+ 1*-1 = -1

nove ažurirane težine su w_1=1, w_2= 1, b=-1
y_{in} = -1*1 + 1*1 + -1 = -1
$Y = f(y_{in})$ => f(-1) => -1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: -1 = -1, dakle promene u težinama nisu zahtevane.

4. red

w_1=1, w_2= 1, b=-1, α=1, i ϑ=0.
dobijamo: $x_1(1)+x_2(1)+-1$
prolazeci 4.red AND tablice (x_1=-1, x_2=-1), dobijamo:
-1*1+-1*1+(-1) = -3
y_{in} = -3
$Y = f(y_{in})$ => f(-3) => -1
proveriti da li je Y jednak ciljnoj vrednosti t ili ne: -1 = -1, dakle promene u težinama nisu zahtevane.

§5 RASPLINUTI SISTEMI

Da bi definisali pojam rasplinutih sistema, prvo uvodimo pojam rasplinute logike i rasplinutih skupova. Konvencionalna logika zasniva se na jasnim pravilima i počiva na teoriji skupova. Dakle, neki element može da pipada nekom skupu ili da ne pripada. Kod konvencionalne logike su precizno utvrđena pravila, a kod konvencionalnog poimanja skupa jasno određene granice (engl. crisp). Nasuprot tome stoji pojam rasplinute logike (engl. fuzzy), koja je logika zaključivanja nad rasplinutim skupovima. Kod rasplinutih skupova nije precizno definisana pripadnost jednog elementa. Često se ta pripadnost izražava u procentima, a vrednosti se obično skaliraju od 0 do 1. Koristi se u brojnim slučajevima gde je nemoguće prezentovati znanje o sistemu na precizan način, te se koriste neprecizne konstatacije (na primer: Ivana je visoka, napolju je toplo). Stoga, ovakve konstatacije se vezane za subjektivno poimanje situacije/događaja. Posledično, rasplinuti skupovi se definišu stepenom pripadanja, tj. reći ćemo da neki element pripada skupu sa određenim stepenom. Rasplinuti skup se definiše funkcijom pripadanja, koja za svaki skup određuje interval u kojem se rasplinjava i kojim stepenom pripadanja. Rasplinuti skupovi se koriste u modeliranju različitih procesa u kojima nailazimo na neodređenost, neizvesnost, višeznačnost, nekompletnosti, itd., kao i nedovoljno preciznih pojava koje je nemoguće modelirati klasičnim matematičkim alatima ili teorijom verovatnoće (Chen & Pham, 2001).

5.1. OSNOVNI POJMOVI RASPLINUTIH SKPOVA

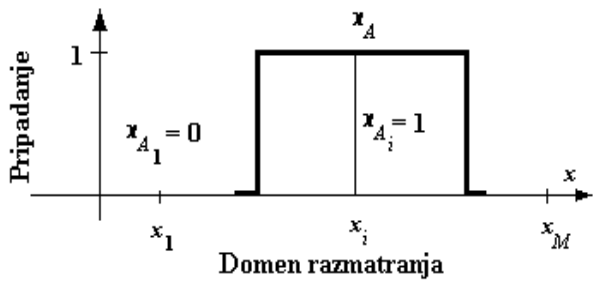
Klasična teorija skupova (engl. crisp set) precizno definiše granicu na osnovu koje tačno znamo da li element pripada skupu ili ne. Za razliku od toga teorija rasplinutih skupova nedovoljno dobro definiše granicu razdvajanja.

5.1.1. Klasična teorija skupova

Podskup *A* skupa *X* definiše se svojom karakterističnom funkcijom χ_A, kao preslikavanje elemenata skupa *X* na elemente skupa {0, 1}:

$$\chi_A : X \rightarrow \{0, 1\}$$

Ovo preslikavanje se može predstaviti skupom uređenih parova, u kome tačno jedan uređeni par odgovara svakom elementu skupa *X.* Prvi element uređenog para je element skupa *X,* a drugi element je element skupa {0,1}. Vrednost 0 drugog elementa se upotrebljava da predstavi nepripadanje, a vrednost 1 se upotrebljava da predstavi pripadanje elementa skupu *X*. Istinitost ili lažnost iskaza „*x* je u *A*“ određena je uređenim parom (x, $\chi_A(x)$). Iskaz je istinit, ako je drugi element uređenog para 1, i iskaz je lažan ako je taj element 0 (Zadeh et al., 1996; Chen & Pham, 2001).



Slika 5.1. Dijagram vrednosti karakteristične funkcije χ_A za neki skup $A \subset X$

5.1.2. Definicija rasplinutog skupa

Rasplinuti podskup *A* skupa *X* može se definisati kao skup uređenih parova, u svakom od kojih je prvi element iz skupa *X*, a drugi element iz intervala [0, 1], pri čemu tačno jedan uređeni par odgovara svakom elementu iz skupa *X.* Ovim je definisano preslikavanje μ_A, elemenata skupa *X* na vrednosti u intervalu [0, 1]:

$$\mu_A : X \rightarrow [0, 1]$$

Vrednost nula se koristi da predstavi potpuno nepripadanje elementa skupu *A*, vrednost jedan se koristi da označi potpuno pripadanje, a vrednosti između nula i jedan se koriste da predstave međustepene pripadanja. Skup *X* mogućih vrednosti posmatrane promenljive *x* naziva se domen razmatranja za rasplinuti podskup *A.* Često se preslikavanje μ_A opisuje kao funkcija pripadanja skupa *A.* Termini funkcija pripadanja i rasplinuti podskup mogu se koristiti jedan umesto drugog. Stepen istinitosti iskaza „*x* je u *A*" (ili „*x* je *A*") određen je uređenim parom $(x, \mu_A(x))$. Stepen istinitosti iskaza je drugi element u uređenom paru (Zadeh et al., 1996; Chen & Pham, 2001).

Definicija 5.1: Neka je $X \neq \emptyset$. Rasplinuti skup *A* u domenu razmatranja *X* je definisan svojom funkcijom pripadanja $\mu_A: X \rightarrow [0, 1]$, pri čemu je $\mu_A(x)$ stepen pripadnosti elemenata *x* skupu *A*[15], za svako $x \in X$.
Prema tome, rasplinuti skup *A* je određen skupom uređenih parova $A = \{(x, \mu_A(x)) \mid x \in X\}$.
Rasplinuti podskupovi realne prave nazivaju se rasplinuti brojevi.

Slika 5.2. Skup *A* i elementi *x*, *y*, *z* i *w*

[15] Što je $\mu_A(x)$ veće, to ima više istine u tvrđenju da element *x* pripada skupu *A*.

Sa slike 5.2. je očigledno da je $\mu_A(x)=1$, $\mu_A(y)=1$, $\mu_A(z)=1$ i $\mu_A(w)=0$.

Funkcija pripadnosti može uzeti i bilo koju drugu vrednost iz intervala [0,1].

Razlikujemo dva slučaja (Chen & Pham, 2001):

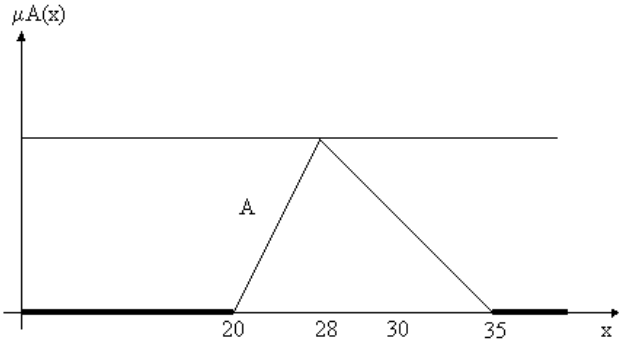
1) Skup $X=\{x_1, x_2, ... ,x_n\}$ je konačan skup elemenata x_i, $i=1,n$. Rasplinuti skup A definisan na skupu X se najčešće prikazuje u obliku:

$$A = \mu_A(x_1)/x_1 + \mu_A(x_2)/x_2 + ... + \mu_A(x_n)/x_n = \sum_{i=1}^{n} [\mu_A(x_i)/x_i]$$

2) Skup X nije konačan. Rasplinuti skup A definisan na skupu X se izražava kao:

$$A = \int x\,[\mu_A(x)/x]$$

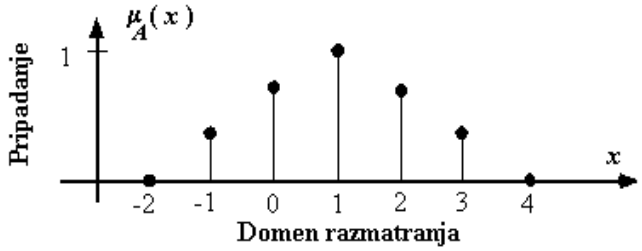
Primer rasplinutog skupa i funkcije pripadnosti prikazan je na slici 5.3:



Slika 5.3. Funkcija pripadnosti $\mu_A(x)$ rasplinutog skupa A

Primer 1. Rasplinuti skup "*blizu broja* 1" u domenu razmatranja (skupu celih brojeva **Z**), prikazan je na slici 5.4 i može se opisati na sledeći način:

$$A = 0.0/(-2) + 0.3/(-1) + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.0/4$$

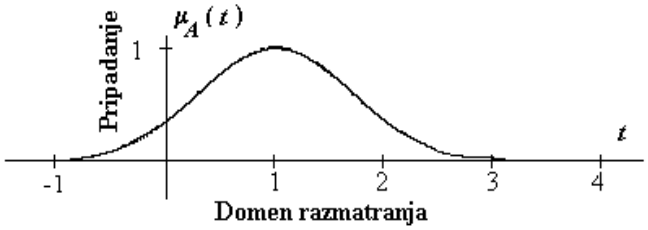


Slika 5.4. Diskretna funkcija pripadanja rasplinutog skupa „blizu broja 1", $x \in \mathbf{Z}$

Primer 2. Rasplinuti skup "*blizu broja* 1" u domenu razmatranja (skupu celih brojeva **R**) prikazan je na slici 5.5 i definiše se svojom funkcijom pripadanja:

$$\mu_A(t) = e^{(-\beta(t-1)^2)}$$

gde je β pozitivan realan broj.

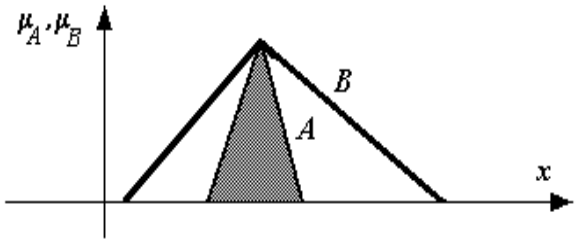


Slika 5.5. Kontinualna funkcija pripadanja rasplinutog skupa „blizu broja 1", $t \in \mathbf{R}$

5.2. OSNOVNE OSOBINE RASPLINUTIH SKUPOVA

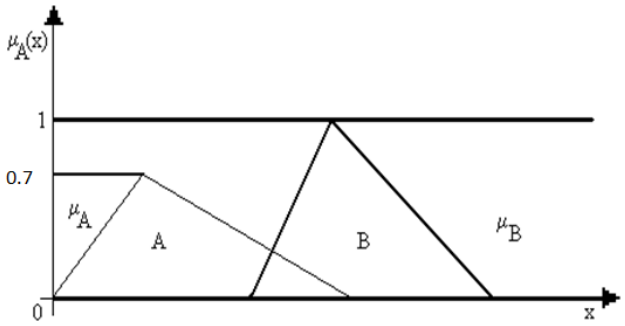
U ovoj sekciji navodimo osnovne osobine rasplinutih skupova: jednakost skupova, definiciju podskupa, te osobine konveksnosti, visine i normalizacije (Zadeh et al., 1996; Chen & Pham, 2001).

1) Rasplinuti skupovi *A* i *B*, definisani na skupu *X*, su *jednaki* (A=B) ako i samo ako važi:
$$\mu_A(x)=\mu_B(x), \text{ za svako } x \in X.$$
2) Rasplinuti skup A je *podskup* rasplinutog skupa B ($A \subset B$) ako i samo ako važi:
$$\mu_A(x) \leq \mu_B(x), \text{ za svako } x \in X.$$



Slika 5.6. Rasplinuti skup *A* je rasplinuti podskup rasplinutog skupa *B*

3) *Visina* rasplinutog skupa predstavlja najveću vrednost stepena pripadnosti. Na slici 5.7 prikazani su rasplinuti skupovi A i B sa visinama 0.7 i 1 respektivno.
4) *Normalizovan rasplinuti* skup ima stepen pripadnosti bar jednog svog elementa jednak jedinici.



Slika 5.7. Funkcija pripadnosti skupa A sa visinom 0.7 i normalizovanog skupa B sa visinom 1

5) Rasplinuti skup A je *konveksan* ako i samo ako važi:

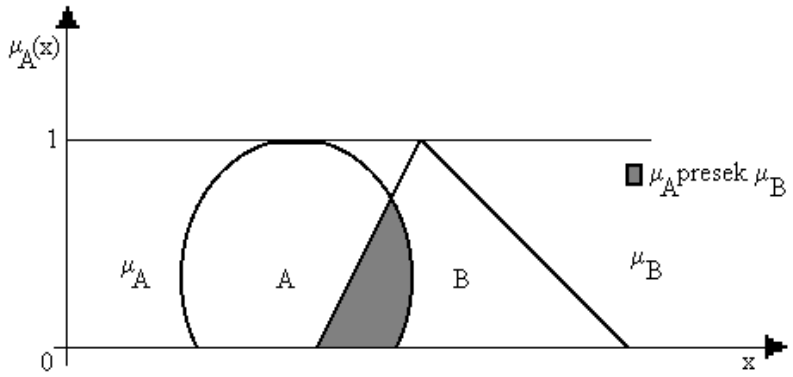
$\mu_A(\lambda x_1 + (1-\lambda)x_2) \geq \mu_A(x_1) \wedge \mu_A(x_2)$, za svako $x_1, x_2 \in X$, i za svako $\lambda \in [0,1]$.

5.3. OPERACIJE NAD RASPLINUTIM SKUPOVIMA

Ova sekcija pokriva definicije osnovnih operacija nad rasplinutim skupovima: presek, uniju, komplement i nacin definisanja Dekartovog proizvoda (Zadeh et al., 1996).

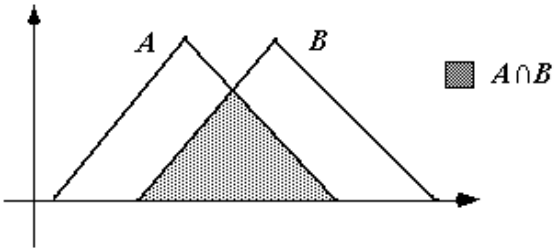
1) *Presek* rasplinutih skupova A i B je najveći rasplinuti skup koji se istovremeno sadrži u A i B:

$$\mu_{A\cap B}(x) = \min\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \wedge \mu_B(x), \forall x \in X$$



Slika 5.8. . Funkcija pripadnosti skupa A i B, i njhov presek

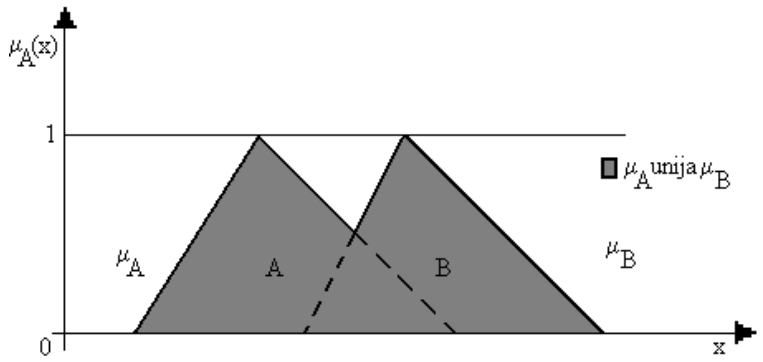
Primer 3. Presek dva trougaona rasplinuta broja



Primer 4. Neka su *A* i *B* rasplinuti podskupovi domena *X* = { -2, -1, 0, 1, 2, 3, 4 }
A = 0.6/(-2) + 0.3/(-1) + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.4/4
B = 0.1/(-2) + 0.3/(-1) + 0.9/0 + 1.0/1 + 1.0/2 + 0.3/3 + 0.2/4
Tada *A* ∩ *B* ima sledeći oblik:
A ∩ *B* = 0.1/(-2) + 0.3/(-1) + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.2/4

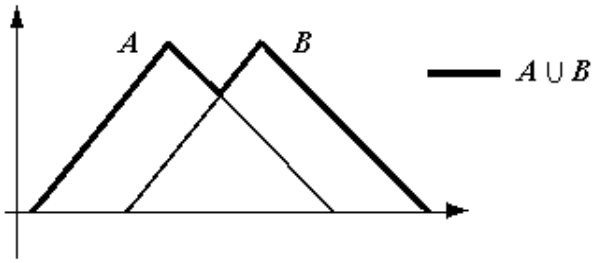
2) *Unija* rasplinutih skupova *A* i *B* je najmanji rasplinuti skup koji istovremeno sadrži i *A* i *B*:

$$\mu_{A\cup B}(x) = \max\{\mu_A(x), \mu_B(x)\} = \mu_A(x) \vee \mu_B(x), \forall x \in X$$



Slika 5.9. Funkcija pripadnosti skupa A i B, i njihova unija

Primer 5. Unija dva trougaona rasplinuta broja



Primer 6. Neka su *A* i *B* rasplinuti podskupovi domena *X* = { -2, -1, 0, 1, 2, 3, 4}:
A = 0.6/(-2) + 0.3/(-1) + 0.6/0 + 1.0/1 + 0.6/2 + 0.3/3 + 0.4/4
B = 0.1/(-2) + 0.3/(-1) + 0.9/0 + 1.0/1 + 1.0/2 + 0.3/3 + 0.2/4 .
Tada *A* ∪ *B* ima sledeći oblik
A ∪ *B* = 0.6/(-2) + 0.3/(-1) + 0.9/0 + 1.0/1 + 1.0/2 + 0.3/3 + 0.4/4

3) *Komplement* fuzzy skupa A je fuzzy skup Ã, takav da je:

$$\mu_{\tilde{A}}(x)=1-\mu_A(x).$$

4) *Dekartov proizvod* rasplinutih skupova $A_1,A_2,...,A_n$ definisanih na skupovima $X_1,X_2,...,X_n$ respektivno, označavamo kao $A_1*A_2...*A_n$, a funkcija pripadnosti se izračunava kao:

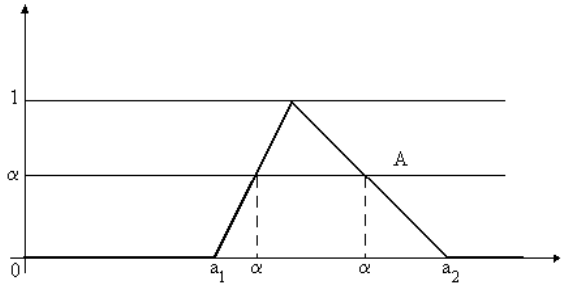
$$\mu_{A_1^* A_2^* ...^* A_n}(X_1,X_2,...,X_n) = \mu_{A_1}(X_1) \wedge \mu_{A_2}(X_2) \wedge ... \wedge \mu_{A_n}(X_n).$$

Rasplinuti skup definisan na *A*x*B*, pri čemu uređeni parovi (*a*,*b*) pripadaju relaciji sa stepenom pripadnosti koji se nalazi u intervalu od 0 do 1, predstavlja rasplinuta *relaciju* između skupova *A* i *B*.

5.4. POJAM RASPLINUTOG BROJA

Pojam rasplinutog broja predstavlja jedan od osnovnih koncepata rasplinutih sistema (Zadeh et al., 1996; Chen & Pham, 2001).

Rasplinuti broj je normalizovan i konveksan rasplinuti skup karakterisan *intervalom poverenja* [a_1, a_2] i *stepenom sigurnosti* α. Na slici 5.10 prikazan je rasplinut broj A i odgovarajući interval poverenja i stepen sigurnosti.



Slika 5.10. Rasplinuti broj, interval poverenja i stepen sigurnosti

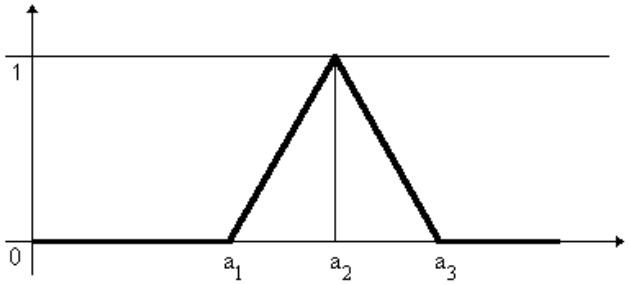
5.4.1. Osnovni oblici rasplinutog broja

U ovoj sekciji nabrojeni su osnovi oblici rasplinutog broja.

Trougaoni rasplinuti broj (slika 5.11) zavisi od funkcije pripadnosti i definisan je sledećim oblikom:

$$A=(a_1, a_2, a_3),$$

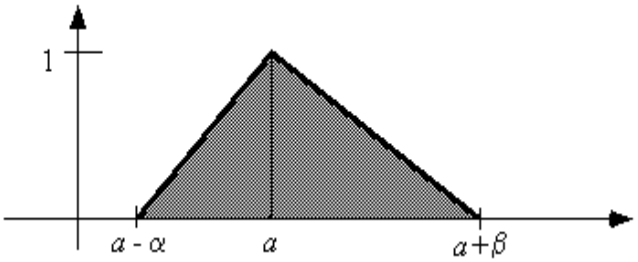
gde je a_1 - donja (leva) granica rasplinutog broja, a_2 - vrednost rasplinutog broja sa najvećim stepenom pripadnosti, i a_3 - gornja (desna) granica rasplinutog broja.



Slika 5.11. Trougaoni rasplinuti broj *A*

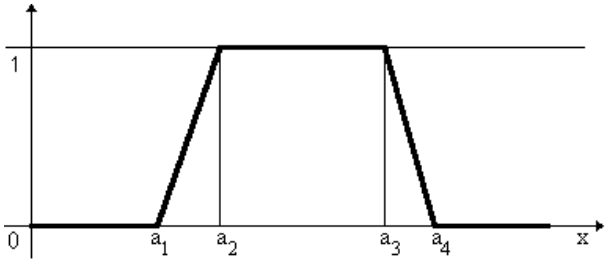
Definicija 5.2: Rasplinuti skup *A*, čiji je domen razmatranja skup **R**, naziva se trougaonim rasplinutim brojem sa centrom *a,* levom širinom $\alpha > 0$ i desnom širinom $\beta > 0$, ako njegova funkcija pripadanja ima oblik dat izrazom:

$$\mu_A(x)=\begin{cases}1-\left[(a-x)/\alpha\right] & ako\ \ je\ \ a-\alpha\le x\le a,\\ 1-\left[(x-a)/\beta\right] & ako\ \ je\ a\le x\le a+\beta,\\ 0 & u\ svim\ ostalim\ slucajevim\ a.\end{cases}$$



Slika 5.12. Primer trougaonog rasplinutog broj

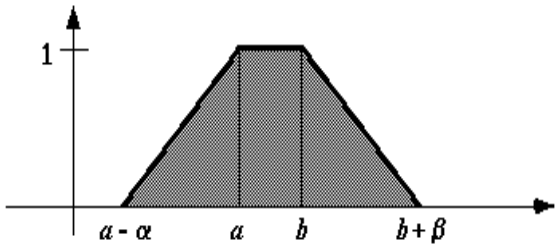
Drugu klasu čine *trapezoidni* rasplinuti brojevi, definisani na sledeći nacin: $A=(a_1, a_2, a_3, a_4)$, sa oblikom prikazanim na slici 5.13.



Slika 5.13. Trapezoidni rasplinuti broj A

Definicija 5.3: Rasplinuti skup *A*, čiji domen razmatranja je skup **R,** naziva se trapezastim rasplinutim brojem sa intervalom tolerancije [*a, b*], levom širinom $\alpha > 0$ i desnom širinom $\beta > 0$, ako njegova funkcija pripadanja ima sledeći oblik:

$$\mu_A(x)=\begin{cases}1-(a-x)/\alpha & ako\ \ je\ \ a-\alpha\le x\le a,\\ 1 & ako\ \ je\ \ a\le x\le b,\\ 1-(x-b)/\beta & ako\ \ je\ \ b\le x\le b+\beta,\\ 0 & x\le a-\alpha,\quad .x\ge b+\beta\end{cases}$$



Slika 5. 14. Primer trapezastog rasplinutog broja

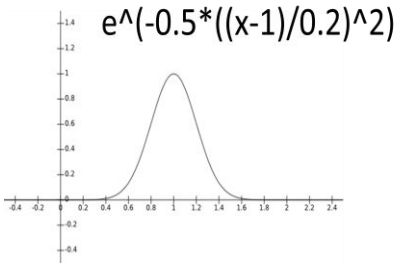
5.4.2. Drugi rasplinuti brojevi

U ovom delu nabrojaćemo varijante rasplinutih brojeva zavisno od njihove funkcije pripadanja.

1) *Gausovska* funkcija pripadanja

$$\mu_A(x) = e^{-\frac{1}{2}\left(\frac{x-c}{\sigma}\right)^2}$$

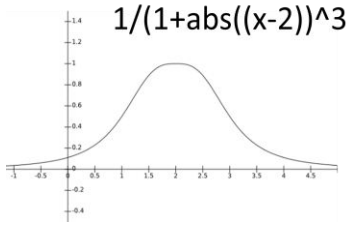
gde su parametri σ i *c* su pozitivni realni brojevi.



2) *Uopštena zvonasta* funkcija pripadanja

$$\mu_A(x) = \frac{1}{1 + \left|\frac{x - c}{a}\right|^{2b}}$$

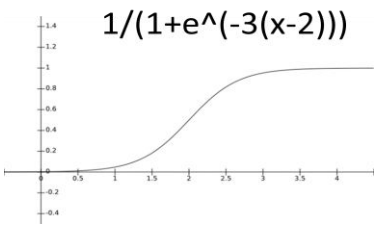
gde su *a*, *b*, *c* su parametri, *b* je obično pozitivan broj.



3) *Sigmoidalna* funkcija pripadanja

$$\mu_A(x) = \frac{1}{1 + e^{[-a(x-c)]}}$$

gde parametar *a* utiče na nagib u srednjoj tački *x* = *c*.



Nad rasplinutim brojevima su definisane i *osnovne operacije,* tako da za rasplinute brojeve sa stepenom uverenosti α: $X\alpha=[x_1\alpha, x_2\alpha]$, i $Y\alpha= [y_1\alpha, y_2\alpha]$ važi:

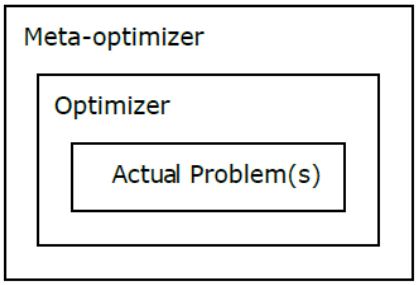
$$X\alpha(\bullet)Y\alpha=[x_1\alpha \bullet y_1\alpha,\ x_2\alpha \bullet y_2\alpha]$$

gde $\bullet$ predstavlja simbol jedne od 4 osnovne računske operacije: sabiranje, oduzimanje, množenje ili deljenje tj. $\bullet \in \{ +, -, *, / \}$.

TREĆI DEO
DODATNE INFORMACIJE

§6 PODEŠAVANJE PARAMETARA

Generalno većinu metaheuristika karakteriše veliki broj parametara u igri. Pošto govorimo o nedeterminističkim algoritmima, selekcija odgovaraćujih vrednosti svih tih parametara je od krucijalnog značaja za efikasnost metaheuristike (Šešum–Čavić, 2020). To je naročito izraženo kod metaheuristika koje su zasnovane na inteligenciji roja. Čak i u slučaju kada se rešava odgovarajući, specificirani problem, selekcija vrednosti parametara je kompjutaciono skupa i delikatna pošto je neophodno odrediti performansu za svaku kombinaciju vrednost parametara. Naime, ako algoritam A nadmaši u performansi algoritam B, to treba da bude posledica činjenice da je algoritam A fundamentalno superiorniji, a ne usled slučajno uspešne selekcije vrednosti parametara. Da bi se eliminisala ta vrsta neizvesnosti, značajan istraživacki napor je urađen i različiti metodi su predloženi i razvijeni, počevši od statističkih procedura (Montgomery, 2012) do automatskog podešavanja vrednosti parametara, npr. meta-evolucije (Hansen, 2006), sekvencijalne parametarske optimizacije (Bartz-Beielstein et al., 2005), estimacije distribucije (Nannen & Eiben, 2007), nadmetanja (engl. racing), (Lopez-Ibanez et al., 2011), izoštravanja(engl. sharpening) (Smit and Eiben, 2009), “adaptive capping” tehnike (Hutter et al., 2009).
Dakle, osnovni smisao podešavanja parametara jeste povećanje performanse samog algoritma. Efikasna inizijalizacija parametara je upravo podešavanje parametara ili tkzv. meta-optimizacija. To takođe ima za posledicu efikasnije poređenje algoritama.
Podešavanje parametara je na neki način slično tkzv. black-box optimizaciji funkcija (Rudolf, 2016). Naime, precizan gradijent optimzacionog problema je nepoznat i fitnes funkcija daje samo neku vrstu mere performanse kandidata za rešenje. Mi možda poznajemo neka ograničenja vezano za dimenzije svakog parametra, ali ne znamo sve međusobne zavisnosti koje odgovarajuće dimenzije nameću jedna drugoj i rezultirajući prostor vrednosti. Možemo reći da osnovni pogled na problem ima sledeću strukturu (slika 6.1) (Pedersen, 2010):



Slika 6.1. Koncept podešavanja parametara (Rudolf, 2016)

Koncept podešavanja parametara se može zamisliti kao black-box optimizer korišćen da bi se ulazni parametri, tj. njihove vrednosti postavile, podesile i ocenile generalno, i za druge optimizacione probleme.
Meta-optimizer traga za optimalnim skupom parametara u slučaju datog ili izabranog optimizacionog metoda. Sam metod podešavanja parametara nema podataka (znanja) o datom problemu koji se rešava.Jedina važna informacija je domen željenih parametara i krijerijum zaustavljanja. Meta-optimizacija je kompjutaciono skupa i zahtevna, jer optimizer se pokreće i izvršava za svaku kandidat-konfiguraciju. Usled izražene kompleksnosti problema i velikog vremena izračunavanja koje zahteva, meta-optimizeri su često limitirani vremenski ili prema maksimalnom broju kandidat-konfiguracija

(Rudolf, 2016). U nastavku su kratko nabrojane neke od najpoznatijih metoda za podešavanje vrednosti parametara.

Meta-Evolucija
Drugi evolutivni algoritam je "iznad" osnovnog algoritma. Populacija jedinki su vektori vrednosti parametara koje se kasnije koriste za glavnu optimizaciju. Dakle, pomoćni evolutivni algoritam određuje skup najboljih vrednosti parametara (Mercer & Sampson, 1978).

Sekvencijalna parametarska optimizacija
U ovom slučaju takođe postoje neke sličnosti sa meta-evolucionim algoritmom. Naime, populacija konfiguracija je evaluirana i na osnovu te evaluacije, regresioni model koji izražava zavisnost parametarske korisnosti je kreiran. Taj trenutni model je nadalje korišćen za testiranje i filterovanje novih konfiguracija (Bartz-Beielstein et al., 2005).

Estimacija distribucije
Cilj ovog metoda je formiranje skupa distribucija parametara koje predstavljaju područje korisnosti. Kandidati konfiguracije su tada izabrane prema svakoj distribuciji (Nannen & Eiben, 2006; Nannen & Eiben, 2007).

Nadmetanje (Racing)
Ovaj metoda se fokusira isključivo na konfiguracije sa dobrom performansom uništavajući one koje čija performansa nije dovoljno dobra. Svaka nova konfiguracija je pokrenuta na malom podskupu test instanci i ukoliko nije značajno gora od do sada najbolje pronađene, ona biva birana za dalja pokretanja. Statistički testovi (kao npr. Friedman-ov test) se obično koriste za poređenje kandidat-konfiguracija (López-Ibáñez et al., 2011).

Izoštravanje (Sharpening)
Obećavajuće konfiguracije parametara su testirane temeljitije i detaljnije od onih koje nisu tako perspektivne. Počinje se sa malim brojem testova i kada je određeni prag dostignut, duplira se broj testova (Smit & Eiben, 2009).

"Adaptive Capping" tehnika
Postoje dva različita pristupa: održavanje trajektorije i agresivnaadaptive capping tehnika (Huter et al., 2009).
U slučaju održavanja trajektorije, svaka nova konfiguracija ima svoj "budžet za izračunavanje" postavljen od strane prethodnog najboljeg pronađenog kandidata. Ako kandidat-konfiguracija nije jednako dobra ili bolja od prethodne, pretraga se nastavlja sa drugim postavljenjem parametara. Svaka iteracija počinje sa inicijalnom konfiguracijom – rešenjem i postaje sve "zahtevnija" da bi se približili boljim rešenjima.
U slučaju agresivnog adaptive capping, kompjutacioni limit zavisi od ukupno najbolje pronađenog rešenja - performanse pomnoženo sa koeficijentomα kojise adaptivno podešava tokom rada algoritma.

§7 NEKE PRIMENE U GEOINFORMATICI

U ovom poglavlju su navedeni primeri primena metoda CI u geoinformatici. Treba napomenuti da neki aplikativni scenariji su više vezani za GIS. Naime, geoinformatika i Geografski Informacioni Sistem (GIS) su međusobno povezane, ali različite oblasti. Geoinformatika je, u svakom slučaju, šira naučna oblast koja obuhvata GIS, kao i druge tehnologije (npr., daljinska detekcija, sistemi globalnog pozicioniranja (GPS) i digitalna kartografija). Takođe, uključuje elemente računarske nauke, statistike i informacione tehnologije za upravljanje i analizu geografskih podataka. Dakle, GIS je alat koji se koristi u okviru geoinformatike, ali geoinformatika je šira oblast koja obuhvata GIS kao i druge tehnologije i discipline.

Nabrojeni primeri su dokaz uspešne primene metoda CI (samostalnih, hibridnih ili kombinovanjem dve metode). Takođe mogu da posluže i kao "startna pozicija" za dalje razmatranje istih i evenutalno usavršavanje dobijenih rezultata.

Primene neuronskih mreža

- (Zhong et al. 2019) razmatra problem klasifikacije letnjih useva koristeći vremenske serije (tj., ulazni podaci su vremenske serije jednog indeksa vegetacije). Primenjene su duboke neuronske mreže sa različitim arhitekturama za klasifikaciju.

- (Pijanowski et al., 2002) razvija model transformacije zemljišta, koji povezuje GIS sa veštačkim neuronskim mrežama za predviđanje promena korišćenja zemljišta. Različiti društveni, politički i ekološki faktori doprinose modelskim prediktorskim varijablama promene korišćenja zemljišta.

- (Li & Gar-On Yeh, 2002) istražuje simulaciju evolucije višestruke upotrebe zemljišta korišćenjem nove metode zasnovane na integraciji neuronskih mreža i ćelijskih automata (engl. cellular automata, CA). Simulacija višestrukih promena korišćenja zemljišta je zahtevna, jer se moraju koristiti brojne prostorne varijable i parametric.

Primene inteligencije bazirane na roju (swarm-based intelligence)

- (Liu et al., 2021) istražuju predviđanje dubine karbonacije za beton od recikliranog agregata modelima mašinskog učenja. Uporebljene su veštačke neuronske mreže (ANN). Svi ANN modeli hibridizovani sa algoritmima inteligencije roja nadmašuju samostalni ANN model. Algoritmi inteligencije roja su korišćeni za podešavanje i optimizaciju parametara u ANN.

- (Bui et al., 2018) verifikuju hibridni inteligentni model za predviđanje koeficijenta kompresije tla (Cc), ključnog parametra u fazi projektovanja građevinskih objekata. Predloženi model je kombinacija metaheurističke optimizacije (PSO) i algoritma MLP neuronskih mreža. MLP neuronske mreže se koriste za konstruisanje funkcije mapiranja koja zaključuje vrednost Cc iz skupa faktora uticaja, dok se PSO metaheuristika koristi za optimizaciju parametara neuronskih mreža.

Primene genetskih algoritama

- (Li & Gar-On Yeh, 2005) predstavljaju kako segenetski algoritmi mogu koristiti sa geografskim informacionim sistemima (GIS) za efikasno rešavanje problema prostornog odlučivanja za optimalno postavljanje *n* lokacija u objektu. Detaljni podaci o populaciji i transportu iz GIS-a se koriste da bi se olakšalo izračunavanje fitnes funkcija. Višestruki ciljevi planiranja su takođe uključeni u GA program.

- (van Dijk et al., 2002) detektuju okvir za GA koji je sposoban da reši određene probleme optimizacije sa kojima se susreću u geografskim informacionim sistemima (GIS). Okvir je posebno pogodan za geografske probleme jer je u stanju da iskoristi njihovu geometrijsku strukturu pomoću novog operatora koji se zove geometrijski lokalni optimizator. Tri takva problema su predstavljena kao studije slučaja: označavanje mape, generalizacija uz očuvanje strukture i pojednostavljenje linija.

Primene rasplinutih sistema

- (Ghorbanzadeh et al., 2020) procenjuju prediktivne performanse adaptivnog neuro-fuzzy sistema zaključivanja sa šest različitih funkcija pripadnosti. Koristeći geografski informacioni sistem (GIS), primenjuju navedeni sistem na mapiranje podložnosti sleganju zemljišta u oblasti istraživanja.

- (Arefiev et al, 2015) razvijaju metod višestruke procene zemljišnih parcela na osnovu teorije rasplinutih skupova integrisane u geografske informacione sisteme (GIS). Izvedeni su numerički eksperimenti za primenu metode u urbanističkom planiranju. Višestruke procene zemljišnih parcela su izračunate i mapirane. Složena vrednovanja zemljišnih parcela vršena su rasplinutim modelom uz korišćenje četiri kriterijuma (tehnološki, ekonomski, ekološki i socijalni) za dva posebna zadatka: 1) ekološki kriterijum je važniji i 2) ekonomski kriterijum je značajniji.

§8 ZAKLJUČAK

Kompjutaciona inteligencija se izdvaja kao jedna od izrazito ekspanzivnih oblasti sa brzim razvojem i brojnim primenama u širokom spektru aplikacija iz realnog života. Spada u podoblast veštačke inteligencije. Njeni metodi su delotvorni u situacijama u kojima ima puno izazova, u okruženjima gde je dinamika veoma izražena, kod izrazito kompleksnih problema. Zbog svega toga, klasično matematičko modeliranje često ne daje adekvatne rezultate. Stoga metode kompjutacione inteligencije su se pokazale kao obećavajuce i uspešne u rešavanju takvih problema.

U knjizi su obrađeni osnovni postulati kompjutacione inteligencije: evolutivni algoritmi (izabrani su genetski algoritmi), metaheuristike bazirane na inteligenciji roja, veštačke neuronske mreže i rasplinuti sistemi.

Metaheuristike bazirane na inteligenciji roja su stekle veliku popularnost u poslednjih par decenija. Pored osobina koje su svojstvene i za ostale prirodno-bazirane metaheuristike (kao sto su npr. evolutivni algoritmi) – robustnost, adaptibilnost, itd., ova vrsta metaheuristika obezbeđuje niz tkzv. “self-*” svojstava (samo-organizacija, samo-učenje, samo-korekcija, ...). Prirodno-bazirane metaheuristike su od izuzetne važnosti, tim pre što su dobijeni rezultati njihove primene na razne aplikativne scenarije obećavajući. Zajedno sa genetskim algoritmima, spadaju u populaciono bazirane algoritme. Nastali prvobitno kao problemski-orjentisani, tokom vremena su razvijene varijante za različite tipove optimizacionih problema. Opšta karakteristka u procesu razvoja ovih metaheuristika je uspešna primena na raznoliku listu optimizacionih problema, ali u velikom broju slučajeva još uvek otvorena brojna teorijska pitanja, naročito pitanja vezano za analizu konvergencije.

Veštačke neuronske mreže poseduju brojne prednosti medju kojima se izdvajaju: nelinearnost, koja je u osnovi distribuirana; ulazno-izlazno preslikavanje, koje se restauriše kroz proces obučavanja; adaptivnost - sposobnost menjanja jačine sinaptičkih veza; kontekstualna informacija - svaki neuron u neuronskoj mreži je pod uticajem globalne aktivnosti ostalih neurona; otpornost na otkaz; adaptivnost - neuronske mreže mogu automatski podesiti svoje parametre (težine) da bi optimizovali svoje ponašanje kao sistemi raspoznavanja oblika, sistemi odlučivanja, regulatori sistema, sistemi predviđanja, itd.

Rasplinuti sistemi se baziraju na osnovnim pojmovima rasplinutih skupova i rasplinute logike. Rasplinuti skupovi se koriste u modeliranju različitih procesa u kojima nailazimo na neodređenost, neizvesnost, višeznačnost, nekompletnosti, itd., kao i nedovoljno preciznih pojava koje je nemoguće modelirati klasičnim matematičkim alatima ili teorijom verovatnoće.

Ova knjiga pruža pregled osnovnih karakteristika najpopularnijih metoda kompjutacione inteligencije, počevši od samog biološkog mehanizma, preko načina na koji je izvedeno matematičko modeliranje, nastanka algoritma, kao i osnovnih osobina samog algoritma. Pored toga, na kraju svakog poglavlja nalaze se primeri primene metoda. U poslednjem, trecem delu (poglavlje 8) knjiga pruža smernice i izvore za dalje proučavanje predstavljenih algoritama, kao i njihovu primenu na konkretne probleme iz geoinformatike.

BIBLIOGRAFIJA

Alam, T., Qamar, S., Dixit, A., & Benaida, M. Genetic Algorithm: Reviews, Implementations, and Applications. CompSciRN: Computer Principles (Topic), 2020.

Arefiev N., Terleev V., BadenkoV. GIS-based Fuzzy Method for Urban Planning, Procedia Engineering, 117:39-44, 2015.

Barth F. Insects and flowers: the biology of a partnership. Princeton University Press, Princeton, New Jersey, 1982.

Bartz-Beielstein T, Lasarczyk C. W. G, Preuss M. Sequential parameter optimization, IEEE Congress on Evolutionary Computation, vol.1, 773–780, 2005.

Bašić B. D., Čupić M., Šnajder J. Umjetne neuronske mreže, Fakultet elektrotehnike i računarstva, Zagreb, 2008

Beasley, D., Bull, D. R. and Martin, R. R. An overview of genetic algorithms: Part 2, research topics, University Computing , 15:170-181, 1993.

Bezdek J.C. Computational Intelligence Defined - By Everyone !. In: Kaynak, O., Zadeh, L.A., Türkşen, B., Rudas, I.J. (eds) Computational Intelligence: Soft Computing and Fuzzy-Neuro Integration with Applications. NATO ASI Series, vol 162. Springer, Berlin, Heidelberg, 1998.

Blass A., Gurevich Y., Algorithms: A Quest for Absolute Definition, Bulletin of European Association for Theoretical Computer Science 81, 2003.

Blum C, Roli A. Metaheuristics in Combinatorial Optimisation: Overview and Conceptural Comparision, ACM Comput. Surv. 35:268-308, 2003.

Bui D. T., Nhu V-H., Hoang N-D. Prediction of soil compression coefficient for urban housing project using novel integration machine learning approach of swarm intelligence and Multi-layer Perceptron Neural Network, Advanced Engineering Informatics, 38:593-604, 2018.

Camazine S., Deneubourg J., Franks N.R., Sneyd J., Theraulaz G. and Bonabeau E. Self-Organization in Biological Systems, Princeton University Press, 2003.

Chen G., Pham T.T. Introduction to Fuzzy Sets, Fuzzy Logic, and Fuzzy Control Systems, CRC Press, 2001.

Cormen T. H., Leireson C.E., Rivest R.L., Stein C. Introduction to Algorithms, The MIT Press, 2009.

Da Silva, I.N. and Spatti, D.H. and Flauzino, R.A. and Liboni, L.H.B. and dos Reis Alves, S.F. Artificial Neural Networks: A Practical Course, Springer International Publishing, 2016

Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a Colony of Cooperating Agents. IEEE Trans. Systems, Man, and Cybernetics, Part B 26(1): 29-41, 1996.

Dorigo M, Stützle T. Ant Colony Optimization, MIT Press, 2004.

Drias H, Sadeg S, Yahi S. Cooperative bees swarm for solving the maximum weighted satisfiability problem, Computational Intelligence and Bioinspired Systems, LNCS, 3512:318–325, 2005.

Duval F. Artificial Neural Networks. CreateSpace Independent Publishing Platform, 2018

Eiben A.E., Smith, J.E. Evolutionary Computing: The Origins. In: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Berlin, Heidelberg, 2015.

Engelbrecht A. Computational Intelligence: An Introduction, Wiley, 2002.

Floyd R.W. Nondeterministic Algorithms, Journal of the ACM, vol .14 (4), 636-644, 1967.

Gaubert L, Redou P, Harrouet F, et al. A First Mathematical Model of Brood Sorting by Ants: Functional Self-Organization without Swarm-Intelligence, Ecological Complexity, 4(4):234-241, 2007.

Ghorbanzadeh O., Blaschke T., Aryal J., Gholaminia K. A new GIS-based technique using an adaptive neuro-fuzzy inference system for land subsidence susceptibility mapping, Journal of Spatial Science, 65:3, 401-418, 2020.

Goldberg, D. E. Genetic Algorithms in search, optimization and machine learning, Addison-Wesley, Massachusetts, 1989.

Goodrich M., Algorithm Design: Foundation, Analysis and Internet Examples, Wiley, 2001.

Hansen N. The CMA Evolution Strategy: A Comparing Review, Springer, 75-102, 2006.

Heylighen F. The Science of Self-Organization and Adaptivity, The Encyclopedia of Life Support Systems, EOLSS Publishers, Oxford, 2001.

Holland, J. Adaptation in Natural and Artificial Systems, 2nd ed., MIT Press, MIT, Cambridge, 1975.

Hutter F, Hoos H.H, Leyton-Brown K, Stützle T. ParamILS: an automatic algorithm configuration framework. Journal of Artificial Intelligence Research, 36:267–306, 2009.

Karaboga D, Basturk B. Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems, Advances in soft computing: foundations of fuzzy logic and soft computing, 4529:789–798, 2007.

Kennedy J, Eberhart R. Swarm Intelligence. Morgan Kaufmann Publishers Inc., 2001.

Kleinberg J., Tardos E. Algorithm Design, Addison-Wesley, 2nd ed., 2006.

Ladner R.E. On the structure of polynomial time reducibility, J. ACM 22:151-171, 1975.

Li X., Gar-On Yeh A. Integration of genetic algorithms and GIS for optimal location search,International Journal of Geographical Information Science, 19(5):, 581-601, 2005.

Li X., Gar-On Yeh A. Neural-network-based cellular automata for simulating multiple land use changes using GIS, International Journal of Geographical Information Science, 16(4): 323-343, 2002.

Liu K, Alam M. S., Zhu J., Zheng J., Chi L. Prediction of carbonation depth for recycled aggregate concrete using ANN hybridized with swarm intelligence algorithms, Construction and Building Materials, vol 301, 124382, 2021.

López-Ibáñez M, Dubois-Lacoste J, Stützle T, Birattari M. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

Lopez-Ibanez M., Dubois-Lacoste J, Stützle T., Birattari M. The irace package, iterated race for automatic algorithm configuration. Tech. Rep. TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011.

Mercer R., Sampson J. Adaptive search using a reproductive meta-plan. Kybernetes, 7(3):215–228, 1978.

Mitchell T. Machine Learning, McGraw-Hill, 1997.

Mlot N.J., Tovey C.A., Hu D.L. Fire ants self-assemble into waterproof rafts to survive floods. *PNAS,* 108(19):7669-7673, 2011.

Montgomery D.C. Design and Analysis of Experiments. John Wiley & Sons, 2012.

Nakrani S., Tovey C. On honey bees and dynamic server allocation in the Internet hosting centers. Adaptive Behaviour 12:223-240, 2004.

Nannen V., Eiben A. A method for parameter calibration and relevance estimation in evolutionary algorithms, 8th Annual Conference on Genetic and Evolutionary Computation, 183–190, 2006.

Nannen V., Eiben A. Relevance estimation and value calibration of evolutionary algorithm parameters. 20th International Joint Conference on Artifical Intelligence, 975–980, 2007.

Nannen V., Eiben, A.E. Relevance estimation and value calibration of evolutionary algorithm parameters. 20th Int. Joint Conf. on Artifical Intelligence, 975-980, 2007.

Osman I.H., Laporte G. Metaheuristics: A bibliography, Ann. Operational Research, 63: 513-623, 1996.

Pedersen M. E. H. Tuning & Simplifying Heuristical Optimization. PhD thesis, University of Southampton, School of Engineering Science, 2010.

Pham DT, Ghanbarzadeh A, Koc E, et al. The bees algorithm - a novel tool for complex optimization problems, IPROMS Conference, 454– 461, 2006.

Pijanowski B. C., Brown D. G., Shellito B. A., Manik G. A. Using neural networks and GIS to forecast land use changes: A Land Transformation Model, Computers, Environment and Urban Systems, 26(6): 553-575, 2002.

Rokach, L., Maimon, O. Decision Trees. In: Maimon, O., Rokach, L. (eds) Data Mining and Knowledge Discovery Handbook. Springer, Boston, MA, 2005.

Rudolf M. Parameter tuning for numerical optimization algorithms, Diploma Thesis, Czech Technical University, Prague, 2017.

Sayama H. Collective Dynamics of Complex Systems, Research Group at Binghamton University, State University of New York, 2010.

Šešum -Čavić V., Swarm-Based Metaheuristics, CET Computer Equipment and Trade, Belgrade, ISBN: 978-86-7991-430-9, 2020.

Shoham Y., Leyton-Brown K., Multiagent Systems: Algorithmic, Game-Theoretic and Logical Foundation, Cambridge University Press, 2009.

Siddique N., Adeli H. Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing, Wiley, 2013.

Skienna S.S. The Algorithm Design Manual, Springer, 2008.

Smit S.K., Eiben A.E. Comparing parameter tuning methods for evolutionary algorithms. IEEE Congress on Evolutionary Computation, 399–406, 2009.

Smit S.K., Eiben A.E. Comparing parameter tuning methods for evolutionary algorithms. IEEE Congress on Evolutionary Computation, 399–406, 2009.

Teichgraeber H, Brandt A.R. , Systematic Comparison of Aggregation Methods for Input Data Time Series Aggregation of Energy Systems Optimization Problems, Editor(s): Mario R. Eden, Marianthi G. Ierapetritou, Gavin P. Towler, Computer Aided Chemical Engineering, Elsevier, 44: 955-960, 2018.

Teodorovic D, Lucic P, Markovic G, et al. D. Bee Colony Optimization: Principles and Applications, 8th Seminar on Neural Network Applications in Electrical Engineering, 151-156, 2006.

Theraulaz G, Jost C, Perna A, et al. A computational model of ant nest morphogenesis, Advances in Artificial Life, ECAL 2011 - Synthesis and Simulation of Living Systems, MIT Press, 404 - 411, 2011.

van Dijk, S., Thierens, D., de Berg, M. Using Genetic Algorithms for Solving Hard Problems in GIS. GeoInformatica 6, 381–413, 2002.

Vose, M. D. Simple genetic algorithm: foundations and theory, MIT Press, 1999.

Wedde H.F., Farooq M., Zhang Y. BeeHive: An Efficient Fault-Tolerant Routing Algorithm Inspired by Honey Bee Behavior, Ant Colony Optimization and Swarm Intelligence: 4th Int. Workshop, 83-94, 2004.

Whitley, L.D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4: 65-85, 1994.

Wong L.P., Low M.Y., Chong C.S. A Bee Colony Optimization for Travelling Salesman Problem. IEEE 2nd Asia Int. Conf. on Modelling & Simulation, 818-823, 2008.

Xing B, Gao W-J. Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms, Intelligent Systems Reference Library, Springer, 2014.

Yang X.S. Engineering Optimizations via Nature-Inspired Virtual Bee Algorithms, Artificial Intelligence and Knowledge Engineering Applications: A Bio-inspired Approach: 1st Int. Work-Conf. on the Interplay between Natural and Artificial Computation, 317—323, 2005.

Yang XS. Nature-Inspired Metaheuristic Algorithms, Luniver Press, 2008

Zadeh L., Klir G. J., Yuan B. Fuzzy Sets, Fuzzy Logic, Fuzzy Systems: Selected Papers, World Scientific, 1996.

Zhong L, Hu L, Zhou H, Deep learning based multi-temporal crop classification, Remote Sensing of Environment, 221: 430-443, 2019.

DODATAK
PRIMERI

Swarm Based algoritmi

Ant Algoritam ACO

MATLAB implementation of ACO for Discrete and Combinatorial Optimization Problems[16]

main.m

```
aco;
```

aco.m

```
clc;
clear;
close all;

%% Problem Definition
model=CreateModel();
CostFunction=@(tour) TourLength(tour,model);
nVar=model.n;
%% ACO Parameters
MaxIt=300;      % Maximum Number of Iterations
nAnt=40;        % Number of Ants (Population Size)
Q=1;
tau0=10*Q/(nVar*mean(model.D(:)));    % Initial Phromone
alpha=1;        % Phromone Exponential Weight
beta=1;         % Heuristic Exponential Weight
rho=0.05;       % Evaporation Rate

%% Initialization
eta=1./model.D;             % Heuristic Information Matrix
tau=tau0*ones(nVar,nVar);   % Phromone Matrix
BestCost=zeros(MaxIt,1);    % Array to Hold Best Cost Values

% Empty Ant
empty_ant.Tour=[];
empty_ant.Cost=[];

% Ant Colony Matrix
ant=repmat(empty_ant,nAnt,1);

% Best Ant
BestSol.Cost=inf;

%% ACO Main Loop
for it=1:MaxIt

   % Move Ants
```

[16] https://www.mathworks.com/matlabcentral/fileexchange/52859-ant-colony-optimization-aco?s_tid=mwa_osa_a

```
    for k=1:nAnt
        ant(k).Tour=randi([1 nVar]);
        for l=2:nVar
            i=ant(k).Tour(end);
            P=tau(i,:).^alpha.*eta(i,:).^beta;
            P(ant(k).Tour)=0;
            P=P/sum(P);
            j=RouletteWheelSelection(P);
            ant(k).Tour=[ant(k).Tour j];
        end
        ant(k).Cost=CostFunction(ant(k).Tour);
        if ant(k).Cost<BestSol.Cost
            BestSol=ant(k);
        end
    end

    % Update Phromones
    for k=1:nAnt
        tour=ant(k).Tour;
        tour=[tour tour(1)]; %#ok
        for l=1:nVar
            i=tour(l);
            j=tour(l+1);
            tau(i,j)=tau(i,j)+Q/ant(k).Cost;
        end
    end

    % Evaporation
    tau=(1-rho)*tau;

    % Store Best Cost
    BestCost(it)=BestSol.Cost;

    % Show Iteration Information
    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);

    % Plot Solution
    figure(1);
    PlotSolution(BestSol.Tour,model);
    pause(0.01);
end

%% Results
figure;
plot(BestCost,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;
```

CreateModel.m

```
function model=CreateModel()
   x=[82 91 12 92 63 9 28 55 96 97 15 98 96 49 80 14 42 92 80 96];
   y=[66 3 85 94 68 76 75 39 66 17 71 3 27 4 9 83 70 32 95 3];
   n=numel(x);
   D=zeros(n,n);
   for i=1:n-1
      for j=i+1:n
         D(i,j)=sqrt((x(i)-x(j))^2+(y(i)-y(j))^2);
         D(j,i)=D(i,j);
      end
   end

   model.n=n;
   model.x=x;
   model.y=y;
   model.D=D;
end
```

RouletteWheelSelection.m

```
function j=RouletteWheelSelection(P)
   r=rand;
   C=cumsum(P);
   j=find(r<=C,1,'first');
end
```

TourLength.m

```
function L=TourLength(tour,model)
   n=numel(tour);
   tour=[tour tour(1)];
   L=0;
   for i=1:n
      L=L+model.D(tour(i),tour(i+1));
   end
end
```

Bee algoritam

Artificial Bee Colony Optimization[17]

ABC.m

```
function [bestsol,bestfitness] = ABC(prob,lb,ub,Np,T,limit)

%% Starting of ABC
f = NaN(Np,1);                % Vector to store the objective function value of the population members
fit = NaN(Np,1);              % Vector to store the fitness function value of the population members
trial = NaN(Np,1);            % Initialization of the trial vector

D = length(lb);               % Determining the number of decision variables in the problem

P = repmat(lb,Np,1) + repmat((ub-lb),Np,1).*rand(Np,D);   % Generation of the initial population

for p = 1:Np
   f(p) = prob(P(p,:));       % Evaluating the objective function value
   fit(p) = CalFit(f(p));     % Evaluating the fitness function value
end

[bestobj, ind] = min(f);      % Determine and memorize the best objective value
bestsol = P(ind,:);           % Determine and memorize the best solution

for t = 1:T

   %% Employed Bee Phase
   for i = 1:Np
      [trial,P,fit,f] = GenNewSol(prob, lb, ub, Np, i, P, fit, trial, f, D);
   end

   %% Onlooker Bee Phase
   % as per the code of the inventors available at https://abc.erciyes.edu.tr/
   % prob=(0.9.*Fitness./max(Fitness))+0.1;
   % MATLAB Code of the ABC algorithm version 2 has been released (14.12.2009) (more optimized
coding)

   probability = 0.9 * (fit/max(fit)) + 0.1;

   m = 0; n = 1;

   while(m < Np)
      if(rand < probability(n))
```

[17] https://www.mathworks.com/matlabcentral/fileexchange/74122-artificial-bee-colony-optimization?s_tid=srchtitle

```
        [trial,P,fit,f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D);
        m = m + 1;
      end
      n = mod(n,Np) + 1;
   end

   [bestobj,ind] = min([f;bestobj]);
   CombinedSol = [P;bestsol];
   bestsol = CombinedSol(ind,:);

   %% Scout Bee Phase
   [val,ind] = max(trial);

   if (val > limit)
      trial(ind) = 0;                % Reset the trial value to zero
      P(ind,:) = lb + (ub-lb).*rand(1,D); % Generate a random solution
      f(ind) = prob(P(ind,:));          % Determine the objective function value of the newly generated
solution
      fit(ind) = CalFit(f(ind));        % Determine the fitness function value of the newly generated solution
   end
end

[bestfitness,ind] = min([f;bestobj]);
CombinedSol = [P;bestsol];
bestsol = CombinedSol(ind,:);
```

CalFit.m

```
function fit = CalFit(f)

if f >= 0
   fit  = 1 / (1+f);
else
   fit = 1 + abs(f);
end
```

ScriptABC.m

```
clc
clear
close all

%% Problem settings
lb = [-100 -100 -100 -100];       % Lower bound
ub = [100 100 100  100];          % Upper bound
prob = @Griewank;                 % Fitness function

%% Algorithm parameters
Np = 10;                          % Population Size
T = 50;                           % No. of iterations
```

```
limit = 3;                    % Parameter limit indicating maximum number of failures
rng(2,'twister')
 [bestsol,bestfitness] = ABC(prob,lb,ub,Np,T,limit)
```

GenNewSol.m

```
function [trial,P,fit,f] = GenNewSol(prob, lb, ub, Np, n, P, fit, trial, f, D)
j = randi(D,1);              % Randomly select the variable that is to be changed
p = randi(Np,1);              % Randomly select the neighbour

while (p == n)               % Ensuring that the neighbour is different from the current solution
   p = randi(Np,1);
end

Xnew = P(n,:);               % Variable to generate a new solution

Phi = -1 + (1-(-1))*rand;      % Generating a random number between -1 and 1

Xnew(j) = P(n,j) + Phi*(P(n,j) - P(p,j));  % Generating a new solution
Xnew(j) = min(Xnew(j),ub(j));    % Bounding the violating variables to their upper bound
Xnew(j) = max(Xnew(j),lb(j));    % Bounding the violating variables to their lower bound

ObjNewSol = prob(Xnew);           % Determining the objective function value
FitnessNewSol = CalFit(ObjNewSol);  % Determining the fitness function value

if (FitnessNewSol > fit(n))
   P(n,:) = Xnew;             % New solution enters the pool of solutions
   fit(n) = FitnessNewSol;      % Update the fitness value
   f(n) = ObjNewSol;          % Update the objective function value
   trial(n) = 0;              % Resetting trial to zero
else
   trial(n) = trial(n)+1;       % Increase the value of the trial counter
end
```

Griewank.m

```
function F = Griewank(x)
d = sqrt(1:length(x));
F = 1+((1/4000)*sum((x).^2)) - prod(cos(x./d.^0.5));
```

Rastrigin(.m

```
function F = Rastrigin(x)
F = sum((x.^2 - 10.*cos(2.*pi.*x) + 10));
```

Rosenbrock.m

```
function F = Rosenbrock(x)
d = length(x);
f = NaN(d-1,1);
```

```
for i = 1: d-1
   f(i) = 100*(x(i)^2 - x(i+1))^2 + (1 - x(i))^2;
end
F =  sum(f);
```

Schaffer.m

```
function F = Schaffer(x)

x1 = x(1);
x2 = x(2);

fact1 = (sin(x1^2 - x2^2))^2 - 0.5;
fact2 = (1 + 0.001*(x1^2 + x2^2))^2;
F = 0.5 + fact1/fact2;
```

SphereNew.m

```
function f = SphereNew(x)
f = sum(x.^2);
```

PSO

Particle Swarm Optimization (PSO)[18]

main.m
```
pso;
```

pso.m
```
clc;
clear;
close all;

%% Problem Definition

CostFunction=@(x) Sphere(x);       % Cost Function
nVar=10;           % Number of Decision Variables
VarSize=[1 nVar];   % Size of Decision Variables Matrix
VarMin=-10;        % Lower Bound of Variables
VarMax= 10;        % Upper Bound of Variables


%% PSO Parameters
MaxIt=1000;      % Maximum Number of Iterations
nPop=100;       % Population Size (Swarm Size)

% PSO Parameters
w=1;           % Inertia Weight
wdamp=0.99;     % Inertia Weight Damping Ratio
c1=1.5;        % Personal Learning Coefficient
c2=2.0;        % Global Learning Coefficient

% If you would like to use Constriction Coefficients for PSO,
% uncomment the following block and comment the above set of parameters.

% % Constriction Coefficients
% phi1=2.05;
% phi2=2.05;
% phi=phi1+phi2;
% chi=2/(phi-2+sqrt(phi^2-4*phi));
% w=chi;         % Inertia Weight
% wdamp=1;       % Inertia Weight Damping Ratio
% c1=chi*phi1;   % Personal Learning Coefficient
```

[18] https://www.mathworks.com/matlabcentral/fileexchange/79814-improved-particle-swarm-optimization-pso-algorithm?s_tid=srchtitle

```
% c2=chi*phi2;    % Global Learning Coefficient

% Velocity Limits
VelMax=0.1*(VarMax-VarMin);
VelMin=-VelMax;

%% Initialization
empty_particle.Position=[];
empty_particle.Cost=[];
empty_particle.Velocity=[];
empty_particle.Best.Position=[];
empty_particle.Best.Cost=[];

particle=repmat(empty_particle,nPop,1);

GlobalBest.Cost=inf;
for i=1:nPop

   % Initialize Position
   particle(i).Position=unifrnd(VarMin,VarMax,VarSize);

   % Initialize Velocity
   particle(i).Velocity=zeros(VarSize);

   % Evaluation
   particle(i).Cost=CostFunction(particle(i).Position);

   % Update Personal Best
   particle(i).Best.Position=particle(i).Position;
   particle(i).Best.Cost=particle(i).Cost;

   % Update Global Best
   if particle(i).Best.Cost<GlobalBest.Cost

      GlobalBest=particle(i).Best;
   end
end

BestCost=zeros(MaxIt,1);

%% PSO Main Loop
for it=1:MaxIt
   for i=1:nPop

      % Update Velocity
      particle(i).Velocity = w*particle(i).Velocity ...
         +c1*rand(VarSize).*(particle(i).Best.Position-particle(i).Position) ...
         +c2*rand(VarSize).*(GlobalBest.Position-particle(i).Position);
```

```
        % Apply Velocity Limits
        particle(i).Velocity = max(particle(i).Velocity,VelMin);
        particle(i).Velocity = min(particle(i).Velocity,VelMax);

        % Update Position
        particle(i).Position = particle(i).Position + particle(i).Velocity;

        % Velocity Mirror Effect
        IsOutside=(particle(i).Position<VarMin | particle(i).Position>VarMax);
        particle(i).Velocity(IsOutside)=-particle(i).Velocity(IsOutside);

        % Apply Position Limits
        particle(i).Position = max(particle(i).Position,VarMin);
        particle(i).Position = min(particle(i).Position,VarMax);

        % Evaluation
        particle(i).Cost = CostFunction(particle(i).Position);

        % Update Personal Best
        if particle(i).Cost<particle(i).Best.Cost

            particle(i).Best.Position=particle(i).Position;
            particle(i).Best.Cost=particle(i).Cost;

            % Update Global Best
            if particle(i).Best.Cost<GlobalBest.Cost

                GlobalBest=particle(i).Best;

            end
        end
    end

    BestCost(it)=GlobalBest.Cost;

    disp(['Iteration ' num2str(it) ': Best Cost = ' num2str(BestCost(it))]);
    w=w*wdamp;

end

BestSol = GlobalBest;

%% Results
figure;
%plot(BestCost,'LineWidth',2);
semilogy(BestCost,'LineWidth',2);
xlabel('Iteration');
```

```
ylabel('Best Cost');
grid on;
```

sphere.m

```
function z=Sphere(x)
   z=sum(x.^2);

end
```

Evolutivni Algoritmi

GA

Travelling Salesman Problem[19]

tsp.m

```
%reading distanceMatrix from csv file
distanceMatrixFile = input('Enter name of the file containing distance matrix (include extension like
.csv/.xls): ','s');
distanceMatrix = readmatrix(distanceMatrixFile);
distanceMatrix = distanceMatrix(:,2:end);

%creating city pairs and converting distance square matrix to distance
%column vector
fprintf('Creating city pairs\n');
numberOfCities = size(distanceMatrix,1); %number of cities
c=1;
for count = 1:numberOfCities:(numberOfCities*numberOfCities)
   cityPairs(count:numberOfCities*c, 1) = c;
   cityPairs(count:numberOfCities*c, 2) = 1:numberOfCities;
   distanceVector(count:numberOfCities*c, 1) = distanceMatrix(c,:)';
   c=c+1;
end
lengthDistanceVector = length(distanceVector);

%% Equality Constraints
fprintf('Creating equality constraints\n');
%Number of trips = number of cityPairs
Aeq = spones(1:length(cityPairs));
beq = numberOfCities;

%Number of trips to a city = 1 and from a city = 1
Aeq =
[Aeq;spalloc(2*numberOfCities,length(cityPairs),2*numberOfCities*(numberOfCities+numberOfCities-
1))]; %allocate a sparse matrix to preallocate memory for the equality constraints;
c=1;
for count = 1:2:((2*numberOfCities)-1)
   columnSum = sparse(cityPairs(:,2)==c);
   Aeq(count+1,:) = columnSum'; % include in the constraint matrix
   rowSum = cityPairs(:,1)==c;
   Aeq(count+2,:) = rowSum';
   c=c+1;
```

[19] https://www.mathworks.com/matlabcentral/fileexchange/54420-travelling-salesman-problem-a-genetic-algorithm-approach?s_tid=srchtitle

```
end
beq = [beq; ones(2*numberOfCities,1)];

%Non-existing routes
nonExists = sparse(distanceVector == 0);
Aeq(2*c,:) = nonExists';
beq = [beq; 0];

% Binary Bounds
%Setting the decision variables as binary variables
intcon = 1:lengthDistanceVector;
lb = zeros(lengthDistanceVector,1);
ub = ones(lengthDistanceVector,1);

%% Optimize Using intlinprog
fprintf('Solving the problem\n');
opts = optimoptions('intlinprog','CutGeneration','Advanced','NodeSelection','mininfeas','Display','off');
[decisionVariables,optimumCost,exitflag,output] =
intlinprog(distanceVector,intcon,[],[],Aeq,beq,lb,ub,opts);

%% Subtour Detection
tours = detectSubtours(decisionVariables,cityPairs);
numberOfTours = length(tours);
fprintf('Number of subtours: %d\n',numberOfTours);

%% Subtour Constraints
A = spalloc(0,lengthDistanceVector,0); % creating sparse inequality constraint matrix
b = [];
while numberOfTours > 1 % repeat until there is just one subtour
   b = [b;zeros(numberOfTours,1)]; % entering inequality constraints RHS
   A = [A;spalloc(numberOfTours,lengthDistanceVector,numberOfCities)]; % entering inequality
constraints LHS
   for count = 1:numberOfTours
      inequalityConstraintNumber = size(A,1)+1;
      subTourId = tours{count}; % Extracting subtour one by one

      % adding subtour constraints (inequality constraints)
      subTourPairs = nchoosek(1:length(subTourId),2);
      for jj = 1:size(subTourPairs,1) % Finding variables associated with the current sub tour
         subTourVariable = (sum(cityPairs==subTourId(subTourPairs(jj,1)),2)) & ...
               (sum(cityPairs==subTourId(subTourPairs(jj,2)),2));
         A(inequalityConstraintNumber,subTourVariable) = 1;
      end
      b(inequalityConstraintNumber) = length(subTourId)-1; % reducing number of trips allowed by One
Ex., A-B-A: 2 -> 1
   end
   % Optimize again
   fprintf('\nsolving the problem again eliminating subtours\n');
```

```
   [decisionVariables,optimumCost,exitflag,output] =
intlinprog(distanceVector,intcon,A,b,Aeq,beq,lb,ub,opts);

   % Check for subtours again
   fprintf('Checking again for subtours\n');
   tours = detectSubtours(decisionVariables,cityPairs);
   numberOfTours = length(tours);
   fprintf('Number of subtours: %d\n',numberOfTours);
end

%% Solution Quality
%smaller the value better the solution
fprintf('\nSolution Quality: %f (lesser the better)\n',output.absolutegap);
fprintf('Optimized tour route:');
celldisp(tours);
fprintf('Note: The numbers correspond to order of cities in the input file\n');
fprintf('Total distance of the optimal route: %d\n', optimumCost);
```

detectSubtours.m

```
function subTours = detectSubtours(x,idxs)
% Returns a cell array of subtours. The first subtour is the first row of x, etc.
x(x<0.0001)=0;
r = find(x); % indices of the trips that exist in the solution
substuff = idxs(r,:); % the collection of node pairs in the solution
unvisitedSubToursSubTours = ones(length(r),1); % keep track of places not yet visitedSubTours
curr = 1; % subtour we are evaluating
startour = find(unvisitedSubToursSubTours,1); % first unvisitedSubToursSubTours trip
   while ~isempty(startour)
      home = substuff(startour,1); % starting point of subtour
      nextpt = substuff(startour,2); % next point of tour
      visitedSubTours = nextpt; unvisitedSubToursSubTours(startour) = 0; % update
unvisitedSubToursSubTours points
      while nextpt ~= home
         % Find the other trips that starts at nextpt
         [srow,scol] = find(substuff == nextpt);
         % Find just the new trip
         trow = srow(srow ~= startour);
         scol = 3-scol(trow == srow); % turn 1 into 2 and 2 into 1
         startour = trow; % the new place on the subtour
         nextpt = substuff(startour,scol); % the point not where we came from
         visitedSubTours = [visitedSubTours,nextpt]; % update nodes on the subtour
         unvisitedSubToursSubTours(startour) = 0; % update unvisitedSubToursSubTours
      end
      subTours{curr} = visitedSubTours; % store in cell array
      curr = curr + 1; % next subtour
      startour = find(unvisitedSubToursSubTours,1); % first unvisitedSubToursSubTours trip
   end
end
```

Neuronske Mreže

Kreiranje i treniranje Feedforward Neuronske Mreže[20] u cilju predviđanja temperature.

- Pročitati podatke sa ThingSpeak kanala meteorološke stanice

ThingSpeak™ kanal 12397 sadrži podatke sa MathVorks® meteorološke stanice, koja se nalazi u Natick-u, Massachusetts. Podaci se prikupljaju jednom u minutu. Polja 2, 3, 4 i 6 sadrže podatke o brzini vetra (mph), relativnoj vlažnosti, temperaturi (F) i atmosferskom pritisku (inHg).

```
data = thingSpeakRead(12397,'Fields',[2 3 4 6],'DateRange',[datetime('January 7, 18'),datetime('January 9, 2018')],... 'outputFormat','table');
```

- Dodeliti ulazne varijable i ciljne vrednosti

Dodeliti ulazne varijable i izračunati tačku “rose” iz temperature i relativne vlažnosti da bi ih koristili kao cilj. Pretvoriti temperaturu iz Farenhajta u Celzijuse i odrediti konstante za vodenu paru (b) i barometarski pritisak (c). Izračunati srednju vrednost 'gama' i dodeliti ciljne vrednosti za mrežu.

```
inputs = [data.Humidity'; data.TemperatureF'; data.PressureHg'; data.WindSpeedmph'];
tempC = (5/9)*(data.TemperatureF-32);
b = 17.62;
c = 243.5;
gamma = log(data.Humidity/100) + b*tempC ./ (c+tempC);
dewPointC = c*gamma ./ (b-gamma);
dewPointF = (dewPointC*1.8) + 32;
targets = dewPointF';
```

- Kreirati i obučiti dvoslojnu feedforward mrežu

Koristiti funkciju *feedforwardnet* da bi kreirali dvoslojnu feedforward mrežu. Mreža ima jedan skriveni sloj sa 10 neurona i izlazni sloj. Koristiti funkciju treniranja da bi obučili mrežu pomoću ulaza.

```
net = feedforwardnet(10);
[net,tr] = train(net,inputs,targets);
```

[20] https://www.mathworks.com/help/thingspeak/create-and-train-a-feedforward-neural-network.html

- Koristiti obučeni model za predviđanje podataka

Nakon što je mreža obučena i validirana, može se koristiti objekat mreže za izračunavanje odgovora na bilo koji ulaz, u ovom slučaju tačku rose za petu tačku ulaznih podataka.

```
output = net(inputs(:,5))
output =    -15.6383
```