SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3927

OPTIMIZACIJA EVOLUCIJSKIM UČENJEM

Josip Užarević

Zagreb, lipanj 2015.

Zahvala

Ovim putem se zahvaljujem mentoru izv. prof. dr. sc. Domagoju Jakoboviću zbog stalne pristupačnosti i ugodne suradnje. Također, mentor me zainteresirao za područja računarske znanosti, za koje ne bih niti znao u ovoj fazi studiranja, kroz Seminar, Projekt te naposljetku Završni rad.

Zahvaljujem se i doc. dr. sc. Marku Čupiću na odličnim predavanjima na kolegijima „Oblikovni obrasci u programiranju“ te „Umjetna inteligencija“, koja su znatno doprinijela poboljšanju implementacije programskog ostvarenja u okviru ovog završnog rada.

Sadržaj

1. Uvod

Pri obradi velike količine podataka, uobičajeni algoritmi, koji dobro rade na manjim skupovima podataka, često nisu dovoljno efikasni i brzi za obradu podataka u konačnom, dovoljno kratkom vremenu. Primjeri velike količine podataka su: aktivnosti korisnika na društvenim mrežama, obrada mjerenja fizikalnih parametara provedenih kroz dulji vremenski period, gibanje velikog broja čestica u prostoru (npr. termodinamika) i dr.

U takvim slučajevima, potrebno je koristiti algoritme koji su prilagođeni obradi velike količine podataka. Takvi algoritmi, pod cijenu optimalnosti, izvode zaključke u konačnom i dovoljno kratkom vremenu za normalno korištenje. Ovisno o težini problema i količini podataka, vrijeme izvođenja može biti od nekoliko sekundi do nekoliko dana ili mjeseci.

Algoritmi koji rade s velikim količinama podataka traže uglavnom „dovoljno dobro rješenje“, jer upravo zbog svoje izvedbe ne jamče optimalno rješenje. Ukoliko bi ovakvi algoritmi tražili optimalno rješenje, opet bismo došli do problema rješavanja u kratkom vremenu. Dapače, ovakvi algoritmi bi vjerojatno radili i sporije nego uobičajeni algoritmi koji su predviđeni za rad s manjom količinom podataka.

Tema ovog završnog rada je „Optimizacija evolucijskim učenjem“, izvorno „Learnable Evolutionary Model“ (skraćeno LEM). LEM je algoritam koji definira okvir za naizmjenično korištenje strojnog učenja i genetskog algoritma pri klasifikaciji rješenja problema. U okviru rada obrađeno je: strojno učenje, genetski algoritam i LEM te je implementirana aplikacija pomoću koje je moguće provesti analizu i izvesti zaključke vezane za LEM, kao apstraktni algoritam te implementaciju, kao konkretizaciju algoritma.

2. Problem optimizacije funkcije

U računarskoj znanosti, jedan od temeljnih problema je optimizacija funkcije. Problem optimizacije funkcije svodi se na problem traženja minimuma ili maksimuma za proizvoljnu funkciju. Ovaj problem je specifičan zbog sljedećih razloga:

- **Računala su diskretna.** Uvijek postoji problem pri obradi kontinuiranih podataka na digitalnom računalu, jer se mogu javiti pogreške, a također pri velikoj količini podataka potrebno je razmatrati neefikasnost rada računala s decimalnim brojevima.

- **Moguće nepoznavanje domene.** Ukoliko je domena ulaznih vrijednosti funkcije beskonačna, javlja se problem određivanja skupa unutar kojeg se podaci ispituju. Budući da je potrebno riješiti problem u konačnom vremenu, domena ulaznih podataka također mora biti konačna, odnosno omeđena. Problem je u odabiru konačne domene, jer uvijek postoji mogućnost da je optimum izvan te domene, ukoliko je originalna domena beskonačna.

- **Deriviranje.** Deriviranje na računalu, pomoću kojeg bi se mogao donekle riješiti ovakav problem je neefikasno i sporo. Nadalje, nisu sve funkcije derivabilne, a još veći problem je da ulazni podaci uopće ne moraju (često i ne mogu) biti predstavljeni egzaktnim funkcijama.

- **Ulazni podaci ne moraju biti definirani egzaktnom funkcijom.** Navedeni primjeri društvene mreže daju naslutiti da ulazni podaci uopće ne moraju biti predstavljeni egzaktnom funkcijom. Veliki broj korisnika je nemoguće prikazati egzaktnom funkcijom (moguće je eventualno aproksimirati), stoga u tom slučaju algoritmi rade sa sirovim podacima (engl. raw data). Iako je ovakve podatke nemoguće prikazati egzaktnom funkcijom i dalje je potrebno (i moguće) tražiti optimume u podacima.

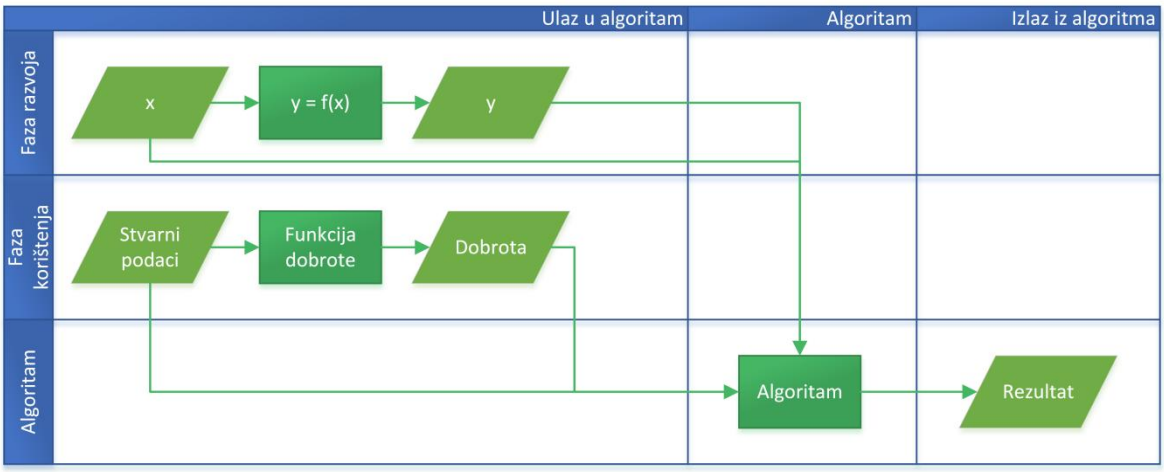
- **Veliki broj atributa koji opisuju objekt.** Porastom dimenzije ulaznih podataka (ekvivalentno porastu broja atributa koji opisuju objekt) vrijeme izvođenja algoritma se povećava, stoga je potrebno voditi računa o tome da

implementirani algoritam može podržavati što je moguće veću dimenziju, a da vrijeme izvođenja i dalje bude razumnog trajanja.

- **Problem lokalnih optimuma.** Budući da se radi s velikim količinama podataka, ne pretražuje se cijeli skup podataka u traženju rješenja – tada bi se algoritmi za rješavanje problema optimizacije funkcije sveli na osnovne algoritme koji rade dobro s manjim skupom podataka. Zbog toga se u većini optimizacijskih algoritama može dogoditi takozvano „zapinjanje u lokalnom optimumu“, odnosno, algoritam počne težiti ka optimumu koji je lokalan. Naposlijetku, algoritam kao rezultat daje lokalni optimum umjesto globalnog.

2.1. Algoritmi za rad s velikim količinama podataka

Zanimljiva činjenica je da se mnogi problemi koji sadrže velike količine podataka mogu zapravo svesti na problem optimizacije funkcije. Upravo zato se proces testiranja i razvijanja algoritama za rad s velikom količinom podataka svodi na to da se algoritmi prvo implementiraju tako da rade s funkcijama – ukoliko rade s funkcijama, vrlo vjerojatno će dobro raditi i sa skupom ulaznih sirovih podataka. Algoritmu se u principu „podmjeste“ ulazne i izlazne vrijednosti neke funkcije (koje se pak računalno generiraju) tako da algoritam u samom početku već radi s gotovim podacima. Na taj način, algoritam nikada ne mora niti raditi direktno s funkcijama, nego s već pripremljenim podacima.



Slika 2-1: Postupak razvoja algoritma

Dijagram toka zorno prikazuje način razvoja i korištenja algoritma. Prvotno se u fazi razvoja koriste spomenute funkcije, a ulazne i izlazne vrijednosti funkcije se

koriste kao ulazni podaci za algoritam. Ulazne vrijednosti funkcije (x) predstavljaju podatke za obradu, dakle atribute koji opisuju podatke, a izlazne vrijednosti (y) se koriste za procjenu koliko je taj ulazni podatak s pripadnim atributima dobar. Algoritam pomoću y procjenjuje dobrotu podataka koji su proizveli taj y: ako se traži minimum funkcije, manji y ima veću dobrotu, dok u slučaju traženja maksimuma veći y proizvodi veću dobrotu.

U fazi korištenja pri radu sa stvarnim podacima potrebno je dodati funkciju dobrote koja će za dani objekt iz stvarnog svijeta procijeniti njegovu dobrotu. Nakon procjene dobrote, algoritam prima jednak oblik podataka kao pri radu s funkcijama – dobiva objekte s pripadnim atributima (x) i pripadnom dobrotom (y).

Opisani način pristupu algoritama je primjenjiv i na dvije velike skupine algoritama, čiji su predstavnici korišteni pri izradi završnog rada: strojno učenje i genetski algoritam. Algoritam LEM, koji je glavna tema završnog rada, kombinira strojno učenje i genetski algoritam te se i sâm LEM, na najapstraktnijoj razini, može opisati na navedeni način. LEM interno vodi računa hoće li podatke i dobrotu prosljeđivati strojnom učenju ili genetskom algoritmu, no o tome će biti više riječi kasnije u poglavlju Learnable Evolutionary Model (LEM).

Budući da optimizacijska funkcija predstavlja funkciju dobrote, korisniku i programeru algoritma uglavnom je unaprijed poznat optimum funkcije. Iako je ovaj podatak poznat, on se ne smije koristiti pri implementaciji algoritma, osim radi provjere ispravnosti rada algoritma, jer pri radu algoritma nad stvarnim podacima optimalno rješenje nije poznato (upravo zbog toga se primjenom algoritma pokušava doći do rješenja). Zbog navedenog problema, pri vrednovanju je li se došlo do konačnog rješenja, ne razmatra se apsolutna vrijednost rješenja (jer je ona nepoznata), nego se gleda stupanj poboljšanja u odnosu na prethodno dobiveno rješenje (npr. manja apsolutna razlika ili omjer novog i starog rješenja bliži jedinici mogu predstavljati manje poboljšanje). Kada je poboljšanje dovoljno maleno, može se zaključiti da je algoritam došao do kraja, odnosno, da ne može više radikalno poboljšati rješenje.

2.2. Prikaz podataka

Budući da je tema završnog rada optimizacija funkcija, podaci koji se koriste su reprezentirani u skladu s tim te se u nastavku rada podrazumijevaju sljedeće teze:

- **Objekti** – predstavljeni su kao vektori dimenzije *n*. Svaki podatak (entitet, ulaz u funkciju, objekt) predstavljen je jednim vektorom. Npr., ukoliko se radi o vektoru dimenzije 2, jedan od objekata u populaciji može biti $x = [2.9, 3.85]$.

- **Atributi** – atribut je jedna komponenta vektora – ranije spomenuti primjer vektora ima dva atributa: prvi je 2.9, a drugi je 3.85.

- **Domena** – kao što je ranije spomenuto, atributi koji imaju beskonačnu domenu stvaraju probleme pri izvođenju algoritama. Stoga se moraju poznavati ograničene domene za svaki atribut objekta, odnosno, za svaki atribut se mora znati minimalna i maksimalna vrijednost. Ukoliko se radi o beskonačnoj domeni može se domena zadati tako da bude dovoljno velika (npr. minimalna i maksimalna vrijednost koju podržava tip podataka *double*) da pokrije eventualno rješenje, a opet da bude konačna. Kasnije će se pokazati da je domena promjenjiv parametar te da algoritam, u vidu poboljšanja rješenja, može mijenjati domenu. Štoviše, može domenu dijeliti u poddomene te samim time podijeliti populaciju na podpopulacije (više u poglavlju Algoritam C4.5).

- **Dimenzija (*d*)** – predstavlja duljinu vektora. U slučaju ranijeg primjera vektora, dimenzija je 2. Dimenzija je jednaka za sve vektore koji pripadaju populaciji.

- **Funkcije** – kao ulazni parametar primaju vektore, a kao izlaz daju evaluaciju vektora. Ako je potrebno naći minimum funkcije, onda će sustav vrednovati vektor tim više što je funkcija vektora manjeg iznosa. Štoviše, u okviru rada razmatraju se isključivo funkcije kojima je potrebno naći minimum. Primjer funkcije može biti $f(x_1, x_2) = x_1^2 + x_2^2$.

- **Populacija** – skup objekata koji se razmatraju. U slučaju optimizacije funkcije, populacija se računalno generira tako da se, uz zadanu veličinu populacije,

nasumično generiraju objekti čiji su atributi unutar zadanih domena. Za sve algoritme koji se nadalje opisuju pretpostavlja se prethodno generirana populacija, tako da u sâm algoritam ulazi već pripremljena populacija, čiji su objekti unutar zadanih domenâ.

- **Veličina populacije (*n*)** – vrlo bitan i unaprijed zadani parametar koji određuje koliko objekata postoji u populaciji. Povećanjem veličine populacije, algoritam je sporiji, no davat će uglavnom bolje rezultate zbog veće vjerojatnosti da se ranije generiraju dovoljno dobra rješenja.

- **Vrednovanje** – budući da se traži isključivo minimum funkcijâ, objekt se vrednuje tim više što je vrijednost funkcije za taj objekt manja. Jasno je da će za prethodno navedeni primjer funkcije optimalno rješenje biti vektor $[0, 0]$, jer je upravo u toj točki minimum funkcije, a povećanjem prve i druge komponente vektora, dobrota rješenja će se smanjivati.

- **Razredi** – unutar populacije odabire se određeni postotak najboljih i određeni postotak najgorih rješenja. Najbolja rješenja pripadaju razredu dobrih rješenja, a najgora razredu loših rješenja. Određeni dio populacije predstavlja osrednja rješenja te se isključuju iz izračunâ unutar algoritama.

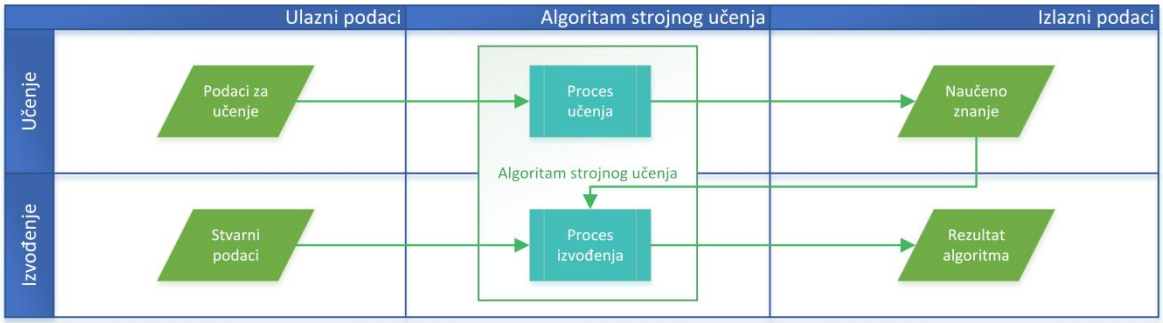
- **Veličina dobre podpopulacije (*g*)** – određuje se na temelju zadanog postotka ukupne populacije unutar kojeg su najbolji članovi populacije.

Navedeni pojmovi koriste se i u strojnom učenju i u genetskom algoritmu, a vrijednosti parametara koji ih opisuju su jednaki za oba algoritma. Samim time, ti parametri se mogu definirati za cijeli LEM, koji ih onda propagira na strojno učenje i genetski algoritam.

3. Strojno učenje

Strojno učenje (engl. *Machine Learning, skr. ML*) predstavlja skup algoritama koji se koristi za klasifikaciju pri obradi velike količine podataka. Algoritam pripadnik strojnog učenja iz ulaznih podataka i njihovih dobrotâ progresivno uči o tome koji atributi utječu na pripadnost kojem razredu. Upravo iz te činjenice i dolazi naziv ove vrste algoritama.

Kada algoritam nauči koji kriteriji su potrebni kako bi podatak pripao određenom razredu, naučene činjenice može koristiti na novom, neviđenom skupu podataka, koji zapravo predstavlja glavni problem koji mora riješiti. Ukoliko skup podataka za učenje algoritma nije bio valjan ili dovoljno reprezentativan u odnosu na stvarne podatke, algoritam strojnog učenja može dati krive rezultate. Kasnije će biti pokazano da se ovaj problem ne odnosi na LEM, jer LEM koristi strojno učenje samo kao dio svog rada te ga uvijek koristi na stvarnim i aktualnim podacima. Opće načelo rada algoritma strojnog učenja prikazano je sljedećim dijagramom toka.



Slika 3-1: Opće načelo rada algoritma strojnog učenja

Primjena strojnog učenja u izvođenju LEM-a se svodi na prikazano opće načelo rada, s iznimkom da je skup ulaznih podataka uvijek skup stvarnih podataka koji se koristi i u procesu učenja i u procesu izvođenja.

Ovisno o vrsti problema, klasifikacija se može provoditi zaista u svrhu klasifikacije (npr. potrebno je klasificirati korisnike društvene mreže u kategorije „aktivan“, „neaktivan“, „periodički aktivan“) ili pak u svrhu klasifikacije na „dobre“ i „loše“ (npr. potrebno je procijeniti koji korisnici društvene mreže su „najdruštveniji“). Klasifikacija na „dobre“ i „loše“ se ustvari svodi na problem optimizacije funkcije – traži se minimum, odnosno maksimum funkcije (ili najveća i najmanja dobrota

ulaznih podataka), ovisno o konkretnom problemu. Stoga se završni rad temelji, gledano sa aspekta strojnog učenja, upravo na traženje „dobrih“ i „loših“ jedinki te je samim time omogućena i jednostavnija i efikasnija implementacija algoritama strojnog učenja. Naime, općenito gledano, algoritmi strojnog učenja podržavaju više razreda pri klasifikaciji, no pri optimizaciji funkcija potrebna je samo „razred dobrih“ i „razred loših“. Samim time lako je uvidjeti da je problem moguće riješiti jednostavnijim i bržim strukturama podataka i postupcima unutar samog algoritma strojnog učenja.

Dobri članovi su definirani kao određeni postotak najboljih članova unutar populacije. U okviru ovog završnog rada, najboljih 20% članova proglašava se dobrima, dok su svi ostali loši.

Pri implementaciji LEM-a korištena su dva algoritma strojnog učenja, koje je moguće unutar aplikacije mijenjati: osnovni, samostalno razvijeni algoritam nazvan BasicML te C4.5, koji je vrlo popularan i poznat predstavnik ovih algoritama.

3.1. BasicML

Osnovna izvedba strojnog učenja smišljena od strane autora je BasicML algoritam. Ovaj algoritam koristi prilično jednostavnu metodu kako bi klasificirao dobra rješenja. Algoritam se može opisati sljedećim pseudokôdom:

```
za svaki objekt unutar populacija.dobričlanovi:
       za svaki indeks => atribut unutar objekt.atributi:
              ako je domena[indeks].min > atribut.vrijednost:
                     domena[indeks].min = atribut.vrijednost
              ako je domena[indeks].max < atribut.vrijednost:
                     domena[indeks].max = atribut.vrijednost
populacija.regeneriraj()
```

Sve što ovaj algoritam radi je sužavanje domene tako da dobri članovi određuju novu domenu. Nakon sužavanja domene, neki od članova populacije nalaze se izvan domene – samo dobri članovi su točno do ruba domene, dok su loši članovi uglavnom izvan nove domene. Primjetimo da loši član može biti i unutar nove domene, jer su, općenito gledano, dobri i loši članovi izmješani unutar domene. Ipak, sužavanjem domene odabire se veći postotak dobrih članova pa samim time i veća vjerojatnost da se u sljedećim generacijama taj postotak povećava sve dok se ne dobije dovoljno malen skup dobrih rješenja i dovoljno malo poboljšanje u odnosu na

prethodnu populaciju da bi se moglo zaključiti da je najbolje od dobrih rješenja upravo rješenje algoritma.

Upravo zbog smanjenja domene, nanovo se mora regenerirati cijela populacija. Regeneracija će proizvesti nove objekte koji će biti unutar nove, smanjene domene. U pravilu će svaki algoritam koji mijenja domenu morati regenerirati populaciju, kako bi populacija zadovoljila zadane granice domene.

Iz pseudokôda se može zaključiti da je složenost ovog algoritma $O(g \times d)$, pri čemu je g veličina dobre podpopulacije, a d dimenzija objekta (broj âtributa). Ovo je, u odnosu na druge algoritme strojnog učenja, relativno mala složenost, jer se ne uzima u obzir loša podpopulacija na temelju koje bi se isto moglo dolaziti do određenih zaključaka. Uz to, algoritam ne vrši nikakve računalno zahtjevne izračune (kao što je to slučaj kod algoritma C4.5).

Upravo zbog svoje jednostavnosti, ovaj algoritam ima jednu veliku manu, a to je izuzetno spora konvergencija. Konvergencija je time sporija što su više izmiješana dobra i loša rješenja, odnosno, što je više lokalnih optimuma. Ipak, zbog spore konvergencije, ovaj algoritam ima izuzetno malu mogućnost zapinjanja u lokalnom optimumu.

3.2. Algoritam C4.5

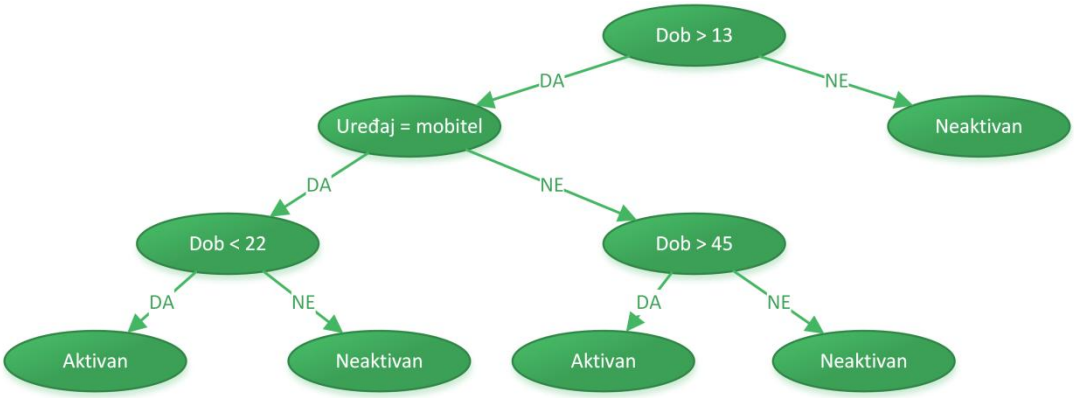
Ovaj algoritam je razvio Ross Quinlan, kao poboljšanu inačicu algoritma ID3, kojeg je razvio isti autor. Algoritam koristi računanje entropije pomoću koje dijeli domenu na poddomene tako da entropija poddomena bude minimalna, odnosno, informacijska dobit bude najveća. U okviru završnog rada implementirana je malo pojednostavljena izvedba budući da ne postoje razredi koji su neviđeni (postoje samo razredi dobrih i loših). Slijedi pseudokôd ovog algoritma:

```
podpopulacija = populacija.kopija()

stabloOdluke = funkcija AlgoritamC45(podpopulacija):
	entropijaMin = 99999999999
	indeksMin = 0
	atributMin = 0
	entropija = 0
	ako podpopulacija.sviSuDobri:
		vrati novi ListOdluke(„dobar“)
	ako podpopulacija.sviSuLoši:
		vrati novi ListOdluke(„loš“)
	za indeks = 0 do dimenzija:
		za svaki indeksObjekta => objekt unutar podpopulacija:
			entropija = računajEntropiju(objekt, indeks)
			ako je entropijaMin > entropijaMin:
				entropijaMin = entropija
				indeksMin = indeks
				vrijednostMin = objekt.atributi[i]
	ovoStablo.dodajDijete(AlgoritamC45(podpopulacija.podpopulacijaManjiOdAtrib
uta(indeksMin, atributMin)))
	ovoStablo.dodajDijete(AlgoritamC45(podpopulacija.podpopulacijaVeciOdAtribu
ta(indeksMin, atributMin)))

stabloOdluke.regenerirajPodpopulacije(populacija)
```

Pseudokôd opisuje algoritam u najgrubljim crtama. Osnova algoritma je generiranje stabla odluke, koje prema vrijednostima atributa klasificira objekte. Sljedeća slika prikazuje primjer stabla odluke za skup podataka o korisnicima na društvenoj mreži. Podaci su fiktivni, a primjer je odabran radi jednostavnijeg razumijevanja konkretnih podataka u odnosu na apstraktne, kakve koriste optimizacijske funkcije.



Slika 3-2: Primjer stabla odluke za algoritam C4.5 s ulaznim podacima o korisnicima društvene mreže

Pretpostavka za ovaj primjer je da je ulaz u algoritam skup objekata korisnika, od kojih svaki ima atribute *Dob* i *Uređaj*. *Dob* poprima vrijednosti od 0 do 100, a

uređaj može biti *„mobitel“* ili *„računalo“*. Postoje dva razreda korisnika: *aktivni* i *neaktivni*.

Prolaskom po svim atributima svih objekata u skupu, C4.5 je pronašao najmanju entropiju pri podjeli skupa na atributu *Dob = 13*. Entropija se računa s obzirom na razred – dakle, idealan slučaj je da su npr. svi stariji od 13 godina aktivni korisnici, a svi mlađi neaktivni. Kasnije će se pokazati da u tom slučaju C4.5 odmah određuje razrede. Ipak, u ovom slučaju, kada su pripadnici razreda izmiješani (ekvivalent lokalnim optimumima), C4.5 ne klasificira podskupine, nego radi rekurzivni izračun za svaku od njih.

Algoritam dijeli populaciju na dvije podpopulacije: korisnike starije od 13 godina i korisnike mlađe od 13 godina. Korijenskom čvoru se rekurzivno dodaje dvoje djece: funkcija koja generira prvo dijete dobiva korisnike starije od 13 godina, a drugo mlađe od 13 godina. U prvom rekurzivnom pozivu postupak se ponavlja i dolazi se do zaključka da je na atributu *uređaj = „mobitel“* najbolje podijeliti skup. Postupak se rekurzivno ponavlja za korisnike koji pomoću mobitela koriste društvenu mrežu.

Podskup korisnika koji su stariji od 13 godina i koriste mobitel opet se dijeli na atributu *dob = 22*. U ovom slučaju, u rekurzivnim pozivima podskupova ovog skupa zaključuje se da su svi korisnici, koji su stari između 13 i 22 godine te koriste mobitel, aktivni. Korisnici koji su pak stariji od 22 godine i koriste mobitel su neaktivni.

Slični zaključci se dovode pri ulasku u granu između *uređaj = „mobitel“* i *Dob > 45*. Naposlijetku, zadnji poziv funkcije će biti za korisnike mlađe od 13 godina – oni su neaktivni.

Naposlijetku, u općem slučaju, jednom gradnjom stabla odluke, moguće je stvaranje više poddomenâ, odnosno više podpopulacijâ. U skladu s tim, potrebno je regenerirati sve podpopulacije pazeći pritom da svaka podpopulacija ima pravedno određen broj jedinki, s obzirom na to da nisu sve poddomene jednake veličine.

Ovaj primjer odlično ilustrira nekoliko važnih stvari koje je ovaj algoritam u stanju izvesti:

- Smanjivanje domene jedne varijable podizanjem donje granice i smanjivanjem gornje granice. Zbog toga, algoritam je u stanju izvesti zaključak da dob mora

biti između 13 i 22 godine. Ako bi algoritam bio u mogućnosti samo jednom mijenjati domenu za jednu varijablu, za *dob > 13* i *uređaj = „mobitel“* algoritam bi morao odlučiti kojem razredu pripadaju ovi korisnici. U najgorem slučaju bi pola korisnika moglo pripadati jednom razredu, a druga polovica drugom te algoritam ne bi ništa mogao zaključiti.

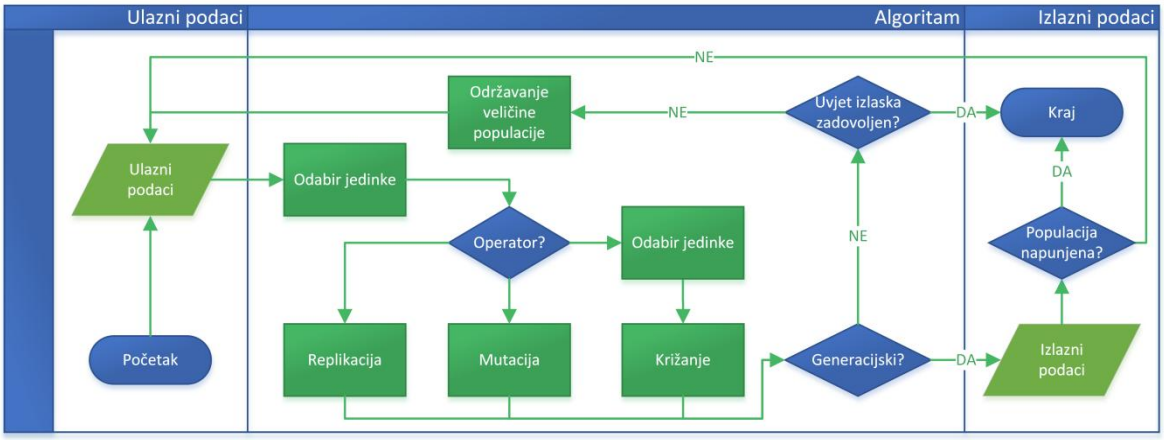
- Zbog navedene promjene, omogućeno je generiranje više poddomena, odnosno podpopulacija i to po svakom atributu. Zbog toga, potrebno je voditi računa i o tome na koji način generirati nasumične populacije, kojih je sada više te svaka od njih ima svoj domenski prostor.

Korištenje matematičkih aparata (u ovom slučaju entropije) u sklopu algoritma svakako daje daleko kvalitetnije rezultate i točniju klasifikaciju u odnosu na jednostavnije metode, kao što je slučaj u BasicML-u. Međutim, ovakav način rada donosi i nedostatke u smislu kompliciranije implementacije te veće složenosti. Minimalna složenost samo jednog rekurzivnog poziva je $O(n^2 \times d)$, jer algoritam u jednom pozivu mora obići sve atribute, za svaki atribut sve objekte, te za svaki par [objekt, atribut] mora proći kroz sve objekte kako bi našao entropiju. Stoga je u implementaciji ovog algoritma jako važno pametno postaviti specijalne slučajeve i prečace kojima se može smanjiti složenost.

4. Genetski algoritmi

Genetski algoritmi predstavljaju veliku skupinu algoritama inspiriranih Darwinovom teorijim evolucije. Skup ulaznih podataka naziva se populacijom jedinki, gdje jedinke predstavljaju objekte. Svaka jedinka ima i gene, koji su zapravo atributi objekta, te dobrotu, koja se u slučaju optimizacije funkcije računa na isti način kao kod strojnog učenja. Svaki genetski algoritam se svodi na konvergenciju ka najboljem mogućem rješenju pomoću manipulacije genima (atributima) te elitističkim odabirom (odabirom najboljih jedinki). Novostvorene ili duplicirane jedinke čine osnovu za novu populaciju, novu generaciju, koja je u pravilu poboljšana u odnosu na prethodnu te predstavlja nove ulazne podatke za algoritam. Algoritam ponavlja radnje sve dok najbolji član populacije ne zadovolji unaprijed definirano pravilo po kojemu se zaključuje da je upravo on rješenje problema. Slično kao i u strojnom učenju, i ovdje se definira prag poboljšanja pri kojemu algoritam prestaje s iteracijama.

Sljedeća slika prikazuje opće načelo rada jedne iteracije genetskog algoritma:



Slika 4-1: Opće načelo rada genetskog algoritma

Ulazni podaci su jednaki kao i za strojno učenje, dakle, populacija ulaznih podataka (objekata). Unutar populacije poznata je dobra poddpopulacija, iz koje se uglavnom uzimaju članovi za daljnju obradu. Ipak, postoje i situacije kada je možda poželjno uzeti nekog člana koji nije među najboljima, npr. kada je potrebno dovesti novi genetski materijal u novostvorenu populaciju.

Ključan faktor algoritma je stoga upravo odabir jedinke i genetski algoritmi se međusobno razlikuju ponajviše u načinu odabira: bira li se nasumični član, najbolji član, neka određena skupina članova i slično.

Nakon odabira jedinke, bira se genetski operator. Operator se najčešće bira slučajnim odabirom, pri čemu je unaprijed zadana vjerojatnost okidanja pojedinog operatora. Najčešći operatori su:

- **Replikacija** – Nepromijenjena jedinka se duplicira u novu jedinku, koja je istovjetna kopija originala. Ova operacija se uglavnom provodi isključivo nad jako kvalitetnim jedinkama, kako bi se zadržao dobar genetski materijal u sljedećoj populaciji. Replikaciju je potrebno izvoditi s malenom vjerojatnošću (otprilike 5 do 15%) kako evolucija ne bi prerano došla u fazu stagnacije. Ekstremni slučaj je da populaciju sačinjavaju sve kopije koje su istovjetne te se samim time evolucija ne može nastaviti, jer nema novog genetskog materijala.

- **Mutacija** – Nasumično odabranom atributu (ili više njih) nasumično se mijenja vrijednost kako bi se doveo novi genetski materijal u novu jedinku. Nova jedinka je dakle nasumično mutirana jedinka originala. Ovaj operator valja koristiti s oprezom, jer se može dogoditi neželjena situacija da se kvalitetna jedinka mutacijom „pokvari“ te samim time degradira buduću populaciju, stoga se i ovaj operator okida s manjom vjerojatnošću (5 do 10%). Ukoliko se događa previše mutacije u procesu evolucije, može se dogoditi ekstremni slučaj u kojem su i najbolje jedinke zapravo previše loše da bi mogle pospješiti evoluciju. Postoje razni načini na koje se mutacija može izvesti, no svi oni uključuju nasumičnu promjenu vrijednosti atributa.

- **Križanje** – Za operaciju križanja potrebne su barem dvije jedinke, stoga je u slučaju odabira ovog operatora, potrebno odabrati još jednu jedinku. Stvara se nova jedinka, čiji atributi poprimaju neke vrijednosti od jednog roditelja, a neke od drugog. Budući da križanje uključuje najmanje dvije jedinke, samim time se otvara mnoštvo načina kako izvesti križanje. U križanje je moguće uključiti i malen faktor mutacije, kako bi se doveo novi genetski materijal u novu jedinku. Ovaj operator se koristi s najvećim postotkom vjerojatnosti (60 – 80%), jer je on zapravo temelj funkcioniranja evolucije jedinki. Ostali

operatori koriste kao potpora ovom operatoru: replikacija čuva kvalitetan genetski materijal, dok mutacija uvodi novi, idealno neviđeni genetski materijal.

Nakon izvršavanja jednog od operatora, stvorena je nova jedinka. Postoje dvije velike skupine genetskih algoritama, te ovisno o tome kojoj algoritam pripada, određeno je što se s tom jedinkom sljedeće događa. Genetski algoritmi mogu biti:

- **Generacijski** – Algoritam od novostvorenih jedinki stvara novu populaciju, koja predstavlja sljedeću generaciju populacije. U sljedećoj iteraciji, ukoliko je potrebna, algoritam može novostvorenom generacijom zamijeniti staru populaciju te ponoviti postupak evolucije nad novim naraštajem. Potrebno je puniti novu populaciju na način da sadrži jednak broj jedinki kao i stara populacija. Dakle, algoritam ponavlja postupak odabira jedinki i operatora te stvaranja novih jedinki sve dok ne napuni novu populaciju. Kada je nova populacija napunjena, jedna iteracija algoritma je gotova.

- **Eliminacijski** – Algoritam ne stvara novu populaciju, nego sve promjene obavlja nad starom populacijom. Populacija je u ovom slučaju uvijek ista u smislu prostora – algoritam praktički samo mijenja sadržaj populacije koristeći genetske operatore i odabir jedinki. Pri radu s jednom populacijom algoritam mora održavati veličinu populacije, uglavnom na način da ili radi operacije nad postojećim jedinkama (bez stvaranja novih) ili stvaranjem novih jedinki i brisanjem starih jedinki. Ispunjavanjem nekog unaprijed zadanog uvjeta, algoritam prestaje s jednom iteracijom (unutar koje je mogao izvršiti mnoštvo operacija).

Nakon odrađenih operacija, algoritam uzima novu generaciju objekata kao ulazne podatke (ako je algoritam generacijski), odnosno, postojeću izmjenjenu populaciju (ako je algoritam eliminacijski) te ponavlja postupak, ukoliko ne postoji jedinka koja zadovoljava pravilo završetka algoritma.

U okviru završnog rada, implementirane su dvije varijante genetskog algoritma, od kojih je jedna generacijska, a druga eliminacijska.

4.1. Jednostavni „rulet“ generacijski algoritam

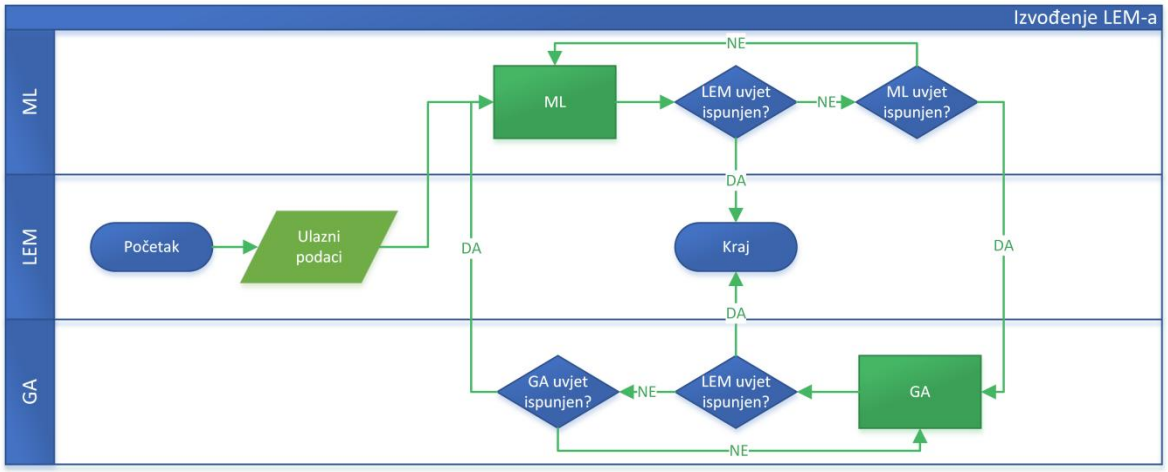
Ideja jednostavnog algoritma (engl. *roulette-wheel*) je nasumičan odabir jedinki te nasumično provođenje genetskog operatora nad njima sve dok se ne popuni nova populacija. Vjerojatnost odabira jedinke proporcionalna je dobroti jedinke, stoga bolje jedinke imaju veću vjerojatnost odabira od lošijih, dok je vjerojatnost provedbe operatora unaprijed određena kao što je ranije opisano.

4.2. 3-turnirski eliminacijski algoritam

Turnirski algoritam nasumično odabire određen broj jedinki (u ovom slučaju tri jedinke) iz populacije. Najlošija od njih se briše iz populacije, a dvije najbolje se križaju kako bi nastala nova, treća jedinka. Postupak se ponavlja *n* puta, odnosno, onoliko puta koliko je velika populacija. Valja primjetiti da postoji mogućnost da se u jednoj iteraciji algoritma novonastala jedinka križa s nekom drugom, pa opet novonastala s nekom trećom itd. Zbog navedenog, moguć je scenarij gdje novonastala jedinka nije direktan potomak neke druge jedinke, već njen *n*-ti potomak. Ovakav scenarij nije moguć u jednostavnom algoritmu – u potonjem, svaka novonastala jedinka je točno direktni potomak stare jedinke. Time ovaj algoritam može polučiti bržu konvergenciju od jednostavnog algoritma.

5. Learnable Evolutionary Model (LEM)

Tema rada je upravo spoj strojnog učenja i genetskog algoritma unutar okvira nazvanog Learnable Evolutionary Model (Model evolucijskim učenjem), skraćeno LEM. Slijedi dijagram toka koji ugrubo opisuje rad LEM-a:



Slika 5-1: Dijaram toka rada LEM-a

LEM kao ulaz prima unificirani oblik ulaznih podataka, jednak onome kakav primaju strojno učenje i genetski algoritam. Podaci se prvotno proslijeđuju algoritmu strojnog učenja koji sužava domenski prostor populacije i generira novu, nasumičnu populaciju unutar novog prostora / novih prostora. Nakon toga ispituje se uvjet za izlazak iz LEM-a: uspoređuje se poboljšanje najbolje jedinke populacije u odnosu na najbolju jedinku iz prošle populacije. Ukoliko je poboljšanje iznad određenog praga, provjerava se je li poboljšanje iznad praga poboljšanja koji je definiran za strojno učenje. Ukoliko je ono i iznad praga izlaska iz strojnog učenja, postupak se ponavlja, dakle novostvorena populacija s novim domenskim prostorima šalje se opet na strojno učenje.

Kada je poboljšanje ispod praga strojnog učenja, dolazi se do zaključka da strojno učenje (barem zasad) ne može više efikasno poboljšati populaciju. Stoga se izlaz iz strojnog učenjate te se najnovija populacija proslijeđuje na genetski algoritam. Genetski algoritam se ponaša na jednak način, što se tiče odlučivanja, kao i strojno učenje: ukoliko nije ispunjen niti prag LEM-a niti prag genetskog algoritma, postupak se ponavlja.

Konačno, u slučaju kada je poboljšanje jedinke ispod praga LEM-a, bilo nakon izvršavanja strojnog učenja ili genetskog algoritma, LEM završava s radom – pretpostavlja se da se ne može doći do daljnjeg bitnog poboljšanja te je rješenje upravo najbolja jedinka iz populacije.

Prag poboljšanja za LEM se definira nekoliko redova niži od pragova za strojno učenje i genetski algoritam. Naime, prag poboljšanja LEM-a na neki način definira dozvoljeno očekivano odstupanje rješenja od stvarnog rezultata. Ukoliko bi ovaj prag bio veći od praga strojnog učenja, LEM se ne bi nikad prebacio u način rada genetskog algoritma. Slično, ukoliko bi bio veći od praga genetskog algoritma, bio bi ostvaren najviše jedan ciklus strojnog učenja i genetskog algoritma.

Strojno učenje i genetski algoritam predstavljaju doista različite koncepte pristupa rješavanju sličnih problema. Upravo zbog toga je kombinacija ove dvije metode dobitna. Slijedi opis utjecaja obje metode na rad LEM-a.

5.1. Uloga strojnog učenja

Strojno učenje se pri optimizaciji funkcija koristi praktički samo za sužavanje domenskih prostora. Zbog toga, strojno učenje niti ne daje direktno rješenje problema, jer bez nasumične regeneracije populacije, početno stvorena populacija bi morala već sadržavati traženo rješenje problema. U tom slučaju, upotreba bilo kakvih algoritama bi bila besmislena.

Sužavanje domenskih prostora može imati za posljedicu preuranjenu nemogućnost pronalaženja konačnog rješenja. Može se dogoditi da su podaci takvi da strojno učenje suzi domenski prostor tako da ne obuhvaća više dio prostora unutar kojeg je traženo rješenje. U tom slučaju, algoritam se nikako ne može „spasiti“, jer se novi domenski prostori generiraju isključivo unutar postojećih.

Ukoliko bi se problem optimizacije funkcije propustio samo kroz strojno učenje, očekivani idealni rezultat bi bio da se nakon određenog broja generacija domenski prostori toliko suze da onemogućavaju poboljšanje iznad zadanog praga. Drugim riječima, sva generirana rješenja bi bila jako blizu jedni drugima i naposlijetku i blizu traženom rješenju.

Algoritmi strojnog učenja su ustvari deterministički, ako se izuzme nasumično generiranje populacije. Zbog toga, strojno učenje ima prilično stabilan rad te analitičkim metodama dolazi do konačnog rješenja, odnosno, konvergira ka istom. Upravo taj determinizam je ujedno i glavna prednost i glavna mana strojnog učenja: s jedne strane osigurava matematički zasnovan i prilično siguran put ka rješenju, a s druge strane nije u stanju nositi se sa stohastikom koju unose stvarni podaci koje je potrebno optimizirati, odnosno klasificirati. Stohastika stvarnih podataka se, radi kvalitetnijeg testiranja, pokušava simulirati unutar samih optimizacijskih funkcija, kako bi se ponašanje optimizacijskih funkcija što više približilo ponašanju stvarnih podataka.

5.2. Uloga genetskog algoritma

Općenito gledano, genetski algoritam izuzetno uspješno rješava probleme. Međutim, za razliku od strojnog učenja, genetski algoritam ne vidi širu sliku problema – jedini indirektni dio šire slike je zapravo funkcija dobrote i uvid u dobru podpopulaciju. Genetski algoritam se ne bavi domenskim prostorima niti ovisnošću dobrote o atributima.

Genetski algoritam dolazi do izražaja upravo nakon odrađenog strojnog učenja, odnosno, kada strojno učenje više ne može dovoljno poboljšati domenski prostor. Tada genetski algoritam već ima dosta dobru i unaprijeđenu populaciju, odnosno, izuzetno sužene domenske prostore unutar kojih se nalazi nasumično generirana početna populacija. S takvim dobrim predispozicijama, genetski algoritam će brzo konvergirati, bez „lutanja“ izvan već suženih domenâ.

Ono što genetski algoritam donosi novog u odnosu na strojno učenje je upravo nedeterminizam. Pomoću nasumičnosti, osigurat će se gubitak pretjeranog fokusa, koji je mana strojnog učenja. Križanjem i repliciranjem dolazit će se do novih, poboljšanih rješenja, do kojih se dolazi (genetski gledano) prilično usmjerenim putem, a mutacijom će se osigurati da LEM ne zablokira unutar lokalnih optimuma.

Nadalje, velika prednost u odnosu na strojno učenje je direktno ili indirektno čuvanje dobrih jedinki. Dok strojno učenje stalno nanovo generira nove, neviđene jedinke, genetski algoritam iz postojećih jedinki gradi nove, tako da se čuva genetski materijal dobrih jedinki.

Nakon odrađenog genetskog algoritma, ukoliko se ne nađe rješenje LEM-a, izvršavanje se ponovno prebacuje na strojno učenje. Strojno učenje će kao prvu generaciju iskoristiti evoluiranu populaciju iz genetskog algoritma te stvoriti nove sužene domenske prostore. Upravo je, dakle, genetski algoritam pomogao strojnom učenju da preskoči prepreku na koju je naletio te nastavi dalje s radom.

6. Programsko ostvarenje

Kao što je spomenuto ranije, ideja završnog rada je implementirati aplikaciju koja će omogućiti korištenje LEM-a te proizvoljnih algoritama strojnog učenja i genetskog algoritma kako bi se moglo doći do analize i zaključaka vezanih za LEM, strojno učenje, genetski algoritam, a i samu implementaciju svega navedenog.

Ideje vodilje pri izradi aplikacije bile su:

- intuitivno grafičko sučelje,
- jednostavno korištenje osobama slabijih tehničkih znanja (*user-friendly*),
- što je moguće brže izvođenje aplikacije, budući da je optimizacija funkcije memorijski i procesorski izuzetno zahtjevan proces,
- grafički prikaz podataka do kojih se dolazi tokom izvođenja,
- jednostavan odabir algoritama koji se koriste pri samom izvođenju,
- jednostavna nadogradnja aplikacije novim genetskim algoritmima i algoritmima strojnog učenja.

Jezik odabran za implementaciju aplikacije je C# iz sljedećih razloga:

- prethodno iskustvo autora u ovom programskom jeziku,
- jaka objektno-orijentirana paradigma,
- odlično razvojno okruženje (Microsoft Visual Studio),
- izuzetno jednostavna izrada grafičkog sučelja,
- jednostavna podrška za višedretvenost,
- mnoštvo pretpostavljeno ugrađenih strukturâ.

6.1. Osnovna struktura aplikacije

Kao što je i sâm LEM strogo komponentan, tako je i aplikacija komponentno izvedena. Na najapstraktnijoj razini, aplikacija se sastoji od sljedećih podsustava:

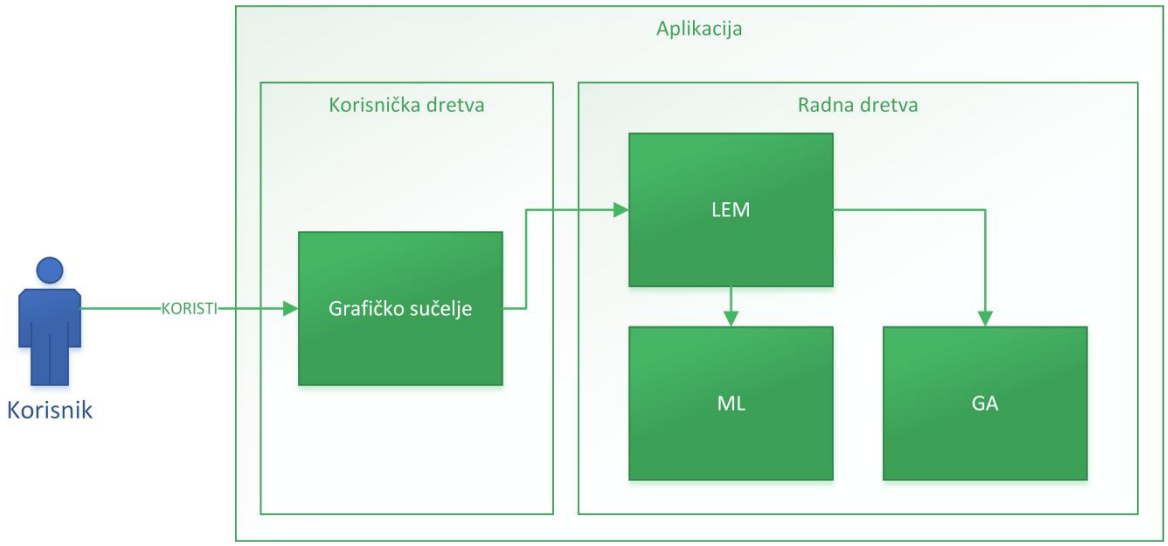
- **Grafičko sučelje** – Korisnik aplikacije ima izravan doticaj jedino s grafičkim sučeljem, kroz koje koristi aplikaciju. Vrlo je bitna višedretvenost pri implementaciji grafičkog sučelja kako bi korisnik imao dojam fluidnosti i responzivnosti aplikacije tokom izvođenja algoritma, koje može potrajati dosta dugo. Korisnik mora biti u mogućnosti zaustaviti i nastaviti izvođenje algoritma u bilo kojem trenutku nakon pokretanja. Također, potreban je aktualni ispis statusa izvršavanja algoritma te grafički prikaz poboljšanja i eventualno drugih parametara koji se mijenjaju tokom izvođenja. Između ostalog, korisnik može podešavati parametre algoritama, birati algoritme koji će se koristiti, kao i optimizacijske funkcije te njihove dimenzije.

- **LEM** – glavni podsustav koji upravlja strojnim učenjem, genetskim algoritmom, generiranjem nasumičnih populacija i ostalim detaljima vezanim za cjelokupni rad algoritma.

- **ML** – podsustav strojnog učenja. Mora imati dobro definiran okvir, kojeg onda moraju poštovati sve varijante strojnog učenja koje je moguće ugraditi u aplikaciju. S programerske strane, vjerojatno najzahtjevniji podsustav zbog najkompliciranijih izračuna u algoritmu C4.5.

- **GA** – podsustav genetskog algoritma. Relativno jednostavan podsustav budući da genetski algoritmi nisu implementacijski pretjerano zahtjevni.

Veze između navedenih podsustava mogu se opisati sljedećim prikazom:

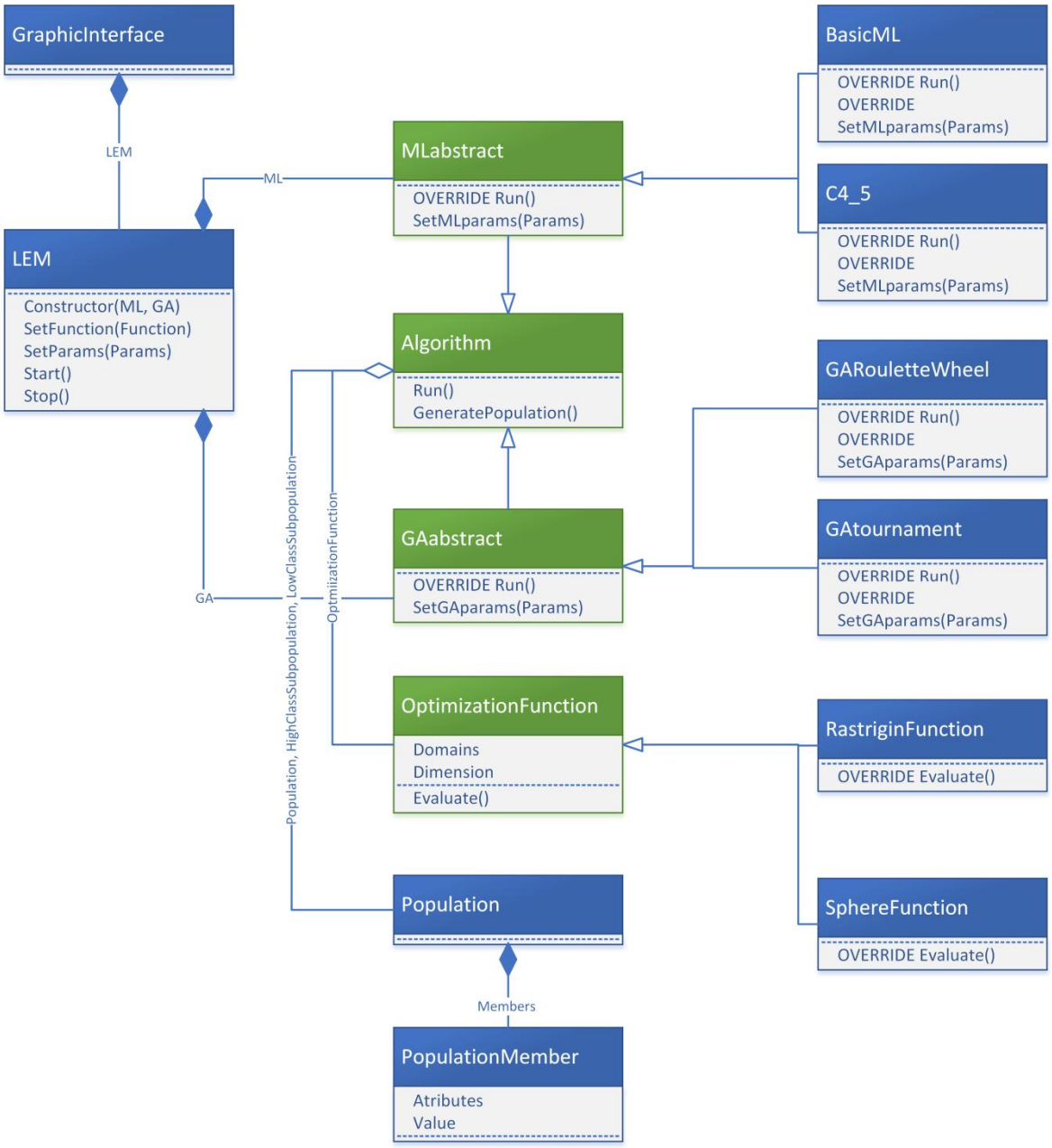


Slika 6-1: Podsustavi aplikacije

Kao što se vidi na prikazu, korisnik koristi samo grafičko sučelje. Potrebno je implementirati grafičko sučelje u takozvanoj korisničkoj dretvi, dok rad sustava treba biti odvojen u posebnu, radnu dretvu. Interakcijom korisnika s grafičkim sučeljem, grafičko sučelje prosljeđuje zahtjeve LEM-u, koji pak upravlja podsustavima strojnog učenja i genetskog algoritma. Tokom izvođenja algoritma, radna dretva je neovisna o korisničkoj, tako da je aplikacija responzivna. Ukoliko algoritam završi s radom, grafičko sučelje provjeravanjem stanja LEM-a saznaje da je algoritam gotov s radom te se korisnik obavještava o završetku rada i pronađenom rješenju.

6.2. Dijagram razreda

U dijagramu razreda, kako bi bio što čitljiviji, nisu prikazani svi implementacijski detalji. Slijedi okvirni prikaz dijagrama razreda aplikacije:



Slika 6-2: Dijagram razreda aplikacije

Na dijagramu razreda apstraktni razredi su obojani zelenom bojom, a konkretni plavom. Počevši od apstraktnih razreda, imamo sljedeće razrede:

- **Algorithm** – predstavlja osnovni okvir za algoritam, bio on iz kategorije strojnog učenja ili genetskog algoritma. Definira virtualnu metodu za pokretanje, koja je prazna i koju svaki konkretni razred mora implementirati. Uz nju, sadrži konkretnu metodu za stvaranje nasumične populacije. Budući da većina algoritama traži ovu funkcionalnost, ovim putem ona je ugrađena u sve

algoritme. Kao članske varijable, Algorithm sadrži populaciju, dobru podpopulaciju, lošu podpopulaciju i optimizacijsku funkciju.

- **OptimizationFunction** – predstavlja sučelje za optimizacijske funkcije. Sadrži virtualnu metodu Evaluate, koju svaka optimizacijska funkcija mora implementirati. Ova metoda ustvari za primljeni objekt (jedinku, vektor) računa vrijednost te indirektno i dobrotu. Članske varijable ovog razreda su Domains, koja sadrži domenske skupove te Dimension, koja čuva podatak o dimenziji objekata.

- **MLabstract** – definira sučelje za algoritme strojnog učenja. Može implementirati metodu za pokretanje algoritma tako da uključi radnje zajedničke svim algoritmima strojnog učenja. Uz to, sadrži virtualnu metodu SetMLparams za postavljanje parametara algoritma. Obje metode izvedeni razredi moraju implementirati.

- **GAabstract** – slično kao i MLabstract, definira sučelje za genetske algoritme.

Konkretni razredi, koji su nužni za rad aplikacije su:

- **GraphicInterface** – predstavlja grafičko sučelje koje korisnik koristi. Ovaj razred zapravo predstavlja glavnu Windows formu koju aplikacija koristi te sve funkcionalnosti koje ista nudi.

- **LEM** – središnji dio rada LEM-a. Pri konstruiranju objekta ovog razreda zadaju se algoritmi strojnog učenja i genetskog algoritma, a posebnim metodama se zadaju parametri i optimizacijska funkcija. Definirane su i posebne metode za pokretanje i zaustavljanje izvršavanja LEM-a, koje se pokreću korisničkim akcijama unutar grafičkog sučelja.

- **Population** – predstavlja populaciju objekata te ne sadrži nikakve posebne metode niti atribute.

- **PopulationMember** – predstavlja jednog člana u populaciji, odnosno objekt (u sučaju optimizacije funkcije, vektor realnih brojeva). On sadrži atribute i vrijednost.

Preostali razredi predstavljaju nadogradivi dio aplikacije. Tako se mogu u aplikaciju dodavati novi algoritmi strojnog učenja, genetski algoritmi te optimizacijske funkcije.

7. Analiza

U okviru analize cilj je usporediti rad LEM-a u odnosu na rad genetskog algoritma. Tokom analize u okviru ovog rada, svaka funkcija će se optimizirati deset puta na genetskom rulet algoritmu, a zatim na LEM-u, koji, uz navedeni genetski algoritam, koristi i algoritam strojnog učenja BasicML.

Sva ispitivanja su provedena s parametrima:

- Veličina populacije: 100
- Prag poboljšanja glavnog algoritma: 10^{-12} (genetskog algoritma u jednom slučaju, u drugom slučaju LEM-a)
- Udio dobrih jedinki: 0.2 (20%)
- Vjerojatnost mutacije: 0.05 (5%)
- Vjerojatnost replikacije: 0.10 (10%)
- Vjerojatnost križanja: 0.85 (85%)
- Prag poboljšanja sporednog algoritma pri radu LEM-a: 10^{-6}

7.1. Optimizacijske funkcije

Slijedi popis korištenih optimizacijskih funkcija, zajedno sa specifičnostima svake od njih.[1]

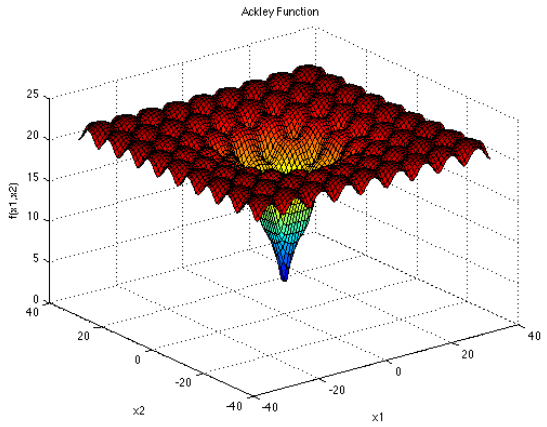
7.1.1. Ackleyeva funkcija

Ackelyeva funkcija je definirana za bilo koju dimenziju *d*, a posebna je po tome što sadrži velik broj lokalnih minimuma. Funkcija je definirana formulom:

$$f(x) = -20\left(-0.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)\right) + 20 + \exp(1)$$

[1] Preuzeto sa http://www.sfu.ca/~ssurjano/optimization.html, (09.06.2015.)

Na sljedećoj slici je prikaz funkcije uz $d = 2$.



Slika 7-1: Ackleyeva funkcija

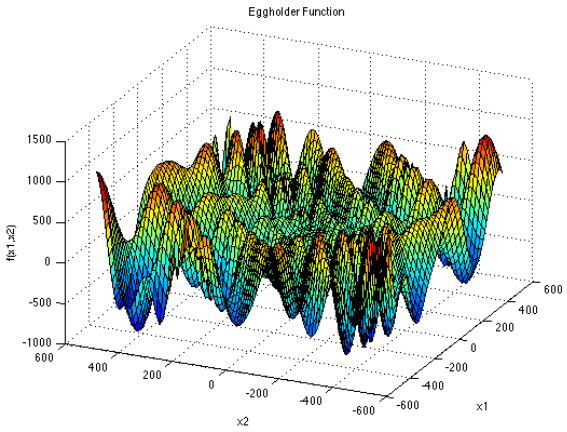
Za domenski prostor se najčešće uzima $x_i \in [-32.768, 32.768], i \in [1, d]$, no domenski prostor se može i suziti. Globalni minimum je u točki $x_i = 0, i \in [1, d]$ te iznosi $f(x) = 0$.

7.1.2. „Eggholder“ funkcija

„Eggholder“ funkcija je definirana za dimenziju 2. Također ima mnoštvo lokalnih minimuma, no u odnosu na Ackleyevu funkciju, oni su mnogo gušće raspoređeni te imaju veće raspone. Uz to, lokalni minimumi međusobno podjednakih vrijednosti raspršeni su po domenskom prostoru. Ova karakteristika znatno otežava rad algoritama, jer je konvergencija ka najboljem rješenju razbacana po domenskom prostoru, nije npr. omeđena na kružni vijenac kao što je to slučaj kod Ackleyeve funkcije. Funkcija je definirana formulom:

$$f(x) = -(x_2 + 47)\sin\left(\sqrt{\left|x_2 + \frac{x_1}{2} + 47\right|}\right) - x_1 \sin\left(\sqrt{|x_1 - (x_2 + 47)|}\right)$$

Na sljedećoj slici je prikaz funkcije.



Slika 7-2: "Eggholder" funkcija

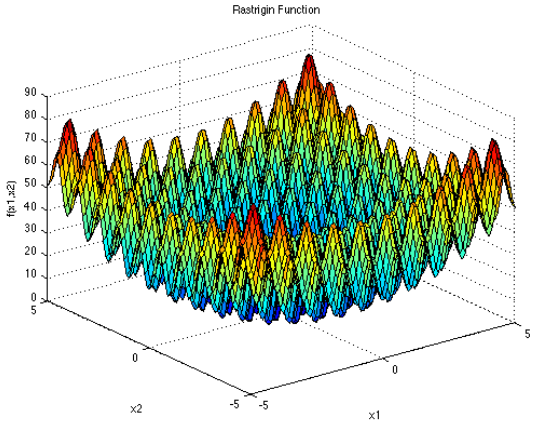
Za domenski prostor se najčešće uzima $x_i \in [-512, 512], i \in [1,2]$. Globalni minimum je u točki $x_{1,2} = (512, 404.2319)$ te iznosi $f(x) = -959.6407$.

7.1.3. Rastrigin funkcija

Rastrigin funkcija je definirana za bilo koju dimenziju *d*. U odnosu na Ackleyevu funkciju ima veće amplitude optimuma, kao i „eggholder“ funkcija, no u odnosu na „eggholder“ funkciju, optimumi su pravilno raspoređeni. Funkcija je definirana formulom:

$$f(x) = -10d + \sum_{i=1}^{d} [x_i^2 - 10\cos(2\pi x_i)]$$

Na sljedećoj slici je prikaz funkcije uz $d = 2$.



Slika 7-3: Rastrigin funkcija

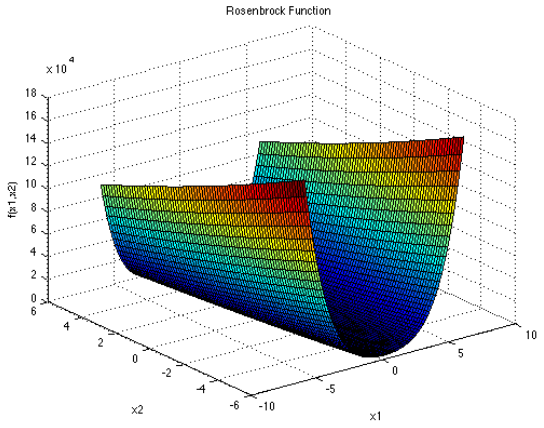
Za domenski prostor se najčešće uzima $x_i \in [-5.12, 5.12], i \in [1, d]$, no domenski prostor se može i suziti. Globalni minimum je u točki $x_i = 0, i \in [1, d]$ te iznosi $f(x) = 0$.

7.1.4. Rosenbrock funkcija

Rosenbrock funkcija je definirana za bilo koju dimenziju d. Posebnost funkcije nije mnoštvo lokalnih optimuma kao kod prethodnih funkcijâ, negoglatka zakrivljena ploha koja sadrži globalni minimum. Iako je jednostavno doći do same plohe, pokazuje se da je problematično iz plohe doći do globalnog minimuma, zbog spore konvergencije, odnosno, blagog nagiba. Funkcija je definirana formulom:

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

Na sljedećoj slici je prikaz funkcije uz d = 2.



Slika 7-4: Rosenbrock funkcija

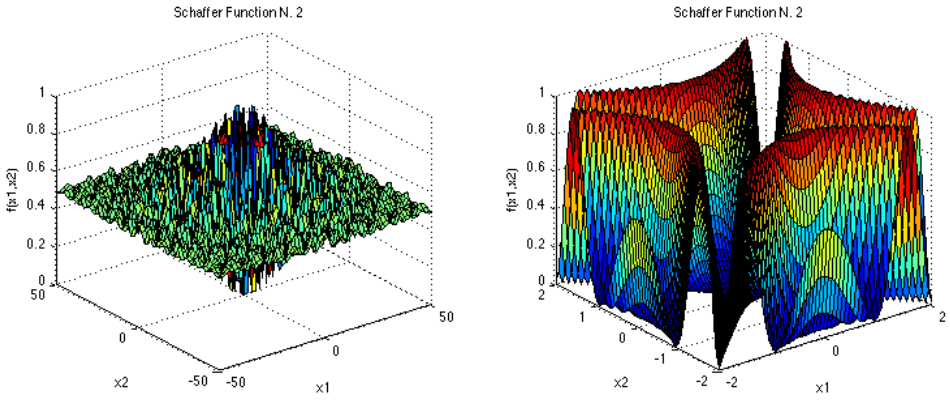
Za domenski prostor se najčešće uzima $x_i \in [-5, 10], i \in [1, d]$, no domenski prostor se može i suziti. Globalni minimum je u točki $x_i = 1, i \in [1, d]$ te iznosi $f(x) = 0$.

7.1.5. Schaffer funkcija №2

Schaffer funkcija №2 je definirana za dimenziju 2. Posebnost funkcije je mnoštvo lokalnih optimuma, a uz to, globalni minimum je ograđen globalnim maksimumima, čime je znatno otežana konvergencija algoritma. Funkcija je definirana formulom:

$$f(x) = 0.5 + \frac{\sin^2(x_1^2 - x_2^2) - 0.5}{[1 + 0.001(x_1^2 - x_2^2)]^2}$$

Na sljedećoj slici je prikaz funkcije, s prikazanim detaljima oko globalnog minimuma.



Slika 7-5: Schaffer funkcija №2

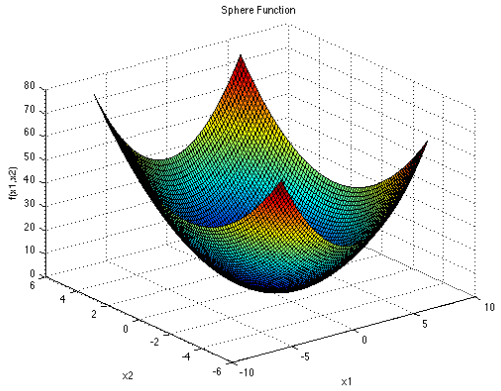
Za domenski prostor se najčešće uzima $x_i \in [-100, 100], i \in [1,2]$, no domenski prostor se može i suziti. Globalni minimum je u točki $x_i = 0, i \in [1,2]$ te iznosi $f(x) = 0$.

7.1.6. Sferna funkcija

Osnovna, jednostavna funkcija s kojom se algoritam prvo može ispitati je sferna funkcija. Ona podržava proizvoljan broj dimenzija. Funkcija je glatka te ima jedan optimum, koji je ujedno i globalni minimum. Funkcija je definirana formulom:

$$f(x) = \sum_{i=1}^{d} x_i^2$$

Na sljedećoj slici je prikaz funkcije.



Slika 7-6: Sferna funkcija

Za domenski prostor se najčešće uzima $x_i \in [-5.12, 5.12], i \in [1, d]$, no domenski prostor se može i suziti. Globalni minimum je u točki $x_i = 0, i \in [1, d]$ te iznosi $f(x) = 0$.

7.2. Analiza potrebnog broja generacija

Uspoređuju se LEM i genetski algoritam u kontekstu potrebnog broja generacija kako bi se dobilo rješenje.

Broj generacija pri izvršavanju LEM-a znatno je veći nego kod genetskog algoritma. Razlog tome je što je kod genetskog algoritma moguća (i česta) situacija da rano otkrivena relativno dobra jedinka ostaje najboljom neko vrijeme. Zbog toga se događa situacija u kojoj se javlja jako malo poboljšanje u odnosu na prethodnu generaciju, iako nije nužno istinito da je nađeno rješenje dovoljno dobro. Genetski algoritam zapinje u lokalnom minimumu i prestaje s radom, jer je poboljšanje premaleno. Ovaj efekt je najizraženiji kod Eggholder funkcije, kod koje genetski algoritam radi jako malo iteracija te ne nalazi kvalitetno rješenje.

S druge strane, pri izvršavanju LEM-a ovakav problem se izbjegava uvođenjem strojnog učenja koje smanjuje domenu rješenja. Nakon smanjenja domene, nove jedinke će biti drukčije raspoređene te će samim time genetski algoritam moći nastaviti s radom.

Ipak, kasnije će se pokazati da upravo strojno učenje može u nekim situacijama nanijeti i štetu prilikom pronalaska rješenja pomoću LEM-a u odnosu na genetski algoritam.

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	14	12	13	18	4	12
2	14	**24**	15	**9**	17	17
3	**5**	14	4	36	22	5
4	9	**3**	15	20	9	**32**
5	9	**3**	4	27	**3**	22
6	6	**3**	13	**41**	**36**	9
7	**54**	5	**3**	**9**	10	7
8	32	**3**	4	18	6	27
9	11	13	**51**	27	29	**3**
10	28	5	21	12	5	4
Prosjek	18,2000	8,5000	14,3000	21,7000	14,1000	13,8000

Tablica 1: Broj potrebnih generacija za dobivanje rješenja pri izvršavanju genetskog algoritma

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	**40**	57	98	58	80	30
2	42	**33**	60	50	112	**17**
3	43	77	47	79	90	35
4	**40**	58	32	**84**	75	68
5	48	98	87	64	**218**	21
6	55	**113**	59	81	81	21
7	46	91	**27**	74	**43**	22
8	**77**	83	**117**	75	119	67
9	70	68	48	**42**	112	**74**
10	48	39	80	72	109	68
Prosjek	50,9000	71,7000	65,5000	67,9000	103,9000	42,3000

Tablica 2: Broj potrebnih generacija za dobivanje rješenja pri izvršavanju LEM-a

7.3. Analiza vremena izvođenja

Vrijeme izvođenja je usko vezano uz broj iteracija. Kod genetskog algoritma se može reći da je ovisnost proporcionalna, no kod LEM-a to nije slučaj. Pri radu LEM-a veći broj iteracija ima za posljedicu dulje vrijeme izvođenja, no proporcionalnost nije strogo definirana kao kod genetskog algoritma. Usporedbom rezultata vidi se kako genetski algoritam troši podjednako vrijeme za svaku generaciju – otprilike 10 ms. Kod LEM-a je za svaku generaciju, ovisno o funkciji, potrebno od otprilike 3 ms u slučaju Schaffer №2 funkcije do 8 ms u slučaju sferne funkcije.

Zanimljivo je da je algoritmu potrebno, grubo govoreći, 2.5 puta više vremena za rad sa sfernom funkcijom u odnosu na Schaffer №2 funkciju, s obzirom na to da je sferna funkcija prilično jednostavna i bez lokalnih minimuma. Razlog leži u činjenici da je skup dobrih jedinki u slučaju Schaffer №2 funkcije puno uži nego što je to slučaj kod sferne funkcije, a direktna posljedica toga je brža konvergencija uslijed sužavanja domene.

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	135	116	119	196	33	121
2	137	**241**	145	93	239	185
3	**42**	139	31	396	217	41
4	81	24	148	228	81	**351**
5	85	22	31	290	**21**	242
6	48	**20**	120	**470**	**348**	88
7	**612**	41	**19**	**90**	96	65
8	326	21	33	191	51	293
9	101	137	**534**	297	277	**21**
10	301	39	214	126	44	34
Prosjek	186,8000	80,0000	139,4000	237,7000	140,7000	144,1000

Tablica 3: Potrebno vrijeme izvođenja za dobivanje rješenja pri izvršavanju genetskog algoritma

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	202	213	305	211	202	219
2	**184**	**115**	527	209	**428**	**78**
3	234	195	268	**566**	282	268
4	192	188	**150**	426	198	653
5	242	290	547	**170**	394	108
6	389	**462**	376	223	**162**	95
7	270	240	153	192	225	160
8	563	293	301	465	312	450
9	**579**	212	307	208	390	667
10	288	129	**559**	302	394	**719**
Prosjek	314,3000	233,7000	349,3000	297,2000	298,7000	341,7000

Tablica 4: Potrebno vrijeme izvođenja za dobivanje rješenja pri izvršavanju LEM-a

7.4. Analiza kvalitete pronađenog rješenja

Kao mjera kvalitete pronađenog rješenja uzeta je euklidska udaljenost između pronađenog najboljeg rješenja i poznatog rješenja problema (globalnog minimuma funkcije). U pogledu kvalitete rješenja, može se općenito reći da prosjek svih deset rješenja pronađenih pomoću LEM-a nije znatno bolji od prosjeka rješenja genetskog algoritma, odnosno, istog je reda veličine. Međutim, rješenja koja daje LEM imaju veću varijancu, stoga su samim time najbolja rješenja koja daje LEM znatno bolja od rješenja koja daje genetski algoritam.

Najveće poboljšanje od čak 12 redova veličine očituje se na Ackleyevoj funkciji, zatim slijede Rastrigin, Schaffer №2 funkcija i sferna funkcija. Ovim funkcijama su zajednička svojstva globalni minimum u ishodištu koordinatnog sustava te uz lokalne minimume (izuzev sferne funkcije) duga tangenta koja ih tangira i vodi ka globalnom minimumu.

Eggholder funkcija predstavlja najveći problem zbog mnoštva, na prvi pogled nepravilno posloženih minimuma. Drugi problem leži i u tome da se globalni minimum nalazi na samom rubu domene te zbog toga strojno učenje vrlo rano može izostaviti područje globalnog minimuma iz pretrage. Budući da strojno učenje sužava domenu, a uz to je BasicML prilično jednostavan algoritam, LEM pokazuje generalno lošije rezultate na ovoj funkciji u odnosu na genetski algoritam.

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	0,246306154	519,9185128	0,036889542	0,824441686	**0,296753818**	0,026710153
2	0,07500954	**997,8957659**	0,012983306	0,700577897	3,476839257	0,052951982
3	0,087355968	455,8081589	0,072137512	0,191303834	4,09314614	0,053704822
4	0,089517836	**65,57618705**	0,050307841	**2,884699294**	**7,519535092**	0,053840432
5	0,050055512	106,4221454	0,072137512	0,187433004	5,499315002	**0,011255262**
6	0,084127968	960,9305945	0,029006328	0,200664812	2,085911256	0,023712544
7	0,122236479	314,7864802	**0,087885252**	0,507394327	4,593153651	0,015034119
8	**0,031635466**	168,081156	0,08618026	0,229581707	7,358234498	0,047956947
9	0,097507314	453,5719383	0,025548651	**0,09244088**	4,419668285	0,048058291
10	**0,336572339**	188,1602954	**0,000620231**	1,13178631	6,538885344	**0,063622903**
Prosjek	0,1220	423,1151	0,0474	0,6950	4,5881	0,0397

Tablica 5: Udaljenost od optimalnog rješenja pri izvršavanju genetskog algoritma

i	Ackley	Eggholder	Rastrigin	Rosenbrock	Schaffer №2	Sphere
1	0,037538652	590,6593375	0,252966295	0,829893562	0,79141008	1,07482E-07
2	5,19321E-14	625,5536999	8,69238E-08	0,809254761	8,310997234	1,90175E-06
3	6,54613E-13	**396,3875915**	**0,995207854**	0,225576807	2,061857074	**3,29274E-08**
4	1,96466E-12	**964,9855419**	8,65922E-08	0,954924375	0,76574599	1,01687E-06
5	**4,70197E-14**	729,7682864	0,945021497	0,200570845	1,513604097	2,2466E-06
6	0,043424651	884,5458597	0,005158293	**0,080821661**	3,236039211	3,21372E-07
7	0,093644964	735,4579863	**5,47942E-08**	0,131687626	**1,12578E-06**	7,57922E-07
8	**0,151952497**	649,2613484	0,402772138	**2,925427912**	5,387452562	**0,141528358**
9	0,013132636	487,1345712	0,016052691	1,704192193	**9,473738218**	0,004080283
10	4,20175E-12	495,3764711	0,015397502	0,929404588	2,473533196	4,58583E-07
Prosjek	0,0340	655,9131	0,2633	0,8792	3,4014	0,0146

Tablica 6: Udaljenost od optimalnog rješenja
pri izvršavanju LEM-a

8. Zaključak

Koncept spajanja strojnog učenja i genetskog algoritma u jedan okvir učinio mi se izuzetno zanimljivim te sam se zato počeo baviti LEM-om još na kolegiju „Projekt iz programske potpore“. Ipak, tokom izrade projekta, nisam nailazio na probleme na koje sam naišao tokom izrade završnog rada, što je i za očekivati, jer je projekt ipak bio manjeg opsega i težine od završnog rada.

Pri izradi Projekta, ideja je bila implementirati osnovne, što jednostavnije algoritme strojnog učenja i genetskog algoritma u LEM tek toliko da se implementira osnovni LEM. U slučaju završnog rada, situacija je nešto složenija, jer je cilj bio implementirati dobro poznate algoritme, provesti analizu i donijeti zaključke.

8.1. Implementacija

Genetski algoritmi ne predstavljaju pretjerani implementacijski problem. No, najveće probleme je uvela implementacija algoritma C4.5, koja je izuzetno zahtjevna (barem u odnosu na ranije implementirani BasicML). Naime, gledano sa stajališta razvoja aplikacije, potrebno je u svakom koraku implementacije razmišljati o optimizaciji i što efikasnijem izvođenju, jer je algoritam prilično vremenski složen. Uz to, algoritam C4.5 ima sposobnost podjele domene u poddomene, što je uzrokovalo potrebu za *refactoringom* gotovo cijele aplikacije. Poddomene su uvele i problem generiranja nasumičnih populacija, kojih je sada više i koje je potrebno „pošteno“ provesti s obzirom na veličine pojedinih poddomena. Općenito, svi algoritmi implementirani prije implementacije C4.5 radili su s jednim domenskim prostorom – sada moraju raditi s više domenskih prostora. Ovime je upravo, između ostalog, došla do izražaja važnost dobrog dizajna arhitekture aplikacije te razmišljanja o mogućim budućim promjenama.

Sljedeći problem se javio pri radu s negativnim vrijednostima. Naime, prvotno se dobrota računala kao recipročna vrijednost funkcije za određenu jedinku. Uvođenjem Eggholder funkcije postale su moguće i negativne vrijednosti te je time dotadašnji sustav izračuna dobrote postao netočan. Dobrota je naposlijetku implementirana kao negativna vrijednost funkcije za određenu jedinku, pri čemu valja voditi računa o ukupnoj dobroti populacije. Naime, pri odabiru genetskog

operatora potrebno je pronaći raspon dobrotâ jedinki, koji se onda dobiva tako da se ukupna dobrota umanji za umnožak dobrote najlošije jedinke i broja jedinki u populaciji. Time se dobrota postavlja u domenu pozitivnih realnih brojeva.

8.2. LEM u odnosu na GA

LEM predstavlja bitno poboljšanje u odnosu na genetski algoritam zbog ranije navedenih razloga. Ipak, bitna stavka unutar LEM-a je odabir strojnog učenja, koji, ukoliko je prejednostavan, ne donosi nikakvu korist, dapače, moguć scenarij je i dobivanje lošijih rješenja i to uz vremenski dulje izvođenje algoritma.

Algoritam C4.5 pokazao se izuzetno dobrim, međutim, zbog svoje kompleksnosti, izuzetno je problematično implementirati ga unutar LEM-a, jer se često događa scenarij u kojem postoji mnoštvo populacija s premalo jedinki po populaciji. U takvom scenariju, genetski algoritam ne može poboljšati populacije jer je premalen domenski prostor, kao i broj jedinki unutar populacije.

Literatura

1. Michalski, Ryszard S. Learnable evolution model: Evolutionary Processes Guided by Machine Learning. http://link.springer.com/content/pdf/10.1023%2FA%3A1007677805582.pdf (01.03.2015)
2. Prikaz rada GA. http://www.zemris.fer.hr/~yeti/studenti/GA.ppt (23.03.2015.)
3. Stokić Ivana. Primjena genetskog programiranja u strojnom učenju. Diplomski rad. Fakultet elektrotehnike i računarstva Sveučilište u Zagrebu, 2011.
4. Treća domaća zadaća iz kolegija Analiza i projektiranje računalom. http://www.fer.unizg.hr/_download/repository/vjezba3.pdf (19.03.2015.)
5. Virtual Library of Simulation Experiments: Test Functions and Datasets. http://www.sfu.ca/~ssurjano/optimization.html (09.06.2015.)

Optimizacija evolucijskim učenjem

Sažetak: Ovaj rad uvodi u problematiku optimizacije funkcije te obrađuje genetske algoritme i algoritme strojnog učenja. Opisuju se osnovni koncepti pri radu sa strojnim učenjem te osnovne genetske operacije koje se koriste kod genetskih algoritama. Obrađen je algoritam LEM, kao okvir za izmjenično korištenje strojnog učenja i genetskog algoritma. Obrađene su optimizacijske funkcije te su opisane specifičnosti vezane za svaku od njih. Opisani su implementacijski detalji razvijene aplikacije te je provedena analiza nad algoritmima i funkcijama u vidu usporedbe rada LEM-a i rada genetskog algoritma.

Ključne riječi: Genetski algoritam, strojno učenje, LEM, optimizacija funkcije, genetski operatori

Optimization by learnable evolution

Abstract: This work describes the problem of function optimization and processes genetic algorithms and machine learning algorithms. It describes the basic concepts concerning machine learning and basic genetic operations used in genetic algorithms. Algorithm LEM is described and explored as a framework for the alternating use of machine learning and genetic algorithm. The work discusses the optimization functions and describes the specific circumstances relating to each of them. Implementation details of developed application are described as well as the analysis of algorithms and functions in a comparison of the LEM and a genetic algorithm.

Keywords: Genetic algorithm, machine learning, LEM, optimization functions, genetic operators