ELEKTROTEHNIŠKI VESTNIK 87(4): 202–208, 2020
IZVIRNI ZNANSTVENI PRISPEVEK

Slovnična evolucija v elektrotehniki – posebnosti in uporabe

Matevž Kunaver

Laboratorij za računalniške metode v elektroniki, Univerza v Ljubljani, Fakulteta za elektrotehniko, Trzčaška cesta 25, 1000 Ljubljana, Slovenija
E-pošta: matevz.kunaver@fe.uni-lj.si

Povzetek. Slovnična evolucija je tehnika s področja evolucijskih algoritmov, ki se lahko uporablja na raznolikih področjih - od računske optimizacije do avtomatske sinteze vezij. Ponuja zelo prilagodljiv pristop k reševanju problemov saj zlahka spremenimo nabor funkcij in spremenljivk, ki se pri tem uporabijo. Tehnika poleg tega omogoča tudi enostavno integracijo specialističnega znanja, s čimer dodatno pospešimo pot do rešitve. Cilj članka je bralcu predstaviti osnove te tehnike, njene zahteve in posebnosti, poleg tega bralcu prikaže tudi dva praktična primera uporabe.

Ključne besede: optimizacija, slovnična evolucija, evolucijski algoritmi, genetsko programiranje

Grammatical Evolution in electrotechnics - requirements and use cases

Grammatical eovlution is an optimization technique that can be easily applied to different fields - from computational optimization to automatic circuit synthesis. It is an extremely flexible technique that can be quickly modified to incorporate different functions, variables and domain knowledge. The aim of this article is to present the method and its requirements to the reader as well as presenting two practical use cases.

Keywords: optimization, grammatical evolution, evolutionary algorithms, genetic programming

1 UVOD

Slovnična evolucija (SE, angl. Grammatical Evolution) je hitro razvijajoča se optimizacijska tehnika s področja *evolucijskih algoritmov* (EA, angl. Evolutionary Computation) ki ponuja visoko stopnjo prilagodljivosti in se lahko uporabi tako na področju priporočilnih sistemov [1] kot avtomatskega načrtovanja vezij [2]. Ta članek predstavlja njene posebnosti in zahteve ter dva praktična primera njene uporabe.

Evolucijski algoritmi so prvič omenjeni v 60. letih, ko sta Fogel [3] in Holland [4] razvila novo metodo, ki je danes znana kot genetski algoritem. Posebnost te metode je v tem, da za reševanje problemov uporablja 'naravne' pristope iz teorije evolucije - mutacijo, selekcijo in križanje. Na metode s področja evolucijskih algoritmov gledamo kot na iterativne metode, ki stremijo k temu, da najdejo najboljšo rešitev s pomočjo manipulacije množice osebkov. Vsak osebek je potencialna rešitev, ki se uporabi za reševanje določenega problema. Pristop nato naredi izbor tistih osebkov, ki so se najbolje odrezali, in iz njih sestavi naslednjo množico potencialnih rešitev. Na ta način EA pristop posnema teorijo evolucije - preživijo samo najmočnejši oziroma tisti, ki so najbolj primerni za reševanje danega problema.

Velika prednost pristopa EA je v njegovi nepredvidljivosti (vsaka potencialna rešitev je naključno generirana, posamezne rešitve lahko naključno mutirajo itd.), zaradi katere pogosto najde rešitev oziroma pot do rešitve, ki bi jo drugače spregledali ali pa sploh nikoli našli. Seveda je to tudi dvorezni meč - iz enakega razloga nimamo dobrege nadzora nad tem, kako bo postopek tekel, in lahko končamo v lokalnem minimumu. Vendar lahko to slabost odpravimo s pravilno nastavljenimi parametri pristopa in večkratnim zagonom pristopa za isti problem.

Prva raba tehnik EA (danes znana kot *genetski algoritem* (GA, angl. Genetic Algorithm) [4]) se je osredotočila na številske vrednosti - iskanje kombinacije številk s pomočjo katere dobimo najboljšo vrednost kriterijske funkcije(glej 2.5). Z rastjo računske moči so znanstveniki začeli EA uporabljati tudi za reševanje bolj kompleksnih problemov in hkrati razvili bolj prilagodljiv pristop. Novi pristop je dobil oznako *genetsko programiranje* (GP, angl. Genetic Programming) [5] in je lahko poleg številk ustvaril tudi čisto nove dele programa - enačbe, kriterije in funkcije. Genetsko programiranje je tako občutno povečalo stopnjo prilagodljivosti pri reševanju problemov in omogočilo, da se je pristop uporabil na več različnih področjih. Kljub temu pa je bilo treba pri njegovi rabi upoštevati še nekaj omejitev, med njimi izstopa omejitev, da morajo biti uporabljene funkcije med seboj kompatibilne - vsaka funkcija mora sprejeti rezultat katerekoli druge funkcije. Posledica te omejitve je, da v večini primerov GP deluje z enim samim podatkovnim tipom (npr. samo s celimi 16-bitnimi števili).

Prejet 26. maj, 2020
Odobren 20. avgust, 2020

Slovnična evolucija [6] te omejitve odpravi z uvedbo proizvodnih pravil, s pomočjo katerih sestavi potencialne rešitve. Vsako pravilo točno opredeli funkcijo ter njene vhodne in izhodne podatke. Tehnika torej za vsako funkcijo ve, katere vhodne/izhodne podatke lahko uporabi, in tako poljubno kombinira podatkovne tipe in funkcije. Poleg tega pravila omogočajo enostavno integracijo specialističnega znanja, zunanjih funkcij, knjižnic in simulatorjev.

V drugem poglavju poglavju prispevka nato predstavimo tehniko SE, njeno slovnico, nastavitve in evolucijske pristope. V tretjem poglavju predstavimo dva primera praktične uporabe tehnike SE, v četrtem pa podamo zaključne misli ter priporočila uporabi te tehnike.

2 SLOVNIČNA EVOLUCIJA

Slovnična evolucija za svoje delovanje potrebuje kar nekaj nastavitev in priprav. V osnovi tehnika deluje tako, da pripravi nabor (generacijo) potencialnih rešitev (osebkov), jih ovrednoti s kriterijsko funkcijo in na podlagi rezultatov pripravi naslednjo generacijo s pomočjo evolucijskih tehnik. To se nadaljuje toliko časa, dokler ne najdemo najboljše rešitve ali ne sproduciramo vnaprej nastavljenega števila generacij. Bistvo tehnike SE pa sta zbirka pravil za sestavljanje potencialnih rešitev in kriterijska funkcija njihovega vrednotenja. Zbirka pravil se imenuje tudi Slovnica, od koder izhaja tudi ime te tehnike. Celotni potek tehnike SE povzamemo kot:

1) Inicializacija sistema: naložimo slovnice in pripravimo kriterijsko funkcijo.
2) Za vsako generacijo: pripravimo izbrano število potencialnih rešitev določenega problema - osebkov. Vsak osebek je predstavljen kot zaporedje naključno generiranih celih števil - kromosomov.
3) Za vsak osebek: osebek ovrednotimo s pomočjo kriterijske funkcije. To pomeni, da osebkovo zaporedje kromosomov prevedemo v problemu primerno obliko (enačba, podvezje, programski ukazi) in ga uporabimo v postopku reševanja problema. Rezultat kriterijske funkcije je osebkova ustreznost - boljša ko je ta vrednost, večja je možnost, da bo osebek postal del naslednje generacije.
4) Ko so vsi osebki ovrednoteni: izberemo delež najboljših osebkov in jih kopiramo v naslednjo generacijo. To potem napolnimo do konca s pomočjo evolucijskih tehnik (mutacija, križanje itd.).
5) Ponavljamo korake od 2 do 4, dokler ne pridemo do zadnje generacije (skladno z nastavitvami).
6) Ponudimo najboljše osebke zadnje generacije kot optimalno rešitev za problem.

Medtem ko SE nudi zagotavlja visoko stopnjo prilagodljivosti pri iskanju rešitve problema, je njegova slabost (in slabost večine drugih pristopov s področja EA) v tem, da zahteva veliko računske moči in časa, saj vrednotenje posameznega osebka pomeni reševanje celotnega problema (in to ponovimo za vsak osebek v vsaki generaciji).

starting symbol	x = <-par->
<-par->	<-exp->|<-n->
<-exp->	(<-par->+<-par->) |(<-par->-<-par->)
<-n->	0 |1 |2 |3 |4 |5

Tabela 1: Primer Slovnice GE

2.1 Slovnica

Slovnica, ki jo uporabimo za pripravljanje osebka je sestavljena iz nabora pravil, ki se delijo v tri kategorije - začetni simbol ter prehodna in končna pravila. Pravila so podana v obliki Backus-Naur, kjer je vsako od njih predstavljeno kot element in njegove potencialne vrednosti.

Začetni simbol predstavlja začetno vozlišče, ki se razširi glede na preostala pravila. Lahko je zelo preprost in predstavlja en sam element (x =<-par->), lahko pa je tudi bolj kompleksen in na primer že na začetku določi največje mogoče število elementov v podvezju (x =<-part-><-part->....<-part->). Prehodna pravila podajajo funkcije, ki se lahko uporabijo pri reševanju problema (npr. vsota, produkt, deljenje in logaritem) in jih je treba dodatno razširiti, da nastavimo njihove vhodne parametre. Končna pravila pa vsebujejo vse elemente, ki jih ni več mogoče razširiti, in tako na tem mestu zaključijo enačbo - številke, spremenljivke in konstante.

Primer enostavne slovnice je podan v tabeli 1. Začetni simbol v tej tabeli je enostaven element <-par->. Pravilo za ta element dovoli SE, da se odloči, ali bo na to mesto postavila številko ali bo razvila bolj kompleksno enačbo s pomočjo ene izmed funkcij v tretji vrstici tabele. Elementa <-par->in <-exp->predstavljata prehodni pravili saj vidimo, da vsaka od njunih vrednosti zahteva še vsaj en korak do zaključitve (če bi v funkcijo vstavili samo številke). Element <-n->pa je končno pravilo saj lahko dobi samo številsko vrednost in se ne more dodatno razširiti.

2.2 Nastavitve pristopa SE

Pristop SE potrebuje kar nekaj nastavitev za pravilno delovanje:

- Število generacij - koliko generacij bomo ovrednotili. Večja ko je številka, več časa bo naš poskus potreboval. Tipična številka je med 50 in 500.
- Velikost generacije - koliko osebkov tvori posamezno generacijo. Pri manjši vrednosti je algoritem hitrejši, vendar je manj verjetno, da bomo našli dobro rešitev. Če je vrednost zelo velika, pa bomo potrebovali več časa za posamezno generacijo, vendar bo veliko bolj verjetno, da najdemo dobro rešitev. Tipična vrednost je med 150 in 300.
- Velikost elite - ta vrednost pove, koliko osebkov bomo iz trenutne generacije kopirali v naslednjo.

Elita je pomembna, ker predstavlja trenutno najboljše rešitve in se uporabi za izdelavo dela osebkov naslednje generacije. Če je vrednost prevelika, bomo v vsaki generaciji videli malo novih potencialnih rešitev in posledično bo pot do rešitve počasnejša. Tipična vrednost je med 5 in 10 odstotki velikosti generacije.

- Verjetnost mutacije - ko pripravljamo naslednjo generacijo se za vsak obstoječi osebek lahko zgodi, da mutira. Verjetnost tega dogodka je ponavadi nastavljena na 5 odstotkov.
- Število kromosomov - to število pove največje mogoče število elementov, ki jih osebek lahko vsebuje. Številka je ponavadi velika - 300 ali več.
- Največja globina - ta nastavitev določi, kdaj (najpozneje) SE preklopi na končna pravila in zaključi izdelavo osebka. Če bi pri naši slovnici iz tabele 1 določili največjo globino 2, bi lahko za osebek dobili enačbo ki ima največ 2 gnezdena oklepaja (na primer $((1+2)-(2+3))$ ali $((1+2)-3)$).

2.3 Osebki in kromosomi

Vsak osebek torej predstavlja potencialno rešitev določenega problema. Sestavljen je iz zaporedja kromosomov, ki se prevedejo v izrazno drevo z uporabo slovnice. Kromosom je naključno generirano celo število med 0 in 256. Dobljeno izrazno drevo se nato uporabi skladno z reševanim problemom - lahko ga prevedemo v del enačbe ali iz njega izgradimo podvezje SPICE.

Interpretacija kromosoma je odvisna od slovničnega pravila, ki je trenutno aktivno (glej primer spodaj), vendar se ponavadi izvede kot naključna izbira ene izmed možnih vrednosti s pomočjo deljenja po modulu. Če ima pravilo tako 3 možne vrednosti, uporabimo deljenje po modulu 3 in dobimo rezultat 0 (prva možnost), 1 (druga možnost) ali 2 (tretja možnost). Če ima torej pravilo dve možni vrednosti, je za vsako izmed njih polovična verjetnost izbire.

2.3.1 Primer izdelave osebka: Pokažimo torej primer izdelave osebka s pomočjo slovnice iz tabele 1. Za poskusni osebek uporabimo zaporedje kromosomov 12,150,33,45,23,4,67,86,90,212. Naš začetni simbol je <-par->. Iz tabele vidimo, da ima lahko dve različni vrednosti, zato prvi kromosom delimo po modulu 2 ($12\%2 = 0$) in skladno z rezultatom izberemo prvo možnost (<-exp->). Tudi ta element ima 2 vrednosti, torej postopek ponovimo za drugi kromosom ($150\%2 = 0$) in izberemo vsoto. Na tej točki se je naš začetni simbol iz <-par->razvil v (<-par->+<-par->). Zdaj moramo določiti vrednost prvega <-par->v oklepaju, za katerega imamo na voljo kromosom 33. Rezultat deljenja po modulu 2 ($33\%2 = 1$) določi, da izberemo element <-n->. Zanj porabimo kromosom 45 in modul 6 saj ima element <-n->6 možnih vrednosti. Rezultat ($45\%6 = 3$) pomeni, da izberemo vrednost 3 in izraz posodobimo v (3+<-par->).

Postopek nato nadaljujemo dokler niso vsi elementi v izrazu končna vozlišča (torej številke), ne porabimo vseh kromosomov ali ne dosežemo največje dovoljene globine izraza.

2.4 Evolucijske tehnike

Delo z osebki in generacijami je pomemben del vseh pristopov EA. Če ga izvedemo pravilno bomo lahko rešitev našli hitreje, v nasprotnem primeru to sicer ne pomeni, da rešitve ne bi našli, temveč bomo za isto rešitev porabili veliko več časa in računske moči. Orodja, ki so nam na voljo, so decimacija, izbor, mutacija, križanje in diverzifikacija.

2.4.1 Decimacija: Decimacija se izvede pri sestavljanju prve generacijo osebkov. Njen cilj je zagotoviti, da bo prva generacija vsebovala čim večje število 'zdravih' osebkov, in bomo lahko zato hitreje prišli do rešitve. 'Zdrav' osebek je osebek, čigar zaporedje kromosomov lahko uporabimo v reševanju določenega problema - torej se prevede v delujoče podvezje, rešljivo enačba ali zanko, ki se pravilno konča.

Ker se osebki sestavljajo naključno obstaja velika možnost, da jih na začetku veliko število ne deluje, saj se njihovi kromosomi prevedejo v izrazna drevesa, ki jih ne moremo uporabiti (upor, ki je vezan sam nase, logaritem negativnega števila itd.). Zato običajno porabimo več generacij, da sestavimo delujoče osebke, s pomočjo katerih začnemo reševati problem. V praksi smo ugotovili, da tako porabimo med 20 do 30 generacij.

Decimacija poskuša to preprečiti tako, da spremeni način izdelave prve generacije. Namesto, da samo izdelamo želeno število osebkov (na primer 50), jih izdelamo krepko več (recimo 500) in med njimi izberemo samo tiste, ki delujejo. Tako povečamo možnost, da začetna populacija vsebuje delujoče osebke in da bomo takoj začeli iskati optimalno rešitev. Pri tem je vredno omeniti, da decimacija ne zagotavlja, da bo vsaka začetna populacija vsebovala take osebke, temveč samo poveča možnost, da se to zgodi. Tipična decimacija izdela število osebkov, ki je od 5- do 100-krat večje od velikosti populacije.

2.4.2 Izbor: Izbor poteka po tem, ko smo ovrednotili vse osebke v trenutni generaciji in smo pripravljeni izdelati naslednjo. Zato izberemo določeno število osebkov, ki imajo najboljšo vrednost kriterijske funkcije in jih označimo za elito trenutne generacije. Ta elita nato postane del naslednje generacije brez kakršnihkoli sprememb in postane skupina osebkov, s katerimi tekmujejo vsi drugi - vsak osebek, ki bo dosegel boljšo vrednost kriterijske funkcije bo tako postal član naslednje elite in iz nje izrinil enega izmed trenutnih osebkov. Elita ima še eno pomembno funkcijo - njene osebke uporabimo pri križanju, in tako najdemo pot do optimalne rešitve.

Velikost elite je izražena kot določen odstotek velikosti generacije, na primer 10 odstotkov populacije. Če je elita velika, pomeni, da želimo ohranjati že najdene rešitve in v vsaki generaciji izdelati majhno količino

novih potencialnih rešitev. Majhna elita, po drugi strani, pa pomeni, da imamo na voljo samo nekaj (trenutno) najboljših osebkov. Ker jih bomo uporabili za križanje, se nam lahko zgodi, da bomo dobili več enakih osebkov v naslednji generaciji. V praksi je elita velika od 10 do 15 odstotkov velikosti populacije.

Preostanek populacije zavržemo in naslednjo generacijo dopolnimo z osebki, ki jih naključno sestavimo, križamo ali mutiramo.

2.4.3 Mutacija: Vsak član elite ima možnost, da bo izbran za mutacijo. Mutacija spremeni osebkove kromosome tako, da naključno izbere enega izmed kromosomov in mu spremeni vrednost. Učinek te spremembe je nato odvisen od položaja kromosoma in je lahko ogromen (spremenijp se celotno vezje in njegovi elementi) ali pa minimalen (spremeni se tretja decimalka številke). Mutacija je uspešna, če lahko osebkovo novo izrazno drevo še vedno uporabimo za reševanje določenega problema. Če mutacija ni uspešna, osebek zavržemo in namesto njega naključno sestavimo novega.

Mutacija ima dve nastavitvi - verjetnost, da se zgodi (ponavadi 5 odstotkov), in tip vozlišča, kjer se lahko zgodi (spremenijo se lahko npr. samo številke, ne pa tudi operacije).

2.4.4 Križanje: Pri križanju ustvarimo dva nova osebka tako, da med seboj pomešamo dva obstoječa. Najprej izberemo dva elitna osebka in pri prvem naključno izberemo en element izraznega drevesa. Nato pri drugem poskusimo najti element istega tipa (na primer pri obeh iščemo operacijo seštevanja). Če nam to uspe, dela med seboj zamenjamo, in tako dobljena osebka označimo za ponovno vrednotenje s kriterijsko funkcijo.

Podobno kot mutacija tudi križanje deluje na podlagi pravil - tipov elementov, ki jih lahko zamenjamo. Pravilo lahko nastavimo 'na polno', kar pomeni, da je križanje dovoljeno povsod, ali pa podamo seznam dovoljenih elementov. Rezultat križanja je lahko ogromen in dobimo dva popolnoma različna osebka ali pa skromen, kjer samo spremenimo manjšo številsko vrednost.

2.4.5 Diverzifikacija: Na vsake toliko generacij moramo preveriti raznolikost generacije, drugače bomo hitro dobili generacijo enakih (oziroma malenkostno drugačnih) osebkov, to pa seveda upočasni našo pot do rešitve. Zato eliminiramo vse osebke, katerih kromosomi se prevedejo v identična izrazna drevesa. Odstranjene osebke nadomestimo z novimi naključno sestavljenimi osebki. Tako pregledamo več potencialnih rešitev in imamo posledično boljšo možnost, da najdemo najboljšo. Diverzifikacijo izvedemo na vsake 10 do 15 ovrednotenih generacij.

2.5 Kriterijska funkcija

S pomočjo kriterijske funkcije ovrednotimo osebek in povemo kako primeren je za reševanje določenega problema. Boljši ko je rezultat, bolj verjetno je, da je izbrani osebek optimalna rešitev. Kriterijska funkcija je seveda odvisna od reševanega problema - v nekaterih primerih je lahko enostavna primerjava točk dveh krivulj, v drugih pa kompleksna večkriterijska primerjava lastnosti izbranega vezja (frekvenca dušenja, ojačenje, red filtra itd.). Kriterijska funkcija nujno potrebuje podatke, s katerimi primerja rezultat trenutnega osebka - nabor realnih podatkov, idealizirano krivuljo odziva ali kateri drugi podatkovni set.

Ne glede na izbrano funkcijo pa je naš cilj, da rezultat strnemo v številsko vrednost, saj lahko tako enostavno primerjamo osebke in izberemo najboljše. V večini primerov to vrednost določimo tako, da manjša vrednost pomeni boljši rezultat, in tako rezultat nič pomeni, da se idealna krivulja in naš rezultat popolnoma ujemata.

Pravilna izbira kriterijske funkcije je kritična, če želimo najti pravilno rešitev. Nepravilna kriterijska funkcija nam ponudi preveč splošne rešitve, saj sprejme vsak delujoč osebek, prestroga kriterijska funkcija pa nam ne ponudi nobene mogoče rešitve, saj sprejme samo popolno rešitev.

To smo preizkusili tudi sami, ko smo tehniko SE uporabili za avtomatsko izdelavo filtrov. Na začetku smo kot kriterijsko funkcijo uporabili *koren povprečne kvadratne napake* (RMSE, angl. Root Mean Square Error), kjer smo primerjali dejanski izmenični odziv vezja z idealizirano krivuljo odziva popolnega filtra. To je delovalo, dokler smo izdelovali filtre nizkega reda; ko smo prešli na tretji in višji red, pa nismo več dobili pravilnega rezultata, saj je kriterijska funkcija kot pravilen rezultat sprejela tudi nižji red. Šele ko smo prešli na večkriterijsko funkcijo, smo ponovno dobili pravilne rezultate.

3 PRAKTIČNI PRIMERI

V tem poglavju opisujemo dva primera uporabe SE, ki smo ju izvedli v jeziku Python. Ugotovili smo, da po začetnem naporu, ki je bil potreben za razvoj okolja, lahko to zelo enostavno in hitro prilagodimo za reševanje več različnih problemov - od dela z velikim naborom podatkov in priporočilnimi sistemi do avtomatskega sestavljanja vezij.

3.1 Optimizacija metode Matrix Factorization

Priporočilni sistemi (PS, angl. Recommender Systems) [7][8] pomagajo uporabniku najti zanimive vsebine s pomočjo analize velikanskih količin podatkov, ki vsebujejo več milijonov ocen. Nalogo PS lahko opredelimo kot čim bolj natančno napovedovanje uporabnikovih bodočih ocen, kar ovrednotimo tako, da napovedane ocene primerjamo z ocenami, ki jih uporabnik dejansko dodeli predlaganim vsebinam. Boljše kot je ujemanje (torej manjša ko je razlika med napovedano in dejansko oceno), boljši je naš priporočilni sistem.

Pri našem primeru smo se osredotočili na tehniko Matrix Factorization (MF) [9], ki je v zadnjih časih eden od standardnih pristopov na tem področju. Temelji na pristopu, podobnem dekompoziciji SVD, ki izvorno

matriko podatkov (matrika uporabnik-vsebina, kjer vrednost v celici predstavlja oceno, ki jo je uporabnik dodelil izbrani vsebini) razbije na matrike latentnih vektorjev. Ti vektorji predstavljajo uporabnikove / vsebinske karakteristike in jih lahko uporabimo za napoved ocene tako, da zmnožimo latentni vektor aktivnega uporabnika z latentnim vektorjem potencialno zanimive vsebine. Dobljena vrednost pa predstavlja napovedano oceno. Če je dovolj velika, vsebino predlagamo uporabniku, drugače jo ignoriramo. Glavni izziv tehnike MF je tako izračun latentnih vektorjev, kar storimo z iterativno tehniko, ki temelji na stohastičnem gradientskem spustu. Koraki tehnike potekajo v sledečem zaporedju:

Algorithm 1 Stohastični gradientni spust

1: Inicializiramo latentne vektorje uporabnikov $\mathbf{p}_u$ in vsebin $\mathbf{q}_i$
2: Izračunamo konstantna zamika b_u, b_i in povprečno oceno celotnega seta μ
3: **for** $k \leftarrow 1$ to f **do**
4: **for** vsaka opazovana ocena r_{ui} **do**
5: Izračunaj napako napovedi e_{ui}
6: **for** $iter \leftarrow 1$ to N **do**
7: Izračunaj latentne vrednosti p_u^k in q_i^k
8: **end for**
9: **end for**
10: **end for**

Uspešnost algoritma MF merimo z RMSE in desetkratno navzkrižno potrditvijo. Tako podatkovni set uporabimo desetkrat - vsakič izberemo eno desetino za ovrednotenje, preostalih devet pa za učenje sistema. V našem primeru smo uporabili podatkovni set MovieLens [10] , ki vsebuje 100.000 ocen 5.000 uporabnikov za 300 vsebin. Podrobnejši opis metode, podatkov in posebnosti je v prispevku [1].

V našem primeru smo se osredotočili na sedmi korak algoritma - izračun latentnih vrednosti. Poenostavljeno lahko ta izračun zapišemo v obliki $x_{n+1} = x_n + stepSize * (error - regularization)$, ki je podana tudi v [11]. Enačba deluje, vendar nas je zanimalo, ali lahko s pomočjo SE izdelamo nadomestno enačbo, ki bi dala boljši rezultat (torej manjšo vrednost RMSE). Slovnico smo sestavili iz spremenljivk (trenutne latentne vrednosti, napaka), številk in računskih operacij (vsota, razlika, produkt, logaritem in koren). Dobljene enačbe smo nato uporabili namesto originalne enačbe in z njimi izvedli celoten postopek MF ter izračunali končni rezultat (vrednost RMSE). Povzetek slovnice eksperimenta je podan v tabeli 2. Rezultati poskusa so bili zanimivi, saj je bila slovnica tako prilagodljiva, da je enkrat izdelala zelo preproste izraze ($x = 1$), drugič pa zelo kompleksne ($x = x + \sqrt(err) - log(x + err)$).

Za vsak zagon poskusa smo ovrednotili 150 generacij, ki so vsebovale po 50 osebkov. Rezultati so bili zelo dobri, saj smo v vsakem poskusu izdelali enačbe, ki so delovale, in to celo od 10 do 30 odstotkov bolje kot originalna (originalna enačba je imela vrednost RMSE 1,27, naša najboljša pa 1,13). Povzetek poskusa je podan v tabeli 3.

začetni simbol	x = <-par->
<-par->	<-exp->|<-var->
<-exp->	(<-par->+<-par->)
	(<-par->-<-par->)
	(<-par->*<-par->)
	(<-par->/<-par->)
	log(<-par->|sqrt(<-par->)
<-var->	x_{old} |err |<-num->
<-num->	0 |1 |2 |3 |4 |5 |6 |7 |8 |9

Tabela 2: Slovnica za tehniko Matrix Factorization

Naloga:	Izdelaj enačbe za izračun latentnih vrednosti.
Kriterij:	RMSE z 10-kratno potrditvijo.
Podatki:	MovieLens z 100 tisoč ocenami
Slovnica	Podana v tabeli 2

Tabela 3: Povzetek poskusa Matrix Factorization

3.2 Avtomatska sinteza vezij

Načrtovanje analognih vezij je področje, ki je v procesu prehoda z ročne zasnove na računalniško podprto načrtovanje z uporabo novih orodij, ki so na voljo razvijalcem. Prva generacija takih orodij so bili simulatorji vezij, ki so uporabnikom omogočili, da testirajo razvito vezje, ne da bi ga dejansko sestavili (dober primer takega orodja je okolje SPICE [12], ki smo ga tudi uporabili v tem primeru), še vedno pa so zahtevali poglobljeno znanje s področja načrtovanja vezij. Naslednja generacija orodij pa to odpravi s pomočjo avtomatske sinteze - uporabnik poda želene lastnosti vezja (impedanca, izmenični odziv, ojačenje itd.), sistem pa sam izdela primerno vezje.

Na tem področju je bilo z uporabo pristopa EA razvitih že kar nekaj rešitev ([2] in [11]) vključno z našo [13]. Naš pristop poenostavi proces načrtovanja visoko ali nizko prepustnih filtrov saj od uporabnika potrebuje le red filtra in mejno frekvenco. Naš program vsebuje vse potrebno znanje o mogočih elementih vezja in jih sestavi v primerno vezje. Deluje v okolju Python skupaj z aplikacijo PyOpus, ki nam omogoči dostop do simulatorja vezij SPICE, da lahko preverimo odziv izdelanega vezja. Postopek izdelave vezja poteka v naslednjih korakih:

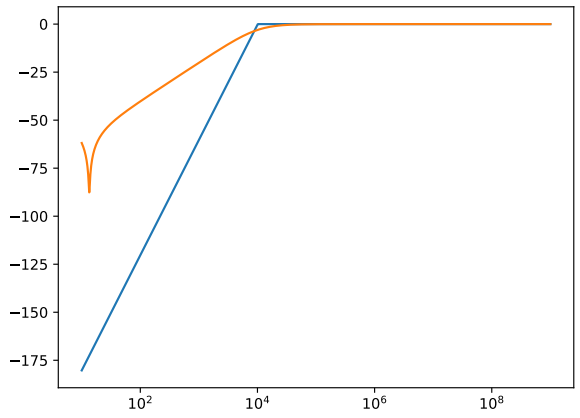
- Inicializacija merilnega vezja.
- Za vsak osebek - izdelamo podvezje na podlagi zaporedja kromosomov.
- Vstavimo podvezje v merilno vezje.
- Izvedemo izmenično analizo podvezja.
- Ovrednotimo rezultat.
- Ponovimo za vsak osebek.

Po prvih nekaj poskusih smo odkrili težavo s kriterijsko funkcijo. Na začetku smo uporabili RMSE, vendar

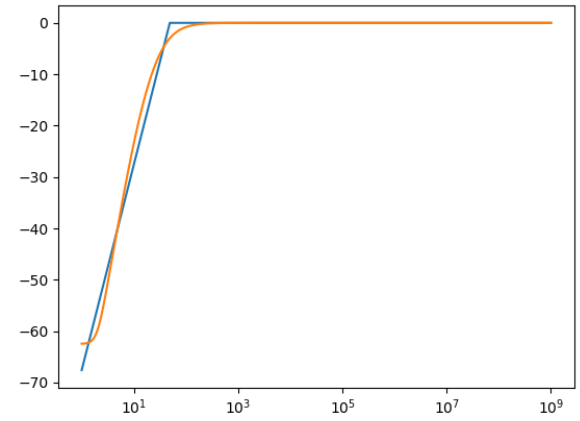
smo hitro ugotovili, da je za naše potrebe preveč ohlapna - ker samo primerja dve krivulji točko za točko, sprejme kot uspešen rezultat tudi filter, ki je nižjega reda, kot je zahtevano. Primer take napake je viden na sliki 1 - modra črta predstavlja odziv idealnega filtra, oranžna pa našo rešitev. Kriterijsko funkcijo smo zato spremenili v večkriterijsko funkcijo, ki je upoštevala štiri različne lastnosti vezja:

- Ojačenje - dvig napetosti pred mejno frekvenco (idealno naj bi bilo enako nič).
- Mejna frekvenca - frekvenca, pri kateri se začne dušenje (naj bo čim bližje zaželeni vrednosti).
- Nihanje - stabilnost napetosti pred dušenjem.
- Dušenje - nivo dušenja, ki ga filter doseže.

S pomočjo utežene vsote smo nato te štiri lastnosti združili v končni rezultat. Ko smo uporabili to kriterijsko funkcijo, je bil vsak nadaljnji poskus uspešen in izdelali smo želeni filter. Primer uspešnega odziva takega filtra je prikazan na sliki 2.



Slika 1: Napaka RMSE.



Slika 2: Večkriterijska funkcija.

začetni simbol	x = <-par-><-par->
	<-par-><-par->
<-par->	<-res->|<-cap->|
<-res->	rXX (<-port->) <-num->
<-cap->	cXX (<-port->) <-num->
<-port->	(in out) |(in 1) |(in 0) |(1 out)
<-num->	<-n->e<-sign-><-n->
<-n->	0 |1 |2 |3 |4 |5 |6 |7 |8 |9
<-sign->	+ |-

Tabela 4: Slovnica za sintezo filtrov

Vsak zagon poskusa je ovrednotil 350 generacij, ki so vsebovale po 150 osebkov. Kljub velikim številkam je bila za vsak poskus potrebna manj kot ura. Slovnica, ki smo jo uporabili v tem primeru, je podana v tabeli 4. V primerjavi s prejšnjim primerom je slovnica tu bolj kompleksna, saj vsebuje več raznolikih elementov. Začetni simbol vsebuje večje število elementov (v celoti 12, v tabeli je podana skrajšana oblika), ker smo tako omejili največje število sestavnih delov vezja in preprečili, da bi vezje raslo v nedogled. Vsak od začetnih elementov lahko tako postane upor, kondenzator ali pa ga izpustimo (vrednost v tabeli). Ko določimo tip elementa, mu nato določimo še sponke ter vrednost upornosti/kapacitivnosti.

Pri tem poskusu smo osebkove kromosome prevedli v podvezje (netlist) in tega uporabili za ovrednotenje osebka. Prednost predstavljenega postopka je, da lahko zelo enostavno dodamo nove elemente - dodamo samo eno novo vrstico v slovnico. Tako smo lahko isti pristop uporabili za zasnovo filtrov, oscilatorjev in vezij z želeno impedančno karakteristiko, kot je prikazano v [13].

4 Zaključek

Slovnična evolucija je pomembna tehnika s področja evolucijskih algoritmov. Predstavili smo njene glavne lastnosti, nastavitve, zahteve in posebnosti. Uporabnost tehnike smo predstavili tudi z dvema praktičnima primeroma.

Menimo, da je slovnična evolucija zelo uporabno raziskovalno orodje, predvsem zato, ker je zelo prilagodljiva in jo lahko z minimalnimi spremembami uporabimo pri različnih problemih. Vse, kar moramo storiti, je, da spremenimo slovnico in izberemo primerno kriterijsko funkcijo. Poleg tega nam omogoča enostavno integracijo specialističnega znanja - tako na primer ni bilo težav pri uporabi programa SPICE, saj nam je slovnica že sama po sebi izdelovala vezja v pravilni obliki (netlist).

Tehniko toplo priporočamo drugim raziskovalnim skupinam, ki se ukvarjajo s področjem optimizacije.

Zahvala

Raziskavo je omogočila agencija ARRS v okviru programa P2-0246 - Informacijsko komunikacijske tehnologije za kakovostno življenje.

LITERATURA

[1] M. Kunaver and I. Fajfar, "Grammatical evolution in a matrix factorization recommender system.," in *ICAISC (1)* (L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, and J. M. Zurada, eds.), vol. 9692 of *Lecture Notes in Computer Science*, pp. 392–400, Springer, 2016.

[2] F. Castejon and E. J. Carmona, "Automatic design of analog electronic circuits using grammatical evolution.," *Appl. Soft Comput.*, vol. 62, pp. 1003–1018, 2018.

[3] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through Simulated Evolution*. Willey, New York, 1966.

[4] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

[5] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[6] M. O'Neil and C. Ryan, *Grammatical Evolution*, pp. 33–47. Boston, MA: Springer US, 2003.

[7] A. L. Buczak, J. Zimmerman, and K. Kurapati, "Personalization: Improving ease-of-use, trust and accuracy of a tv show recommender," in *Proceedings of the 2nd Workshop on Personalization in Future TV*, 2002.

[8] A. Difino, B. Negro, and A. Chiarotto, "A multi-agent system for a personalized electronic programme guide," in *Proceedings of the 2nd Workshop on Personalization in Future TV*, 2002.

[9] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009.

[10] GroupLens, "Movielens," 2017. (accesed 19 March 2018).

[11] J. R. Koza, "Human-competitive results produced by genetic programming.," *Genetic Programming and Evolvable Machines*, vol. 11, no. 3-4, pp. 251–284, 2010.

[12] P. W. Tuinenga, *SPICE: a Guide to Circuit Simulation and Analysis using PSpice*. Englewood Cliffs, NJ: Prentice Hall, first ed., 1988.

[13] M. Kunaver, "Grammatical evolution-based analog circuit synthesis," *Informacije MIDEM*, vol. 49, pp. 229–239, 2019.

Matevž Kunaver received his B.Sc.and Ph.D. degree from Faculty of Electrical Engineering of University of Ljubljana in 2004 and 2009. He is currently working as an Assistant at the same university. His research interests are data mining, evolutionary computation and optimization.