Генетичко програмирање

Семинарски рад у оквиру курса
Методологија стручног и научног рада
Математички факултет

Александра Стојановић, Ивана Ивановић,
Александар Стефановић, Оливера Поповић
mi16048@alas.matf.bg.ac.rs, mi16120@alas.matf.bg.ac.rs,
ai16222@alas.matf.bg.ac.rs, mi16064@alas.matf.bg.ac.rs

30. март 2020.

Абстракт

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da koristite!) kao i par tehničkih pomoćnih uputstava. Pročitajte tekst pažljivo jer on sadrži i važne informacije vezane za zahteve obima i karakteristika seminarskog rada.

Садржај

1 Увод

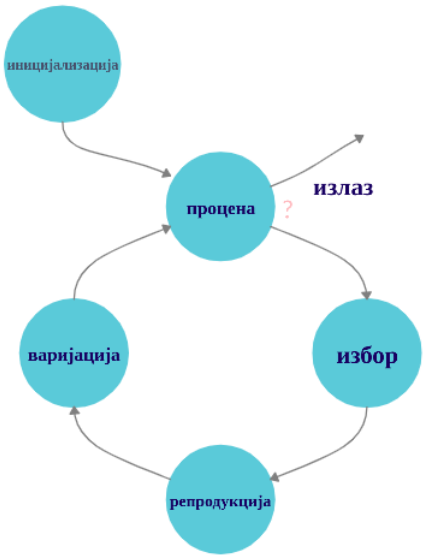
Kada budete predavali seminarski rad, imenujete datoteke tako da sadrže redni broj teme, temu seminarskog rada, kao i prezimena članova grupe. Precizna uputstva na temu imenovnja će biti data na formi za predaju seminarskog rada. Predaja seminarskih radova biće isključivo preko veb forme, a NE slanjem mejla. Link na formu će biti dat u okviru obaveštenja na strani kursa. Vodite računa da prilikom predavanja seminarskog rada predate samo one fajlove koji su neophodni za ponovno generisanje pdf datoteke. To znači da pomoćne fajlove, kao što su .log, .out, .blg, .toc, .aux i slično, **ne treba predavati**.

2 Историјат

Први запис предлога за развијање програма вероватно је запис Алана Тјуринга из 1950. године. Дошло је до размака од 25 година пре објављивања књиге „Прилагођавање природним и вештачким системима“ Џона Холанда, која је поставила теоријске и емпиријске темеље науке. Џон Холанд постао је познат по генетском алгоритму, али о програмима је већ говорио у свом семинарском раду из 1962. године. Дуго се сматрало да излагање рачунарских програма случајним силама мутације и рекомбинације неће донети одрживе програме. Сматрало се да је рачунарски код сувише слаб да би се могао побољшати случајним генерисањем у еволуцији. Појам генетско програмирање описује истраживачко подручје у пољу еволуцијског рачунања које се бави еволуцијом рачунарског кода. Њени алгоритми имају за циљ да олакшају решења проблема машинског учења или да индукују прецизна решења у облику граматички исправних (језичких) структура за аутоматско програмирање рачунара.На структуре које одређују понашање рачунара се примењује општи поступак приказан на слици 1.

Дуго времена је прошло док није утврђено да није немогуће еволуирати рачунарски код. Овај развој захтевао је неколико различитих (мањих) интермедијарних корака. Пре свега оригиналне генетске алгоритме који користе низове битова фиксне дужине да представе бројеве у проблемима оптимизације. Прво схватање било је да се систем правила, уместо бројева који представљају проблематична решења, може подвргнути еволуцији. У другом семинарском раду, Холанд и Реитман представили су систем класификатора који је омогућио да се правила развију током еволуције. Кључни допринос система класификатора будућем пољу генетског програмирања је био да су сва правила знак обележја програмских језика и извршавања сложенијих рачунарских кодова.

Убрзо након Холанда и Реитмана, Смит је увео репрезентацију променљиве дужине. Омогућио је да се уједине правила у програме засноване на правилима који могу да реше задатак дефинисан функцијом прилагођавања (енг. *fitness function*).Ричард Форсит је 1981. године демонстрирао успешну еволуцију малих програма, представљених као дрвеће, за извршење класификације доказа о месту злочина за матичну канцеларију у Великој Британији. Форсит је објавио логички систем правила с параметрима који нам омогућавају класификацију примера у различите класе. За обучавање класификатора се користи низ обука узорака. Програми би логички и нумерички процењивали узорке података да би добили закључке о класи примера.



Слика 1: Општи поступак за генетичко програмирање

Форсит је то сажео на следећи начин:

„Видим три разлога за ову врсту вежбе. Прво, занимљиво је само по себи, друго, правила се понашају на занимљив начин и треће, изгледа да делује. На првом месту је забавно испробати мало апстрактно баштованство, узгајати воћњак бинарних стабала. Могло би бити корисно у другом смислу. На крају крајева, ми смо овде само на основу принципа природне селекције, укључујући и ВИ раднике, и пошто је тако моћан у стварању природне интелигенције, потребно је да га сматрамо методом за гајење вештачке разноликости.“

У другој половини 1980-их, број раних генетски програмираних система се повећавао. Крамер је 1985. представио два еволуциона програмска система заснована на различитим једноставним језицима које је конструисао. Хиклин, Фујики и Дикинсон написали су прекурсорске системе за одређене апликације користећи стандардни програмски језик ЛИСП , а Коза је 1989. коначно документовао методу која је користила универзални језик и примењена је на много различитих проблема. Генетско програмирање је наставило са напредовањем објављивањем књиге Џона Козе 1992. године. Коза је објавио серију од 4 књиге, почев од 1992. године са пратећим видео снимцима. Након тога, дошло је до огромног повећања броја публикација са Библиографијом о генетском програмирању, премашивши 10 000 уноса.

Коза је 1996. започео годишњу конференцију о генетском програмирању коју је 1998. пратила годишња конференција ЕуроГП, и прва књига у ГП серији коју је уредио Коза. 1998. године представљен је и први уџбеник ГП-а. Рик Риоло је наставио процват ГП-а, што је довело до првог специјалистичког часописа опште праксе, а три године касније, 2003. године основана је годишња радионица Теорије и праксе генетичког програмирања. Радови о генетском програмирању и даље се објављују на различитим конференцијама и повезаним ча-

сописима. Данас постоји деветнаест ГП књига укључујући неколико за студенте.

3 Опис методе

Генетичко програмирање је метода која је настала као један од покушаја да се спроведе аутоматизација прављења рачунарских програма за решавања проблема задатих на веома апстрактном нивоу. Основни циљ је да ови проблеми буду тако дефинисани да је познато једино шта треба да буде урађено, без додатних информација о форми или структури решења.

Идеја на којој ова метода почива је процес еволуције који се јавља у природи. Самим тим, генетичко програмирање је техника која спада у групу алгоритама **еволутивног израчунавања** [3]. Еволуција подразумева промену особина укупне популације кроз више смењених генерација. Постојало је током времена више теорија еволуције, док је данас опште прихваћена теорија чији је творац британски биолог Чарлс Роберт Дарвин. По његовој теорији, сви живи организми се развијају процесом природне селекције [2]. Тим процесом јединке које су боље прилагођене имају већу вероватноћу да преживе и да оставе потомство, које је углавном једнако или чак и боље прилагођено од родитеља. Да би могли да оставе потомство, у ћелијама сваког живог бића постоје хромозоми, који су сачињени од гена. Ген је основна јединица наслеђивања, која преноси поруке из генерације у генерацију. Дакле, репродукција подразумева комбинацију гена родитеља уз мале количине мутације. Карактеристике, односно генетски материјал прилагођених јединки углавном опстаје кроз генерације и у неком тренутку почиње да доминира, док остале особине најчешће ишчезну.

У овом одељку биће описано како се процес еволуције осликава на генетичко програмирање, које су његове основне компоненте као и то које одлуке треба донети при прилагођавању алгоритма задатом проблему.

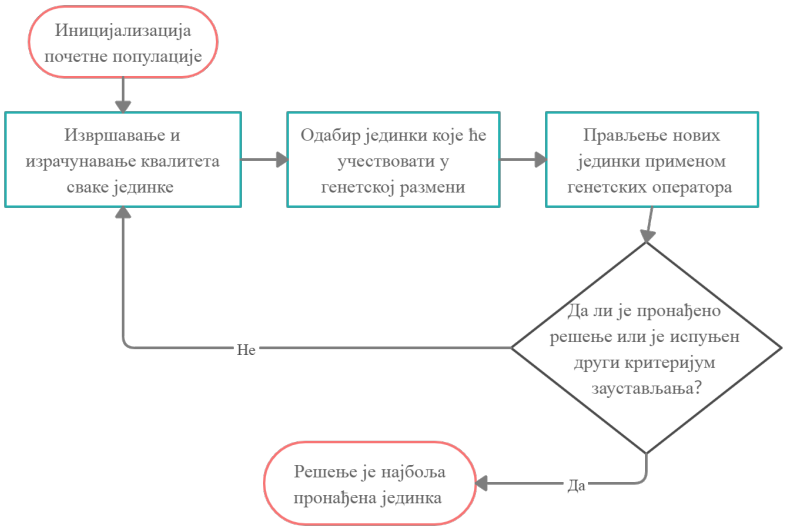
3.1 Општи алгоритам

У генетичком програмирању популацију чине рачунарски програми. Еволуцијом се од почетних, најчешће на случајан начин генерисаних програма, добијају нови, у нади бољи и ефикаснији програми. Обзиром да је то случајан процес, резултат никада не може бити гарантован. Ипак, управо та случајност даје могућност превазилажења неких проблема које имају други детерминистички алгоритми [8]. На слици 2 приказан је дијаграм контроле тока опште верзије алгоритма.

У наредним поглављима биће детаљно описан сваки од наведених корака алгоритма.

3.1.1 Репрезентација јединки

У генетичком програмирању, јединке су заправо рачунарски програми који се могу извршавати. Иако то на први поглед делује логично, оне неће бити представљене линијама кода. Разлог томе је што



Слика 2: Општи алгоритам генетичког програмирања

је неопходно да над изабраном репрезентацијом буду дефинисани генетски оператори, тако да извршавање буде ефикасно. Због тога се репрезентација у виду **синтаксних стабала** [1] најчешће користи.

Код синтаксних стабала, основно је да се променљиве и константе налазе у листовима (најнижи чворови) и називају се терминалима, док се у осталим чворовима налазе функције. Функције могу бити различите природе, на пример аритметичке, тригонометријске или логичке, али могу бити и различите програмске конструкције попут условних израза и петљи. Свако овакво стабло мора имати скуп дефинисаних терминала и функција који заједно чине такозвани **примитивни скуп** генетичког програмирања.

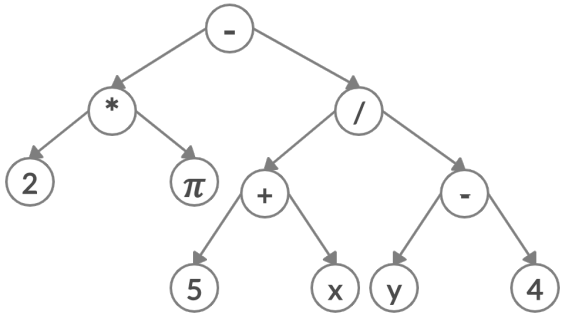
Пример 3.1 *Нека је проблем задат аритметичком формулом:*

$$2 * \pi - \frac{5 + x}{y - 4} \tag{1}$$

*Одговарајући скуп терминала: {2, π, 5, x, y, 4}, скуп функција: {+, -, *, /}, док је одговарајуће синтаксно стабло дато на слици 3.*

Репрезентација стаблима има неколико импликација које треба имати на уму [3]:

- Величина (висина), облик (какве су гране чворова) и сложеност јединки нису фиксирани, већ се могу разликовати. Те особине се чак и код једне јединке такође могу мењати током времена, услед примене оператора за репродукцију.
- За сваки проблем се посебно мора дефинисати примитиван скуп. Као додатак, могу бити дефинисана и семантичка правила која обезбеђују конструкцију исправних стабала. На пример: *Делилац никада не сме да буде једнак 0.*



Слика 3: Синтаксно стабло које одговара формули (1)

3.1.2 Иницијализација популације

Генерисање почетне популације обично се врши на случајан начин, при чему се поштују сва дефинисана ограничења, уколико она постоје. Две најстарије и најједноставније методе су **потпуна метода** (eng. *full method*) и **метода раста** (eng. *grow method*), док се често користи и њихова пола-пола комбинација [8].

Прва метода зове се потпуна јер креира потпуна стабла. Стабло је потпуно уколико су му сви листови на истој дубини. **Дубина чвора** представља број грана које треба прећи да би се од корена дошло до тог чвора. Дубина чвора са вредношћу π приказаног на слици 3 износи 2. По потпуној методи, вредности у чворовима се бирају на случајан начин из скупа функција све док се не дође до максималне дубине. Максимална дубина је један од унапред задатих параметара. Након тога се на случајан начин вредности бирају из скупа терминала. По другој методи, методи раста, чворови се бирају на случајан начин из целог примитивног скупа док се не дође до максималне дубине, а након тога се такође бирају само терминали. Основна разлика између ове две методе је што метода раста дозвољава креацију стабала са више различитих величина и облика.

Ипак, ниједна од ове две методе не пружа претерано велику разноврсност. Због тога се данас нашироко користи њихова комбинација коју је 1992. предложио Џон Коза [?]. Она подразумева да се половина генерације креира коришћењем потпуне методе, а друга половина методом раста.

3.1.3 Функција прилагођености и селекција

Функција прилагођености (eng. *fitness function*) даје оцену **квалитета јединке**. Она треба да буде одабрана тако да може да се израчуна за сваку јединку а да израчунавање притом буде што ефикасније [5]. Зависи од проблема који се решава, али се у генетичком програмирању најчешће реализује тако што постоји скуп тест случајева којима се свака јединка евалуира. То значи да постоји одређени број улаза за које се знају очекивани излази, па се евалуира колико излаза је јединка, односно програм погодио. Осим као мера квали-

тета јединке, функција прилагођености се такође може користити да додели пенале оним јединкама које нису прихватљиве или нису семантички тачне.

Селекција се користи како би се пронашле боље прилагођене јединке које ће учествовати у репродукцији. То се врши коришћењем функције прилагођености. На основу њене вредности јединке се узимају са одређеном вероватноћом. Најчешће се користе **турнирска селекција** и селекција пропорционала вредности функције прилагођености, али се могу користити и друге селекционе методе еволутивног израчунавања [3]. Код турнирске селекције бира се одређени број јединки из популације, на случајан начин. Оне се упоређују и бира се најбоља од њих. Она ће учествовати у репродукцији. Постоје и разне друге варијанте турнирске селекције, али је њена предност у сваком случају то што и средње квалитетне јединке могу добити прилику да оставе потомке. Пример селекције пропроционалне вредности функције прилагођености је **рулетска селекција** (eng. *roulette wheel selection*). По њој, вероватноћа да ће јединка бити изабрана једнака је:

$$p_i = \frac{f(i)}{\sum_j^N f(i)} \tag{2}$$

где је f(i) вредност функције прилагођености јединке i, док је N укупан број јединки у популацији [5].

3.1.4 Репродукција и мутација

Репродукција представља начин на који се од два селекцијом изабрана родитеља добија потомство. Код генетичког програмирања најчешће се користи метод укрштања подстабала. На случајан начин одаберу се тачке укрштања код оба родитеља, након чега се једноставно замене подстабла родитеља са кореном у тачки укрштања. Постоје две опције, креирање једног или два детета разменом. Проблем код овог метода је то што се најчешће барата са стаблима код којих сваки чвор има најмање двоје деце, што доводи до тога да већи број чворова припада скупу терминала. Последица тога је што се размењују мање количине генетског материјала, јер се чешће на случајан начин погоди управо неки од терминала. Да би се избегао овај проблем, Коза је 1992. предложио широко коришћен приступ бирања функција 90% времена, а листова преосталих 10%. Постоји још много видова репродукције који се користе код генетичког програмирања [8].

Мутација се примењује како током времена јединке не би постале сувише сличне. Такође помаже у избегавању локалних екстремума, у ком би се највероватније јединке заглавиле уколико не би било мутације. Мутација се најчешће имплементира тако да се на случајан начин изабере тачка јединке, па се подстабло са кореном у тој тачки замени случајно генерисаним стаблом. Још један чест начин имплементације је да се такође на случајан начин изабере тачка у стаблу јединке, али се сада вредност у том чвору замени случајно одабраном вредношћу исте арности из примитивног скупа. Ако нема других примитива исте арности чвор се не дира.

Основна разлика ове две имплементације је у примени. Примена прве подразумева мутацију тачно једног подстабла, док се код друге

обично она примени на сваки чвор са одређеном вероватноћом што омогућава да више чворова буде мутирано.

Избор који ће се од оператора (укрштање или мутација) применити врши се са одређеном вероватноћом, која се задаје као параметар алгоритма. У генетичком програмирању оператори су обично ексклузивни, што значи да се примењује укрштање или мутација.

3.2 Прилагођавање генетског програмирања

Генетичко програмирање у свом основном облику не користи специфичности ниједног проблема, те се може применити на широк спектар различитих проблема. Међутим, да би се могао користити за решавање неког конкретног проблема потребно га је прилагодити. Одлуке које притом треба донети [8]:

1. Како изгледа примитивни скуп?
Примитивни скуп треба да буде дефинисан тако да је могуће изразити решење коришћењем садржаних примитива. Нажалост, врло често није могуће предвидети примитивни скуп тако да задовољава ово својство. Ипак, захваљујући природи генетског програмирања, врло често иако решење не може да се представи, алгоритам може развити програме који представљају његову апроксимацију. То је најчешће и сасвим довољно. Примери функција дати су у табели 1.

Табела 1: Пример функција генетичког програмирања.

природа функције	примери
аритметичка	$+$, -, $\times$, $\div$
математичка	sin, cos, exp
логичка	$\wedge$, $\vee$, /
условни изрази	if-else
петље	for, while

2. Како изгледа функција прилагођености?
Квалитет, односно прилагођеност јединке може се рачунати на разне начине, а то који ће се користити зависи од природе проблема. На пример може се рачунати као грешка између излаза и жељеног излаза, као тачност у класификацији објеката или као растојање од жељеног резултата.

3. Који ће параметри бити коришћени за контролу извршавања?
Пре покретања алгоритма потребно је подесити неколико параметара, међу којима су величина популације, вероватноће примене оператора, максимална величина програма и слични детаљи. Иако вредности ових параметара зависе од проблема, генетско програмирање је у пракси углавном веома моћно те ће више параметара дати задовољавајуће резултате. Због тога постоје неке препоруке од којих вредности кренути. Правило је да величина популације буде највећа са којом систем који користимо може да ради, а да је то најмање 500.

Укрштање се обично примењује са највећом вероватноћом, која је 90% или више. Насупрот томе, мутација се обично примењује са веома малом вероватноћом, обично око 1%. Вероватноћа мутације је обично толико мала јер би у супротном алгоритам брзо прерастао у случајну претрагу услед превише измена.

4. Који ће бити критеријум заустављања?
Као критеријум заустављања најчешће се користи унапред дефинисан максималан број генерација. Поред тога може бити дефинисан још и неки предикат успеха који је специфичан за проблем који се решава. Такође, извршавање се обично прекида уколико се пронађе решење које задовољава проблем. Као резултат се најчешће узима најбоља пронађена јединка до тог тренутка.

4 Примери примене

Примена генетичког програмирања у индустрији је разнолико и рапидно се повећава годинама. Биће речи о неким од важнијих или занимљивих радова написаних о изабраним дисциплинама, што не значи да не постоји доста радова, односно симулација зарад решавања других проблема у разним другим дисциплинама.

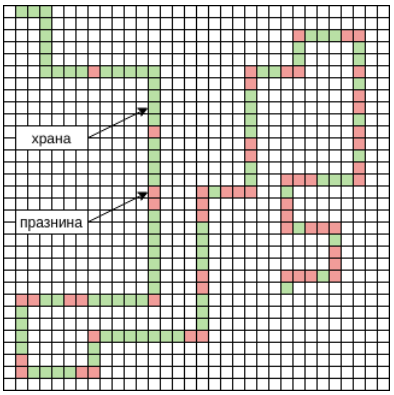
4.1 Роботика

Један од проблема који се симулира јесу два робота који играју познату игру јурке. Оба робота/играча поседују информацију о томе ко јури и где му се апроксимативно налази противник. Циљ игре је што краће бити онај који треба да јури противника. Играју се четири рунде од којих се свака састоји од 25 симулација. Улоге се, у току једне рунде, мењају кад онај који јури буде од свог противника на дужини од једног возила. Играчев резултат за ту рунду рачуна се као број колико пута није био онај који јури подељен са 25. [9]

Такође, постоји и проблем навигације робота како би нашао храну која је постављена дуж неправилне линије; где робот зна да извршава примитивне операције: померање напред, окретање лево/десно као и "мирисање" хране. Овај проблем решен је уз помоћ програмског језика Lisp и коришћењем с-израза. [7] Путање које су коришћене су следеће:

- "Santa Fe" путања (приказана на слици 4) - направљена од 89 комада хране, где између њих постоји једна, две или три празнине без хране који могу бити и на угловима линије кретања робота
- "Los Altos Hills" путања - састављена од 157 комада хране, где 105. комад представља 89. комад односно крај из претходне путање. Након тог комада, додате су још две иргуларности у путањи:
  - налажење хране два места лево или десно од претходног комада хране; прво појављивање после 116. комада хране
  - померање једно место унапред, затим проналажење хране два месте лево или десно; прво појављивање после 136. комада хране

За прву путању успешно је решен проблем у 21. генерацији, док је за другу путању било потребно стићи до 19. генерације. [6]



Слика 4: Изглед "Santa Fe" путање

4.2 Медицина

Иако генетичко програмирање не може бити једино решење када се анализирају биолошки подаци, оно је важно као додатан аналитички алат који има потенцијал да пружи изненађујуће и непредвидиве резултате. Способност да пронађе корисне комбинације својстава и да их прикаже у облику разумљивом за људе га чине погодним за истраживање болести попут рака. [12]

Коришћење унапређеног генетичког програмирања је показало боље резултате у проблему класификације података од класичних алгоритама машинског учења попут линеарне регресије, неуронских мрежа и класификације базиране на суседима. [11]

4.3 Економија

Можда једна од најбитнијих проблема са којима се сусреће ова дисциплина је предвиђање банкрупта компанија. Коришћењем генетичког програмирања дошло се до занимљивих карактеристика података ових предузећа. [10]

Такође, у економији је од велике важности предвидети податке на тржиштима хартија од вредности (берзама), како због смањења губитака од куповине и продаје акција, тако и зарад максимизовања профита. [4]

5 Мета-генетичко програмирање

Мета-генетичко програмирање подразумева упоредно еволуирање програма који врши генетичко програмирање, које замењује ручно подешавање параметара и одлика алгоритма.

Постоје различити начини да се примени мета-генетичко програмирање. Према подели датој у **citat!** то су:

- Они који додају додатни генетички материјал у геном,
- Они који праве колекције модула и имплицитно мењају *језик* изражавања гена,

- Они који мењају фиксни скуп параметара, попут параметра учесталости мутације
- Они који еволуирају операторе генетичког програмирања, тако да се не користе фиксни, унапред-познати оператори.

У даљем тексту ће пажња бити посвећена последњем начину.

Треба узети у обзир да мета-генетички алгоритми изискују знатно више рачунарских ресурса, јер сада свака јединка на првом метанивоу сада захтева извршавање алгоритма генетичког програмирања, ради израчунавања њене прилагођености, и сходно томе, треба проценити да ли је додатно улагање ресурса вредно могућих добитака.

5.1 Еволуирање оператора генетичког алгоритма

Обично су оператори генетичког програмирања унапред задати и фиксни, и најчешће се ради о комбинацији укрштања и мутација, али су истраживани и неки алтернативни оператори **referenca?**, и испоставља се да у неким случајевима, коришћење тих алтернативних оператора, заједно са „уобичајеним" операторима даје боље резултате. Дакле, паметним одабиром и укључивањем одређених оператора мутације, могуће је постићи бољи учинак програма који врши генетичко програмирање.

Међутим, одабир оператора није тривијалан, јер је могуће да неки јако добар оператор не буде ни лак за тумачење, ни интуитиван, односно, мало је вероватно да би нека особа сама осмислила такав оператор, што сасвим одговара мотивацији генетичких алгоритама уопште, где је простор претраге јако велик, и посао проналажења решења је препуштен рачунару, а не особи.

Као и у типичном генетичком програмирању, јединке се приказују помоћу синтаксних стабала, с тим што оваква стабла садрже инструкције које представљају операторе мутације. У овом случају, прилагођеност је сразмерна прилагођености јединки које јединка мета-генетичког програмирања може да генерише својим извршавањем. Више о томе шта би функција прилагођености могла да укључује дато је у 5.1.2.

5.1.1 Елементи оператора

У **referenca**, коришћени су ови листови:

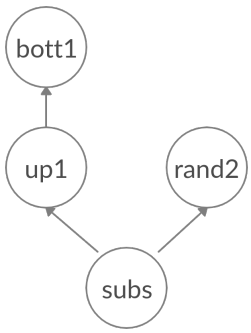
- `rand1` — Вратити прво стабло, са насумично изабраним чвором
- `rand2` — Вратити друго стабло, са насумично изабраним чвором
- `bott1` — вратити прво стабло, са изабраним кореним чвором
- `bott2` — вратити друго стабло, са изабраним кореним чвором

И следећи унутрашњи чворови:

- `top` — одсећи део стабла тако да изабрани чвор сада постане нови корен
- `up1` — вратити исто дрво, али променити изабрани чвор са тренутног на његово прво дете (ако такво не постоји, вратити неизмењено дрво)

- `up2` — вратити исто дрво, али променити изабрани чвор са тренутног на његово друго дете (ако такво не постоји, вратити неизмењено дрво)
- `down` — вратити исто дрво, али променити изабрани чвор са тренутног на његовог родитеља (ако родитељ не постоји, вратити неизмењено дрво)
- `down1` — исто као `down` (како оператори не би имали склоност према `up` операторима)
- `subs` — вратити дрво које је први аргумент, тако да је његов изабрани чвор замењен дрветом које је други аргумент.

На пример, оператор који има структуру задату сликом 5 се може тумачити као "прво дете корена првог стабла замени са другим стаблом".



Слика 5: Пример једног оператора

ovde mogu da napravim ceo primer kako bi izgledalo spajanje jedinki, ako ne budem ništa drugo imao da dopunim do tri strane..

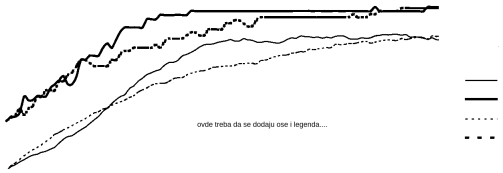
5.1.2 Разматрања за могућа побољшања алгоритма

Према истраживању датом у **referenca**, долази се до закључка да се, због похлепног приступа алгоритма, јако брзо конвергира ка пар ефективних оператора, што умањује разноврсност популације и тиме онемогућује да се дође до неких оператора који постају ефективни тек када се развију. Због тога, функцију прилагођености треба конструисати тако да она узима у обзир колико јединка утиче на разноврсност популације, да ли производи повремене значајне напретке, или равномерне благе напретке и слично.

Примећује се да ограничавање висине стабла оператора негативно утиче на учинак.

5.2 Поређење са генетичким програмирањем

Када се упореди са алгоритмом генетичког програмирања, увиђа се да резултати некада могу бити знатно бољи, а некада конвергирају ка истој вредности (овом пасусу је потребна промена).



Слика 6: Поређење ГП и МГП

6 Закључак

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

Литература

[1] A Aho, M.S. Lam, R. Sethi, and J. Ullman. *Compilers: principles, techniques and tools*. Addison-Wesley Longman Publishing Co., 2006.

[2] Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859.

[3] A.P. Engelbrecht. *Computational Intelligence: An Introduction*. John Wiley & Sons, Ltd, 2007.

[4] Sasaki T. Iba H. Using genetic programming to predict financial data. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99*, pages 244–251, july 1999.

[5] Predrag Janičić and Mladen. Nikolić. *Vestačka inteligencija*. Matematički fakultet u Beogradu, 2019.

[6] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. The MIT Press, London, 1992.

[7] John McCarthy. Lisp, 1958. on-line at: https://lisp-lang.org/.

[8] R Poli, W.B. Langdon, and N.F. McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[9] Craig W. Reynolds. Competition, Coevolution and the Game of Tag. In *Proceedings of Artificial Life IV*, pages 59–69, jan 1994.

[10] Terje Lensberg Thomas E. McKee. Genetic Programming and rough sets: A hybrid approach to bankruptcy classification. *European Journal of Operational Research*, 1(138):436–451, 2002.

[11] Stephan Winkler, Michael Affenzeller, and Stefan Wagner. Using enhanced genetic programming techniques for evolving classifiers in the context of medical diagnosis. *Genetic Programming and Evolvable Machines*, 10:111–140, 06 2009.

[12] William Worzel, Jianjun Yu, Arpit Almal, and Arul Chinnaiyan. Applications of genetic programming in cancer research. *The International Journal of Biochemistry & Cell Biology*, 41:405–413, 02 2009.

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe.