3. domaća zadaća – a. genetski algoritam

Uputa za treću domaću zadaću – Genetski algoritam

Potrebno je implementirati SASEGASA koji rješava problem identifikacije sustava (traženja onih 6 parametara koje ste u drugoj zadaći tražili gradijentnim spustom u okviru zadatka 4. zadaće).

Rješenje treba biti napravljeno općenito, tako da može raditi s bilo kojim problemom, a Vi ga ispitajte na tom problemu (drugim riječima – modelirajte problem sučeljem).

Inicijalne postavke za SASEGASA neka budu:

- početna veličina populacije: 40
- početni broj "sela" u koja se ta populacija dijeli: 5
- minimalni nužan postotak uspješne djece: 40%
- faktor usporedbe: uvijek 1 (pa dijete mora biti strogo bolje od oba roditelja)
- maksimalni broj generirane djece prije no što se u generaciji proglasi neuspjeh (ako nismo generirali traženi broj uspješne djece) modelirat ćemo faktorom, i neka bude 30, što se uvijek interpretira kao 30 * trenutna veličina populacije/sela nad kojom GA radi
- u svakom selu zasebno radite algoritam Offspring selection, i iz generacije u sljedeću generaciju uvijek prekopirajte jedno najbolje rješenje iz trenutne generacije, tako da algoritam bude elitistički
- generaciju koja bude neuspješna odbacite (drugim riječima, ako u pop+pool nema zadani postotak uspješne djece – ne mjenjajte postojeću populaciju; dogodila se konvergencija pa idete na sljedeće selo
- implementirajte dva operatora križanja: jedan koji svaki parametar zasebno linearno interpolira između parametara roditelja, te drugi koji radi to isto, ali s 20% vjerojatnosti za svaki parametar (u 80% ga preuzima iz prvog roditelja); svaki puta kada zatrebate operator križanja, slučajno odaberite koji ćete koristiti
- implementirajte dva operatora mutacije: oba s vjerojatnošću od 30% mutiraju svaki od parametara: prvi dodaje slučajno generiranu vrijednost iz normalne distribucije sa sigmom 0.9 a drugi sa sigmom 0.1. Kad trebate operator, slučajno odaberite koji ćete koristiti u tom trenutku.
- svakog od roditelja birate zasebnim 2-turnirom

Uz ove postavke, algoritam bi trebao kvalitetno rješavati zadani problem. Za svako selo eventualno možete dodati "tvrdu granicu" na maksimalni broj generacija koje dopuštate da se selo razvija (primjerice: 100000 generacija) pa ako se to dogodi, krenite na sljedeće selo.

Napomena: pri radu algoritma ispisujte barem trenutke kada se prelazi s *i* sela na *i*-1 sela, te za svako selo koje je najbolje rješenje na početku rada, a koje na kraju, te koliko je generacija odrađeno prije konvergencije. Ovo će Vam dati informacije kako algoritam "diše" prilikom rada.

3. domaća zadaća – b. genetsko programiranje

U sklopu ove domaće zadaće genetskim programiranjem rješavat ćete problem simboličke regresije. U nastavku je dan kratki uvod u genetsko programiranje, nakon čega su opisani zadaci koje je potrebno riješiti.

Priprema

Kao kratak uvod u genetsko programiranje pročitajte:

http://www.geneticprogramming.com/Tutorial/

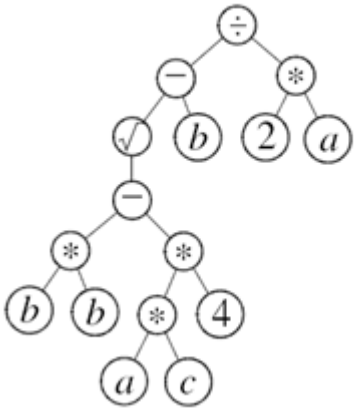
U tom tekstu spominju se tri operatora: reprodukcija, križanje i mutacije. Uočite da se, za razliku od klasičnog genetskog algoritma, oni ne primjenjuju slijedno već se primjenjuje samo jedan od njih.

- Da bismo stvorili nove programe križanjem, biramo dvije jedinke iz trenutne populacije, primjenjujemo neki od operatora križanja i dobivenu djecu ubacujemo u sljedeću generaciju.
- Odaberemo li mehanizam reprodukcije, iz trenutne populacije biramo jednu jedinku i nju direktno prebacujemo u sljedeću generaciju.
- Odaberemo li mehanizam mutacije, iz trenutne populacije biramo jednu jedinku, primjenjujemo nad njom operator mutacije i nastali program ubacujemo u novu generaciju.

Obratite pažnju da ovi svi operatori ne smiju mijenjati roditelje – operatore treba primijeniti nad kopijom roditelja ako oni nad tim stablima rade modifikacije jer roditelji moraju u trenutnoj generaciji ostati nepromijenjeni kako bi mogli biti korišteni i za stvaranje ostale djece (sve dok se ne popuni nova generacija).

Prikaz rješenja stablima i provođenje različitih operacija ilustrirat ćemo na primjeru izgradnje funkcijskog izraza. Svaki se funkcijski izraz može prikazati u obliku stabla. Primjer je dan u nastavku.

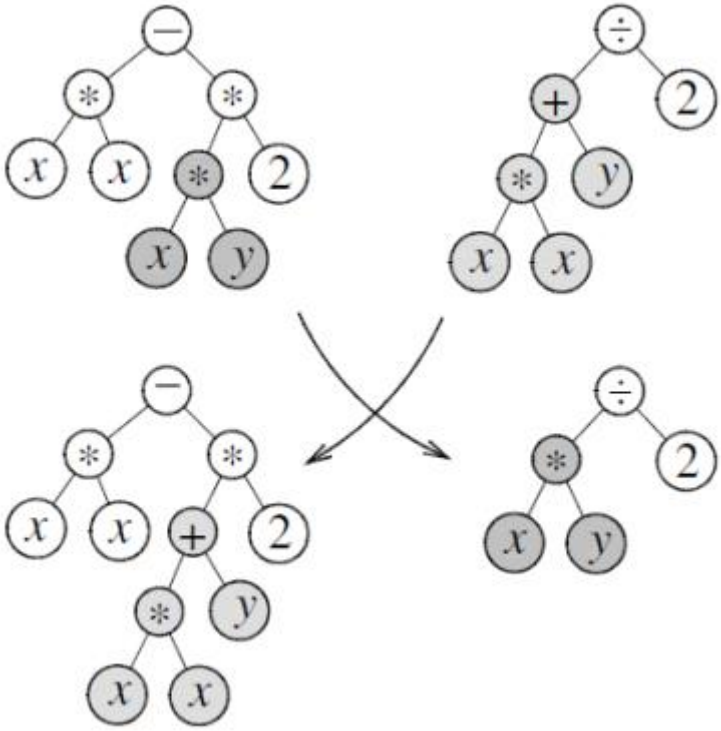
$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$



Za izgradnju stabla na raspolaganju nam stoje nezavršni znakovi – definirat ćemo ih skupom funkcija F, te završni znakovi – definirat ćemo ih skupom terminalnih znakova T. U prethodnom primjeru, F={%, +, -, *, √} dok je T={a, b, c}∪{1,2,3,4}.

Križanje zamjenom podstabala

Ovo križanje grafički je ilustrirano na slici u nastavku. Nakon što su odabrana dva roditelja, u svakom se roditelju s uniformnom vjerojatnošću odabire jedan čvor u stablu. Potom se tako odabrana podstabla zamijene između roditelja čime se dobiju dva nova djeteta.



Križanje može vrlo brzo dovesti do stvaranja stabala koja su vrlo duboka ili imaju ekstremno puno čvorova. Stoga se može definirati maksimalna dubina koju stablo može imati odnosno maksimalni broj čvorova koje stablo može imati. Ukoliko operator križanja stvori stablo koje krši ta ograničenja, tipično možemo postupiti na jedan od dva načina:

- možemo kao rezultat vratiti jednog od (ili oba) roditelja, čime se križanje pretvara u reprodukciju, ili
- možemo križanje proglasiti neuspjelim; u tom slučaju "pozivatelj" križanja mora biti tako napisan da se s time može nositi (primjerice, odustati od odabranih roditelja, odabrati neka druga dva i pokušati opet).

Mutacija

Mutacija se najčešće provodi tako da se u odabranoj jedinki slučajno prema uniformnoj distribuciji odabere jedan čvor i on se zamijeni nekim posve slučajno generiranim podstablom. Izgradnja tog podstabla treba biti tako napravljena da omogući da se kao podstablo dobije i samo jedan terminalni simbol čime omogućava mutacije u kojima se jedan terminalni simbol mijenja drugim ili pak situacije u kojima se čitavo podstablo zamjenjuje jednim terminalnim simbolom. Kako i mutacija može napraviti preveliko stablo, takve situacije treba obraditi na sličan način kao što to radi i križanje. Alternativno, ako se zna na kojoj je dubini odabrani čvor koji će biti zamijenjen, moguće je izračunati koliko novostvoreno podstablo smije biti duboko pa se ta informacija može koristiti kako bi se takvo stablo izgradilo metodom *grow* ili *full*, odnosno možemo unaprijed slučajno odabrati koliko duboko podstablo želimo (random između 1 i te dubine) pa izgradimo takvo podstablo. Pri tome i dalje postoji mogućnost da ukupni broj čvorova bude prevelik.

Stvaranje početne populacije

Dva su osnovna načina izgradnje početne populacije, odnosno kolekcije početnih programa. Početne programe možemo generirati metodom *full* ili metodom *grow*. Obje metode započinju tako da iz skupa funkcija F slučajno odaberu jednu funkciju i nju postave kao korijen stabla. Nakon toga za svaki se argument funkcije bira novi primitiv s uniformnom vjerojatnošću. Ako je duljina staze od korijena do trenutnog čvora manja od maksimalne dopuštene duljine, novi se primitiv bira iz skupa funkcija F (ako koristimo metodu *full*) odnosno iz unije skupa funkcija F i skupa terminalnih simbola T (ako koristimo metodu *grow*). Ako je duljina staze od korijena do trenutnog čvora dosegla maksimalnu dopuštenu duljinu, tada se primitiv kod obje metode bira isključivo iz skupa terminalnih simbola T. Postupak se nastavlja sve dok izgradnja stabla nije gotova.

Prilikom izgradnje početne populacije htjeli bismo da se u populaciji nađe što više raznovrsnih stabala. Jedna mogućnost kako ovo osigurati jest koristiti metodu *ramped-half-and-half* sa specificiranom maksimalnom dubinom. Ideja ove metode je da za svaku dubinu do zadane pola stabala stvori metodom *grow* a pola stabala metodom *full*. Primjerice, neka je maksimalna dubina postavljena na 6. To znači da trebamo stvoriti stabla dubina 2, 3, 4, 5 i 6, tj. Imamo 6-2+1=5 različitih dubina stabala. U populaciji želimo imati 1/5=0.2 odnosno 20% stabala svake od tih dubina. Pola od tih stabala trebamo izgraditi metodom *grow* a pola metodom *full*. Konkretno, gradimo li populaciju veličine 200 jedinki, imat ćemo 200/(6-2+1)=200/5=40 jedinki svake od dubina. To znači da ćemo graditi 20 jedinki metodom *grow* uz dubinu 2, 20 jedinki metodom *full* uz dubinu 2; 20 jedinki metodom *grow* uz dubinu 3, 20 jedinki metodom *full* uz dubinu 3; 20 jedinki metodom *grow* uz dubinu 4, 20 jedinki metodom *full* uz dubinu 4; 20 jedinki metodom *grow* uz dubinu 5, 20 jedinki metodom *full* uz dubinu 5; 20 jedinki metodom *grow* uz dubinu 6, 20 jedinki metodom *full* uz dubinu 6. S obzirom da metoda grow ne garantira da ćemo dobiti stablo željene dubine, konačni postotak stabala pojedinih dubina ne mora nužno biti jednak ciljanom.

Detaljnije o ovoj zadaći

Genetsko programiranje

Implementirajte generacijski genetski algoritam koji radi s populacijom od 500 jedinki. Inicijalnu populaciju izgradite metodom *ramped-half-and-half* uz maksimalnu dubinu postavljenu na 6. Kao operator križanja koristite operator zamjene podstabala a kao operator mutacije koristite zamjenu slučajno odabranog čvora novim podstablom.

Postavite ograničenje maksimalne dubine stabla s kojom algoritam može raditi (provjeravati prilikom križanja i mutacija) na 7.

Kao mehanizam selekcije koristite 7-turnirsku selekciju za odabir svake od jedinki.

Algoritam neka bude elitistički: najbolje trenutno rješenje uvijek kopirajte u novu generaciju.

Isprobajte rad algoritma uz ograničenje na 100 generacija. Neka je vjerojatnost reprodukcije 1%, vjerojatnost mutacije 14% a vjerojatnost križanja 85%. Ako se ovi parametri pokažu neadekvatnima, slobodno ih promijenite.

Simbolička regresija

U okviru ove domaće zadaće rješavat će se problem simboličke regresije. Cilj simboličke regresije jest na temelju niza ulaznih i izlaznih vrijednosti, odnosno skupa

$\{(x_{11}, x_{12}, \ldots, x_{1k}, y_1), \ldots, (x_{n1}, x_{n2}, \ldots, x_{nk}, y_n)\}$, odrediti simbolički oblik funkcije koja opisuje to preslikavanje.

Ulazi i izlaz funkcije zapisani su u datoteci na način da se u svakom retku nalazi jedno mjerenje, pri čemu se svaki redak sastoji od *k* stupaca (vrijednosti u svakom stupcu su međusobno odvojene tabulatorom), gdje prvih *k-1* vrijednosti predstavlja ulaze u funkciju, a posljednja vrijednost predstavlja izlaz iz funkcije. Primjer ulaza za neku funkciju $f(x_1, x_2)$ bi izgledao:

```
x1      x2      f(x)
0       0       0
0.2     0.2     0.24
0.4     0.4     0.56
0.6     0.6     0.96
0.8     0.8     1.44
…
```

Za izgradnju simboličkog izraza funkcije iskoristite sljedeće operatore: +, -, *, /, sin, cos, sqrt, log (logaritam po bazi 10), exp (eksponent prirodnog broja e). Operatore /, sqrt i log definirajte u obliku „zaštićenih“ operatora, što znači da ako se kao argument operatora dobije vrijednost za koju on nije definiran (1/0, sqrt(-5), log(0)), onda kao rezultat operacije vratite vrijednost 1.

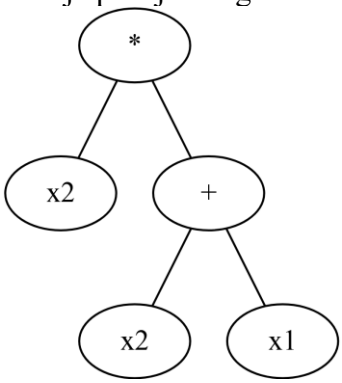
Skup terminalnih čvorova neka se sastoji od skupa ulaznih varijabli funkcije (x1, x2....), i konstanti. Broj varijabli određuje se automatski na temelju ulazne datoteke, dok se raspon iz kojeg se generiraju konstante zadaje kao dodatni parametar. Vrijednost čvorova koji predstavljaju konstante nasumično se odabire iz zadanog intervala u onom trenu kada se i sam čvor generira u jedinci (primjerice kod stvaranja jedinki u populaciji na početku algoritma, ili kada se u mutaciji nasumično generira novo podstablo) te se vrijednost tog čvora ne mijenja tijekom rada algoritma. Ako raspon za vrijednosti konstanti nije zadan, onda se one ne koriste u programu.

Kaznu jedinke (*cost*) računajte kao srednje kvadratno odstupanje vrijednosti dobivenih od izraza evoluiranog genetskim programiranjem i očekivanog izlaza:

$$cost = \frac{1}{N}\sum_{i=1}^{N}(\mathrm{t}_i - y_i)^2$$

Gdje N predstavlja ukupni broj primjera, y izlaz dobiven od izraza evoluiranog genetskim programiranjem, a t očekivanu vrijednost.

Ako je primjerice genetsko programiranje pronašlo izraz koji je zadan sljedećim stablom



Onda za prethodni primjer ulaza možemo izračunati izlaze za dobiveni simbolički izraz i izračunati srednju kvadratnu pogrešku:

x_1	x_2	t	y	$(y-\mathrm{t})^2$
0	0	0	0	0
0.2	0.2	0.24	0.08	0.0256

0.4	0.4	0.56	0.32	0.0576
0.6	0.6	0.96	0.72	0.0576
0.8	0.8	1.44	1.28	0.0256
Srednja kvadratna greška				0.03328

Kriterij zaustavljanja algoritma mora biti definiran kao maksimalni broj izračuna funkcije kazne, čija je maksimalna dozvoljena vrijednost 1000000. Možete uzeti i manji broj evaluacija funkcije kazne ako uočite da genetsko programiranje prije konvergira ili možete definirati i dodatni kriterij kojim ispitujete stagnira li najbolje rješenje određeni broj generacija (stagnaciju možete definirati na način da se kazna najbolje jedinke nije promijenila u zadnjih *n* generacija). Maksimalna dubina stabla mora biti postavljena na 7. Ostale parametre odaberite proizvoljno.

Parametre genetskog programiranje mora biti moguće postaviti korištenjem konfiguracijske datoteke. Oblik konfiguracijske datoteke je:

```
FunctionNodes: +, -, *, /, sin, cos, sqrt, log, exp
ConstantRange: -3, 3
PopulationSize: 500
TournamentSize: 3
CostEvaluations: 1000000
MutationProbability: 0.3
MaxTreeDepth: 7
UseLinearScaling: 0
```

Parametar „FunctionNodes“ definira koji će se funkcijski čvorovi koristiti u genetskom programiranju, „ConstantRange“ interval iz kojeg se mogu generirati vrijednosti za konstante, dok ostali retci predstavljaju preostale vrijednosti parametara genetskog programiranja. Za konstante zadaju se donja i gornja granica intervala (granice su uključene u interval), a ako se konstante ne koriste, onda za to koristite oznaku N/A, npr. „ConstantRange: N/A“. U datoteci možete definirati i ostale parametra koje koristite u algoritmu, no parametri koji su navedeni u primjeru moraju se obavezno moći definirati kroz datoteku. Parametar „UseLinearScaling“ određuje koristi li se linearno skaliranje ili ne prilikom izvođenja.

Algoritam mora na standardni izlaz ispisati simbolički izraz i kaznu najbolje jedinke svaki put kada se pronađe jedinka s manjom kaznom.

Primjer pokretanja programa može biti:

java -cp staza hr.fer.zemris.gp.SymbolicRegressionSolver *input.txt*

gdje input.txt predstavlja datoteku s podacima o funkciji koja se uči genetskim programiranjem.

Zadatak 1
Riješite problem simboličke regresije koji je zadan s ulaznom datotekom 03-f1.txt. Zadana funkcija je oblika $f(x) = x + x^2 \sin x$. Na njoj isprobajte radi li viša implementacija.

Zadatak 2
Riješite problem simboličke regresije zadan s ulaznom datotekom 03-f2.txt u kojoj se nalaze podaci za funkciju $f(x, y) = xy + \sin(x + y) * \cos y$.

Zadatak 3

Riješite problem simboličke regresije koji je zadan s ulaznom datotekom 03-f3.txt. Zadana funkcija je oblika $f(x) = 0.5x^2 + 1.3$. Na njoj isprobajte radi li viša implementacija. Za početak problem riješite bez uporabe konstanti. Kakva se rješenja dobivaju? Nakon toga uključite korištenje konstanti iz raspona [-1,1]. Poboljšava li korištenje konstanti rezultate koje dobiva program?

Konačno, implementirajte postupak linearnog skaliranja, kojim će prilikom evaluacije jedinke vaše genetsko programiranje dodatno pokušati „pogoditi" konstante u simboličkom izrazu koji se evaluira. Ako y predstavlja izlaz iz genetskog programiranja za neke ulazne podatke, onda linearnim skaliranjem pokušavamo pronaći parametre a i b za koje će izraz $a+by$ dati najmanju grešku. Na temelju ulaznih podataka

$$b = \frac{\sum_i (t_i - \bar{t})(y_i - \bar{y})}{\sum_i (y_i - \bar{y})^2}$$
$$a = \bar{t} - b\bar{y}$$

Pri čemu y_i predstavlja izlaz iz stabla dobivenog genetskim programiranjem za primjer i, t_i predstavlja željeni izlaz za primjer i, a $\bar{y}$ i $\bar{t}$ srednje vrijednosti za dobivene i željene vrijednosti za sve ulazne podatke.
Ponovno pokrenite program s linearnim skaliranjem s i bez korištenja konstanti. Što možete zaključiti o ponašanju algoritma uz korištenje ovog postupka? Na kraju obavezno ispišite i koeficijente a i b za najbolju jedinku koja je pronađena.