Ogrodje za genetsko programiranje z Mathematico
Seminarska naloga

Jaka Močnik
jaka@gnu.org

Povzetek

V članku predstavim metodo reševanja problemov brez eksplicitnega programiranja: genetsko programiranje. To metodo povežem s koncepti jezika Mathematice in predstavim izvedbo knjižnice za reševanje problemov z genetskim programiranjem v Mathematici ter opišem uporabo te knjižnice. Izvedbo preizkusim na treh problemih iz različnih domen: problemu usmerjanja mravlje, ki išče hrano ter problemih simbolične regresije in konstrukcije Booleovih funkcij.

1 Uvod

Genetsko programiranje, ena izmed podvrst genetskih algoritmov [1], je pomembna metoda reševanja problemov brez eksplicitnega programiranja.

Izvedba sistema za genetsko programiranje zahteva rešitev dveh osnovnih problemov:

- problem *predstavitve* programov in *rokovanja* z njimi ter
- problem *izvajanja* oziroma *interpretiranja* programov

Pokazal bom, da so programi pisani v jeziku Mathematice izjemno primerni za uporabo v sistemih za genetsko programiranje, saj je spričo asketske sintakse jezika rokovanje z njimi enostavno, poleg tega pa Mathematica sama omogoča rokovanje z lastnimi programi. Pomembno prednost predstavlja tudi širok nabor obstoječih funkcij, primernih za gradnike programov razvitih s postopkom genetskega programiranja.

Mathematica elegantno reši tudi problem interpretiranja programov: Mathematicin program ne le gradimo v sami Mathematici temveč ga v njej tudi izvajamo.

Predstavil bom izvedbo knjižnice za reševanje poljubnega problema s postopkom genetskega programiranja v Mathematici. V drugem razdelku predstavim osnovne pojme iz področja genetskega programiranja. V tretjem razdelku te koncepte preslikam v domeno Mathematice, četrti razdelek pa opiše izvedbo in uporabo razvite knjižnice. Zaključimo z povzetkom opravljenega dela ter smernicami za nadaljnje delo na tem področju.

2 Genetsko programiranje

Pri uporabi genetskega programiranja prevedemo reševanje problema na iskanje programa, ki reši dani problem.

Program predstavimo kot drevesno strukturo, katere notranja vozlišča so funkcije, listi pa spremenljivke ali konstante.

Problem opišemo na visokem nivoju, običajno tako, da navedemo množico učnih primerov. Učni primer tvorijo vhodne vrednosti spremenljivk ter željeni rezultat programa.

Kvaliteto vzgojenih programov ocenjujemo glede na *funkcijo uspešnosti.*

Genetsko programiranje je iterativen postopek, ki poteka takole:

- tvorimo naključno začetno populacijo programov,
- vsak program izvedemo oz. interpretiramo in ocenimo rezultat,
- glede na uspešnost in izbrani postopek selekcije (o slednjem več v [2]) izberemo programe, ki tvorijo novo populacijo,
- na programih nove populacije izvedemo *križanje* in *mutacijo* [2],
- postopek ponovimo od interpretacije in ocenjevanja dalje,
- postopek ustavimo po določenem številu iteracij (generacij) ali, ko najdemo program, ki reši dani problem.

Rešitev je najbolje ocenjeni program, ki smo ga na ta način dobili (ne nujno v zadnji generaciji).

Vhod genetskega algoritma predstavljajo:

- množici funkcij F in končnih vrednosti (spremenljivk, konstant) T,
- funkcija uspešnosti f,
- parametri genetskega algoritma: velikost populacije N_p, največje število generacij N_g, verjetnosti križanja in mutacije p_x, p_m.

Izhod pa je program, ki najuspešneje reši dani problem.

3 Evolucija v Mathematici

Programski jezik Mathematice je zasnovan kot funkcijski programski jezik: program v Mathematici je izraz, ki se neposredno preslika v drevesno strukturo, s katero delamo med postopkom genetskega programiranja. Preprosta preslikava je posledica asketske sintakse, običajne v funkcijskih programskih jezikih in majhne izbire objektov s katerimi rokujemo v programu: pravzaprav je edini objekt funkcija.

Mathematica pozna mehanizma *samozavedanja* (*ang. reflection*) in *ostvarjanja* (*ang. reification*) [3], ki Mathematici omogočata, da s svojimi programi in nenazadnje tudi s samim interpreterjem rokuje na enak način kot s problemskim področjem: Mathematica vsebuje bogat nabor funkcij za rokovanje z in preoblikovanje lastnih programov ter nadzor postopka interpretiranja programov. Te funkcije nam omogočajo, da celoten sistem za genetsko programiranje izvedemo v Mathematici, za jezik v katerem bodo predstavljeni vzgojeni programi pa vzamemo kar njen programski jezik. S tem se izognemo definiciji novega programskega jezika, potrebi po interpreterju zanj in implementaciji preoblikovanja programov, saj nam vse to ponuja že Mathematica sama.

Postopek genetskega programiranja bomo v Mathematici izvedli kot množico funkcij za kombiniranje in preoblikovanje programov po načelih genetskega programiranja [1]. Za interpretiranje, ovrednotenje in ocenjevanje tako nastalih programov, pa bo služil meta-interpreter, napisan kar v Mathematici sami!

Podoben pristop je sicer mogoč tudi v nekaterih drugih jezikih oz. sistemih: Common Lispu, Haskellu, Prologu in celo Javi; vendar ima Mathematica naslednje prednosti:

- že omenjeno preprosto sintakso,
- odličen vmesnik za preoblikovanje lastnih programov,
- bogat nabor obstoječih funkcij, ki jih lahko uporabimo kot gradnike naših programov,
- poleg čistega funkcijskega programiranja s takojšnjim vrednotenjem izrazov (*ang. eager evaluation, call-by-value*) ponuja možnost uporabe

mnogih drugih programskih paradigem, od počasnega vrednotenja izrazov (*ang. lazy evaluation, call-by-name*) do proceduralnega in objektnega programiranja [4].

4 Knjižnica GP

Za določanje parametrov postopka, osnovnih gradnikov programov in samega problema knjižnica uporablja množico simbolov, katerih vrednost je prepuščena definiciji uporabnika: uporabnik mora definirati vrednosti parametrov genetskega algoritma, število učnih primerov in same učne primere ter funkcijo uspešnosti.

Postopek poženemo preko ovrednotenja funkcije `GPEvolve`: rezultat ovrednotenja je seznam, ki vsebuje, po vrsti, naslednje elemente:

- najboljši vzgojeni program,
- surovo uspešnost programa [2],
- standardizirano uspešnost programa [2],
- prilagojeno uspešnost programa [2],
- normalizirano uspešnost programa [2].

Izvedba knjižnice je v Mathematicinem zvezku *GP.nb*.

4.1 Parametri genetskega algoritma

Uporabnik sme nastavljati naslednje parametre genetskega algoritma:

`GPMaxDepth`: največja dovoljena globina drevesa, ki predstavlja program,

`GPCrossoverP`: verjetnost križanja osebkov, `GPMutationP`: verjetnost mutacije osebka,

`GPPopulationSize`: velikost populacije (število programov v eni generaciji),

`GPGenerations`: največje dovoljeno število generacij.

4.2 Gradniki programov

Gradnike programov določimo z definicijo seznama `GPPrimitives`. Ta vsebuje tako funkcije kot spremenljivke in konstante.

Funkcije definiramo z izrazom z glavo `Func`: `Func[Hold[Sin[X,Y]],2]`. Prvi argument je generična oblika funkcije - ta mora biti zaščitena z uporabo `Hold`, ki prepreči prezgodnje ovrednotenje funkcije. Imena argumentov funkcije niso pomembna. Drugi argument predstavlja število argumentov funkcije.

Spremenljivke definiramo z izrazom `Var[Hold[X]]`; tudi tu moramo ime spremenljivke zaščititi z uporabo `Hold`.

Celoštevilske konstante definiramo z izrazom `Const[Integer, Low, High]`, kjer sta `Low` in `High` spodnja oz. zgornja meja domene za konstante.

Gradnike, iz katerih lahko sestavljamo majhno podmnožico polinomov ene spremenljivke, bi lahko opisali z

```
GPPrimitives:={
  Func[Hold[Plus[X,Y]],2], Func[Hold[Times[X,Y]],2],
  Var[Hold[X]], Const[Integer,-10,10]
}
```

4.3 Učni primeri

Učne primere podamo s seznamom `GPTestCase`. Ta vsebuje izraze z glavo `TC`, prvi argument vsebuje seznam začetnih prireditev spremenljivk (predstavljenih z izrazom `Assign[Ime, Vrednost]`), s katerim vzpostavimo testno okolje v katerem se program izvaja, drugi argument pa je pričakovani rezultat programa.

Eden izmed učnih primerov za iskanje funkcije $x^2 + y + 1$ je npr.

```
TC[{Assign[X,3], Assign[Y,1]}, 11]
```

Število učnih primerov nastavimo s simbolom `GPTestCaseCount`.

4.4 Funkcija uspešnosti

Uporabnik mora definirati dve funkciji uspešnosti: surovo uspešnost (*ang. raw fitness*) in standardizirano uspešnost (*ang. standardized fitness*) [2]:

```
GPRawFitness[Result, TestCase]
GPStdFitness[Result, RawFitness, TestCase]
```

Standardizirana uspešnost se izračuna na osnovi surove, pri čemer nižje vrednosti predstavljajo uspešnejše programe.

Funkciji uspešnosti za problem iskanja funkcije $x^2 + y + 1$ sta:

```
GPRawFitness[Res, TC]:=Abs[Res - TC[[2]]]
GPStdFitness[Res, Raw, TC]:=Raw
```

5 Zaključek

Razvil in preizkusil sem generično knjižnico za reševanje splošnih problemov s pomočjo genetskega programiranja v Mathematici. Izkaže se, da je Mathematica spričo odličnih možnosti za rokovanje z lastnimi programi primerna platforma za uporabo metode genetskega programiranja. Bogat nabor obstoječih funkcij, ki jih lahko uporabljamo za gradnike vzgajanih programov in pa velika izbira paradigem konstrukcije programov, ki jih ponuja, dodatno pripomoreta k temu, da je čas potreben za formulacijo problema v kontekstu genetskega programiranja čim krajši.

V prihodnje bi veljalo seznam osnovnih funkcij za gradnjo programov opremiti še z informacijo o tipih vhodnih parametrov in rezultata, kar bi omogočilo sprotno preverjanje pravilnosti programa že med njegovim spreminjanjem in ne šele med interpretiranjem - to bi močno zmanjšalo količino nepravilnih programov, ki evolucijo le upočasnjujejo.

Literatura

[1] Koza, J. R., Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press, 1992, ISBN 0-262-11170-5

[2] Goldberg, D. E., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Publishing, 1989, ISBN 0-201-15765-5

[3] Smith, B., Reflection and Semantics in a Procedural Language, Ph. D. Thesis, MIT Laboratory for Computer Science, 1982

[4] Wolfram, S., The Mathematica Book, Fourth Edition, Cambridge University Press, 1999, ISBN 0-521-64314-7