ORODJE ZA ANALIZO KOMPLEKSNOSTI PROGRAMOV

Vili Podgorelec, Peter Kokol, Janez Brest
Fakulteta za elektrotehniko, računalništvo in informatiko
Univerza v Mariboru, Smetanova 17, 2000 Maribor
vili.podgorelec@uni-mb.si

POVZETEK

Pri uresničevanju zahtev po kakovostnih programskih izdelkih se prej ali slej srečamo z uporabo programskih metrik kompleksnosti. Prav tako kot obstaja veliko različnih metrik, imamo na voljo tudi mnogo orodij, ki omogočajo analizo programov z nekaterimi izmed teh metrik. Vendar pa je vsako orodje vezano samo na določene programske metrike, zato moramo za obširnejše analize uporabiti celo vrsto različnih orodij. Želeli smo si okolja, ki bi združevalo vse najbolj uporabljane programske metrike in ker nove metrike tudi sami razvijamo, smo izdelali orodje za analizo kompleksnosti programov, ki ob klasičnih temelji predvsem na novih, bolj splošnih fraktalnih metrikah.

ABSTRACT

As we try to fulfill the requirements upon quality of the software products we cannot avoid the use of the complexity metrics. There is a lot of different metrics and also there are hundreds of tools for analyzing the software with some of those metrics. However, since all tools available are concentrated only on some specific programming metrics, for a comprehensive analysis one has to use a lot of different tools. We wanted to derive an environment that would include all of the mostly used metrics, and since we are also developing new metrics ourselves, we have developed a tool called Software Complexity Analyzer, based besides upon some classical metrics primarily upon new, more general ones.

Uvod

Če želimo uresničiti zahteve po kakovostnih programskih izdelkih, moramo te izdelke med njihovim razvojem tudi redno analizirati. Eden izmed najbolj pogostih in najbolj uveljavljenih načinov analiziranja programov je uporaba programskih metrik kompleksnosti [3,12]. V članku bomo opisali programsko orodje za analizo kompleksnosti programov Software Complexity Analyzer, ki smo ga izdelali za okolje Windows 95 oz. Windows NT. Orodje omogoča analizo programov s pomočjo nekaterih najbolj uporabljanih programskih metrik kompleksnosti in s pomočjo novih fraktalnih metrik, ki smo jih sami razvili [1,6-10]. Poleg same analize programov lahko med seboj primerjamo tudi posamezne programske metrike, kar je v takšnih orodjih izredno pomembno.

Z opisanim programom smo uspeli združiti vse pomembnejše metode za analizo kompleksnosti računalniških programov v enovito celoto, ob tem pa smo dodali še lastne fraktalne metrike. Na ta način bomo lahko s predstavljenim izdelkom nadomestili mnogo posameznih orodij za analiziranje na različnih sistemih, kar bo občutno skrajšalo čas analiziranja programov, prav tako pa ne bo več potrebe po učenju dela z različnimi, ponavadi za uporabnika nič kaj prijaznimi orodji.

Ob samem prikazu programskega orodja bomo na kratko predstavili še novo fraktalno metriko a in podali nekaj zgledov, kako lahko to fraktalno metriko uporabimo tudi v nekatere manj običajne namene.

Kompleksnost računalniških programov in fraktalne metrike

Kompleksnost je težko definirati. Po Morowitzu [11] si kompleksni sistemi delijo nekatere lastnosti, npr. vsebujejo veliko število elementov in predstavljajo širok prostor različnih možnosti. So večnivojske hierarhije, od katerih ima vsak nivo svoje lastne principe, zakone in kompleksnosti. Gell-Mann [4] predlaga, da bi morali opraviti mnogo različnih meritev, če bi hoteli zajeti vse naše intuitivne zamisli o tem, kar predstavlja kompleksnost. Druga, bolj konkretna meritev kompleksnosti, ki temelji na generalizaciji entropije [5], je korelacija. To lahko relativno enostavno izračunamo za posebno vrsto sistemov, ki se jih da predstaviti z zaporedjem simbolov.

Računalniške programe ponavadi analiziramo s pomočjo računske kompleksnosti, merimo jih z metrikami kompleksnosti [3,12]. Lahko pa pogledamo na program iz drugega zornega kota in si ga predstavljamo

kot zaporedje simbolov. Tedaj lahko izračunamo daljnosežne korelacije med simboli; torej uporabimo pristop, ki je že bil uspešno uporabljen pri dekodiranju DNK [2] in pri analizi pisanih besedil [13].

Ta pristop smo uporabili tudi sami in razvili novo fraktalno metriko, ki temelji na analizi daljnosežnih korelacij zaporedja simbolov - metriko α. V dosedanjem raziskovanju smo pokazali, da daljnosežne korelacije v računalniških programih obstajajo [9], in da je karakteristični koeficient teh korelacij α na neki način skladen z nekaterimi klasičnimi metrikami kompleksnosti, kot npr. Halsteadov volumen, McCabeovo ciklomatično število, LOC, itd. Prav tako smo pokazali, da lahko v nekem omejenem smislu povežemo α s kvaliteto programa [7,8].

Programsko orodje za analizo kompleksnosti programov

Ker smo želeli bolj natančno preveriti naše teoretične izsledke o metriki α, smo potrebovali program oz. orodje, ki nam bi pri tem pomagalo. Prav tako pa smo želeli imeti na enem mestu zbrane vse tiste klasične metrike, ki smo jih prej uporabljali s pomočjo različnih orodij na različnih sistemih. Kot rezultat teh potreb smo razvili programsko okolje Software Complexity Analyzer, ki poleg vsega naštetega združuje še nekatere dodatne lastnosti.

V glavnem želimo analizirati izvorni kod računalniških programov. Zato moramo zagotoviti podporo najrazličnejšim programskim jezikom na čimbolj transparenten način, saj izbira programskega jezika ne sme bistveno vplivati na nadaljnje delo, če naj zagotovimo splošnost. V ta namen vsak programski jezik opišemo z množico atributov, ki jih bomo uporabili pri analizi. Podporo nekaterim programskim jezikom (Pascal, C, C++, Java, Fortran) smo že zagotovili, ostale pa lahko uporabniki kasneje dodajo še sami. Za analizo novih fraktalnih metrik in primerjavo z ostalimi smo dodali še možnost generiranja naključnih, sintaksno pravilnih programov, glede na izbrani programski jezik.

Za analizo z metriko α je potrebno programe pretvoriti v model Brownovega gibanja, pri čemer moramo dopustiti možnost izbire načina kodiranja. Iz modela Brownovega gibanja lahko nato izračunamo regresijsko funkcijo, Fourierjevo transformacijo, ipd. Vse te podatke si lahko predstavimo tudi vizualno v obliki grafov funkcij.

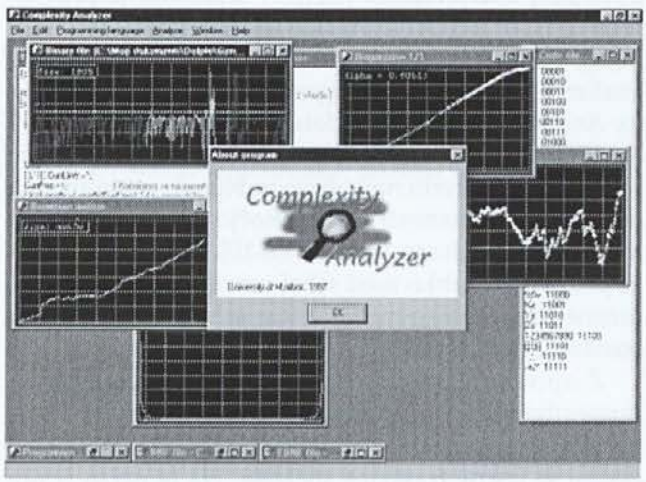
Vsekakor bomo največkrat merili izvorni kod programov, občasno pa bi si gotovo želeli analizirati tudi podatke kakšne druge vrste, npr. izvršljivi kod prevedenega programa. V ta namen smo zagotovili podporo analizi podatkov v dvojiški obliki in na ta način omogočili analiziranje česarkoli, ne le računalniških programov.

Ob metriki α bi radi merili programe tudi z ostalimi klasičnimi metrikami kompleksnosti, zato smo dodali možnost analize z nekaterimi najbolj pogosto uporabljanimi programskimi metrikami. Ker imamo podporo programskim jezikom zagotovljeno in s tem poznamo vse njihove osnovne podatke, za računanje teh klasičnih metrik ne bomo potrebovali veliko dodatnih podatkov in je zato celoten proces analize zelo preprost in neutrudljiv.

Da nam ne bo treba vedno znova ponavljati istih postopkov in analizirati že analiziranih programov, smo dodali podatkovno bazo, v katero lahko shranjujemo vse potrebne podatke, ki nas o nekem programu zanimajo. Shranjujemo lahko osnovne podatke, kot npr. rezultate opravljenih analiz, dodatne podatke o avtorju programa, času pisanja programa, ipd. ter tudi same programe, ki jih imamo tako pregledno zbrane na enem mestu.

Ob izračunih in analizah, ki jih ponuja samo orodje, bi radi včasih opravili tudi nekatere manj uporabljane statistične analize, ali pa dobljene podatke vključili v poročila, izdelana z za to namenjenimi programskimi orodji. Zato lahko izračunane podatke shranimo v obliki, primerni za nekatere splošne (npr. Excel) ali specializirane programe (npr. SPSS, Statistica).

Vse navedene funkcije smo torej združili v enovito programsko okolje. Omogočiti smo želeli čimbolj preprosto delo, vendar zaradi tega nismo zanemarili funkcionalnosti. Poskušali smo poiskati pravo razmerje med zmogljivostjo in preprosto uporabo, ob tem pa pustiti dovolj prostora za bodoče razširitve. Na sliki 1 si lahko ogledamo delovno okolje programskega orodja Software Complexity Analyzer.



Slika 1: Delovno okolje programskega orodja

Podpora programskim jezikom

Pri opisu programskega jezika nas dejansko zanima opis sintakse. Podatke o posameznem programskem jeziku lahko v glavnem razdelimo na tri skupine:

- osnovni podatki o jeziku (ime, način zapisa komentarjev in znakovnih nizov, ločevanje med velikimi in malimi črkami, ipd.),
- Backus-Naurova oblika (BNF) zapisa sintakse jezika, in
- kategorije jezika.

Za opis sintakse jezika uporabljamo nekoliko spremenjeno Backus-Naurovo obliko. Dodali smo namreč nekaj metasimbolov, ki določajo obliko izvornega koda (zamikanje, prehodi v nove vrstice), predvsem pa skrbijo za avtomatsko generiranje naključnih programov. Z njimi določamo verjetnosti prehoda v posamezne veje strukture BNF, omejujemo maksimalno dovoljeno globino rekurzije pri razširjanju neterminalov, omejujemo kompleksnost posameznih programskih blokov (npr. izrazov), določamo bližnjice iz rekurzivnih produkcij, ipd. Vse te podatke zapišemo v razširjeno obliko BNF (EBNF, extended BNF), ki se lahko tvori iz osnovne BNF tudi samodejno, v tem primeru pač dobijo parametri določene privzete vrednosti.

Za nekatere izmed metrik kompleksnosti moramo poznati pomen besed, ki se pojavljajo v izvornem kodu programa. V ta namen smo vse besede programskega jezika razdelili na štiri kategorije: rezervirane besede, operatorji, simboli in standardne funkcije. Za čimbolj enostavno določanje kategorij besedam jezika smo vpeljali možnost interaktivnega sledenja izvornemu kodu, kjer lahko ob prehodu skozi izvorni kod za vsako neprepoznano besedo določimo, v katero kategorijo sodi. Sintaksne lastnosti programskega jezika določamo s pomočjo pogovornega okna na sliki 2.

Slika 2: Eden od zavihkov pogovornega okna za opis sintakse programskega jezika

Analiza z metriko α

Ker je analiziranje programov s klasičnimi metrikami kompleksnosti dobro poznano in je bilo o tem že tudi precej napisanega [3,12], se bomo tukaj omejili predvsem na analizo programov z našo metriko α.

Osnovni potek izračuna koeficienta α izgleda takole: najprej pretvorimo podatke, ki jih želimo analizirati (ponavadi računalniški program, ni pa nujno), v model Brownovega gibanja. Iz tega modela izračunamo nato regresijsko krivuljo, iz nje pa lahko dobimo vrednost α.

Za tvorjenje modela Brownovega gibanja moramo najprej definirati način kodiranja, torej opisati pretvorbo simbolov programa v zaporedje ničel in enic. Na voljo imamo dve možnosti. Za analizo izvornega koda programa (ali kakšnega drugega tekstovnega zapisa) uporabljamo kodirno tabelo, v kateri vsakemu znaku ASCII priredimo ustrezen dvojiški niz. V drugem primeru, ko želimo analizirati poljubne podatke, pa določimo, katere bite bomo uporabili pri tvorbi modela Brownovega gibanja.

Slika 3: Model Brownovega gibanja

Model Brownovega gibanja (slika 3) je osnova za izračun regresijske funkcije in s tem koeficienta α. Za izračun lahko uporabimo celoten model gibanja, lahko pa določimo le posamezen del (ali več delov) ter izračunamo regresijsko krivuljo samo za ta del. Ker je izračun regresije odvisen le od relativnih razlik med točkami v modelu Brownovega gibanja, je popolnoma vseeno, kateri del izberemo. Z analizo posameznih delov lahko med drugim ugotavljamo vrinjenost »čudnih« delov, ki se ne skladajo s celoto in imajo bistveno različno vrednost koeficienta α.

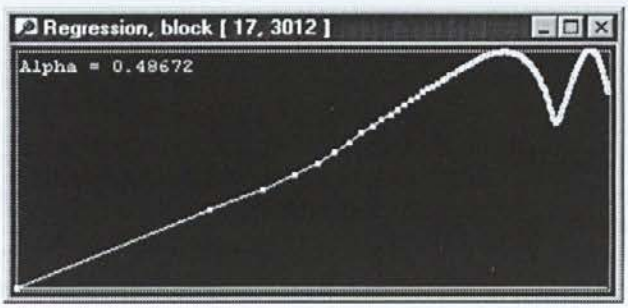
Tista zanimiva veličina, ki jo želimo izračunati iz modela Brownovega gibanja, je regresijska funkcija $F(l)$, ki jo izračunamo:

$$F^2(l) \equiv \overline{[\Delta y(l)]^2} - \overline{\Delta y(l)}^2$$

kjer je $\Delta y(l)=y(l_0+l)-y(l)$ relativna razlika med dvema točkama v modelu Brownovega gibanja in prečna črta pomeni povprečje prek vseh pozicij l.

Ker je izračun te funkcije časovno precej zahteven, smo omogočili več različnih možnosti nastavitev koraka l in s tem natančnosti izračuna same funkcije. Ločimo med logaritemskim in linearnim korakom.

Izračunano regresijsko funkcijo prikažemo kot graf $F(l)$, l na dvojni logaritemski skali (slika 4). Če je skalirna lastnost opisana s potenčnim zakonom, je dobljena krivulja linearna in naklon krivulje (izračunan z metodo najmanjših kvadratov) predstavlja vrednost koeficienta α. Dejansko se za izračun uporabi le prvi del krivulje, ki je linearen, temu pa ponavadi sledi značilno iznihanje (kar je vidno tudi na sliki 4). Do sedaj smo za analizo večinoma uporabljali le linearni del krivulje, pri čemer smo dokazali obstoj daljnosežnih korelacij v računalniških programih [9], pokazali smo, da koeficient α meri sintaksno kompleksnost [10] in jasno smo definirali kritično vrednost [6]. Prav gotovo bi bil za analizo primeren tudi zadnji, nelinearni del krivulje in trenutno se ukvarjamo ravno s potrditvijo hipotez, ki so nastale ob eksperimentiranju. Kot primer lahko navedemo povezanost nihanja krivulje in podobnosti med deli izvornega programa. Število nihajev je namreč identično številu podobnih si delov v programu, amplituda nihajev pa ponazarja koliko so si ti deli med seboj dejansko podobni.



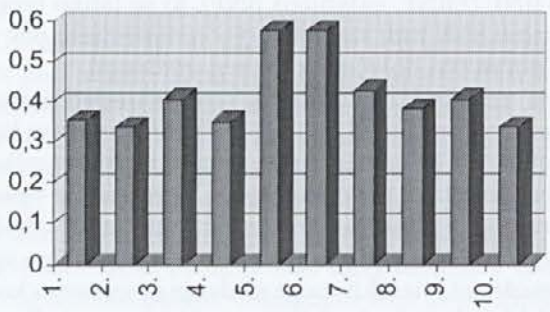
Slika 4: Graf regresijske funkcije

Primer uporabe programskega orodja in metrike α

Za konec si poglejmo še primer uporabe orodja Software Complexity Analyzer in fraktalne metrike α za odkrivanje vrinjenih delov v originalni program. Možnost iskanja posameznih delov, ki se bistveno razlikujejo od preostalega dela programa in na splošno od programa kot celote, lahko koristno uporabimo v mnogih primerih. Če se omejimo na klasično analizo izvornega koda, je seveda jasno, da deli programa, ki očitno izstopajo, zahtevajo še posebno natančen pregled. Po drugi strani lahko na ta način iščemo napake v podatkih, ki so nastale kot posledica aparaturnih okvar ali tehničnih motenj. Tretja možnost uporabe pa bi bilo odkrivanje neželenih vrinjenih delov, kot npr. iskanje virusov.

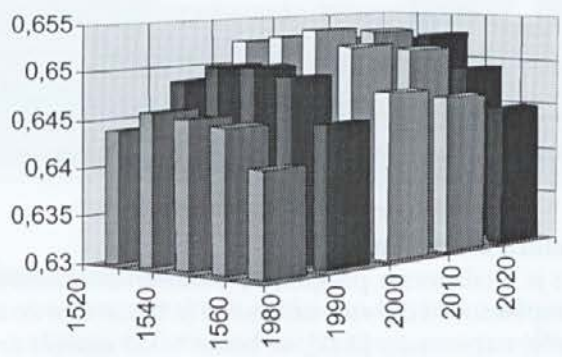
V našem primeru smo v originalen dokument (vzeli smo del besedila iz uvoda tega članka) dolžine 2820 znakov ASCII vrinili na naključno mesto del dolžine 440 znakov (za to smo uporabili del izvornega programa, ki se po pomenu seveda precej razlikuje od navadnega besedila). Tako spremenjen dokument je bil dolg 3260 znakov, kjer je bil v intervalu od 1550. do 1990. znaka vrinjen »neželjeni« del. Ko smo izmerili vrednosti koeficienta α za originalni dokument, za del, ki smo ga vrinili in za končni, spremenjeni dokument, smo ugotovili, da se te vrednosti med seboj precej razlikujejo (original je imel vrednost $\alpha=0.46861$, vrinjeni del $\alpha=0.64970$ in sestavljeni dokument $\alpha=0.57702$). Ti podatki nam seveda lahko služijo le za orientacijo, saj jih sicer ne poznamo, ko se lotimo analize nepoznanega dokumenta. Vidimo pa, da je vključitev relativno majhnega dela (nekaj nad 10% končnega dokumenta) povzročila precejšnjo spremembo vrednosti α.

Iskanja smo se lotili s pretvorbo spremenjenega dela v model Brownovega gibanja, ki smo ga razdelili na več delov in za vsak del posebej izračunali vrednost koeficienta α (slika 5).



Slika 5: Vrednosti α za posamezne dele

Koeficienta α za 5. in 6. del se bistveno ločita od vseh ostalih, zato pričakujemo, da se iskani del nahaja v okolici tega območja (to območje zaseda interval od 1304. do 1956. znaka). Ker je na tem območju vrednost α precej višja kot drugod, bomo poiskali tisti del, na katerem je α najvišji. S postopnim krčenjem območja ugotovimo, da se del z najvišjo vrednostjo α nahaja približno na intervalu od 1540. do 2000. znaka (slika 6). Po natančnem pregledu tega območja najdemo natančen interval, to je od 1540. do 2002. znaka. Če sedaj primerjamo najdeno območje z resničnim stanjem (interval 1550-1990), vidimo, da je poiskana ocena zelo dobra.



Slika 6: Vrednosti α na posameznih območjih

Zaključek

Namen tega članka je bil prikaz programskega orodja Software Complexity Analyzer za analiziranje programov s pomočjo nekaterih najbolj uporabljanih programskih metrik kompleksnosti in s pomočjo nove fraktalne metrike α. Do sedaj se je orodje izkazalo kot učinkovito, predvsem smo z njim analizirali programe, ki so jih izdelali študentje v okviru vaj in kot seminarske naloge.

V prihodnje želimo predvsem opraviti še dodatna testiranja na čim več različnih primerih. Vključili bomo nekatere dodatne metrike kompleksnosti, ki jih doslej še nismo zajeli. Prav tako bomo poskušali izboljšati lastne fraktalne metrike oz. vpeljati tudi nove, ko bomo za to imeli dovolj teoretične podlage.

Literatura

[1] *Brest J., Kokol P., Mernik M., Žumer V.,* Orodje PROMIS in njegova uporaba pri analizi programov, Uporabna informatika 3(4):26-29, 1995.

[2] *Bunde A., Havlin S. (eds.), 1994,* Fractals in Science, *Springer Verlag.*

[3] *Conte S.D., Dunsmore H.F., Shen V.Y.,* Software engineering metrics and models, *Benjamin/ Cummings, 1986.*

[4] *Gell-Mann M.,* What is Complexity, Complexity 1(1):16-19, 1995.

[5] *Harrison W.,* An Entropy-Based Measure of Software Complexity, IEEE Transactions on Software 18(11):1025-1029.

[6] *Kokol P.,* Searching for Fractal Complexity in Computer Programs, SIGPLAN 29(1), 1994.

[7] *Kokol P., Brest J., Mernik M., Žumer V.,* Fractal Program Metrics: a new way to measure the characteristics of computer programs, Computational methods and experimental measurements (CMEM '95), str. 41-48, Capri, Italy, 1995.

[8] *Kokol P., Brest J., Mernik M., Žumer V.,* Automatic Generation of Software Quality Analysis Tools - The Case of Fractal Metrics, Proceedings of the 4th Software Quality Conference, str. 423-432, Dundee, Scotland, UK, 1995.

[9] *Kokol P., Brest J., Žumer V.,* Long Range Correlations in Computer Programs, Proceedings of the 13th European Meeting on Cybernetics and Systems, Vienna, Austria, 1996.

[10] *Kokol P., Brest J., Žumer V., Software Complexity - An Alternative View,* SIGPLAN 31(2):35-41, 1996.

[11] *Morowitz H.,* The Emergence of Complexity, Complexity 1(1):4, 1995.

[12] *Oman P., Pfleger S.L. (eds.),* Applying Software Metrics, *IEEE CS Press, 1997.*

[13] *Schenkel A., Zhang J., Zhang Y.,* Long Range Correlations in Human Writings, Fractals 1(1):47-55, 1993.

[14] *Schroeder M.,* Fractals, Chaos, Power Laws - Minutes from an Infinite Paradise, *W.H.Freeman and Company, 1991.*

[15] *Tsonis A.A., Schultz C., Tsonis P.A.,* Zipf's Law and the Structure and Evolution of Languages, Complexity 2(5):12-13, 1997.

◆

Vili Podgorelec, dipl. ing. rač., je zaposlen na univerzi v Mariboru, na fakulteti za elektrotehniko, računalništvo in informatiko, kot raziskovalec. Diplomiral je leta 1996 in je trenutno vpisan na podiplomski študij. Raziskovalno se ukvarja s programskimi jeziki, analizo programov, teorijo sistemov in z genetskim oz. evolucijskim programiranjem. Je član IEEE in ACM.

◆

Dr. Peter Kokol je zaposlen na univerzi v Mariboru, na fakulteti za elektrotehniko, računalništvo in informatiko, kot docent s področja računalništva. Doktoriral je leta 1992. Raziskovalno se ukvarja s programskimi jeziki, analizo programov, razvojem informacijskih sistemov in teorijo sistemov. Je avtor mnogih člankov, objavljenih v zbornikih mednarodnih konferenc in v mednarodnih revijah ter je član IEEE, ACM, ISCA in Slovenskega društva Informatika.

◆

Janez Brest, dipl. ing. rač., je zaposlen na univerzi v Mariboru, na fakulteti za elektrotehniko, računalništvo in informatiko, kot stažist-asistent. Diplomiral je leta 1995 in je trenutno vpisan na podiplomski študij. Raziskovalno se ukvarja s programskimi jeziki, analizo programov in generatorji prevajalnikov. Je avtor večih člankov, objavljenih v zbornikih mednarodnih konferenc in v mednarodnih revijah ter je član IEEE.