

VILNIAUS UNIVERSITETAS  
MATEMATIKOS IR INFORMATIKOS FAKULTETAS  
PROGRAMŲ SISTEMŲ KATEDRA

# **Neuroninių tinklų panaudojimas testavimo proceso gerinimui**

**Using neural networks to improve the testing process**

Bakalauro darbas

Atliko:	Ričardas Mikelionis	(parašas)
Darbo vadovas:	asist. dr. Vytautas Valaitis	(parašas)
Darbo recenzentas:	partn. prof., dr. Aldas Glemža	(parašas)

Vilnius – 2018

## **Santrauka**

Glaustai aprašomas darbo turinys: pristatoma nagrinėta problema ir padarytos išvados. Santraukos apimtis ne didesnė nei 0,5 puslapio. Santraukų gale nurodomi darbo raktiniai žodžiai.

**Raktiniai žodžiai:** raktinis žodis 1, raktinis žodis 2, raktinis žodis 3, raktinis žodis 4, raktinis žodis 5

## **Summary**

Santrauka anglų kalba. Santraukos apimtis ne didesnė nei 0,5 puslapio.

**Keywords:** keyword 1, keyword 2, keyword 3, keyword 4, keyword 5

## TURINYS

ĮVADAS .....	4
1. PROGRAMINIO KODO SUDĖTINGUMO IR DEFEKTYVUMO SĄSAJOS .....	5
1.1. McCabe sudėtingumo metrikos .....	5
1.2. Halstead sudėtingumo metrikos .....	6
2. SUDĖTINGUMO METRIKŲ PANAUDOJIMAS APRAŠANT PROGRAMINĖS ĮRAN- GOS DEFEKTYVUMĄ NUSPĖJANČIĄ BESIMOKANČIĄ MAŠINĄ .....	7
2.1. Sprendimų medžio metodu paremta besimokanti mašina .....	7
2.2. Giliuoju neuroniniu tinklu paremta besimokanti mašina .....	7
3. DUOMENŲ ATRINKIMAS .....	8
3.1. Rankiniu būdu ieškant atributų kurie yra tiesiogiai proporcingi kitiems atributams ....	8
3.2. Duomenų analizė naudojant įrankį Weka .....	8
4. NEURONINIO TINKLO EFEKTYVUMO GERINIMAS NAUDOJANT ATRINKTUS DUOMENIS .....	9
REZULTATAI IR IŠVADOS .....	10
SANTRUMPOS .....	11
PRIEDAI .....	11
1 priedas. Neuroninio tinklo struktūra .....	12
2 priedas. Eksperimentinio palyginimo rezultatai .....	13

## Įvadas

Smarkiai augant Continuous Delivery principų populiarumui auga poreikis testuoti daugiau per trumpesnę laiko tarpą. Paprastas sprendimas šiai problemai būtų be abejo nuolat besisukantis automatinių testų paketas, tačiau automatiniais testais padengti programinį kodą 100% praktiškai neįmanoma. 100% padengimas įmanomas tik pagal kokią nors specifinę metriką, o vaiktis tikrojo 100% padengtumo testais pagal kiekvieną metriką ar funkcinę sritį prilygsta šviesos geičio vaikymuisi, ku arčiau esame tikslo tuo daugiau pastangų ir resursų reikia pasistūmėti į priekį. Todėl, žinoma, vis dar išlieka poreikis rankiniam testavimui. Norint išleisti programinės įrangos versiją ku dažniau ištestuoti visko kiekvieną kartą neįmanoma. Taip šis problemos sprendimas iškelia dar vieną, mažesnę, problemą: kaip efektyviai pasirinkti testavimo sritis kiekvienai naujai programos laidai (angl. release). Alan M. Davis [**Davis:1995:PSD:203406**] tegia, jog pareto principą galima pritaikyti ir programinės įrangos testavime: čia 80% kode esančių defektų surandami 20% viso kodo.

Tobulame pasaulyje iteracijai pasirinktos testavimo sritys ir apims tuos 20% problematiškojo kodo. Tačiau dažnai net remiantis visa turima istorine informacija testuotojui atsakingam už testavimo sričių parinkimą yra sunku efektyviai atrinkti testavimo sritis, kuriose atliktas darbas turės didžiausią įtaką programinės įrangos kokybei.

Sieniant sėkmingai įgyvendinti darbo tikslą siekiama įgyvendinti šiuos uždavinius:

1. Išrinkti metrikas darančias įtaką programinės įrangos sudėtingumui
2. Palyginti efektyvumą tarp sprendimų medžio besimokančios mašinos ir neuroninio tinklo naudojant surinktus duomenis
3. Atrinkti duomenis turinčius mažiausią įtaką galutiniam spėjimui
4. Naudojantis mažesniu atributų kiekiu iš naujo apmokyti neuroninį tinklą ir palyginti rezultatus su pilnų duomenų rinkinių apmokyto neuroninio tinklo

# 1. Programinio kodo sudėtingumo ir defektyvumo sąajos

Kiekvieną kartą renkatis testuojamas programinės įrangos vietas, žinoma, jei nėra testuojamas visas programinės įrangos funkcionalumas, reikia turėti atskaitos tašką, kuris leistų nuspręsti kurios programinės įrangos kodo vietos, ar programų paketo moduliai turi didžiausią riziką būti defektyvūs. Į šią kategoriją dažnai pakliūva seniai testuoti, nauji, atnaujinti bei anksčiau daug defektų turėję programinės įrangos moduliai. Didelė problema, su kuria vis dažniau susiduriama yra ką daryti, jei nėra laiko ištestuoti visiems šiems moduliams ir kaip tuomet pasirinkti atitinkamus programinės įrangos modulius testavimui, kad nešvaistydami laiko padarytumėme didžiausią įmanomą įtaką programinės įrangos kokybės užtikrinimui. Reikėtų sukurti detalesnę programinės įrangos metrikų sistemą nei, „naujumas“ ar „prieš tai buvusių defektų kiekis“, kuri tiesiogiai atspindėtų būsimą (ar esamą) programinės įrangos defektyvumą. O gal būt užtenka pritaikyti egzistuojančias programinio kodo metrikas randant koreliaciją su programų defektyvumu.

Ir iš tiesų, būtų galima teigti, jog programinės kokybės metrikos kurias būtų įmanoma tiesiogiai susieti su programinio kodo defektyvumu egzistuoja. Arčiausiai tokių metrikų yra programinio kodo sudėtingumo metrikos. Be abejo yra ne vienas būdas matuoti programos kodo sudėtingumą, tačiau, šiam tyrimui pasirinktos dvi, galima būtų sakyti, populiariausios metrikos kodo sudėtingumui atvaizduotii: Maurice H. Halstead aprašytosios 1977-aisiais [**Halstead:1977:ESS:540137**] bei Thomas J. McCabe aprašytosios 1976-aisiais [**McCabe:1976:CM:800253.807712**].

Ryšys tarp kodo sudėtingumo bei defektų buvimo jame, o gal būt derėtų sakyti defektų buvimo galimybės kode, atrodytų, gana logiškas: kuo sudėtingesnis kodas, tuo sunkiau jį skaityti, kuo sunkiau kodas skaitomas tuo sunkiau jį plėsti, todėl kyla rizika defektų atsiradimui. Tokia prielaida tiriama nuo pat sudėtingumo metrikų aprašymo. Žinoma, dėl kodo sudėtingumo tiksliai defektų kiekiui daromo efekto kyla nestuorimų. Atsiranda teigiančių, jog Halstead metrikos neturi jokio įrodomo ryšio su defektų buvimu, tačiau yra ir tyrimų pagrindžiančių kodo sudėtingumo bei defektų buvimo jame koreliaciją [**Schroeder1999APG**]. Dažniausiai tyrėjai lieka šio argumento viduryje nei visiškai neigdami, nei patvirtindami šią koreliaciją. Gana dažnas sprendimas yra pamažinti metrikų kiekį ir patikrinti metrikas priklausomai nuo situacijos, taip užtikrinant, kad pasirinktosios metrikos tikrai koreliuoja su tyrimo metu ieškomu fenomenu, šiuo atveju kodo defektyvumu [**Metrics in Evaluating Software Defects:2013**].

Šiame dokumente aprašomame tyrime bus naudojamos visos Halstead ir McCabe metrikos kartu, bei keletas šių atributų poaibių susiaurintų tiek rankiniu būdu tiek duomenų analizės įrankiais.

## 1.1. McCabe sudėtingumo metrikos

1976-aisiais Thomas J. McCabe išleido tyrimą kuriame pasiūlė naują programinės įrangos kodo sudėtingumo matavimo būdą, pavadintą Ciklomatiniu sudėtingumu (angl. Cyclomatic Complexity). Šis matavimo vienetas kiekybiškai matuoja nepriklausomų kelių per kodo elementus kiekį.

Ciklomatinis sudėtingumas skaičiuojamas naudojant orientuotą grafą kuris vaizduoja programos kodą. Tokiu atveju grafo viršūnės atspindi komandų kurios vykdomos kartu grupes, o kraštinė su kryptimi sujungia dvi viršūnes jei jos gali būti vykdomos viena po kitos. Tokį ciklominio sudėtingumo skaičiavimo metodą galima taikyti ir aukštesnės abstrakcijos programinio kodo vienetams kaip funkcijos, metodai, klasės ar kt.

## **1.2. Halstead sudėtingumo metrikos**

- 2. Sudėtingumo metrikų panaudojimas aprašant programinės įrangos defektyvumą nuspėjančią besimokančią mašiną**
- 2.1. Sprendimų medžio metodu paremta besimokanti mašina**
- 2.2. Giliuoju neuroniniu tinklu paremta besimokanti mašina**



### **3. Duomenų atrinkimas**

- 3.1. Rankiniu būdu ieškant atributų kurie yra tiesiogiai proporcingi kitiems atributams**
- 3.2. Duomenų analizė naudojant įrankį Weka**

#### **4. Neuroninio tinklo efektyvumo gerinimas naudojant atrinktus duomenis**

## **Rezultatai ir išvados**

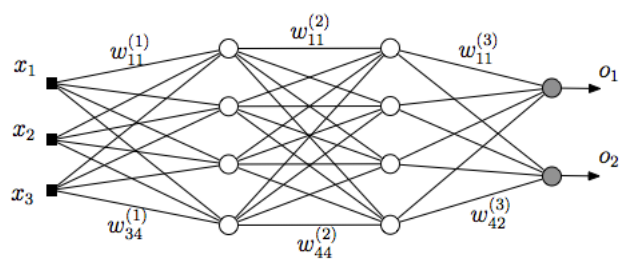
Rezultatų ir išvadų dalyje išdėstomi pagrindiniai darbo rezultatai (kažkas išanalizuota, kažkas sukurta, kažkas įdiegta), toliau pateikiamos išvados (daromi nagrinėtų problemų sprendimo metodų palyginimai, siūlomos rekomendacijos, akcentuojamos naujovės). Rezultatai ir išvados pateikiami sunumeruotų (gali būti hierarchiniai) sąrašų pavidalu. Darbo rezultatai turi atitikti darbo tikslą.

## **Santrumpos**

Sąvokų apibrėžimai ir santrumpų sąrašas sudaromas tada, kai darbo tekste vartojami specialūs paaiškinimo reikalaujantys terminai ir rečiau sutinkamos santrumpos.

## Priedas nr. 1

### Niauroninio tinklo struktūra



1 pav. Paveikslėlio pavyzdys

## Priedas nr. 2

### Eksperimentinio palyginimo rezultatai

1 lentelė. Lentelės pavyzdys

Algoritmas	$\bar{x}$	$\sigma^2$
Algoritmas A	1.6335	0.5584
Algoritmas B	1.7395	0.5647