KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Kristina Smilgytė

Dirbtinių neuroninių tinklų taikymas parenkant testavimo metodą

Magistro darbas

Darbo vadovė

m.d. dr. Jovita Nenortaitė

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

INFORMACIJOS SISTEMŲ KATEDRA

Kristina Smilgytė

Dirbtinių neuroninių tinklų taikymas parenkant testavimo metodą

Magistro darbas

Recenzentas
prof. dr. Eduardas Bareiša

2011-05-30

Darbo vadovė
m.d. dr. Jovita Nenortaitė

2011-05-23

Atliko
IFM-9/4 gr. studentė
Kristina Smilgytė

2011-05-23

Kaunas, 2011

Artificial Neural Networks Application in Software Testing Selection Method

Summary

The importance of software testing is growing as a concurrent part of software development. In order to improve the financial allocation of the software testing, software developers have to make a choice between automatic and manual testing methods. The solution related to the problematic choice of testing methods is presented in this work. The main concept of the method is to present the recommendation whether the automatic or manual testing method is better to use or whether their usage is simply adequate. The choice of the testing method is based on the application of the artificial neural networks. Experimental investigations on artificial neural networks structure selection and method evaluation showed that presented idea could be worth as substantiation of the choice selection. The results of the method should be considered as the recommendation since the accuracy depends on the data which was used for training. According to this reason this method is more suitable for those companies or customers which already have historical testing data of the project.

Keywords: artificial neural networks, software testing, project manager, experiments

Turinys

1. Įvadas

Programinės įrangos kūrimas dažniausiai remiasi proporcingu laiko ir kokybės suderinimu. Daugumai kompanijų projektų užbaigimas laiku ir neviršijant skirto biudžeto reikalauja daug pastangų, norint tai pasiekti dažnai stokojama dėmesio kokybei. Išleidus produktą, dėl kūrimui skirto laiko trūkumo vartotojai dažnai susiduria su įvairiomis funkcionalumo neišpildymo ar nekorektiško įgyvendinimo problemomis. Vienas iš kokybės gerinimo būdų – tinkamas sistemos testavimas. Vykdant didelius projektus didelę reikšmę turi tikslūs ir motyvuoti sprendimai. Renkantis testavimo metodą nemažai diskusijų susilaukia rankinio ir automatinio testavimo parinkimas. Vis dėlto, vieningo atsakymo į klausimus: kada, kodėl, kaip ir kuris metodas turi būti naudojamas nėra.

Siūlomas metodas skirtas palengvinti ir pagrįsti testavimo metodo parinkimą. Metodas padės nuspręsti, ar reikalingi automatiniai testai. Tokiu būdu bus galima ne tik įvertinti testavimo tipo privalumus ir trūkumus, bet ir kuriamo metodo gautus rezultatus. Kuriant metodą, skirtą testavimo būdo parinkimui, siekta išlaikyti tikslumą, kad jo taikymas testavimo procese galėtų garantuoti testavimo efektyvumą bei kokybę. O metodo įgyvendinimui pasirenkant dirbtinio intelekto metodą.

Todėl šio darbo tyrimo sritis – programinės įrangos testavimo ir dirbtinio intelekto metodai, o tyrimo objektas – dirbtiniu intelektu pagrįstas metodas, skirtas testavimo metodo parinkimui. Siekiant surasti tinkamą sprendimą buvo analizuojama tiek dirbtinio intelekto metodų savybės, panaudojimo galimybės, tiek ir automatinio, rankinio testavimo metodų parinkimą lemiantys kriterijai.

Darbo tikslas – pagerinti testavimo metodų parinkimą, pasiūlant intelektualų testavimo tipo parinkimo metodą, paremtą dirbtinio intelekto metodų taikymu.

Darbo uždaviniai:

1. Išanalizuoti, testavimo metodus, automatinius testus bei jų parinkimo ir naudojimo galimybes.
2. Išanalizuoti dirbtinio intelekto metodus bei jų taikymo galimybes programinės įrangos testavime.
3. Remiantis atlikta analize nuspręsti, kuris iš dirbtinio intelekto metodų (dirbtiniai neuroniniai tinklai, genetiniai algoritmai, spiečiaus intelektas ir pan.) galėtų būti panaudotas testavimo metodo parinkimo kūrime.
4. Pasiūlyti intelektualų programinės įrangos testavimo metodą, kuris galėtų būti naudojamas projekto vadovo ar testuotojo, ir kuris padėtų pasiūlyti testavimo metodo parinkimo įrankį. Intelektualaus programinės įrangos testavimo metodo

taikymas testavimo procese turėtų garantuoti testavimo efektyvumą bei testavimo tikslumą.

5. Atlikti pasiūlyto testavimo metodo parinkimo įrankio analizę, bei nustatyti naudotinus parametrus (pvz.: jei bus nutarta naudoti neuroninius tinklus reiks nustatyti įėjimų skaičių, parinkti neuroninio tinklo struktūrą ir pan.).
6. Įvertinti intelektualaus testavimo metodo taikymo galimybes ir perspektyvas.
7. Sukurti prototipą pasiūlyto metodo veikimo demonstravimui.

Nagrinėjant testavimo specifiką ir testavimo parinkimą lemiančius veiksnius, buvo remiamasi [1-7], [26] šaltiniais, kuriuose aprašoma testavimo metodai, optimalus testavimo kiekis, rankinio ir automatinio testavimo savybės, metrikos. Tyrinėjant dirbtinio intelekto metodus, buvo analizuojami dirbtiniai neuroniniai tinklai [8], [9], [22], [30], miglotos informacijos tinklas [9], [10], genetiniai algoritmai [8], [11], [12], spiečiaus intelektas [13], [14], [15], [16], [17], [18], skruzdžių kolonijos optimizavimo algoritmas [18], [19], dalelių spiečiaus optimizavimo algoritmas [15], [18], [20], [21], [22], [23], genetinio algoritmo ir spiečiaus intelekto palyginimas [24], [25], [27]. Sprendimo gerinimo galimybės paremtos [33], [34] šaltiniais.

Vykdant magistro tiriamąją dalį atliktos testavimo metodų, dirbtinio intelekto analizės, išsiaiškinti visų šių metodų privalumai, trūkumai. Atsižvelgiant į gautus analizės rezultatus nuspręsta kurti testavimo metodą, paremtą dirbtiniais neuroniniais tinklais, skirtą testavimo tipo (automatinis ar rankinis) parinkimo palengvinimui. Net jeigu šis metodas ir neišspręs pagrindinės parinkimo problemos, jis gali būti naudojamas, kaip papildoma priemonė renkantis testavimo tipą.

Nustačius 17 testavimo parinkimą lemiančių kriterijų, eksperimentiniu būdu surasta dirbtinių neuroninių tinklų struktūra. Optimalūs rezultatai gauti naudojant daugiasluoksnį neuroninį tinklą su dviem paslėptais sluoksniais, 6 neuronai pirmame ir 5 antrame sluoksniuose, tikslumo funkcija *SSE* ir tinklą apmokant *Conjugate Gradient with Powell/Beale Restarts* algoritmu. Eksperimentams naudota 1000 duomenų imtis, kuri paruošta atsižvelgiant į kriterijus, kurie yra neuroninio tinklo įėjimai, o jų atsakymai – tinklo išėjimo reikšmės.

Skirtingai nei kiti egzistuojantys parinkimo būdai tarp rankinio ir automatinio testavimo, pasiūlytas metodas apibendrina parinkimą įtakojančius kriterijus ir kompanijos sukauptą patirtį panaudoja ateities prognozėms. Neuroninis tinklas išmoksta sudėtingas sąsajas tarp turimos patirties duomenų greičiau nei per minutę (remiantis eksperimentų rezultatais tūkstančiui duomenų išmokti pakanka 3 sekundžių). Kuriant parinkimo metodą vienas pagrindinių vertinimo aspektų – pasiekiamas tikslumas. Remiantis eksperimentinių

tyrimų duomenimis apmokant tinklą su 1000 duomenų gauta, kad naujų duomenų prognozė bus pateikta su 80% tikslumo tikimybe.

Siūlomas metodas gali būti vertingas pagalbininkas testavimo sprendimu suinteresuotiems žmonėms, kai nėra pakankamai laiko analizuoti ankstesnių atvejų, kai trūksta patirties, dvejojama ar reikia parinkimo pagrįstumo. Metodo idėja gali būti pritaikoma ir kitoms dalykinėms sritims, kuriose sprendžiama parinkimo problema.

Ateityje galima gerinti metodo rezultatų tikslumą derinant neuroninius tinklus su kitais skaitinio intelekto metodais. Tarptautinėje hibridinių dirbtinio intelekto sistemų konferencijoje pristatomi tyrimai patvirtina susidomėjimą ir teikiamą naudą. Parinkimo metodo atveju, galbūt pavyktų pasiekti, kad neuroninis tinklas mokytųsi tik iš kiekvieną kartą atrinktų duomenų, taip sumažinant paklaidą didinančių įrašų kiekį.

Antrame skyriuje pateikta tyrimo srities analizė. Išnagrinėta tiek testavimo problemos, esamos testų parinkimo galimybės, metrikos, tiek dirbtinio intelekto metodai testavime, parinkimo uždaviniuose. Suformuluota pasiūlyto metodo idėja. Trečiame skyriuje pateikta metodo specifikacija. Ketvirtame skyriuje detalizuota metodo koncepcija, naudojami kriterijai, formalizuota neuroninio tinklo architektūra. Penktame skyriuje aprašytos realizavimo priemonės, optimalios struktūros ieškojimui naudoti parametrai. Šeštame skyriuje pateikti atrinkti eksperimentinių tyrimų duomenys, supažindinta su nagrinėtais neuroninio tinklo parametrais. Apibendrinta nustatyta ANN struktūra, duomenų kiekio įtaka tinklo apmokymui ir pateiktos sprendimo taikymo rekomendacijos. Septintame skyriuje suformuluotos darbo išvados.

Darbo rezultatai buvo pristatyti:

1. 2010 m. konferencijoje „Informacinė visuomenė ir universitetinės studijos“, straipsnis priimtas spausdinti leidinyje „Informacinės technologijos“, ISSN 2029-249X ir pateiktas 1 ir 2 prieduose.
2. 2011 m. tarptautinėje konferencijoje „*Hybrid Artificial Intelligence Systems*“, straipsnis išspausdintas konferencijos leidinyje [35] ir pateiktas 3 priede.

2. Metodo analizė

2.1. Analizės tikslas

Išanalizuoti esamus testavimo metodus, rasti jų privalumus ir trūkumus (jų efektyvumą, kaštus). Nustatyti, kokie gali būti ar yra testavimo metodų vertinimo parametrai. Išanalizuoti dirbtinio intelekto metodus ir remiantis analizės metu gautais rezultatais nuspręsti, kokiais metodais turi būti grindžiamas siūlomas metodas.

Tyrimo analizei atlikti pasirinkti mokslinės literatūros analizės ir apibendrinimo metodai.

2.2. Tyrimo sritis, objektas ir problema

Magistrinio darbo tyrimo objektas – dirbtiniu intelektu pagrįstas metodas, skirtas testavimo tipo parinkimui.

Įvairių programų testavimas turi didelę įtaką galutinio produkto bendrai kokybei ir patikimumui. Aptikus įvairias klaidas dar kūrimo procese, jos kainuoja daug mažiau nei, kad jau atidavus produktą vartotojui(-ams). Programinės įrangos testavime susiduriama su eile problemų, tokių kaip testavimo tikslumas, testavimo efektyvumas, testų pakartotinis panaudojimas, testavimo tipo (automatinis ar rankinis) parinkimas ir pan. Ne visada reikia ir apsimoka automatizuoti testavimą [1], todėl yra svarbu tinkamai įvertinti testavimo tipo parinkimą. Siekiant, sutaupyti testavimui skiriamą laiką nesumažinant testavimo rezultatų gavimo efektyvumo ir tikslumo, buvo nuspręsta ištirti programinės įrangos testavimo ir dirbtinio intelekto metodus. Išnagrinėjus šiuos metodus tikimasi sukurti intelektualų testavimo metodą, kuris galėtų būti naudojamas testų ar jų tipų parinkime, bei pačiame testavimo procese, atitinkamai „apmokant“ testus, t. y. keičiant jų parametrus, pasiūlymo galimybę. Literatūroje randami adaptyvių bei intelektualių testų pavyzdžiai rodo, kad tokie testavimo metodai yra ypač taikytini testuojant dideles ir sudėtingas sistemas. Siekiant sukurti intelektualų metodą, skirtą programinės įrangos testavimo būdo parinkimui, yra svarbu išanalizuoti šiandieninius testavimo metodus, rasti jų privalumus ir trūkumus, kad vėliau būtų galima geriau suprasti, ar reikalingas naujas, koks nors metodas ir jeigu taip, tai koks jis turėtų būti ir kokias problemas jis turėtų spręsti.

2.3. Tyrimo objekto analizė

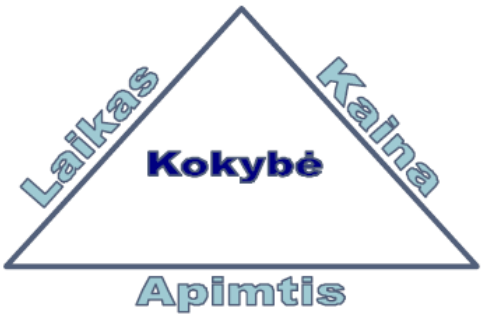
Programinės įrangos testavimas – tai procesas, kurio metu tikrinamas produkto veikimas, kad galėtume atsakyti į klausimą „Ar programinė įranga veikia, kaip yra apibrėžta ir tikimasi?“. Vykdant testavimą bandoma surasti skirtumų tarp gautų ir laukiamų rezultatų, nes kuo ankščiau aptinkama klaida, tuo jos ištaisymas kainuoja mažiau. Reikia nepamiršti, kad testavimas parodo klaidų buvimą, bet ne jų nebuvimą.

1 pav. pateikiamas testavimo proceso ciklas, kuriame atrenkami reikalavimai sistemai ar jos daliai, vertinimo kriterijai, testavimo strategija, planas ir testiniai rinkiniai. Vėliau duotai sistemai, objektui ar programai šie testai vykdomi bei fiksuojami rezultatai. Lyginant gautas reikšmes su laukiamomis sprendžiama apie aptiktas klaidas ir sistemos atitikimą reikalavimams.

1 pav. Testavimo ciklas [2]

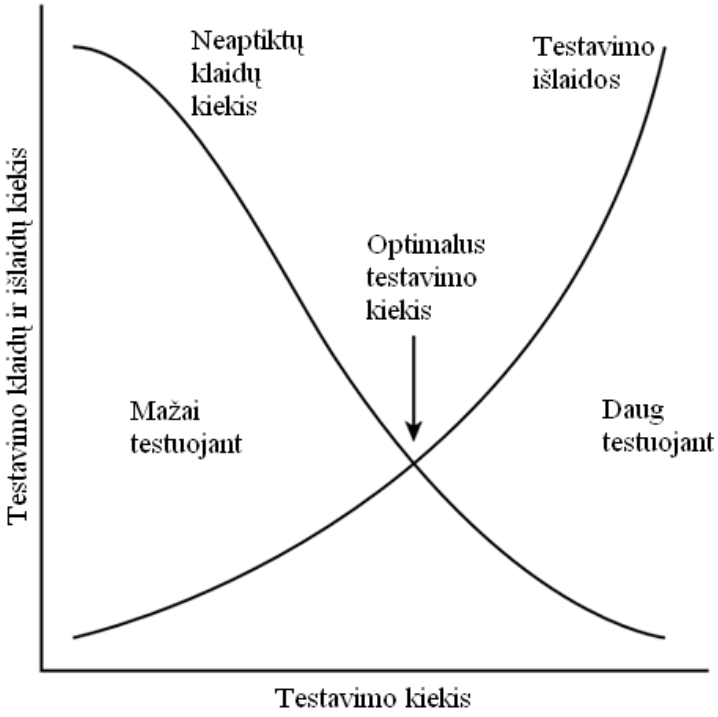
Testavimo metodų yra daug ir įvairių (2 pav.). Ne visi jie yra naudojami kartu ar tai pačiai programai ar paprogramei. Geriausiai tinkantis metodų rinkinys atrenkamas testavimui siekiant sumažinti darbo poreikį bei rezultatų gausą. Kuo programinė įranga yra sudėtingesnė, tuo daugiau laiko gali reikėti ją ištestuoti (3 pav.) – patikrinti jos atitikimą reikalavimams, specifikacijoms, jos funkcinį stabilumą, veikimą, tinkamumą naudoti. Laikas tampa labai svarbus kriterijus programinės įrangos kūrime, nes jis tiesiogiai susijęs su kaina. Laikome, kad apimtis yra pastovi.

2 pav. Testavimo metodų pavyzdžiai [3]



3 pav. Projekto kokybė [4]

Kartais, kai vėluoja pirmesnės projekto dalys, bandoma sutaupyti testavimo laiko sąskaita, taip paliekant sistemą prasčiau ištestuotą. Iš kitos pusės, geras ir mažiau laiko reikalaujantis testavimas įmonei gali laimėti laiko, taip ji atliktų projektus greičiau nei įprastai ir galėtų padidinti savo pelną ar būti pranašesnė konkurentų tarpe.



4 pav. Kiekvienas IT projektas turi testavimo kiekio optimalią reikšmę [5]

Ištestuoti viską yra sudėtinga ne tik dėl laiko ir išlaidų sąnaudų, bet ir todėl, kad pats testavimas parodo tik klaidų buvimą, bet neparodo, kad daugiau klaidų jau nebėra. 4 pav. pavaizduota kokybės ir testavimo kiekio priklausomybė nuo testavimo kainos ir nesurastų klaidų. Bandant ištestuoti viską aptinkamos klaidos pradeda neatsipirkti pagal kainą, nes jų suradimui išleidžiama daug daugiau. Jeigu testuojama taupant lėšas, ar blogai atsirenkant, ką testuoti, yra paliekama daug svarbių klaidų, kurios pateikus sistemą vartotojui kainuos dar daugiau [5]. Todėl reikia kiekvienam projektui surasti ar nuspręsti, koks optimalus kokybės ir išlaidų variantas tenkina projektą ir įmanomas jį kuriančiai organizacijai.

Testavimo procesą pagreitina automatinių testų naudojimas. Jis naudojamas testavimo atvejų generavime, pavyzdžiui, galima greičiau ir daugiau patikrinti ar vartotojo įvedami veiksmai yra teisingi ir ar gaunamas rezultatas, kurio tikimasi. Taip pat naudojamas atgaliniame testavime, kurio tikslas yra patikrinti naujų programinės įrangos (PĮ) galimybių funkcionavimo teisingumą ir įsitikinti, kad nauji PĮ pakeitimai nesukėlė klaidų tarp jau ištaisytų. Bandant ištestuoti neautomatiniu būdu, patikrinama nedidelė programinės įrangos dalis arba išeikvojama daug laiko ir sąnaudų. Testuojant PĮ kartais yra geriau naudoti rankinį testavimą, kartais automatinį, o kartais juos abu. Kada kurį metodą geriau naudoti galima nuspręsti iš jų privalumų ir trūkumų palyginimo [6].

Rankinio testavimo

- Privalumai:
    1. Rankinis testavimas gali būti naudojamas tiek mažuose, tiek dideliuose projektuose.
    2. Nesudėtinga sumažinti ar padidinti testinius atvejus atsižvelgiant į projekto eigą.
    3. Pradedantiesiems rankinis testavimas yra lengvai įsisavinamas.
    4. Rankinis testavimas yra patikimesnis lyginant su automatiniu (neretai automatizuotas testavimas neapima visų galimų testavimo atvejų).
    5. Rankinis testavimas suteikia testuotojui galimybę geriau vykdyti *ad-hoc*[1] testavimą. Daugiau klaidų aptinkama naudojant *ad-hoc* būdą, nei automatinį.
- Trūkumai:
    1. Rankiniu būdu realių apkrovų ir spartų išbandymas dideliems vartotojų skaičiams sunkiai įmanomas.
    2. Testų vykdymas rankiniu būdu yra labai daug laiko reikalaujantis darbas.
    3. Sudėtinga atlikti spalvų kombinacijų atskyrimą.

Automatinio testavimo

- Privalumai:
    1. Nesudėtinga išbandyti daug testavimo atvejų per ribotą laiką.
    2. Galima atlikti apkrovos, įvairius našumo testus naudojant specialius įrankius.
    3. Atgalinio testavimo metodui – pats geriausias pasirinkimas.

[1] Ad-hoc testavimas vykdomas nesuvokiant testinių atvejų sukūrimo struktūros ir be išankstinio suplanavimo. Viena iš paskirčių testavimo pilnumui patikrinti ir įvairūs klaidų atradimai.

4. Naudojamas ne vieną kartą automatinis testavimas padeda sutaupyti labai daug laiko.
5. Automatinį testavimą tuo pačiu metu galima atlikti skirtingose operacinėse sistemose.

❖ Trūkumai:

1. Testavimo atvejų realizavimas į testavimo programinį kodą ir jo atnaujinimas reikalauja daug laiko.
2. Automatinis testavimas yra brangesnis lyginant su rankiniu testavimu.
3. Automatinio testavimo įrankiai net jei ir aptinka grafinius elementus, jie negali pasakyti, ar jų požymiai yra pakankamai informatyvūs ar ne. Tokiuose atvejuose tik pats testuotojas gali tinkamai įvertinti.

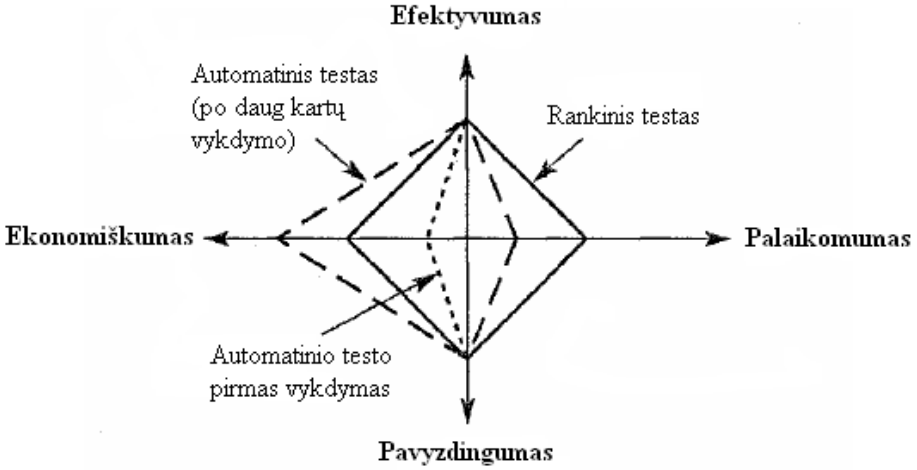
1 lentelėje esantys apibendrinti testavimo tipų privalumai ir trūkumai yra bendro pobūdžio, todėl konkrečiam projektui reikia detalesnės informacijos norint parinkti testavimo tipą. Pavyzdžiui, vertinant pagal testavimo aprėptį, jeigu reikia ištestuoti platesnę sritį, tai geriau rinktis rankinį testavimą. Jeigu svarbu tikslus peržiūrėjimas tos pačios srities, ar siauresnės srities gilesnis ištestavimas, geriau rinktis automatinį. Praktikoje automatinis testavimas negali visiškai pakeisti rankinio testavimo, todėl parinkimo problema išlieka aktuali.

1 lentelė. Testavimo tipų palyginimas

Kriterijus	**Rankinis**	**Automatinis**
Kaina	(+) Pigesnis	(–) Brangesnis
Laikas	(–) Vykdomi lėčiau	(+) Vykdomi greičiau
	—	(–) Testų kodo parengimas lėtas
Reikalavimų pasikeitimai	(+) Nesudėtingai įgyvendinami	(–) Reikalingas testų kodo pakeitimas
Testavimo aprėptis	(+) Platesnė	(–) Fiksuota
Teksto skaitymas	(+) Skaito	(–) Neskaito, nesuvokia prasmių
Spalvų skyrimas	(–) Priklauso nuo jų kombinacijų	(+) Gerai skiria
Atgalinis testavimas	(–) Sudėtingas, laiko reikalaujantis	(+) Puikiai tinka
Išmokstamumas	(+) Nesudėtingas	(–) Sudėtingas, ypač pradedantiems
Klaidų aptikimas žvalgantis, *Ad-hoc*	(+) Daugiau klaidų randama	(–) Mažiau

Papildomam apibendrinimui pateikiamas 5 pav., vaizduojantis nuo ko priklauso testavimo atvejų „gerumas“. Efektyvumas parodo, ar surandama klaidų, ar bent jau yra polinkis aptikti klaidas. Pavyzdingumas parodo, kiek turi sugebėti testuoti dalykų, taip apimant kelis testavimo atvejus. Kiti du kriterijai apimą kainą: kiek yra ekonomiška vykdyti,

šalinti defektus, analizuoti konkretų testavimo atvejį ir kiek reikalauja priežiūros, palaikymo kiekvieną kartą programinei įrangai pakitus ar pasikeitus [7].



5 pav. Testavimo atvejų „gerumas“ pagal Keviat diagramą [7]

Iš 5 pav. galima vaizdžiau palyginti automatinį ir rankinį testavimus. Automatinio testavimo nauda priklauso nuo pakartotino panaudojimo skaičiaus. Jeigu testas bus naudojamas vieną kartą, geriau naudoti rankinį testavimą, nes šiuo atveju jis yra pranašesnis už automatinį. Tačiau ši diagrama nenurodo, kurį būdą geriau naudoti konkrečiu atveju. Pavyzdžiui, turimas konkretus IT projektas ir projektų vadovui yra gan sudėtinga nuspręsti, kurį metodą geriau taikyti, ar apsimoka išleisti papildomas išlaidas automatiniam testavimui.

2.4. **Metrikos**

Norint sėkmingai įvertinti testavimo metodus nėra konkretaus ir vieningo atsakymo, koki matai ar parametrai turi būti naudojami. Naudojant metrikas (kiekybinius matus, parodančius, kiek sistema ar jos komponentas atitinka nustatytus atributus) atliekamos defektų duomenų, metodų, projektų analizės. Įvairiuose literatūros šaltiniuose geromis metrikomis vadinamos tos, kurios yra objektyvios, išmatuojamos, reikšmingos, paprastos ir sudarytos iš nesudėtingai gaunamų duomenų. Nusistatant metrikas pirmiausiai reikia nuspręsti, ką norima įvertinti ir kurie matavimai būtų reikšmingiausi. Iš eilės nustatinėjant visas žinomas metrikas tik eikvojamas laikas, o naudos iš to ne daug [26]. Todėl sudarant aktualių klausimų sąrašą palengvėja metrikų pasirinkimas. Klausimai gali būti:

1. Kiek laiko užtrunka paruošti testavimo scenarijų?
2. Kiek kartų naudosime pasirinktus testus?
3. Kiek reikia testuotojo/sistemos laiko įvykdyti pasirinktus testus?
4. Kiek reikia testuotojų įvykdyti pasirinktiems testams?
5. Kaip apibrėžiama testų aprėptis (*KLOC, FP* ir pan.)?
6. Kiek laiko užtrunka klaidų analizavimas?

7. Kokiu tikslumu norima įvertinti testus? Minutės, sekundės ar mikrosekundės.
8. Ar turima pakankamai resursų (laiko, įrankių, žmonių) automatizuoti testus?
9. Kokį kainos ir gaunamos vertės santykį sudaro automatinių testų palaikymas?

Kitas būdas nusistatyti metrikas gali būti išsirenkant svarbius testavimo atributus. Paprastai išskiriami palaikomumo, efektyvumo, patikimumo, prisitaikomumo, panaudojamumo, tvirtumo, portatyvumo ir nuodugnumo atributai. Testavimo duomenų analizė yra atliekama naudojant metrikas – kiekybinius matus, parodančius, kiek sistema ar jos komponentas atitinka nustatytus atributus.

Keletas pavyzdžių, kokios gali būti testavimo kokybės metrikos:

- *MTTF* (angl. *Mean Time to Failure*) arba *MTBF* (angl. *Mean Time Between Failure*) – vidutinis laikas tarp sistemos lūžimų. *MTTF* metrika nusako laiką tarp dviejų klaidų ar lūžimų, arba kaip ilgai vidutiniškai programinė įranga gali veikti be klaidų. Ši metrika dažniausiai naudojama kritinio saugumo sistemose. $MTTF = \frac{N_F}{N_M}$, čia NF – sistemos lūžimų skaičius per nagrinėjamą laikotarpį, N_M – mėnesių skaičius nagrinėjamame laikotarpyje.
- *DD* (angl. *Defect Density*) – defektų tankis. Defektų kiekio matu laikomas per tam tikrą laikotarpį surastų defektų kiekis, pvz. surastų defektų skaičius nuo modulio sukūrimo iki esamos datos. *DD* naudojamas palyginti defektų tankius skirtinguose programinės įrangos komponentuose. Tai padeda identifikuoti kandidatus detalesnei peržiūrai, testavimui, struktūros pertvarkymui (angl. *re-engineering*) ar pakeitimui. Kuo defektų tankis mažesnis tuo geresnė kokybė. $DD = \frac{N_D}{S}$, čia N_D – žinomų defektų kiekis, S – dydis, paprastai matuojamas *KLOC* (angl. *Kilo Lines Of Code*) tūkstančiais kodo eilučių, bet gali būti naudojami ir funkciniai taškai.
- *DDP* (angl. *Defect Detection Percentage*) – procentinis defektų aptikimas. $DDP = \frac{F_D}{N_D} \cdot 100\%$, čia F_D – aptikti defektai testavimo metu, N_D – žinomų defektų kiekis.
- *AB* (angl. *Achieving Budget*) – testavimui skirtų finansinių lėšų išnaudojimas. Ši metrika padeda įvertinti testavimui skirtų lėšų išnaudojimą. Naudojant *AB* galima sužinoti ar rankinis, ar automatinis testas sunaudoja mažiau pinigų ir

neviršija nustatyto limito. $AB = \frac{C_A}{C_B}$, čia C_A – reali kaina išnaudota testavimui, C_B – testavimui numatytas biudžetas.

- *DDT* (angl. *Defect Detected in Testing*) – testavimo metu aptiktų defektų skaičius. Pagal šią metriką galima spręsti, kuriuo metodu (automatinis, rankinis) aptinkame daugiau sistemoje esančių defektų. $DDT = \frac{D_D}{T_D}$, čia D_D – testavimo metu aptikti defektai, T_D – visi sistemoje esantys defektai.
- *CE* (angl. *Coverage Extension*) – testavimo apimties padidinimas. Palyginama surastos klaidos rankiniu testavimu su papildomom klaidom, kurias padėjo aptikti automatiniai testai (rankiniu testavimu jos buvo praleistos, neaptiktos). Šis įvertinimas naudingas tuo, kad galima palyginti automatinių testų pridėtinę vertę klaidų aptikime.
- Laikas skirtas analizuoti automatinių testų pateiktoms klaidoms. Rankinių testų metu aptiktos klaidos nereikalauja papildomos analizės, nes iškart žinoma, ką vykdant ir kokia klaida buvo aptikta.
- Klaidingų klaidų skaičius pateiktas automatinių testų metu. Tai klaidos, kurios iš tikro nėra klaidos. Toks įvertinimas gali padėti nustatyti, ar automatiniai testai daugiau linkę padėti ar klaidinti.

Siekiant palyginti testavimo metodus reikia remtis keliais vertinimo kriterijais, taip galima gauti tikslesnį sprendimą, kiek verta automatizuoti testavimą arba kuris testavimas (rankinis ar automatinis) yra naudingesnis konkrečioje situacijoje.

2.5. Vartotojų analizė

2.5.1. Vartotojų aibė, tipai, savybės

Intelektualių programinių įrangų testavimo metodais naudojasi ir vadovaujasi jų testuotojai, projektų vadovai.

2.5.2. Vartotojų tikslai ir problemos

Vartotojai susiduria su įvairiomis problemomis, tokiomis kaip atliekamo testavimo tikslumas, patikimumas, efektyvumas, pakartotinis testų scenarijų panaudojimas, testavimo tipo parinkimas. Pagrindinis vartotojų tikslas yra kuo efektyviau ištestuoti programinę įrangą.

Vis dažniau kuriamos programinės įrangos testavimui yra pasirenkami automatiniai testai. Ruošiant automatinius testus yra siekiama užtikrinti testavimo tikslumą (automatiniai testai padengia visus aprašytus scenarijus ir yra išvengiama žmogiškųjų klaidų, kai

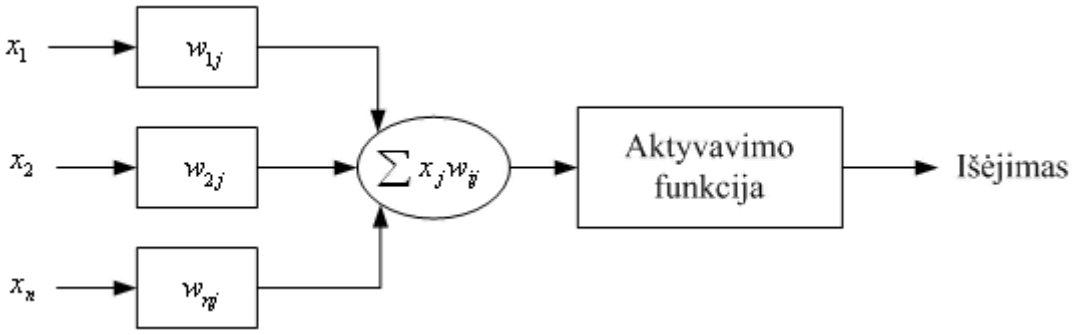
pamirštama pratestuoti vieną ar kitą scenarijų), o taip pat yra siekiama sutaupyti testavimui skiriamą laiką (automatiniai testai patikrina visus aprašytus scenarijus greičiau nei tai padarytų testuotojas; automatinio testo vykdymo metu testuotojas gali atlikti kitus jam priskirtus darbus). Tačiau taip pat svarbu žinoti, kad automatinių testų paruošimas yra daug laiko reikalaujantis darbas. Todėl sprendžiant, ar tam tikro komponento testavimui bus kuriamas automatinis testas, būtina įvertinti laiko sąnaudas, kaštus ir automatinių testų teikiamą naudą. Pagrindinis asmuo, kuris turi planuoti atliekamus darbus, nustatyti darbų trukmes bei atitinkamai sekti kaip yra vykdomi darbai, yra projekto vadovas. Darbų planavimas, jų terminų nustatymas yra atliekamas konsultuojantis su tam tikros srities specialistais, kurie geriau išmano tam tikrą dalykinę sritį ir gali aiškiai nusakyti darbų sudėtį, jų trukmę ir panašius dalykus. Tokiu pačiu principu yra nustatoma ir testavimo darbų trukmė. Tačiau visais atvejais projekto vadovas turi pateikti galutinį sprendimą dėl vienų ar kitų darbų atlikimo, įvertindamas projekto apimtį, kaštus ir laiką. Testavimo etape projektų vadovai siekdami tinkamai koordinuoti viso projekto vykdymą turi atitinkamai įvertinti testavimo tipo parinkimą. Pagrindinė problema yra ta, kad ne visada projektų vadovai turi pakankamai žinių, leidžiančių teisingai parinkti testavimo tipą, o net jei ir tų žinių pakanka, testavimo tipo parinkimas nėra vienareikšmis ir turi būti įvertinta eilė faktorių, tam kad būtų galima nutarti, kad testavimui yra geriau naudoti rankinį ar automatinį testavimą, ar jų derinius konkrečiam testavimo procesui.

2.6. **Esamų sprendimų analizė**

2.6.1. Dirbtinio neuroninio tinklo pristatymas

Dirbtinių neuroninių tinklų (angl. *Artificial neural networks – ANN*) idėja kilo iš neurobiologijos. Neuroniniai tinklai yra sudaryti iš tarpusavyje susijungusių neuronų, suskirstytų į sluoksnius. Į kiekvieną tinklo neuroną yra patalpinama perdavimo funkcija. Kiekvieno sluoksnio neuronai yra sujungti su gretimų sluoksnių neuronais. Dažniausiai to paties sluoksnio neuronai nėra sujungti. Tokia neuronų ir jungčių visuma vadinama dirbtiniu neuroniniu tinklu. Neuroniniai tinklai turi įėjimo, išėjimo ir tarpinius sluoksnius [8]. Labiausiai paplitusios yra dvi dirbtinių neuroninių tinklų rūšys:

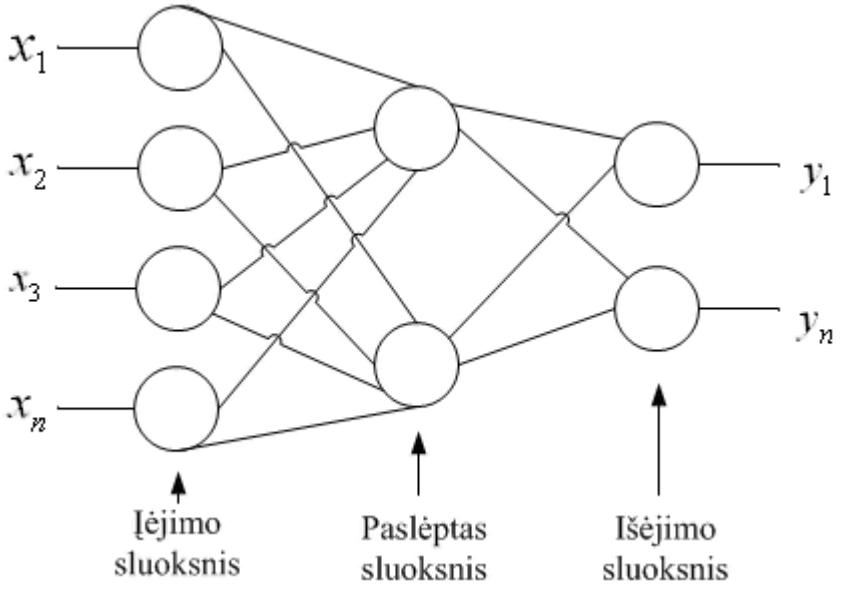
- *vienasluoksnis perceptronas* arba tiesiog perceptronas (angl. *single layer perceptron – SLP*) – tiesiog vienas neuronas, struktūra pateikta 6 paveiksle.



6 pav. Paprastas neuronas

Dažnai aktyvavimo funkcija naudojama sigmoidinė funkcija $f(x) = \frac{1}{1+e^{-net}}$, čia $net = \sum x_j wij$.

- *daugiasluoksnis perceptronas* (angl. *multilayer perceptron – MLP*) – daug neuronų, išdėstytų sluoksniais. Kiekvieno sluoksnio neuronų išėjimai sujungti su kito iš eilės sluoksnio neuronų įėjimais. Įėjimo sluoksnis – pradiniai duomenys, išėjimo sluoksnis – paskutiniame sluoksnyje esantys neuronai ir jų išėjimai. Visi kiti sluoksniai vadinami paslėptais, tarpiniais. Neuroninio tinklo struktūra pateikta 7 paveiksle.



7 pav. Neuroninio tinklo struktūra

Neuroninius tinklus aprašo viena arba kelios matematinės funkcijos. Ryšius tarp neuronų charakterizuoja svoriai. Neuroninio tinklo mokymas yra tų svorių radimas pasirinktu matematinio optimizavimo metodu. Neuroniniai tinklai yra elegantiškas būdas modeliuoti sistemas, kurių įėjimo – išėjimo priklausomybė nežinoma, negalima pakankamai tiksliai išmatuoti stebimos priklausomybės charakteristikų. Neuroniniai tinklai nežinomą priklausomybę „išmoksta“ iš pavyzdžių aibės, kuri apibūdina šią priklausomybę. Neuroniniai tinklai gali būti naudojami kaip neparametriniai klasifikavimo įrankiai. Jais klasifikuojant

nereikia žinoti duomenų pasiskirstymo. Mokydamiesi neuroniniai tinklai minimizuoja nuostolių funkciją, tačiau nėra tiesiogiai minimizuojama paklaida. Neuroniniai tinklai daugiausia naudojami klasifikavimo ir prognozavimo uždaviniams spręsti. Tuo pasinaudojant *ANN* naudojamas automatizuoto orakulo nustatymui, kuris padeda sumažinti programinės įrangos testavimo laiką ir sąnaudas [9].

2.6.2. Orakulo gavimas remiantis *ANN*

Norint sukurti programos automatizuotą orakulą remiantis *ANN* atliekami du etapai: *ANN* apmokomų būti orakulu ir įvertinimo [9].

2.6.2.1. Apmokymo etapas.

Atsitiktinai sugeneruojami įėjimai pagal programos specifikacijas. Nors atsitiktinis generavimas neužtikrina visų galimų įėjimų atvejų, tačiau jis užtikrina, kad nebus spiečiaus įėjimų. Neuroninis tinklas gerai apmokomas, jeigu įėjimai apima kuo platesnį spektrą pagal programos specifikacijas. Testuojama programa įvykdoma su gautais įėjimais ir pagal kiekvieną įėjimo atvejį gauname atitinkamus išėjimus. *ANN* parametrai turi būti nustatyti priklausomai nuo pasirinkto apmokymo algoritmo. Dažniausiai naudojamas tinklo treniravimo (angl. *backpropagation*[2]) metodas. Apmokytas *ANN* naudojamas kaip automatizuotas orakulas, nes jis „įsisavino“ pagal kokius įėjimus, kaip turi elgtis programa.

8 pav. ANN apmokymo procesas

2.6.2.2. Įvertinimo etapas

Į išbandytą programą specialiai įterpiama keletas klaidų, kad turėtume keistų ar su defektais programos versijų. Po klaidų įterpimo sukuriami atskiri įėjimų komplektai tokiu pat būdu, kaip buvo sukurti apmokomai programai. Po to, įvykdome klaidingą programą su gautais įėjimais ir gavę atitinkamus išėjimus panaudojam juos naujiems apmokymams. Tada apskaičiuojame nuotolį tarp neuroninio tinklo išėjimo ir atitinkamo programos išėjimo.

[2] *Backpropagation* – neuroninio tinklo treniravimo metodas, kur pradinis įėjimas į sistemą (tinklą) lyginamas su pageidaujamu išėjimu ir sistema pritaikoma tol, kol skirtumas tarp įėjimo ir išėjimo yra minimizuojamas – įėjimas supanašėtų su išėjimu.

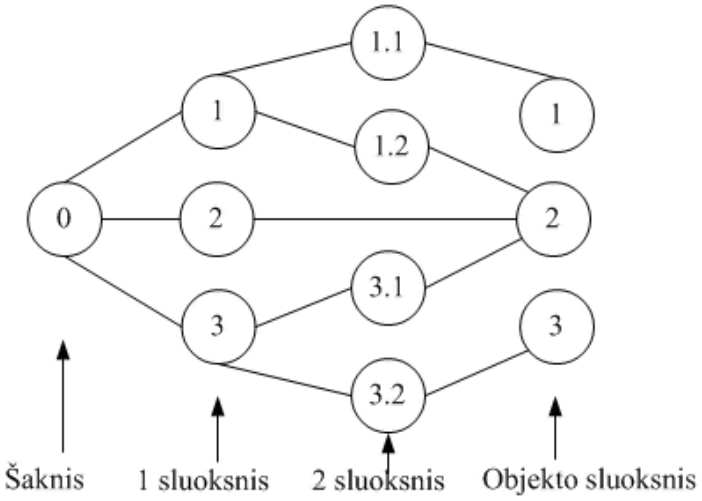
Nuotolis kartu su *ANN* išėjimu ir klaidingos programos išėjimu naudojami nustatyti, ar klaidingos versijos gautas išėjimas iš tikrųjų yra blogas ar ne.

9 pav. ANN įvertinimo procesas

Duomenų palyginimas naudojant *ANN* yra kaip alternatyva testuotojui žmogui, kuris palyginęs klaidingos ir geros versijos rezultatus pasako, ar gautas išėjimas yra blogas ar ne.

2.6.3. Miglotos informacijos tinklo pristatymas

Miglotos informacijos tinklas (angl. *Information Fuzzy Network – IFN*) yra metodas, sukurtas žinių atskleidimui ir duomenų gavybai. Metodą sukūrė Mark Last bendradarbiaujant su Oded Maimon ir Abraham Kandel [10]. *IFN* gali turėti kintamą sluoksnių skaičių ir vieną objekto (taikinio) sluoksnį. Kiekvienas *IFN* paslėptas sluoksnis yra atributo įėjimas, o ne svorių suma, kaip kad *ANN*. *IFN* turi šakninį mazgą, o n-tasis paslėptas sluoksnis yra sudarytas iš visų galimų įėjimo atributo kombinacijų. Kadangi mazgas bet kokiame paslėptame sluoksnyje atstovauja ekvivalentiškų klasių kombinaciją, kiekvienas testuojamas atvejis gali būti jungiamas su vienu ir tiktai vienu mazgu kiekviename paslėptame sluoksnyje, pagal jo įėjimo atributų vertes.



10 pav. IFN struktūra

Miglotos informacijos tinklo struktūra pateikta 10 paveiksle. Tinklas yra dviejų sluoksnių, kurie pasako, kad turime du įėjimo atributus. Pirmas sluoksnis atitinka pirmą įėjimo atributą, o antras sluoksnis – antrą atributą. Pirmas atributas turi tris reikšmes, tai pirmo sluoksnio mazgai 1, 2 ir 3, kurie sudaro tris ekvivalenčias klases. Iš jų 1 ir 2 mazgas yra išskaidytas konstruojant tinklą. Antras sluoksnis turi 4 mazgų derinius gautus iš dviejų antro

įėjimo atributų ir dviejų suskaidytų mazgų iš pirmo sluoksnio. Objekto sluoksnis – tai objekto atributas, kuris susideda iš trijų mazgų.

IFN gali būti naudojami numatyti nežinomas objekto reikšmes, šiek tiek panašiai į sprendimo medžius. Daugiasluoksnis tinklas naudojamas, kai neturima išankstinių duomenų apie atributus. Vieno ar dviejų sluoksnių tinklas sudaromas, jeigu iš anksto turime informaciją apie atributus, jų ryšius. Miglotos informacijos tinklas, kaip ir dirbtinio intelekto, gali būti naudojamas automatizuoto orakulo nustatymui.

2.6.4. Orakulo gavimas remiantis *IFN*

Norint sukurti programos automatizuotą orakulą remiantis *IFN* atliekami du etapai: tinklo konstravimo ir įvertinimo [9].

2.6.4.1. Tinklo konstravimo etapas

Apmokyti miglotos informacijos tinklus naudojamas stebėjimų metodas.

11 pav. Miglotos informacijos tinklo konstravimo procesas

Įėjimų generavimas atliekamas atsitiktiniu būdu. Ištestuota programa įvykdoma su gautais įėjimais ir gauname įėjimų su atitinkamais išėjimais failą, kuris panaudojamas konstruojant miglotos informacijos tinklą. Tinklo indukcijos algoritmas yra pagrįstas išankstiniu trumpinimo (angl. *pre-pruning*) metodu – kai joks požymis nesukelia statistinio reikšmingo entropijos mažėjimo, tinklo konstravimas stabdomas.

2.6.4.2. Įvertinimo etapas

Šis etapas yra panašus į neuroninių tinklų įvertinimo etapą. Patikrinimo failas, sugeneruotas neuroninio tinklo, yra įvykdomas apmokyto *IFN* ir klaidingos programos versijos, tada jų išėjimai palyginami įvertinti, ar programos išėjimai yra geri ar ne.

2.6.5. *ANN* ir *IFN* palyginimas

Tiek *ANN*, tiek *IFN* naudojami orakului nustatyti, abu turi kintamą sluoksnių skaičių. Tačiau dirbtinio neuroninio tinklo apmokymui naudojamas *online* metodas, o miglotos informacijos tinklui – *offline*. Struktūrinis skirtumas *IFN*, jog jis turi šakninį mazgą.

Remiantis [9] straipsnio atliktais tyrinėjimais abu būdai yra efektyvūs, kai klaidingų atvejų procentas yra pakankamai didelis. *ANN* yra geresnis nuolat vertinamų išėjimų klasifikatorius. Ypač, kai mokomų įrašų skaičius yra mažas. Pagrindinis *IFN* pranašumas

prieš *ANN* yra tai, kad *IFN* yra daug greitesnis. O dėl savo struktūros abu tinklai tinka intelektualiam testavimui vykdyti.

2.6.6. Genetiniai algoritmai

Genetinių algoritmų (angl. *Genetic Algorithms – GA*) pradininku laikomas John Holland [11]. 1975 metais jis pasiūlė matematinį optimizavimo metodą, paremtą natūraliais gamtoje vykstančiais procesais: natūraliąja individų atranka, kryžminimu ir mutacija. Metodas buvo pradėtas plačiai taikyti atsiradus kompiuteriams. *GA* – vienas populiariausių meta euristinių algoritmų – stochastinis optimizavimo metodas, kadangi vienas iš procesų yra atsitiktinė, bet į konkretų tikslą orientuota individų atranka. Pasinaudojus evoliucijos mechanizmais, galima sukurti programas, kurios sprendžia problemas netgi tada, kai nėra pilnai aiškus sprendimo kelias [11]. Genetiniai algoritmai dažniausiai naudojami intelektualiai paieškai, optimizavimui ir kompiuterių apmokymam. *GA* yra dažnai naudojamas su įvairiais dirbtinio intelekto metodais. Šiuo metu *GA* yra naudojami kartu su neuroniniais tinklais ir miglota logika spręsti sudėtingas problemas. Dėl jų jungtinio panaudojimo daugeliui problemų spręsti neuroniniai tinklai ir miglota logika vadinama „*soft-computing*“ [8].

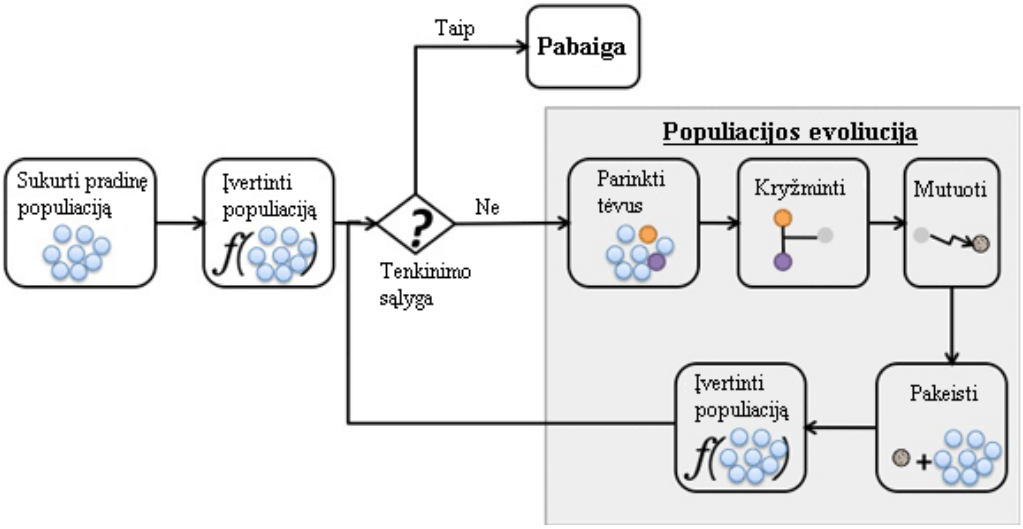
Pagrindinis evoliucijos mechanizmas – tai natūrali atranka. Jos esmė – labiau prisitaikę turi daugiau šansų išlikti ir palikti po savęs palikuonis. Genetinės informacijos perdavimo dėka palikuonys paveldi būdingus tėvų bruožus. Todėl stiprių individų palikuonys taip pat santykinai gerai bus prisitaikę, o jų dalis bendroje masėje augs.

GA metodas paremtas individų populiacija. Kiekvienas individas yra užkoduotas atskiras tikslo funkcijos sprendinys. Kuo jos reikšmė didesnė, tuo individas arčiau optimalaus sprendinio. Individai dažniausiai užkoduojami dvejetainėje sistemoje, tačiau galimas ir kitoks kodavimas. Kiekviena gardelė užkoduojama 1 arba 0 vadinama genu, o iš jų sudarytas užkoduotas individas – chromosoma. Pirmiausia suformuojama pradinė individų populiacija. Po to, kartojami ciklai, kol pasiekiamas norimo tikslumo uždavinio sprendinys. Neretai vienas sudėtingiausių uždavinių pasiruošiant genetiniam algoritmui yra tikslo funkcijos suformulavimas [11].

Kiekvienoje generacijoje atsitiktinai atrenkami individai iš esamos populiacijos. Jie yra tėvai, naudojami sukurti vaikus kitai generacijai. Generacijų metu populiacija konverguoja į optimalų sprendinį. *GA* naudoja tris pagrindinius žingsnius kurti kitą generaciją iš esamos populiacijos:

1) Atranka – atrenkami individai, vadinami tėvais, kurių genai naudojami naujai populiacijai kurti;
2) Kryžminimas – sujungiami tėvų genai ir sukuriami vaikai kitai populiacijai;

3) Mutacija – populiacijos individams atsitiktinai keičiami genai.



12 pav. Genetinio algoritmo struktūra [12]

Genetinio algoritmo idėją iliustruoja 12 pav. Pasirinkus pradinę populiaciją, evoliucija prasideda nuo visiškai atsitiktinių kitimų. Gavus naują populiacijos kartą (kandidatus) įvertinamas jos tinkamumas, atrenkamas tam tikras naujos kartos individų skaičius, pagal atrankos kriterijų. Atrinktieji individai pakeičiami darant mutacijas arba rekombinacijas ir sukuriama nauja populiacija. Vėliau viskas kartojama, atrenkant naujus tinkamiausius individus, sukuriant naują populiaciją. Ciklas kartojamas, kol gaunamas užduotį tenkinantis sprendimas.

2.6.7. Spiečiaus intelektas

„Išlieka tik tie, kurie darosi altruistiški. Visos populiacijos, kurios nebuvo altruistiškos, išmirė. Tie, kurie vienas kitam padeda, mažiau rizikuoja“, - tvirtina profesorius Šarūnas Raudys [13].

Spiečiaus intelektas (angl. *Swarm Intelligence – SI*) apibūdina decentralizuotų, saviorganizuotų, natūralių ar dirbtinių sistemų kolektyvinį elgesį. Literatūroje sutinkami spiečiaus intelekto apibūdinamai yra:

- Bet koks bandymas suprojektuoti algoritmus ar problemas spendžiančius įrenginius, remiantis vabzdžių kolonijų ir kitų gyvūnų visuomeniniu elgesiu [14].
- „Kvailos dalelės, tinkamai sujungtos į spiečių duoda protingus rezultatus.“ Kevin Kelly
- Intelektualios sąveikos atsiradimas tarp atskirų grupės agentų [14].
- *SI* terminas naudojamas išreikšti dirbtinio intelekto sistemas, kur paprastų individų kolektyvinis elgesys sukelia nuoseklių sprendimų ar struktūrų susidarymą [15].

Spiečiaus intelekto sąvoka kilo studijuojant įvairių vabzdžių kaip skruzdžių, bičių ar paukščių gyvenimą. Pavienės bitės arba skruzdėlės nepasižymi protiniais sugebėjimais, bet jų spiečiai ir kolonijos – taip! Kaip kolonijiniai gyvūnai organizuoja savo kolonijos veiklą: kuria greitkelius, stato lizdus bei organizuoja koordinuotus reidus. Tai ir yra pagrindiniai „spiečių“ teorijos uždaviniai. Skruzdėlės nėra maži inžinieriai, architektai arba kariai – bent jau kaip atskiri individai. Stenfordo universiteto biologė Debora Gordon pasakoja: „Atskirtos skruzdėlės nėra protingos, tačiau jų spiečiai, taip.“ Tiktai kolonija sugeba išspręsti tokius uždavinius, kaip trumpiausio kelio paieška suradus geriausią maisto šaltinį, individų paskirstymas įvairioms funkcijoms atlikti, teritorijos apsauga nuo kaimynų bei stambių ir kur kas pavojingesnių įsibrovėlių nei pačios skruzdėlės [16].

Kaip viskas taip yra koordinuojama, jeigu kolonijose nėra jokių vadybininkų arba generolų, nei vienas kolonijos individas nemato bendro vaizdo? Kaip bebūtų kolonijos dažnai elgiasi kaip vientisas organizmas. Viskas veikia vykstant nesuskaičiuojamai daugybei kelių paprastų sąveikų tarp kolonijos narių. Tokį reiškinį mokslininkai vadina saviorganizacija [16].

Spiečiaus intelekto vieni iš privalumų yra, kad jis gali pasiūlyti sprendimus įvairioms problemų rūšims spręsti. Sistemos yra labai tvirtos ir lanksčios, atsparios aplinkos pasikeitimams. Vientisos sistemos elgesys viršija vieno individo turimus elgesio gebėjimus [17]. *SI* populiariausi optimizavimo algoritmai yra skruzdžių kolonijos ir dalelių spiečius [18].

2.6.7.1. Skruzdžių kolonijos optimizavimo algoritmas

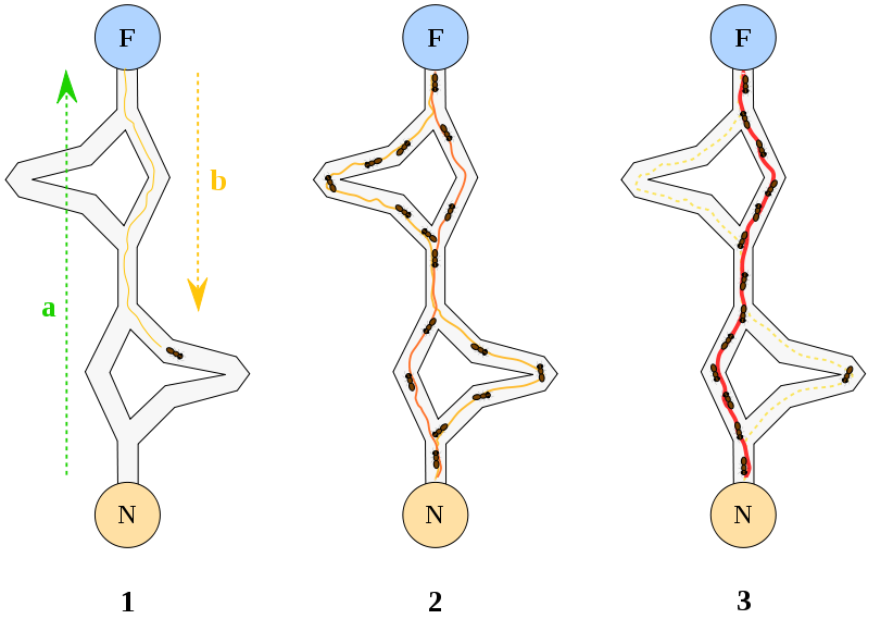
Skruzdžių kolonijos optimizavimo (angl. *Ant Colony Optimization – ACO*) algoritmą 1992 metais pasiūlė Marco Dorigo. Skruzdžių kolonijos elgesys yra vienas iš populiariausių spiečiaus elgesio modelių. Pavienės skruzdėlės elgiasi atsitiktinai ir be kažkokio pastebimo tikslo, bet kai atsiranda kolektyvinės sąveikos tarp skruzdžių, galima stebėti jų spiečiaus intelektą ir elgesį, kuris gali spręsti daug problemų. Skruzdžių spiečius gali nustatyti trumpiausią kelią į maisto šaltinį, pamaitinti visą koloniją, pastatyti didelę struktūrą, ir prisitaikyti prie įvairių situacijų. Tinka spręsti įvairias problemas, kurias galima pavaizduoti grafu.

ACO algoritmo pagrindinė idėja kilo stebint skruzdžių maisto atsargų paieškas. 13 pav. pavaizduota skruzdžių maisto paieška detalizuojama trim punktais [18]:

1. Pirma skruzdėlė, kuri atsitiktiniu būdu surado maisto (žymima F) parneša jį į lizdą (žymima N) žymėdama visą nueinamą kelią feromonu[3]. Grįžta tuo pačiu keliu, kuriuo ir nuėjo.

[3] Feromonai – gyvūnų egzokrininių liaukų sekretai, perduodantys informaciją tos pačios rūšies gyvūnams.

2. Skruzdėlės keliauja visais įmanomais keliais, šiuo atveju jų yra keturi. Dėl didesnio feromonų kiekio stiprėjantis takas tampa patrauklesniu keliu, taip pat tai yra ir trumpiausias maršrutas nuo lizdo iki maisto.
3. Palaipsniui skruzdėlės renkasi trumpiausią maršrutą, o kiti ilgesni keliai išblėsta, nes silpnėja ir išnyksta paliktas feromonų kiekis.



13 pav. Skruzdžių kolonijos natūralus optimizavimo algoritmas [19]

2.6.7.2. Dalelių spiečiaus optimizavimo algoritmas

Dalelių spiečiaus optimizavimo (angl. *Particle Swarm Optimization – PSO*) algoritmą 1995 metais pasiūlė Russ Eberhart ir James Kennedy [20], [21]. *PSO* sukurtas remiantis paukščių bandos elgesiu ir žuvų mokyklų stebėjimais [15]. Šis algoritmas yra panašus į *GA*. Sistema sužadinama su atsitiktinių sprendimų populiacija ir ieškoma optimalumo atnaujinant kartas. Tačiau priešingai nei *GA*, *PSO* nenaudoja kryžminimo ir mutacijų. *PSO* algoritme potencialūs sprendimai skrieja per probleminę erdvę sekant palankiausias sąlygas. Kiekvienas atskiras sprendimas paieškos erdvėje yra tarsi „paukštis“, kuris pavadinamas dalele. Visos dalelės turi tinkamumo reikšmes, kurios siekiant optimizuoti nustatomos tinkamumo funkcija. Taip pat dalelės turi kryptį ir greitį, kuris nukreipia dalelių skriejimą. Dalelės skrieja per probleminę erdvę sekdamos dabartinę optimalią dalelę, kuri vadinama gidu [18].

Vadinasi, spiečius susideda iš dalelių rinkinio, kuriame kiekviena dalelė atstovauja potencialiam sprendimui. Dalelės skrieja per probleminę erdvę, kur kiekvienos dalelės padėtis keičiama pagal jos pačios patirtį ir jos kaimynus. Pažymėkime dalelės $P_i \in P_i(t)$ probleminėje erdvėje poziciją $\vec{x_i}(t)$ laiko momentu t. Dalelės padėtis pakeičiama pridedant kryptingą greitį $\vec{v_i}(t)$ į einamą poziciją. Greičio vektorius $\vec{v_i}(t)$ valdo optimizavimo procesą ir atspindi

socialiai apsikeistą informaciją. Kiekvienos dalelės padėties pakeitimas gali būti išreiškiamas formule [22]:

$$\vec{x_i}(t) = \vec{x_i}(t-1) + \vec{v_i}(t) \quad (1)$$

Detalizuojamas dalelių spiečiaus algoritmas gali remtis geriausia dalelės asmenine savybe (angl. *individual best*), geriausia bendra visų dalelių savybe (angl. *global best*) arba vietinio rato geriausia dalelių savybe (angl. *local best*). Pirmuoju atveju, kiekvienos dalelės veiklos rezultatai F dabartinėje jos padėtyje lyginami su jos geriausiais rezultatais ankstesnėje padėtyje (*pbest*). Kitų dalelių informacija nenaudojama. Jeigu $F(\vec{x_i}(t)) < pbest_i$, tai

$$pbest_i = F(\vec{x_i}(t)) \quad (2)$$

$$\vec{x}_{pbest_i} = \vec{x_i}(t) \quad (3)$$

Po veiklos rezultatų palyginimo pakeičiama dalelės kryptis ir greitis pagal formulę:

$$\vec{v_i}(t) = \vec{v_i}(t-1) + \rho(\vec{x}_{pbest_i} - \vec{x_i}(t)) \ , \quad (4)$$

čia ρ reikšmė yra atsitiktinis teigiamas skaičius. Jį pasirenka vartotojas. Mažos ρ reikšmės labiau linkę į sklandžią trajektoriją, o didelės – įtakoja svyravimus atsirandančius trajektorijoje. Tada kiekviena dalelė atskirai perkeliama į naują padėtį:

$$\vec{x_i}(t) = \vec{x_i}(t-1) + \vec{v_i}(t) \quad (5)$$

$$t = t + 1 \quad (6)$$

ir kartojamas procesas nuo veiklos rezultatų palyginimo, kol dalelės supanašėja.

Antruoju atveju, bendra visų dalelių savybė *gbest* pavaizduoja *PSO* žvaigždės kaimynystės struktūrą. Pagrindinis skirtumas lyginant su ankstesniu atveju, kad naudojama ir kitų dalelių informacija. Palyginama kiekvienos dalelės veiklos rezultatai su bendrais geriausiais veiklos rezultatais

$$gbest_i = F(\vec{x_i}(t)) \quad (7)$$

$$\vec{x}_{gbest_i} = \vec{x_i}(t) \quad (8)$$

Tada dalelės kryptis ir greitis pakeičiami pagal formulę:

$$\vec{v_i}(t) = \vec{v_i}(t-1) + \rho_1(\vec{x}_{pbest_i} - \vec{x_i}(t)) + \rho_2(\vec{x}_{gbest} - \vec{x_i}(t)), \quad (9)$$

čia ρ_1 ir ρ_2 yra atsitiktiniai skaičiai.

Trečiuoju atveju, labai panašus principas į antrąjį atvejį, tik vaizduojama rato kaimynystės struktūra. Vietoj *gbest* naudojamas *lbest* 7, 8 ir 9 formulėse. Naudojant *lbest* vyksta lėtesnis panašėjimas negu naudojant *gbest*, todėl įtakoja geresnius spendimus ir vykdo paiešką didesnėje paieškos erdvėje [22].

PSO algoritmas, palyginti su kitais evoliuciniais algoritmais (skruzdžių optimizavimo algoritmais, genetiniais algoritmais), yra pranašesnis, nes mokymo metu nėra eliminuojami silpniausiai pasirodę individai. Ši savybė yra labai svarbi dirbant su sparčiai, kritiškai kintančiomis aplinkomis, nes yra svarbu adekvačiai sureaguoti ir į skirtingas situacijas, kurios dažnai gali būti visiškai priešingos esamai situacijai. Testuojant sistemas ar jų dalis galima tik spėlioti ar jau yra aptiktos visos klaidos, o gal ištestuotose srityse atlikus pakeitimus atsirado naujų klaidų. Tokiais atvejais yra didelė galimybė, kad spiečiaus intelekto algoritmo pritaikymas, priimant sprendimus, galėtų gerokai pagerinti galinius testavimo rezultatus.

PSO yra panaudotas tokiuose praktiniuose taikymuose, kaip dirbtiniuose neuroniniuose tinkluose ir gramatiniuose vystymosi modeliuose [15]. Šis metodas gali būti taikomas ne tik vertybinių popierių rinkų pokyčiams prognozuoti, bet taip pat ir sprendimams priimti telekomunikacijų, logistikos ir kt. srityse [23].

2.6.8. Genetinio algoritmo ir spiečiaus intelekto palyginimas

Tiek *GA*, tiek *SI* yra optimizavimo algoritmai. *GA* ir *PSO* yra panašios struktūros, naudoja tinkamumo, tikslo funkcijas. Dalelė *PSO* algoritme yra panaši į chromosomą, kuri yra *GA* populiacijos narys. Tiek chromosoma, tiek dalelė atstovauja galimiems problemos sprendimams [24]. *PSO* neturi kryžminimų ir mutacijų, nes remiasi paukščių bandos elgesiu. *PSO* algoritme dalelės atnaujina save su vidiniu greičiu, jos taip pat turi atmintį, kuri yra svarbi algoritmui [25]. *GA* remiasi natūralia individų atranka, kurioje neprisitaikę individai miršta, ko nėra *PSO* algoritme. Dėl šios priežasties *GA* yra greitesnis, tačiau, jeigu dirbama su jautria pokyčiams ir nuolat besikeičiančia sistema geriau tinka *PSO* [27]. Šiuo atveju *GA* trūkumas dirbant su tokiomis sistemomis, kad kritiškai pasikeitus situacijai gali būti reikalingi prarasti duomenys.

2.7. Siekiamas sprendimas

Siekiama sukurti metodą, kuris būtų skirtas testavimo metodo parinkimui ir nurodytų, koks testavimo metodas galėtų būti naudojamas tam tikro testavimo atveju. Parenkant testavimo metodą siekiama įvertinti šiuos esminius kriterijus: testavimo efektyvumą ir tikslumą. Atlikus tyrimo objektų, vartotojų, esamų spendimų analizes pastebėta, kad dažniausiai pasirenkami testavimo tipai pagal jų privalumus, trūkumus, ar testuotojo, vadovo intuiciją. Daugelis testavimo atvejų papildo ar dalinai dengia vienas kitą, todėl projektų vadovui gali būti sudėtinga nuspręsti, kurį testavimo metodą (automatinį ar rankinį) yra geriau naudoti, nes smulkesnius testavimo metodus pasirenka patys testuotojai. Testuotojų komanda gali tik patarti projektų vadovui ar apsimoka kurti ar pirkti konkretų automatizuotą testavimo įrankį, ar užtenka rankinio ištestavimo, nes už patį projektą atsakingas lieka vadovas. Jis žino,

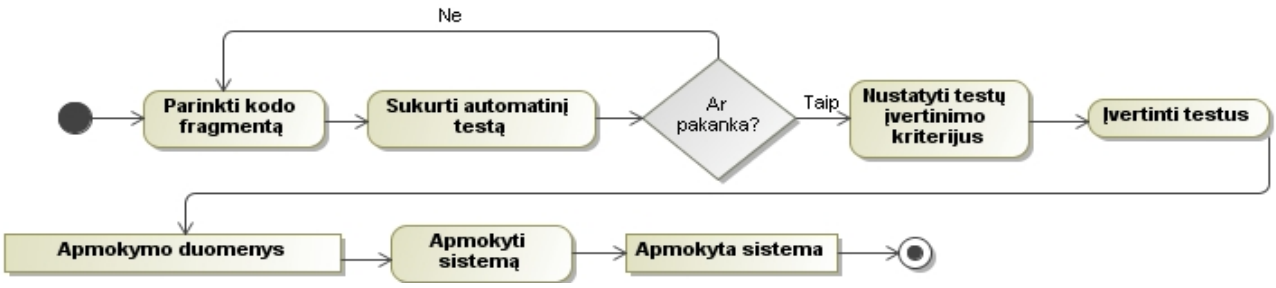
kiek galima ar ne išleisti papildomų pinigų ir skirti papildomų resursų testavimui, nes įvertina ir kitus vykdomo projekto etapus bei jiems atlikti reikalingas sąnaudas.

Siūlomas sprendimas galėtų padėti projektų vadovui lengviau pasirinkti ir atsakingiau įvertinti, kurį testavimo metodą (automatinį ar rankinį) geriau naudoti vieno ar kito testavimo atveju. Ypač šis sprendimas yra aktualus, jeigu automatizuojančią priemonę reikia kurti, koreguoti ar pirkti. Jeigu organizacija jau turi ją iš ankščiau, tai nesudaro papildomų rūpesčių ją sėkmingai adaptuoti ir naudoti, kaip papildomą priemonę klaidų aptikimui, tačiau ir tokiu atveju testavimo metodo parinkimas turi būti vertinamas, nes priemonės adaptavimas tam tikram testavimo atvejui taip pat gali pareikalauti papildomų kaštų bei resursų.

Kuriamas metodas remiasi dirbtinio intelekto pritaikymu siekiant nustatyti, ar konkrečiu atveju yra geriau naudoti automatinį, rankinį ar abu testavimo metodus. Tokiam metodui pirmiausiai reikia apmokyti sistemą, kad būtų galima gauti norimus rezultatus. 14 pav. pavaizduotas sistemos apmokymo procesas:

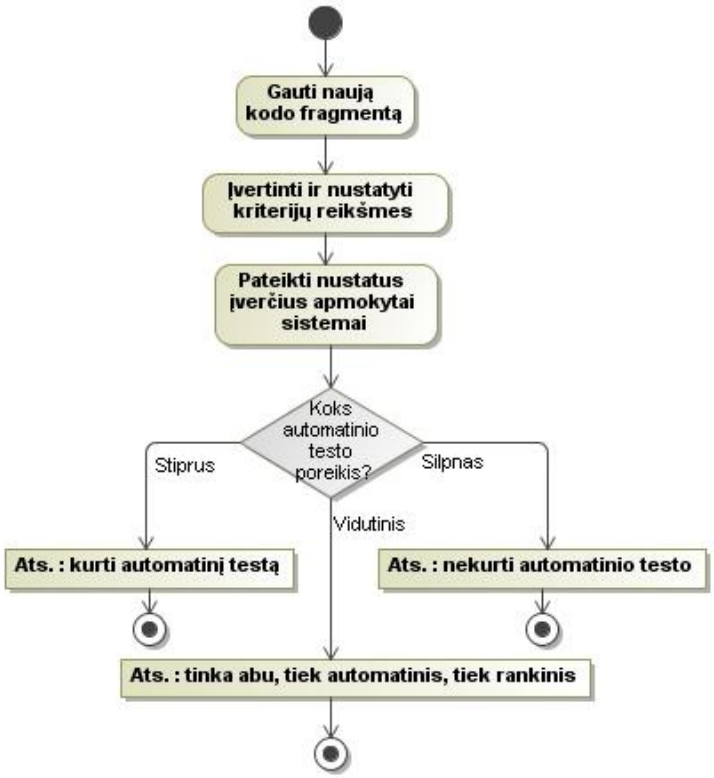
1. Parenkamas tam tikras skaičius kodo fragmentų ir jiems sukuriami automatiniai testai.
2. Nustatomi testų įvertinimo kriterijai.
3. Įvertinami visi turimi testai ir kaupiami gauti duomenys sistemos apmokymui.
4. Atliekamas parinkimo įrankio (sistemos) apmokymas.

Sistemos apmokymui pasirinkta naudoti dirbtinį neuroninį tinklą. Testavimo metodui apmokyti *ANN* pasirinktas remiantis analizės gautomis išvadomis, kad jis yra geresnis klasifikatorius nuolat vertinant išėjimus nei *IFN*. Ypač, kai mokomų įrašų yra mažai.



14 pav. Sistemos apmokymas

Kai turimas apmokytas įrankis (sistema) galima gauti įvertinimą svarbų nagrinėjamai problemai išspręsti, kurį testavimo metodą (automatinį ar rankinį) geriau naudoti. Kuriamo metodo principinė schema pateikta 15 pav., pateikus apmokytai sistemai naujo kodo fragmento duomenis (ciklų, kodo eilučių kiekį) bei kitą reikalingą informaciją detalizuotą 13 lentelėje, sistema įvertinus naują duomenų rinkinį pateikia rekomendaciją apie automatinio testo kūrimo poreikį.



15 pav. Kuriamo metodo principinė schema

2.8. **Analizės išvados**

1. Testavimo metodų analizė parodė, kad egzistuoja testavimo tipo parinkimo problema tarp automatinių ir rankinių testų.
2. Dirbtinio intelekto metodų analizės metu buvo išnagrinėta dirbtinis neuroninis tinklas, miglotos informacijos tinklas, genetiniai algoritmai, spiečiaus intelektas (skruzdžių kolonijos ir dalelių spiečiaus optimizavimo algoritmai).
3. Orakulo nustatymo analizė parodė, kad dirbtiniai neuroniniai ir miglotos informacijos tinklai tinka intelektualiam testavimui vykdyti. Kai mokomųjų įrašų skaičius yra mažas, geresnis nuolat vertinamų išėjimų klasifikatorius yra dirbtinis neuroninis tinklas.
4. Genetinių algoritmų analizės metu nustatyta, kad jie yra greitesni už dalelių spiečiaus intelektą ir tinka sistemoms, kuriose nėra staigių pokyčių, nes atmetinėjami neteikiantys vilčių sprendimai.
5. Dalelių spiečiaus intelekto analizė parodė, kad jis pasižymi tvirtumu ir lankstumu besikeičiančioms sistemoms, kuriose iš esmės gali pasikeisti situacija.
6. Įvertinus problemos aktualumą, nutarta kurti metodą, kuris padėtų projektų vadovui nuspręsti, ar reikalingi automatiniai testai.
7. Įvertinus dirbtinio intelekto metodus ir sprendžiamą problemą, nutarta metode pritaikyti dirbtinį neuroninį tinklą.

3. Metodo reikalavimų specifikacija ir analizė

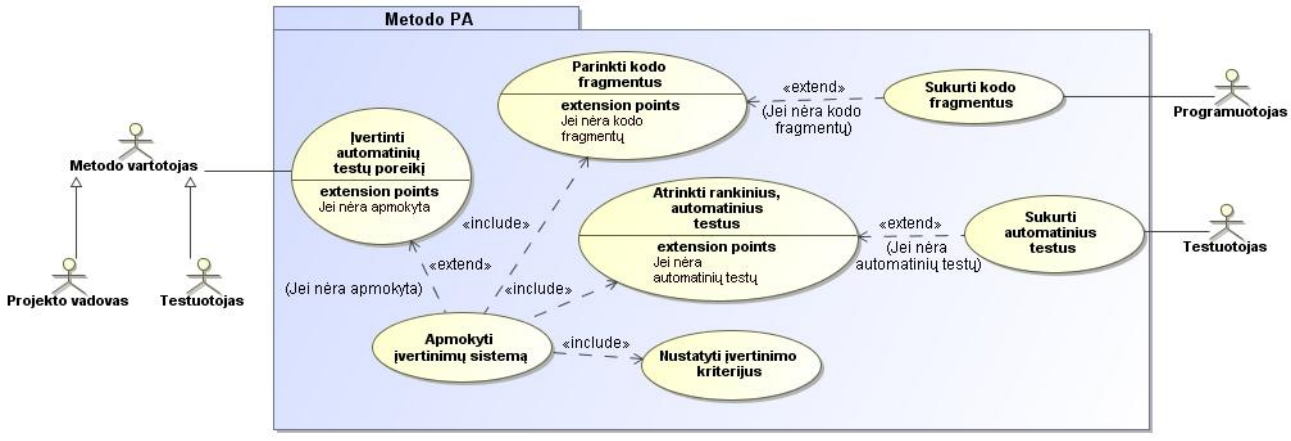
3.1. Taikymo sritis, sąlygos ir prielaidos

Įvertinant gautus testavimo metodų analizės rezultatus, kuriamo metodo taikymo sritis parenkama iš sprendžiamos problemos srities. Kuriamas metodas yra orientuotas į projektų vadovus ir testuotojus, bet juo naudotis gali ir kiti suinteresuoti asmenys, kurie domisi informacinių sistemų testavimu. Kai nėra žinoma, kokį testavimo būdą (automatinį ar rankinį) yra geriau naudoti į pagalbą galima pasitelkti dirbtiniais neuroniniais tinklais paremtą testavimo tipo parinkimo metodą.

Metodo pateikiamus rezultatus reikia vertinti kaip rekomendacinio, patariamojo pobūdžio, kadangi tikslumas priklauso nuo to, kokiais duomenimis vartotojai apmokė metodą. Dėl šios priežasties metodas tinkamesnis toms įmonėms ar vartotojams, kurie jau turi vykdytų projektų ir minėtiems projektams atliktas testavimas. Tokiu būdu įmonė gali parengti metodo taikymą (dirbtinio neuroninio tinklo apmokymą) remiantis įgyta praktika, nes žino, kokiais atvejais automatinių ar rankinių testų naudojimas jai buvo pasiteisinęs ir nepasiteisinęs. Jeigu įmonė neturi testavimo praktikos, gali naudoti jau parengtą metodą. Šiuo atveju reikalingas teikiančios įmonės ar kito subjekto leidimas naudoti parengtą metodą pagal jų duomenis.

3.2. Funkciniai reikalavimai

Detalizuojant 2.7 poskyryje pateiktą metodo idėją sudaryti panaudojimo atvejai (16 pav.), kuriuos turi apimti kuriamas metodas. Panaudojimo atvejų specifikacijos pateiktos 2 – 8 lentelėse ir 17 – 19 paveikslėliuose.

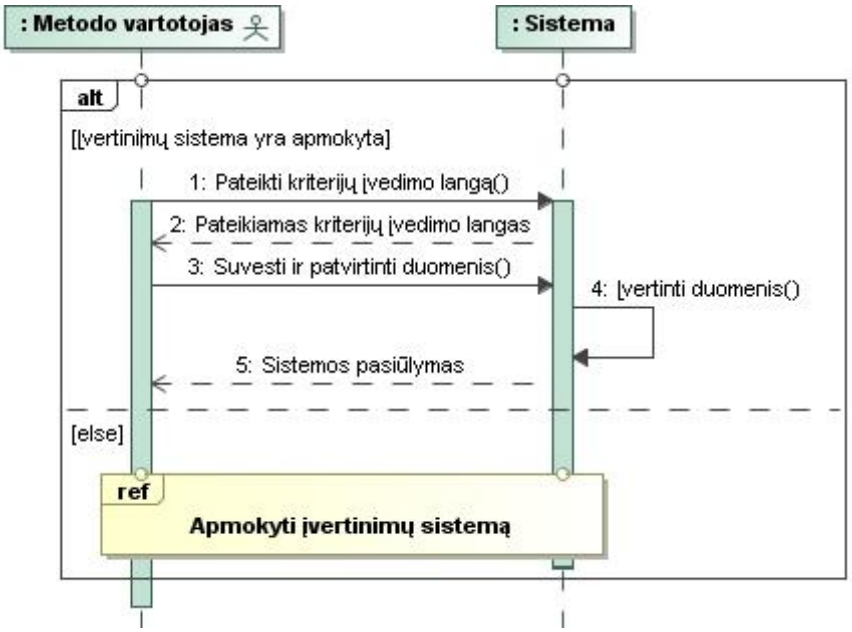


16 pav. Metodo panaudojimo atvejų diagrama

2 lentelė. Panaudojimo atvejo „Įvertinti automatinių testų poreikį“ specifikacija

1 PA „Įvertinti automatinių testų poreikį“	
Aprašymas. Tai pagrindinis metodo naudojimosi panaudojimo atvejis. Vartotojas sužino metodo siūlymą dėl automatinių testų naudojimo.	
Prieš sąlyga	Išrinktos aktualaus kodo fragmento įvertinimų vertės (ANN tinklo įėjimai skaitiniame pavidale).
Aktorius	Metodo vartotojas

Sužadinimo sąlyga		Atsiradęs poreikis sužinoti, ar būtų verta naudoti automatinius testus.
Susiję panaudojimo atvejai	Išplečia PA	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas paprašo pateikti kriterijų įvedimo langą.		1.1 Sistema pateikia kriterijų įvedimo langą.
2. Vartotojas suveda duomenis ir patvirtina.		2.1 Sistema įvertina pateiktus duomenis ir pateikia atsakymą vartotojui.
3. Baigiamas PA		
Po sąlyga:		Sistema pateikė atsakymą vartotojui.
Alternatyvūs scenarijai		
A1. Nėra apmokyta įvertinimų sistema.		Vykdomas PA „Apmokyti įvertinimų sistemą“.

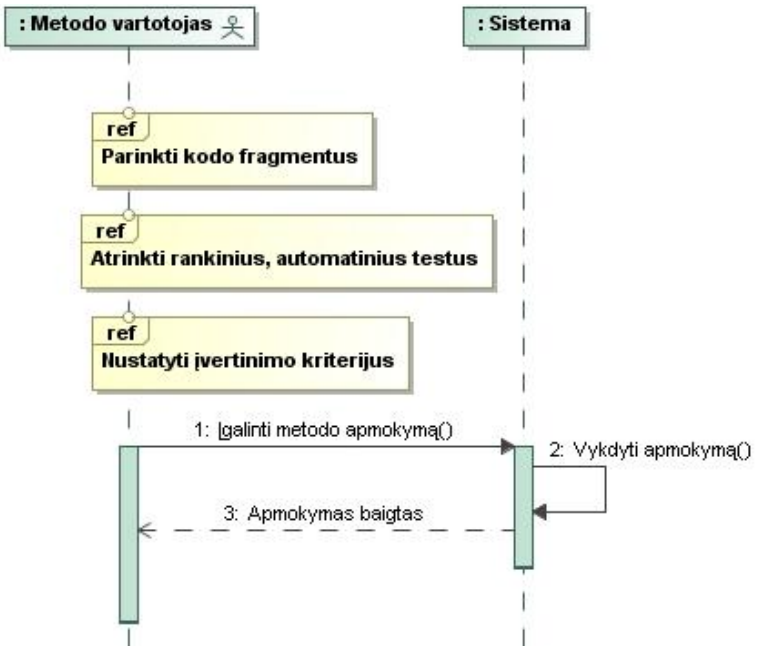


17 pav. PA „Įvertinti automatinių testų poreikį“ sekų diagrama

3 lentelė. Panaudojimo atvejo „Apmokyti įvertinimų sistemą“ specifikacija

2 PA „Apmokyti įvertinimų sistemą“		
Aprašymas. Tai pagrindinis metodo apmokymo panaudojimo atvejis. Šis PA išplečia automatinių testų poreikio įvertinimą ir apima kodo fragmento parinkimą, automatinių testų sukūrimą, įvertinimo kriterijų nustatymą.		
Prieš sąlyga		Įvertinimų sistema nėra apmokyta, parengta.
Aktorius		Metodo vartotojas
Sužadinimo sąlyga		Atsiradęs poreikis sužinoti, ar būtų verta naudoti automatinius testus, bet neturima apmokytos sistemos.
Susiję panaudojimo atvejai	Išplečia PA	Įvertinti automatinių testų poreikį
	Apima PA	Parinkti kodo fragmentus, atrinkti automatinius testus, nustatyti įvertinimo kriterijus.
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojui reikia parinkti kodo fragmentus.		1.1 Vykdomas PA „Parinkti kodo fragmentus“.
2. Vartotojui reikia atrinkti rankinius, automatinius testus.		2.1 Vykdomas PA „Atrinkti rankinius, automatinius testus“.
3. Vartotojui reikia nustatyti įvertinimo kriterijus.		3.1 Vykdomas PA „Nustatyti įvertinimo kriterijus“.

4. Vartotojas įgalina metodo apmokymą.	4.1 Sistema vykdo apmokymą.
5. Baigiamas PA	
Po sąlyga:	Apmokyta įvertinimų sistema (įrankis).
Alternatyvūs scenarijai	
-	



18 pav. PA „Apmokyti įvertinimų sistemą“ sekų diagrama

4 lentelė. Panaudojimo atvejo „Parinkti kodo fragmentus“ specifikacija

3 PA „Parinkti kodo fragmentus“		
Aprašymas. Šis PA skirtas išrinkti programinio kodo fragmentus tinkamus, norimus naudoti metodo apmokymui ir yra PA „Apmokyti įvertinimų sistemą“ dalis.		
Prieš sąlyga		Įvertinimų sistema nėra apmokyta, parengta; kodo fragmentai neatrinkti.
Aktorius		Metodo vartotojas
Sužadinimo sąlyga		Atsiradęs poreikis parinkti kodo fragmentus metodo apmokymui.
Susiję panaudojimo atvejai	Išplečia PA	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas susiranda turimus kodo fragmentus.		
2. Vartotojas išrenka reikalingiausius kodo fragmentus.		
3. Baigiamas PA		
Po sąlyga:		Kodo fragmentai yra parinkti.
Alternatyvūs scenarijai		
A1. Pakankamai ar visai nėra kodo fragmentų.		Vykdomas PA „Sukurti kodo fragmentus“

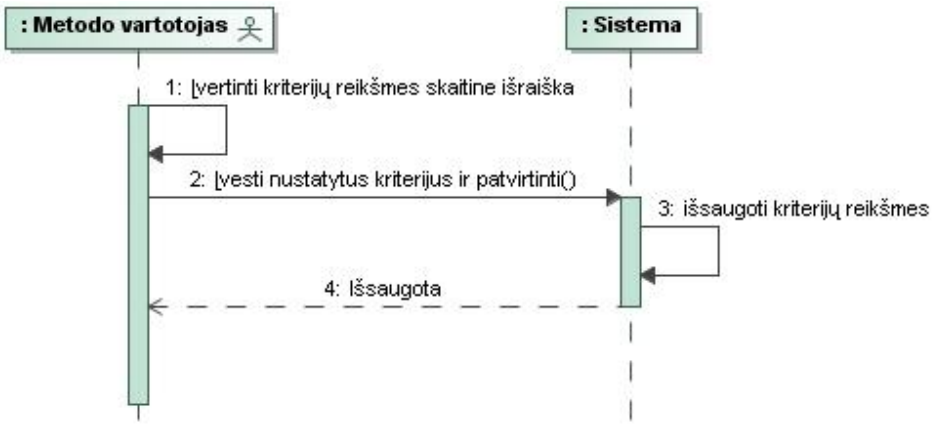
5 lentelė. Panaudojimo atvejo „Atrinkti rankinius, automatinius testus“ specifikacija

4 PA „Atrinkti automatinius testus“	
Aprašymas. Šis PA skirtas atrinkti esančius sukurtus rankinius, automatinius testus išrinktiems kodo fragmentams ir yra PA „Apmokyti įvertinimų sistemą“ dalis.	
Prieš sąlyga	Įvertinimų sistema nėra apmokyta, parengta; rankiniai,

		automatiniai testai neatrinkti.
Aktorius		Metodo vartotojas
Sužadinimo sąlyga		Atsiradęs poreikis išrinkti sukurtus rankinius, automatinius testus parinktiems kodo fragmentams.
Susiję panaudojimo atvejai	Išplečia PA	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas susiranda turimus rankinius, automatinius testus.		
2. Vartotojas išrenka testus susijusius su parinktais kodo fragmentais.		
3. Baigiamas PA		
Po sąlyga:		Rankiniai ir automatiniai testai yra atrinkti.
Alternatyvūs scenarijai		
A1. Pakankamai ar visai nėra automatinių testų.		Vykdomas PA „Sukurti automatinius testus“.

6 lentelė. Panaudojimo atvejo „Nustatyti įvertinimo kriterijus“ specifikacija

5 PA „Nustatyti įvertinimo kriterijus“		
Aprašymas. Šis PA skirtas nustatyti įvertinimo kriterijų reikšmes parinktiems kodo fragmentams su jiems pritaikytais testais ir yra PA „Apmokyti įvertinimų sistemą“ dalis.		
Prieš sąlyga		Įvertinimų sistema nėra apmokyta, parengta; įvertinimo kriterijų reikšmės nenustatytos.
Aktorius		Metodo vartotojas
Sužadinimo sąlyga		Atsiradęs poreikis nustatyti įvertinimo kriterijų reikšmes.
Susiję panaudojimo atvejai	Išplečia PA	-
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas įvertina kriterijų reikšmes skaitine išraiška.		
2. Vartotojas suveda nustatytus kriterijus ir patvirtina.		2.1 Sistema išsaugo kriterijų reikšmes.
3. Baigiamas PA		
Po sąlyga:		Parengti duomenys sistemos (įrankio) apmokymui.
Alternatyvūs scenarijai		
-		



19 pav. PA „Nustatyti įvertinimo kriterijus“ sekų diagrama

7 lentelė. Panaudojimo atvejo „Sukurti kodo fragmentus“ specifikacija

6 PA „Sukurti kodo fragmentus“		
Aprašymas. Šis PA skirtas parengti kodo fragmentus, kurie bus testuojami ir naudojami metodo apmokymui. Taip pat išplečia kodo fragmentų parinkimą.		
Prieš sąlyga		Įvertinimų sistema nėra apmokyta, parengta; nėra jokių galimų testuoti kodo fragmentų.
Aktorius		Programuotojas
Sužadinimo sąlyga		Atsiradęs poreikis sukurti kodo fragmentus.
Susiję panaudojimo atvejai	Išplečia PA	Parinkti kodo fragmentus
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas sukuria prasminį programinį kodą.		
2. Vartotojas kodą suskirto į fragmentus.		
3. Baigiamas PA		
Po sąlyga:		Kodo fragmentai sukurti.
Alternatyvūs scenarijai		
-		

8 lentelė. Panaudojimo atvejo „Sukurti automatinius testus“ specifikacija

7 PA „Sukurti automatinius testus“		
Aprašymas. Šis PA skirtas sukurti automatinius testus ruošiant duomenis metodo apmokymui. Taip pat išplečia rankinių, automatinių testų atrinkimą.		
Prieš sąlyga		Įvertinimų sistema nėra apmokyta, parengta; nėra reikalingų automatinių testų.
Aktorius		Testuotojas
Sužadinimo sąlyga		Atsiradęs poreikis sukurti automatinius testus tam tikriems kodo fragmentams.
Susiję panaudojimo atvejai	Išplečia PA	Atrinkti rankinius, automatinius testus
	Apima PA	-
	Specializuoja PA	-
Pagrindinis įvykių srautas		Sistemos reakcija ir sprendimai
1. Vartotojas išanalizuoja testo poreikio prasmę.		
2. Vartotojas sukuria automatinius testus.		
3. Baigiamas PA		
Po sąlyga:		Automatiniai testai sukurti.
Alternatyvūs scenarijai		
-		

3.3. **Nefunkciniai reikalavimai**

Nefunkciniai reikalavimui metodui (sistemai) išskirti remiantis ISO 9126 standartu ir pateikiami 9 – 11 lentelėse.

9 lentelė. Funkcionalumo reikalavimai

Funkcionalumo reikalavimai		
1.1	Naudojimo tikslumas	
	Numeris:	1
	Panaudojimo atvejai:	1
	Pagrindimas:	Reikalinga tam, kad būtų prasminis metodo naudojimas.
	Tinkamumo kriterijus:	Iš atliekamų bandymų 80% atvejų pirmas bandymas

		turi būti teisingas.
	Užsakovo patenkinimas:	5
	Užsakovo nepatenkinimas:	4
	Priklausomybės:	Nėra
	Konfliktai:	Nėra

10 lentelė. Efektyvumo reikalavimai

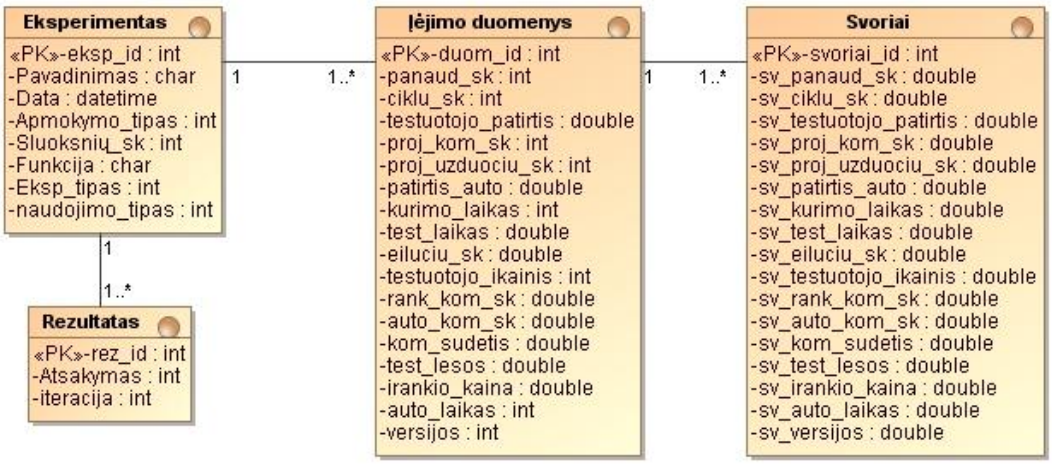
Efektyvumo reikalavimai		
4.1	Laiko ir išteklių paskirstymas	
	Numeris:	2
	Panaudojimo atvejai:	1
	Pagrindimas:	Reikalingas tam, kad būtų siekiama sumažinti testavimo sąnaudas ar palengvinti parinkimą.
	Tinkamumo kriterijus:	Metodo naudojimas turi sumažinti laiko ir išteklių naudojimą nei kad buvo prieš pradedant jį naudoti.
	Užsakovo patenkinimas:	5
	Užsakovo nepatenkinimas:	4
	Priklausomybės:	Nėra
	Konfliktai:	Nėra

11 lentelė. Palaikomumo reikalavimai

Palaikomumo reikalavimai		
5.1	Keitimo galimybės	
	Numeris:	3
	Panaudojimo atvejai:	1 – 5
	Pagrindimas:	Reikalingas tam, kad esant poreikiui įmonė galėtų prisitaikyti pagal savo sukauptą testavimo būdų parinkimo patirtį.
	Tinkamumo kriterijus:	Galima apmokyti tinklą su įmonėje sukauptais testavimo parinkimo duomenimis.
	Užsakovo patenkinimas:	5
	Užsakovo nepatenkinimas:	4
	Priklausomybės:	Nėra
	Konfliktai:	Nėra
5.2	Testavimo galimybės	
	Numeris:	4
	Panaudojimo atvejai:	1 – 5
	Pagrindimas:	Reikalingas tam, kad esant poreikiui įmonė galėtų pasitikrinti, kaip veikia sistema po apmokymo.
	Tinkamumo kriterijus:	Galima panaudoti sukauptus papildomus duomenis įsitikinti, kokiu tikslumu veikia prognozavimas.
	Užsakovo patenkinimas:	5
	Užsakovo nepatenkinimas:	4
	Priklausomybės:	Tiesiogiai priklauso nuo keitimo reikalavime apmokymui naudotų įmonės duomenų.
	Konfliktai:	Nėra

3.4. **Dalykinės srities modelis**

Apibendrinant panaudojimo atvejų ir veiklos diagramas sudarytas dalykinės srities esybių modelis aprašantis metodo apmokymo ir naudojimo duomenis (20 pav.). Detalus įėjimo duomenų aprašymas pateiktas 13 lentelėje. Kiekvienas eksperimentas gali būti sudarytas iš vieno ir daugiau apmokymui skirtų duomenų ir rezultatų. Apmokymui pateiktų duomenų ir rezultatų skaičius turi būti vienodas, nes kiekviena duomenų aibė turi turėti jai priklausantį atsakymą. Naudojimo atveju, eksperimentas turės vieną duomenų rinkinį, kuriam bus metodo (sistemos) pateikiamas prognozuojantis atsakymas dėl automatizavimo poreikio. Kiekvienam duomenų rinkiniui yra bent po vieną svorių rinkinį (atitinkamai įėjimo reikšmei yra po vieną svorio reikšmę). Vykdant apmokymą tam pačiam duomenų rinkiniui, gali būti keli skirtingi svorių rinkiniai, nes keičiant įėjimo duomenų svorių reikšmes siekiama priartėti prie norimo rezultato. Naudojimo metu duomenų rinkinys turi naujausią svorių rinkinį (su šiais svoriais apmokymo metu gaunamas tiksliausias pageidautas rezultatas).



20 pav. Dalykinės srities esybių klasių diagrama

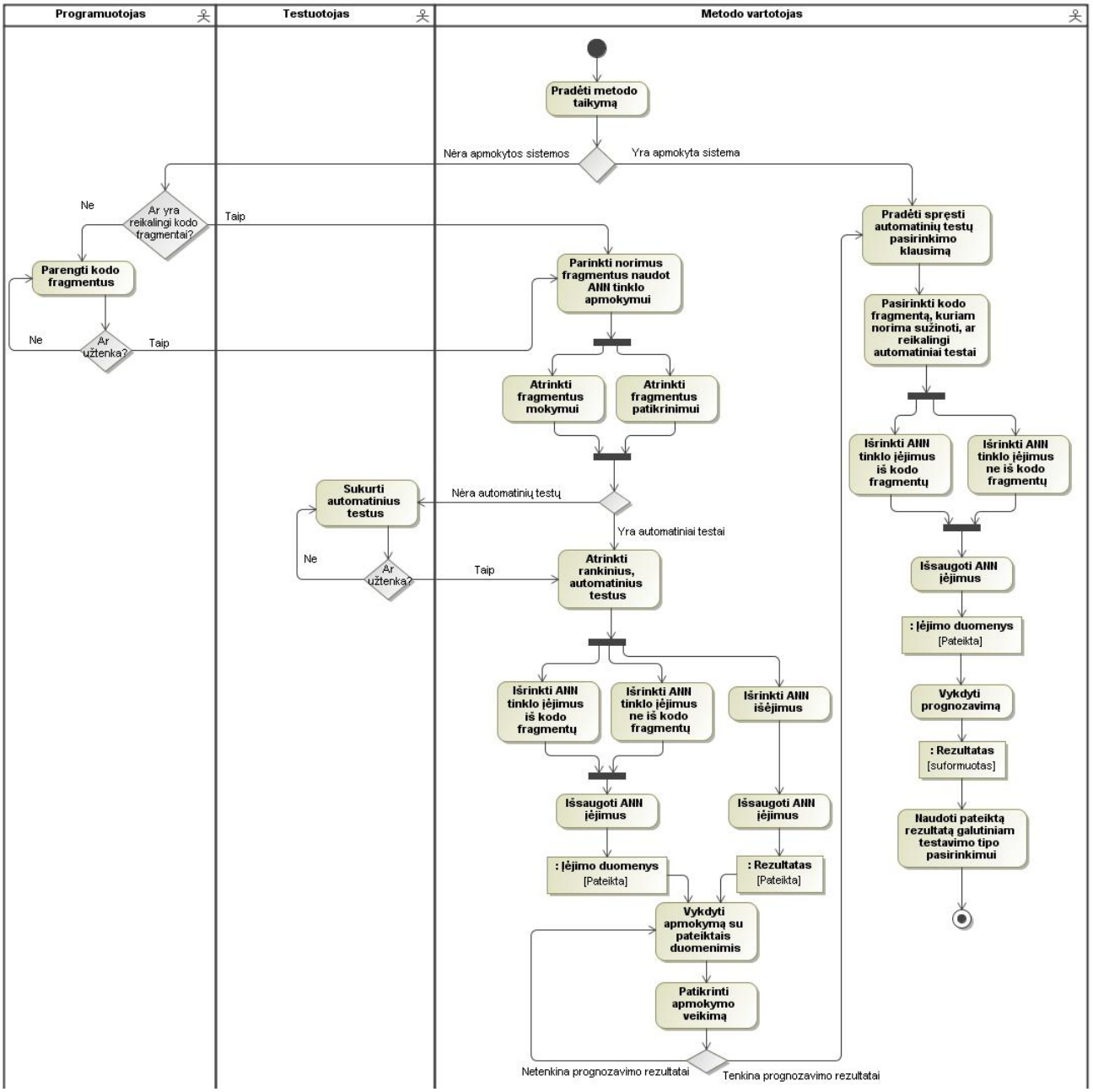
3.5. **Reikalavimų analizės apibendrinimas**

Reikalavimų analizės metu konkretizuota metodo taikymo sritis, detalizuoti funkciniai reikalavimai, išskirti nefunkciniai reikalavimai ir nustatytas dalykinės srities modelis. Pagrindinė metodo funkcija yra įvertinti automatinių testų poreikį, bet ji negali būti vykdoma, jeigu prieš tai nėra apmokyta sistema, kurios apmokymui naudojamas dirbtinis neuroninis tinklas. Siekiant metodo naudojimo pagrįstumo iškeltas nefunkcinis gaunamų rezultatų tikslumo reikalavimas. Metodo (sistemos) dalykinės srities modelis apima metodo naudojimą ir apmokymui skirtų duomenų saugojimą.

4. Metodo aprašas

4.1. Metodo taikymas

Bendras metodo taikymo vaizdas yra pateiktas dar 2.7 poskyryje 14 paveikslėlyje yra bendra apmokymo schema, o 15 pav. – apmokytos sistemos (metodo) taikymas. Siekiant aiškesnio metodo taikymo suvokimo yra sudaryta 21 pav. veiklos diagrama detalizuojanti metodo panaudojimą. Detalizuotoje metodo taikymo veiklos diagramoje apžvelgiami veiksmai nuo metodo taikymo pradžios iki pateiktų rezultatų padedančių metodo vartotojui priimti sprendimą dėl automatinių testų poreikio. Įvertinama situacija, kai metodo naudotojai turi arba neturi apmokytą sistemą, pateikiama detalesnių veiksmų seka norint pritaikyti ar naudotis metodu.



21 pav. Detalizuota metodo taikymo veiklos diagrama

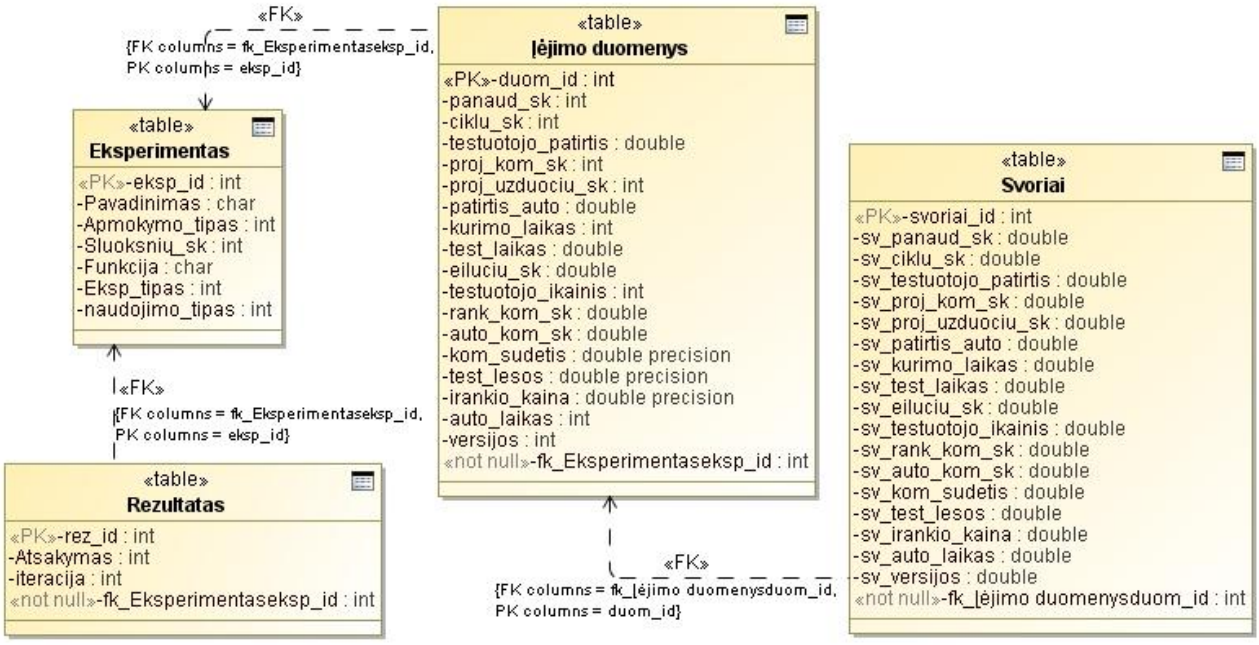
4.2. Metodo realizavimo programinė įranga

Įvertinant veiklos analizės dalyje pasiūlytą testavimo tipo parinkimo metodo idėją nuspręsta metodo veikimo demonstraciją ir eksperimentus atlikti naudojant *MathWorks* kompanijos sukurtą *Matlab* produktą (7.8.0 versiją). *Matlab* savo galimybes praplečia naudojant įvairių sričių priemonių komplektus (angl. *toolbox*). Testavimo tipo parinkimo metodas yra paremtas dirbtiniais neuroniniais tinklais, todėl iš daugybės įvairių priemonių komplektų, sprendžiamai situacijai aktualus yra *Neural Network Toolbox* (6.0 versija; toliau – priemonių komplektas).

Priemonių komplektas leidžia projektuoti, vykdyti, įsivaizduoti ir imituoti neuroninius tinklus [28]. Patikrinant galimybes įsitikinta, kad galima tiek keisti, tiek susidaryti pageidaujamą tinklo struktūrą (nustatyti apmokymo tipą, funkciją, sluoksnių skaičių, neuronų kiekį). Galimybė keisti įvairius parametrus ypač svarbi, nes vykdant įvairius eksperimentus apmokant neuroninį tinklą, galima stebėti, kaip koks parametras įtakoja metodo veikimą. Dėl metodui reikalingų ir esamų funkcijų *Neural Network Toolbox* įrankyje pasirinkta naudoti *Matlab* taikomąją programą.

4.3. Metodo naudojamų duomenų modelis

Iš dalykinės srities modelio (20 pav.) sudarytas siūlomas duomenų modelis. 22 pav. pateikiama duomenų bazės struktūra yra vienas iš galimų variantų, kaip galėtų būti saugomi metodui reikalingi duomenys.



22 pav. Siūlomas metodo duomenų modelis

12 lentelė. Duomenų bazės lentelių aprašas

Atributas	Tipas	Paskirtis
Eksperimentas		
eksp_id	int(10)	Pirminis raktas. Eksperimento identifikatorius.
Pavadinimas	char(20)	Eksperimento pavadinimas, turintis prasminę reikšmę.
Data	datetime	Eksperimento vykdymo data ir laikas.
Apmokymo_tipas	int(1)	Nurodoma apmokymo tipas: su prižiūrėtoju (1); be prižiūrėtojo (2).
Sluoksniu_sk	int(3)	Neuroninio tinklo paslėptų sluoksnių skaičius eksperimento vykdymo metu.
Funkcija	char(20)	Neuroniniam tinkle naudojama funkcija.
Eksp_tipas	int(1)	Nurodomas eksperimento tipas: apmokymo (1); prognozavimo, naudojimo (2).
naudojimo_tipas	int(1)	Nurodomas naudojimo tipas: apmokymo (1); prognozavimo, naudojimo (2).
Rezultatas		
rez_id	int(10)	Pirminis raktas. Rezultato identifikatorius.
Atsakymas	int(3)	Tam tikro duomenų rinkinio atsakymas: rankinis (-1); nesvarbu (0); automatinis (1)
Iteracija	int(10)	Skaičius nurodantis, kurios iteracijos metu gautas atsakymas, jei naudojama apmokymo duomenis – tai iteracijos reikšmė lygi nuliui.
eksp_id	int(10)	Išorinis raktas. Eksperimento identifikatorius.
Iejimo_duomenys		
duom_id	int(10)	Pirminis raktas. Įėjimo duomenų identifikatorius.
panaud_sk	int(10)	Paaiškinimas 13 lentelė. 1 eil. nr.
ciklu_sk	int(10)	Paaiškinimas 13 lentelė. 2 eil. nr.
testuotojo_patirtis	double	Paaiškinimas 13 lentelė. 3 eil. nr.
proj_kom_sk	int(10)	Paaiškinimas 13 lentelė. 4 eil. nr.
proj_uzduociu_sk	int(10)	Paaiškinimas 13 lentelė. 5 eil. nr.
patirtis_auto	double	Paaiškinimas 13 lentelė. 6 eil. nr.
kurimo_laikas	int(10)	Paaiškinimas 13 lentelė. 7 eil. nr.
test_laikas	double	Paaiškinimas 13 lentelė. 8 eil. nr.
eiluciu_sk	double	Paaiškinimas 13 lentelė. 9 eil. nr.
testuotojo_ikainis	int(7)	Paaiškinimas 13 lentelė. 10 eil. nr.
rank_kom_sk	double	Paaiškinimas 13 lentelė. 11 eil. nr.
auto_kom_sk	double	Paaiškinimas 13 lentelė. 12 eil. nr.
kom_sudetis	double	Paaiškinimas 13 lentelė. 13 eil. nr.
test_lesos	double	Paaiškinimas 13 lentelė. 14 eil. nr.
irankio_kaina	double	Paaiškinimas 13 lentelė. 15 eil. nr.
auto_laikas	int(10)	Paaiškinimas 13 lentelė. 16 eil. nr.
versijos	int(3)	Paaiškinimas 13 lentelė. 17 eil. nr.
eksp_id	int(10)	Išorinis raktas. Eksperimento identifikatorius.
Svoriai		
svoriai_id	int(10)	Pirminis raktas. Svorių identifikatorius.
sv_panaud_sk	double	Pakartotinio testo panaudojimo svoris.
sv_ciklu_sk	double	Kode esančių ciklų svoris.
sv_testuotojo_patirtis	double	Testuotojo patirties svoris.
sv_proj_kom_sk	double	Projekto komandos svoris.

Atributas	Tipas	Paskirtis
sv_proj_uzduociu_sk	double	Projekto užduočių svoris
sv_patirtis_auto	double	Testavimo komandos (testuotojo) patirties automatizavime svoris.
sv_kurimo_laikas	double	Projekto kūrimo laiko svoris.
sv_test_laikas	double	Testavimui skirto laiko svoris.
sv_eiluciu_sk	double	Kodo eilučių svoris.
sv_testuotojo_ikainis	double	Testuotojo įkainio svoris.
sv_rank_kom_sk	double	Rankinio testavimo komandos svoris
sv_auto_kom_sk	double	Automatinio testavimo komandos svoris.
sv_kom_sudetis	double	Testavimo komandos sudėties svoris.
sv_test_lesos	double	Testavimui skirtų lėšų svoris.
sv_irankio_kaina	double	Testavimo įrankio kainos svoris.
sv_auto_laikas	double	Laiko reikalingo sukurti automatinį testą svoris.
sv_versijos	double	Numatomų projekto versijų iki užbaigimo svoris.

13 lentelė. Dirbtinių neuroninių tinklų įėjimai-kriterijai

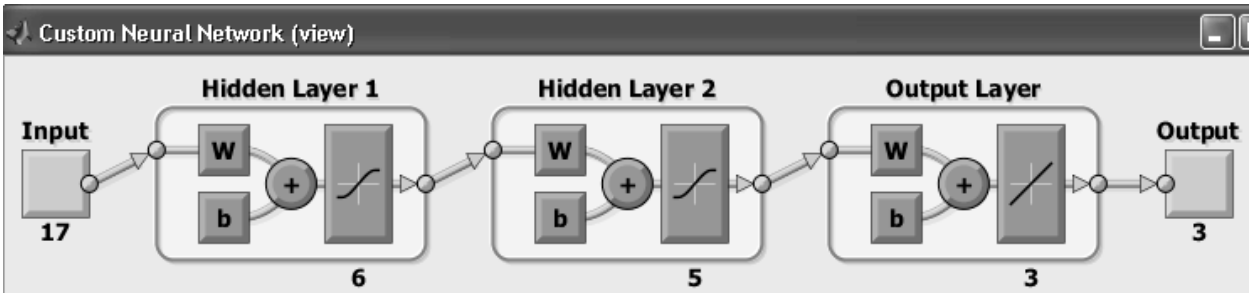
Eil. nr.	Kriterijaus pavadinimas	Paskirtis	Išraiška, formulė	Interpretacija	Reikšmių sritis	Matavimo vienetai
1	Pakartotinis testo panaudojimo sk.	Nustatyti, kiek kartų naudojamas konkretus testas	X = A A = konkretaus testo naudojimo skaičius	Kuo didesnė A reikšmė, tuo didesnis automatinio testo poreikis	0<=A A reikšmėmis gali būti nulis ir sveikųjų teigiamų skaičių aibė	vnt.
2	Kode esančių ciklų sk.	Įvertinti pastangas reikalingas ciklų patikrinimui	X = A A = konkretaus testuojamo kodo bendras ciklų skaičius	Kuo didesnė A reikšmė, tuo didesnis automatinio testo poreikis	0<=A A reikšmėmis gali būti nulis ir sveikųjų teigiamų skaičių aibė	vnt.
3	Testuotojo patirtis	Įvertinti testuotojo patirtį, kad nustatyti, ar pakanka žinių pradėti kurti automatinius testus	X = A A = testuotojo patirtis dirbant testavimo srityje	Kuo didesnė A reikšmė, tuo brandesni susiformavę testavimo įgūdžiai. Esant poreikiui galima pereiti į automatinių testų sritį.	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	metai
4	Projekto komandos sk.	Įvertinti projekto komandos narių skaičių, kad nustatyti testavimo poreikio dažnumą ir projekto dydį.	X = A A = projekto komandos narių skaičius.	Kuo didesnė A reikšmė, tuo dažniau gali tekti testuoti tuos pačius komponentus; didesnis vykdomas projektas.	1<=A A reikšmėmis gali būti teigiamų sveikųjų skaičių aibė nuo 1	vnt.
5	Projekto užduočių kiekis	Įvertinti projektui atlikti reikalingų užduočių kiekį	X = A A = projektui įgyvendinti reikalingas užduočių kiekis.	Kuo didesnė A reikšmė, tuo didesnis gali būti vykdomas projektas ir reikalingas didesnis testų stebėjimas.	1<=A A reikšmėmis gali būti teigiamų sveikųjų skaičių aibė nuo 1	vnt.

6	Testavimo komandos (testuotojo) patirtis automatizavime	Įvertinti komandos (testuotojo) patirtį, kad nustatyti, ar pakanka žinių kurti automatinius testus	X = A A = testuotojo (komandos) patirtis dirbant automatinio testavimo srityje	Kuo didesnė A reikšmė, tuo didesnė įgyta patirtis ir įgūdžiai. Tikslingiau kuriami automatiniai testai	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	metai
7	Projekto kūrimo laikas	Įvertinti, kaip sugebės automatiniai testai atsipirkti	X = A A = numatytas projektui kurti laikas	Kuo didesnė A reikšmė, tuo daugiau gali atsipirkti automatinis testas	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	mėnesiai
8	Testavimui skirtas laikas	Nustatyti iki kurio laiko turi atsipirkti automatinis testas	X = A A = konkretus testavimui skirtas laikas	Kuo didesnė A reikšmė, tuo daugiau gali atsipirkti automatinis testas	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	mėnesiai
9	Funkciniai taškai arba kodo eilučių sk.	Įvertinti, kokią dalį apima testuojama sritis	X = A A = testuojamos srities funkcinių taškų arba tūkstančių kodo eilučių skaičius	Kuo didesnė A reikšmė, tuo sudėtingiau ištestuoti vien tik rankiniu ar automatiniu būdu, todėl didėjant A reikšmei didėja abiejų testavimo tipų poreikis	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	vnt.
10	Testuotojo įkainis	Reikalinga žinoti vykdant testavimą, kad neviršyti testavimui skirtų lėšų	X = A A = testuotojo darbo įkainis	Jei A reikšmė 0, tai testuotojas gali būti praktikantas. Kuo didesnė A reikšmė, tuo svarbiau tinkamai nukreipti testuotojo darbą, pvz., skirti automatinių testų sudarymą	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	Lt per valandas, mėnesius
11	Testavimo komandos sk. (rankinio testavimo)	Įvertinti galimybes kurti automatinius testus	X = A A = rankinio testavimo testuotojų skaičius	Kuo A reikšmė didesnė, tuo įvairesnį galima atlikti testavimą ir paskirti kelis geriausius testuotojus kvalifikuotis automatinių testų kūrime	0<=A A reikšmėmis gali būti nulis ir sveikųjų teigiamų skaičių aibė	vnt.

12	Testavimo komandos sk. (automatinio testavimo)	Įvertinti galimybes kurti automatinius testus	X = A A = automatinio testavimo testuotojų skaičius	Kuo A reikšmė didesnė, tuo daugiau resursų turima kurti automatinius testus	0<=A A reikšmėmis gali būti nulis ir sveikųjų teigiamų skaičių aibė	vnt.
13	Testavimo komandos sudėtis	Įvertinti testavimo komandos sudėtį	X = A/B A = automatinio testavimo testuotojų skaičius B = bendras testuotojų skaičius	0<=X<=1 Kuo X reikšmė artimesnė 1, tuo daugiau turima automatinių testuotojų	0<=A, 0<=B, 0<=X<=1 A ir B reikšmėmis gali būti nulis ir teigiamų skaičių aibė. X reikšmės gali būti nuo 0 iki 1.	santykinė išraiška
14	Testavimui skirtos lėšos	Įvertinti testavimui skirtų lėšų ribas	X = A A = testavimui skirtų pinigų suma	Kuo A reikšmė didesnė, tuo daugiau testuotojo darbo valandų galima skirti testavimui	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	litai
15	Testavimo įrankio kaina	Įvertinti reikalingas išlaidas įsigyti tinkamam testavimo įrankiui	X = A A = testavimo įrankio pirkimo kaina	Kuo A reikšmė mažesnė, tuo greičiau atsipirks investicija ir pirkimas patrauklesnis už kūrimąsi. Jei A = 0, tai testavimo įrankis nėra perkamas	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	litai
16	Laikas reikalingos sukurti automatinį testą	Įvertinti, kiek laiko reikia sukurti konkrečiam automatiniam testui	X = A A = automatinio testo sukūrimo laikas	Kuo A reikšmė mažesnė, tuo patraukliau, nes likusį laiką galima skirti kitiems darbams	0<=A A reikšmėmis gali būti nulis ir teigiamų skaičių aibė	valandos
17	Numatomas projekto versijų skaičius iki projekto užbaigimo	Papildomai įvertinti testų pakartotinio panaudojimo galimybę	X = A A = numatytas versijų skaičius iki užbaigimo dienos	Kuo A reikšmė didesnė, tuo svarbesnis automatinių testų sukūrimas	1<=A A reikšmėmis gali būti teigiamų sveikųjų skaičių aibė nuo 1	vnt.

4.4. Metodo dirbtinio neuroninio tinklo struktūra

Parengus detalų reikalingų metodo sprendimui gauti kriterijų sąrašą gaunamas dirbtinio neuroninio tinklo įėjimo sluoksnio elementų skaičius. Įėjimų sluoksnį sudaro 17 skirtingų įėjimo duomenų (13 lentelė) iš kurių gaunamas vienas sprendimo rezultatas. *ANN* išėjimas gali būti trijų tipų: rankinis testavimas; automatinis testavimas; abu pasirinkimai yra adekvatūs. Metodas pateikia vieną sprendimą iš trijų galimų ir *ANN* išėjimas sudarytas iš trijų neuronų (23 pav.). Paslėptų sluoksnių ir neuronų kiekis juose nustatomas bandymų metu. Taip pat vykdant eksperimentus nustatomi ir kiti neuroninių tinklų parametrai kaip, pavyzdžiui, apmokymo algoritmas, aktyvavimo funkcija.



23 pav. Metodo dirbtinio neuroninio tinklo struktūra

4.5. Pateikto formalaus aprašo pagrindimas

4.5.1. Neuroninio tinklo apmokymas

Norint apskaičiuoti neuroninio tinklo išėjimą, reikia turėti tinklo įėjimo reikšmes, neuronų svorius, kurie iš pradžių priskiriami atsitiktine tvarka, bei žinoti tikslo reikšmes, kokias norime, kad neuroninis tinklas išmoktų.

Iš pradžių kiekvienam neuronui vykdomas svorių sumų priskyrimas pagal (10) formulę, po to gauta svorių suma pateikiama perdavimo funkcijai (11) formulėje. Tokiu būdu gaunama konkretaus neurono išeiga. Tarp skirtingų sluoksnių vienų neuronų išeigos tampa kitų neuronų įeigomis.

Turint neuroninio tinklo išėjimus ir tęsiant apmokymo procedūrą yra atliekamas svorių paslėptuose ir išėjimo sluoksniuose perskaičiavimas. Naujus svorius skaičiuoti patartina nuo išėjimo sluoksnio, nes paslėpto sluoksnio svorių skaičiavime yra naudojamos išėjimo sluoksnio neuronų paklaidos. Naujas svoris apskaičiuojamas pagal (16) formulę, į kurią įstatoma atnaujinimo svorio pokyčio (15) ir po to neurono paklaidos (14) formulės. Paslėpto sluoksnio nauji svoriai gaunami pritaikant minėtą (16) formulę, tik į ją įstatant (13) ir po to (12) formules.

4.5.2. Neuroninio tinklo naudojamos formulės

Svorių priskyrimo formulė [30] (svorių suma pasiekianti neuroną iš i-tojo įėjimo į j-ąjį):

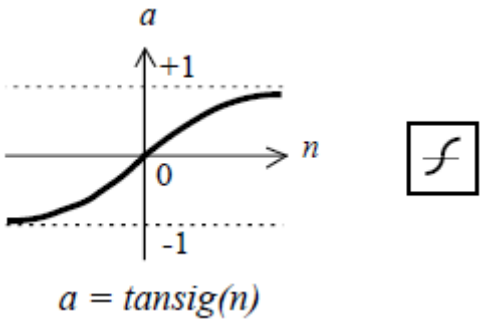
$$n_j = \sum_{i=0}^{d} w_{ji} x_i \qquad (10)$$

kur n_j - j-tasis neuronas, kuriam skaičiuojama svorių įtaka, d – įėjimų kiekis, w_{ji} - svoris iš i-tojo įėjimo į j-ąjį, x_i - įėjimo reikšmė. Pastaba: formulėje yra įtraukta w_0 *bias* reikšmė, kurios $x_0 = 1$.

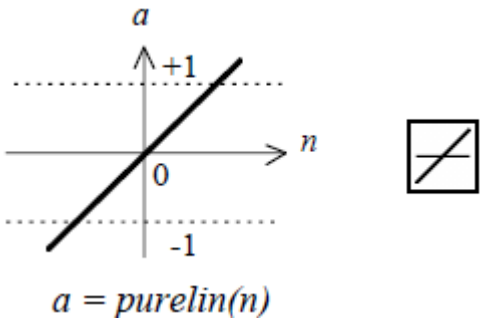
Neurono išėjimo formavimo formulė:

$$a_j = g(n_j) \qquad (11)$$

kur a_j - j-tojo neurono išėjimo reikšmė, n_j - j-tojo neurono svorių suma gauta (10) formulėje, $g(...)$ - neuronų sluoksniui taikoma perdavimo funkcija, darbe yra naudojamos hiperbolinio tangento ir tiesinė funkcijos [22] (24 pav. ir 25 pav.).



24 pav. Hiperbolinis tangentas arba *tansig*



25 pav. Tiesinė funkcija arba *purelin*

Svorių perskaičiavimo formulės paslėptam ir išėjimo sluoksniui skaičiuojamos skirtingai:

- Paslėpto sluoksnio paklaidų skaičiavimas:

$$\delta_{a_j} = a_j(1 - a_j)\sum_{i=1}^{c} w_{ji}\delta_{y_j} \qquad (12)$$

kur δ_{a_j} - paslėpto sluoksnio j-tojo neurono paklaida, a_j - neurono išėjimo reikšmė gauta (11) formulėje, c – išėjimų kiekis kitame sluoksnyje, w_{ji} - svoris iš j-tojo paslėpto sluoksnio neurono į i-tąjį kito sluoksnio neuroną, δ_{y_j} - išėjimo sluoksnio j-tojo neurono paklaida gauta (14) formulėje (kai neuroninis tinklas sudarytas iš dviejų ir daugiau paslėptų sluoksnių, naudojama δ_{a_j+1} - kitas neuronų sluoksnis, reikšmė gaunama iš (12) formulės).

$$\Delta w_{ji} = \eta \delta_{a_j} x_i \qquad (13)$$

kur Δw_{ji} - svorių pokytis tarp i-tojo ir j-tojo neuronų gretimuose sluoksniuose, η - žingsnio dydis, δ_{a_j} - paslėpto sluoksnio j-tojo neurono paklaida gauta (12) formulėje, x_i - i-tojo įėjimo reikšmė (kai neuroninis tinklas sudarytas iš dviejų ir daugiau paslėptų sluoksnių, naudojama a_i kaip įėjimo reikšmės iš prieš tai buvusio sluoksnio).

- Išėjimo sluoksnio paklaidų skaičiavimas:

$$\delta_{y_j} = y_j(1-y_j)(t_j - y_j) \qquad (14)$$

kur δ_{y_j} - išėjimo sluoksnio j-tojo neurono paklaida, y_j - išėjimo sluoksnio j-tojo neurono gauta reikšmė pagal (11) formulę, t_j - j-tojo neurono tikslo reikšmė.

$$\Delta w_{ji} = \eta \delta_{y_j} a_i \qquad (15)$$

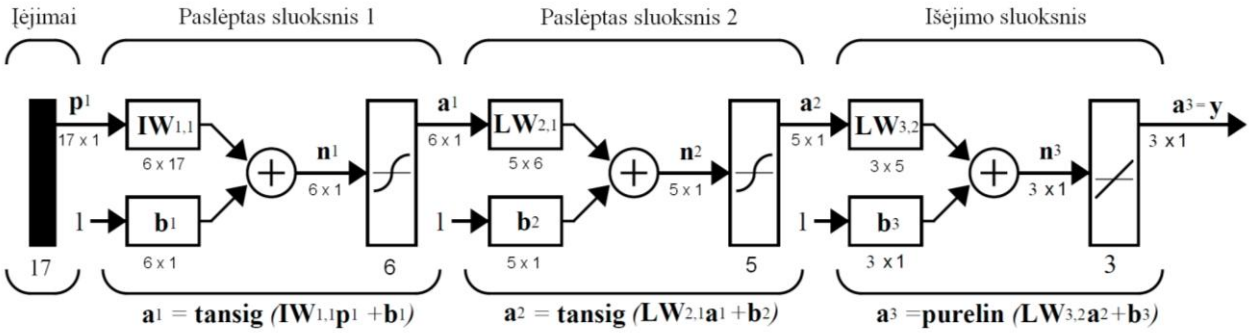
kur Δw_{ji} - svorių pokytis tarp i-tojo ir j-tojo neuronų tarp paslėpto ir išėjimo sluoksnių, η - žingsnio dydis, δ_{y_j} - išėjimo sluoksnio j-tojo neurono paklaida gauta (14) formulėje, a_i - i-tojo įėjimo reikšmė iš paslėpto sluoksnio.

Naujo svorio reikšmės suradimo formulė:

$$naujas_{w_{ji}} = senas_{w_{ji}} + \Delta w_{ji} \qquad (16)$$

4.5.3. Neuroninio tinklo architektūra testavimo tipo parinkimo metodui

Atlikus *ANN* parametrų analizę (detaliau - 6 skyriuje) nustatytos *ANN* architektūros vaizdas pateikiamas 26 pav.



26 pav. Neuroninio tinklo architektūra sprendimui realizuoti

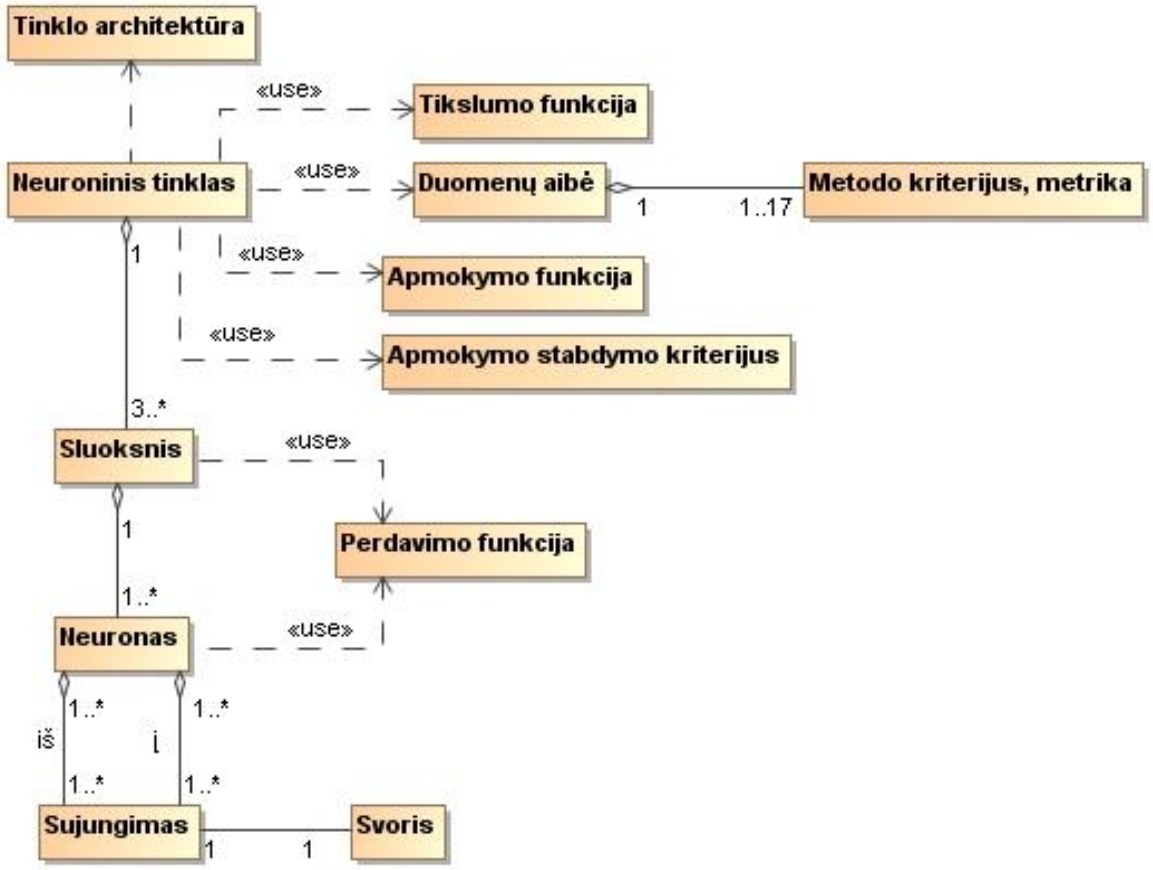
5. Sprendimo realizacija

Siekiant išbandyti darbe pristatytą testavimo tipo parinkimo metodą, buvo sukurtas programinis kodas skirtas įgyvendinti pasiūlyto metodo idėją (4 priedas). Realizacijai panaudota *Matlab* taikomoji programa ir neuroninių tinklų priemonių komplektas. Norint

pasinaudoti neuroninių tinklų priemonių komplektu reikia būtinai turėti įsidiegus *Matlab* paketą [31], [32]. Darbo metu naudotos versijos:

- *MATLAB* 7.8.0 (R2009a). Minimalūs diegimo reikalavimai: 700MB instaliacinis failas, 4GB suinstaliuotas paketas, 512 RAM.
- *Neural Networks Toolbox* 6.0. Turi būti įdiegtas *Matlab* paketas.

Atsižvelgus į neuroninio tinklo įgyvendinimą *Maltab* aplinkoje, jo struktūros fragmentas pateikiamas 27 pav. Neuroninis tinklas ir jo formuojami rezultatai priklauso nuo jį sudarančios tinklo struktūros, naudojamų tikslumo, apmokymo ir perdavimo funkcijų. Daugiasluoksnis neuroninis tinklas mažiausiai gali turėti 3 jį sudarančius sluoksnius (įėjimo, paslėptą ir išėjimo). Kiekvienas sluoksnis turi po kelis neuronus, kurių svarbą sprendimo priėmime nurodo svorio reikšmė. Neuroniniam tinklui naudojamą duomenų aibę sudaro pasiūlyto testavimo tipo parinkimo metodo kriterijai, kurie detaliau aprašyti 13 lentelėje.



27 pav. Naudojamo daugiasluoksnio neuroninio tinklo struktūra

Sukurto programinio kodo logika skirta tiek metodo pagrįstumo analizei, tiek metodo struktūros išgryninimui, kurią sudaro:

1. Pradinių parametrų nustatymas;
2. Apmokymo duomenų ir svorių nuskaitymas;
3. Nustatymas kiek ir kokio tipo duomenų naudojama;
4. Daugiasluoksnio neuroninio tinklo sukūrimas;

5. Daugiasluoksnio neuroninio tinklo apmokymas;
6. Daugiasluoksnio neuroninio tinklo imitavimas;
7. Gautų neuroninio tinklo duomenų atvertimas į pradinį formatą;
8. Naujų duomenų aibės pateikimas apmokytam tinklui;
9. Galutinių rezultatų pateikimas, išsaugojimas.

Vykdant metodo tyrimus, bandymų pavadinimai išreiškiami specifiniu formatu. Perceptronų tinklo eksperimentų numeris prasideda raide P, o daugiasluoksnio – raide M. Pavyzdžiui, M1HL5N kodas šifruojasi tokiu principu:

M daugiasluoksnis (angl. *Multilayer*).

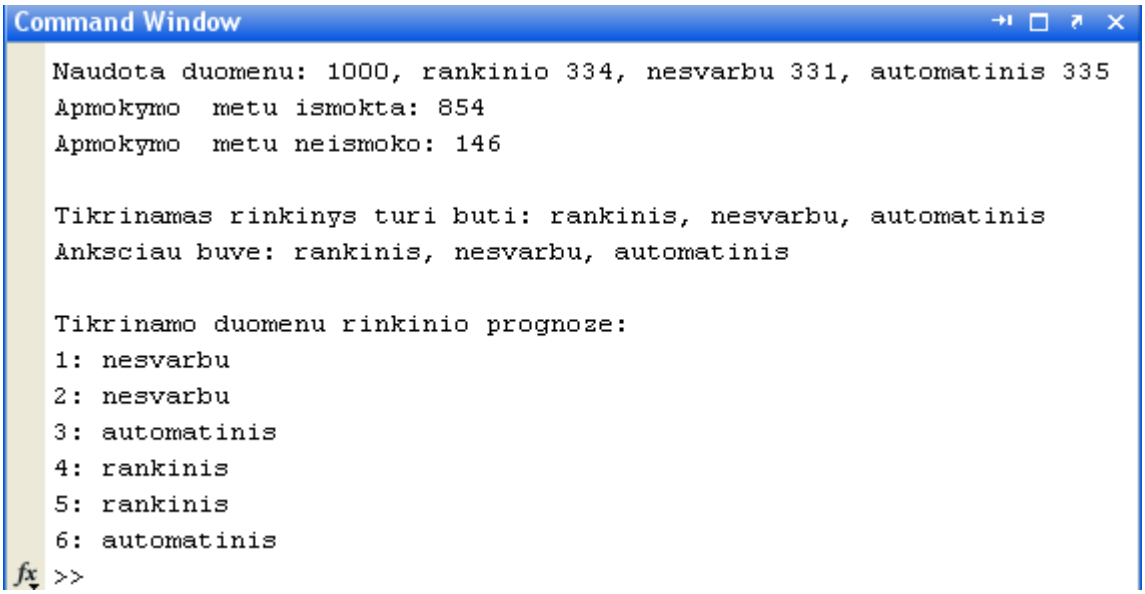
1 sluoksnių kiekis.

HL paslėptas sluoksnis (angl. *Hidden Layer*).

5 neuronų kiekis paslėptame sluoksnyje.

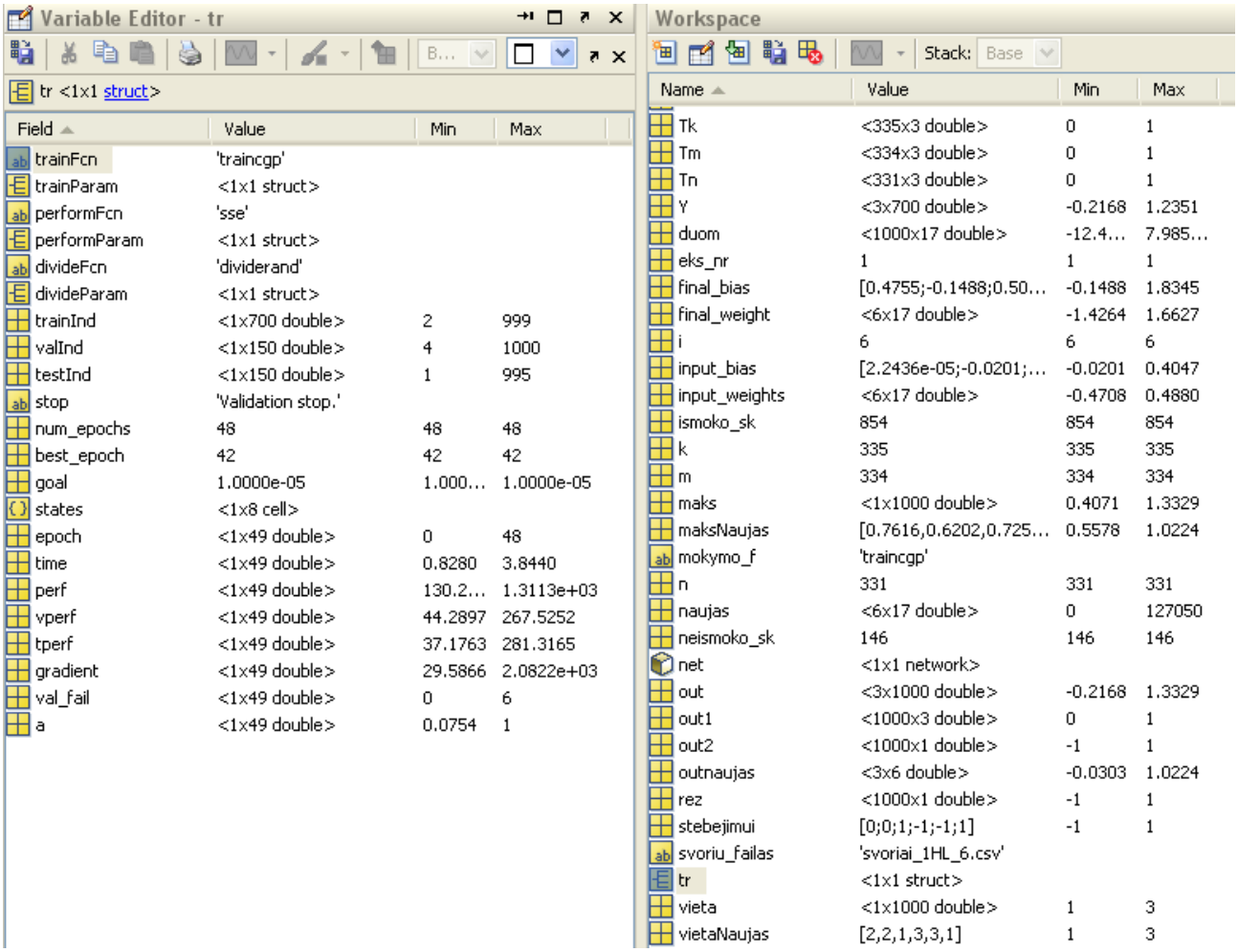
N Neuronai (angl. *Neuron*).

Atliekant M2HL6-5N bandymą, naudojant *traincgp* apmokymo ir suminę kvadratinės paklaidos tikslumo funkcijas, atsidarius programinį kodą reikia jį paleisti įvykdyti. Speciali grafinė sąsaja nėra kurta, nes tai nėra tyrimo objektas. Po kodo įvykdymo *Matlab* paketo komandų lange gauti rezultatai pateikiami 28 pav. Kita informacija peržiūrima per kintamuosius (29 pav.).



28 pav. Papildomos informacijos išvedimo pavyzdys

Analizuojant pateiktus 28 pav. rezultatus, galima įvertinti kiek neuroninis tinklas išmoksta duomenų iš apmokymui skirtos aibės, kiek kokių duomenų pateikta, kaip nuspėja naujus pateiktus duomenis po tinklo apmokymo. Iš 29 pav. galima analizuoti duomenis susijusius su neuroninio tinklo struktūra (pasiektas tikslumas, tikslumo reikšmės kiekvienoje iteracijoje, apmokymo stabdymo priežastis), įvertinti pateiktus tinklo išėjimus.



29 pav. Informacijos per kintamuosius peržiūra

6. Eksperimentinis tyrimas

6.1. Neuroninio tinklo parametrų analizė

Nustatinėjant *ANN* struktūrą geriausiai tinkančią parinkimo problemai spręsti pirmiausiai testams buvo paruošta duomenų aibė skirta tinklo mokymams. *ANN* tinklo įėjimų duomenys aprašomi 13 lentelėje. Nustatant tinklo išėjimo struktūrą buvo įvertinta *ANN* naudojama perdavimo funkcija. Kadangi tinklo išėjime gali būti trijų rūšių reikšmės (automatinis, rankinis, nesvarbu), perdavimo funkcija pasirinkta naudoti *tan-sigmoid*. *Tan-sigmoid* perdavimo funkcijos reikšmės gali būti tarp -1 ir 1. Siekiant tinkamai interpretuoti gaunamas *ANN* reikšmes, nuspręsta *ANN* išėjimo sluoksnį sudaryti iš trijų neuronų. Priskyrus metodo reikšmes skaitinėms gauname:

- Rankinis testavimas yra -1 ir [0 0 1];
- Automatinis testavimas yra 1 ir [1 0 0];
- Nesvarbu, kuris testavimas yra 0 ir [0 1 0].

Pavyzdžiui, jeigu tinklas išėjime pateikia [0.8 0.3 -0.06], tai suradus maksimalią reikšmę ir jos pozicijoje įrašius 1, o visas kitas reikšmes pakeitus į 0, gaunama [1 0 0]. Pagal

ankstesnį pažymėjimą šis kodas reiškia, kad yra gauta rekomendacija automatizuoti testą. Tokiu būdu galima tiksliai atskirti *ANN* pateikiamas reikšmes.

Pasiruošus duomenis kiekvieno eksperimento metu buvo pateikiama tinkui 1000 mokymų duomenų, kurie atitinkamai padalinti į 70% mokymui, 15% validavimui ir 15% testavimui. Nustatymas leistinas epochų skaičius 100 ir pageidautinas apmokymo tikslumas 0,00001, nes tinklo veikimo tikslumas yra vienas iš pagrindinių galutinio tinklo vertinimo kriterijų. Atliekant eksperimentus buvo nagrinėjama perceptrono ir daugiasluoksnio neuroninio tinklo tinkamumas testavimo parinkimo problemai spęsti. Percentronų tinklas buvo atmestas kaip netinkamas dėl didelių paklaidų ir mažo išmokstamų bei naujų atpažįstamų duomenų skaičiaus. Dėl to buvo atlikti eksperimentai siekiant nustatyti daugiasluoksnio neuroninio tinklo struktūrą: sluoksnių skaičių, neuronų skaičių kiekviename sluoksnyje, tinklo apmokymo funkciją, paklaidos funkciją.

Eksperimentų metu nagrinėtos tinklo apmokymo funkcijos:

- *trainlm – Levenberg-Marguardt*;
- *trainbr – Bayesian Regularization;*
- *trainbfg – BFGS Quasi-Newton;*
- *trainrp – Resilient Backpropagation;*
- *trainscg – Scaled Conjugate Gradient;*
- *traincgb – Conjugate Gradient with Powell/Beale Restarts;*
- *traincgf – Fletcher-Power Conjugate Gradient;*
- *traincgp – Polak-Ribiere Conjugate Gradient;*
- *trainoss – One Step Secant;*
- *traingdx – Variable Learning Rate Gradient Descent;*
- *traingdm – Gradient Descent with Momentum;*
- *traingd – Gradient Descent.*

Kiekvieno tipo eksperimentas buvo atliekamas po 30 kartų, kad apskaičiavus vidurkį būtų galima tiksliau įvertinti *ANN* pateikiamus rezultatus. Vykdant eksperimentus įvertinant bendrą spėjimų įvertį (naujų duomenų teisingą atpažinimą), pasiektą išmokimo tikslumą, išmoktų įrašų skaičių ir apsimokymo kreivę geriausiai pasirodė tinklas turintis 6 neuronus pirmame paslėptame sluoksnyje ir 5 antrame. Apmokymo funkcija yra *trainbr* ir suminė vidutinės kvadratinės paklaidos tikslumo funkcija (angl. *sum squared error - SSE*). Pagal 14 lentelėje pateiktus duomenis įvertinus išmoktų duomenų skaičių (iš viso galėjo išmokti 1000 duomenų), teisingai atspėtų naujų ir pasiektą tikslumą vienareikšmiškai pasakyti, kuris eksperimentas yra geriausias sudėtinga. Todėl 14 lentelės duomenys analizuojami kartu su 30 pav. esančiomis apmokymo kreivėmis. M2HL6-5N tinklas turi optimalius parametrus ir

apmokymo kreivę, kuri tolygiai ir be didesnių svyravimų gerina apsimokymo tikslumą. Kuo negali pasigirti M2HL1-5N tinklas turintis didžiausią naujų duomenų atpažinimo koeficientą, nes jo apsimokymo kreivė iš pradžių per greitai didina tikslumą ir po to turi staigų lūžio kampą bei pasiektas tikslumas yra prasčiausias lyginant su kitais eksperimentais. M2HL4-10N, M2HL8-10N kreivės taip pat per greitai krinta, todėl jei turi vieną kurį iš parametrų geresnį, kiti labiau išsiskiria. Pavyzdžiui, M2HL8-10N pasiekia vieną iš geresnių tikslumų ir išmoktų duomenų, bet sunkiau nuspėja naujus duomenis. M2HL8-5N turi didžiausią išmoktų duomenų skaičių ir geriausią pasiektą tikslumą, bet prasčiausiai pavyko nuspėti naujus duomenis, tai gali būti dėl to, kad jo apmokymo kreivės tikslumas pirmomis iteracijomis buvo pats mažiausias. Todėl viską apibendrinus nuspręsta tolimesniems tyrimams pasirinkti M2HL6-5N tinklą.

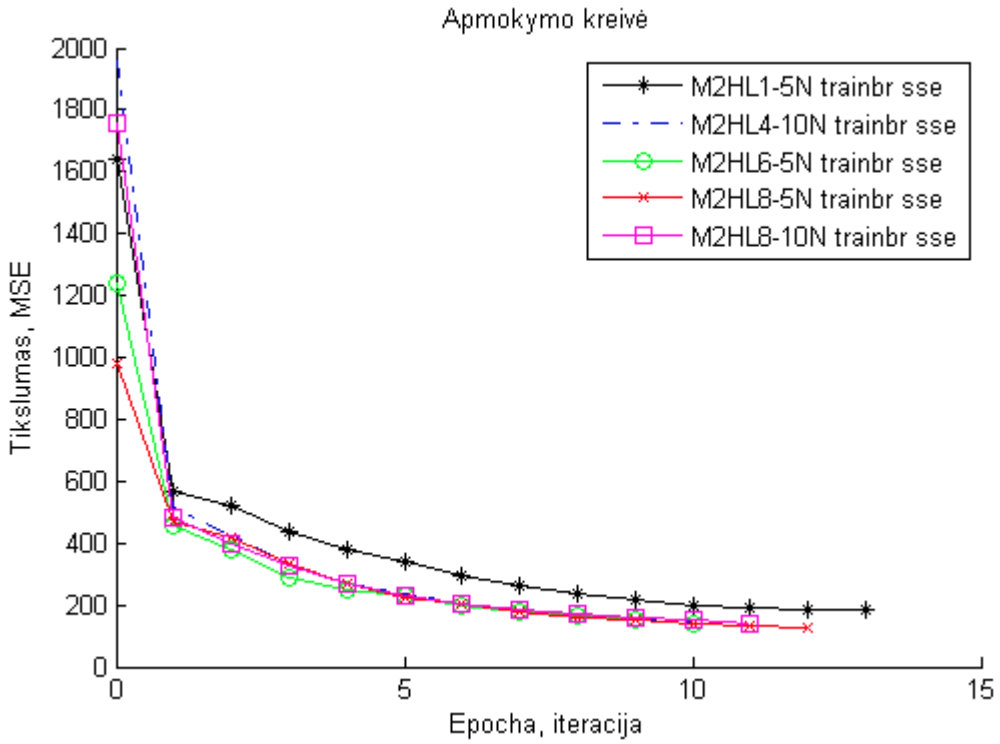
Vykdant tolimesnius tyrimus atlikus apmokymo funkcijų eksperimentus (15 lentelė), kai naudota vidutinės kvadratinės paklaidos (angl. *mean squared error - MSE*) tikslumo funkcija geriausiai pasirodė *traincgp* apmokymo funkcija, pralenkdama naudotą *trainbr* funkciją. Tačiau atlikus tuos pačius funkcijų eksperimentus naudojant *SSE, traincgp* apmokymo funkciją pralenkė *traincgb* funkcija. Rezultatus galima palyginti grafiškai 31 pav. Žvaigždute pažymėti eksperimento duomenys piešiami prie pat x ašies, nes tikslumo funkcija naudojama *MSE*, o visų kitų eksperimentų *SSE*, dėl to toks ryškus skirtumas, kai nėra sumuojamos paklaidos. Grafiškai geriausiai atrodo *traincgb sse* kreivė, nes nėra didelių apsimokymo svyravimų, tolydžiai gerinamas tikslumas.

14 lentelė. Atrinktų eksperimentų duomenys parenkant neuronų skaičių

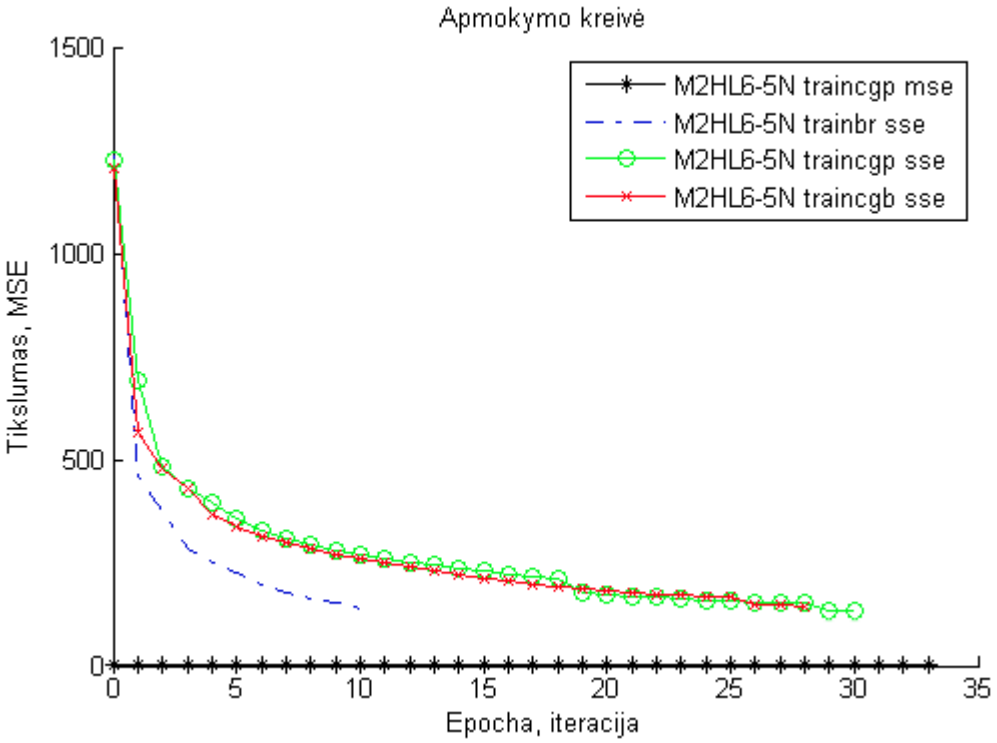
Savybė	**M2HL1-5N**	**M2HL4-10N**	**M2HL6-5N**	**M2HL8-5N**	**M2HL8-10N**
Išmokta duomenų	819.6	854.3	853.4	871.3	862
Atspėta naujų	86.67%	75.00%	71.67%	63.33%	70.00%
Pasiektas tikslumas	41.27	38.15	37.07	34.30	34.48
Teisingai suklasifikuota	81.96%	85.43%	85.34%	87.13%	86.20%
Laikas, sek.	2.8188	3.6811	3.4204	5.0547	6.8171

15 lentelė. Atrinktų eksperimentų duomenys parenkant apmokymo funkciją

Savybė	**traincgp mse**	**trainbr sse**	**traincgp sse**	**traincgb sse**
Išmokta duomenų	841.1	853.4	822.1	839.0
Atspėta naujų	78.33%	71.67%	75.00%	80.00%
Pasiektas tikslumas	0.0812	37.07	41.67	38.75
Teisingai suklasifikuota	84.11%	85.34%	82.21%	83.90%
Laikas, sek.	2.9781	3.4204	2.6064	2.4939



30 pav. Atrinktų eksperimentų apmokymo kreivės parenkant neuronų skaičių



31 pav. Atrinktų eksperimentų apmokymo kreivės parenkant apmokymo funkciją

Naudojant M2HL6-5N *traincgb sse* tinklas geriausiai atspėja naujus duomenis. Kiekvieno eksperimento metu, tinklas turėjo atpažinti naujus šešis duomenų rinkinius: 2 iš rankinio testavimo, 2 iš nesvarbu, 2 iš automatinio testavimo. Susumavus kiekvieno eksperimento metu iš 30 kartų, kiek teisingai atspėjo turimą gauti atsakymą, minėtas tinklas

pateikia tokius rezultatus: rankinis 10 ir 30, nesvarbu 29 ir 13, automatinis 29 ir 30. Gauti rezultatai rodo, kad reikia atkreipti dėmesį dėl kokių priežasčių tinklas vieną rankinio testavimo duomenų aibę atspėja tik 10 kartų iš 30, o kitą kartą visus 30 iš 30. Tai rodo, kad yra reikalingi tolimesni tyrimai, siekiant gerinti tikslumą. Reikia panagrinėti kaip elgiasi gautas neuroninis tinklas, kai turimi skirtingi nauji duomenys, kai yra kitokia apmokymui skirtų duomenų aibė.

6.2. **Neuroninio tinklo parametrų analizės išvados**

Atlikti ANN eksperimentai parodė, kad jų taikymas testavimo tipo parinkimo problemai yra tinkamas. Tyrimų metu nustatyta, kad nagrinėjamą problemą geriausiai sprendžia daugiasluoksnis neuroninis tinklas su dviem paslėptais sluoksniais, iš kurių pirmame turi 6 neuronus, o antrame 5, naudojama *SSE* tikslumo funkcija, tinklas apmokomas naudojant *Conjugate Gradient with Powell/Beale Restarts* algoritmą. Tačiau dar yra reikalingi tolimesni tyrimai, kad būtų galima surasti geresnę *ANN* struktūrą, padėsiančią pasiekti dar didesnį prognozavimo tikslumą. Įvertinant, kad testavimo tipo parinkimo metodas skirtas projektų vadovams ir testuotojams bei pagrindinis siekiamas tikslas, kad galėtų palengvinti ir padėtų atsakingiau įvertinti, kuris testavimo tipas yra geriau tinkamas konkrečiomis sąlygomis, turi būti nagrinėjama apmokymo duomenų aibės sudėtis, kaip įtakoja tinklo veikimą pateikiant jam skirtingus apmokymo duomenis, kiek tinklas gali būti atsparus parinkti teisingą sprendimą.

6.3. **Pradinių duomenų imties įtaka neuroninio tinklo apmokymui**

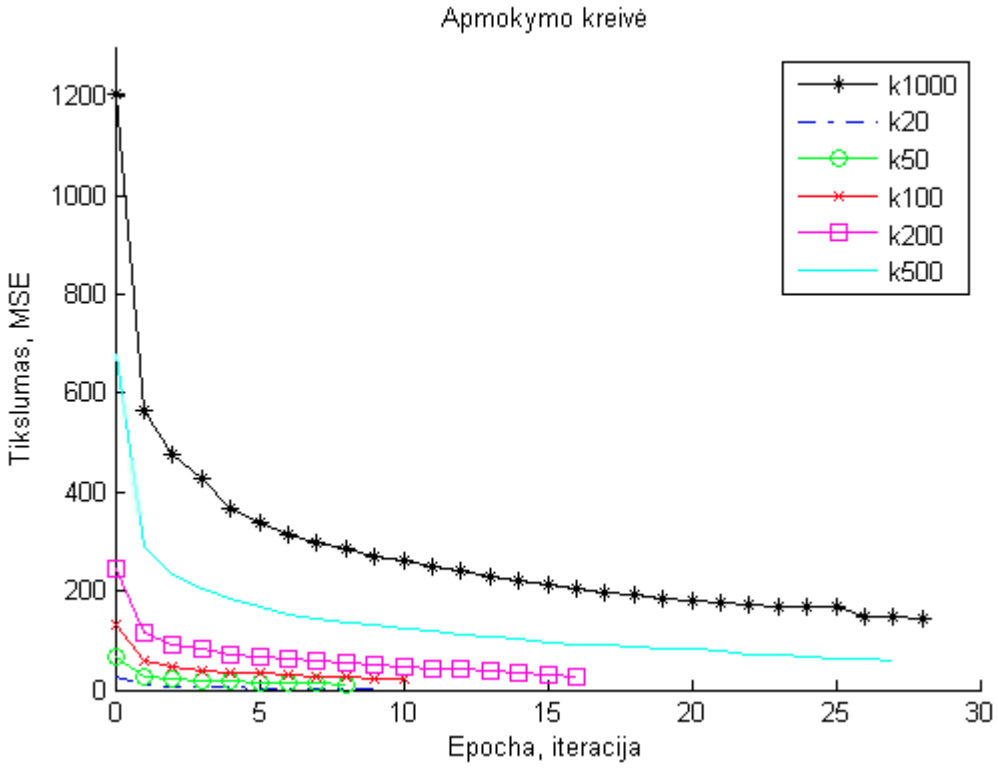
Nustačius neuroninio tinklo struktūrą yra labai svarbu žinoti, kaip gaunamų duomenų tikslumas kinta esant skirtingam mokomųjų įrašų kiekiui. Neuroninio tinklo parametrų analizei buvo naudojama 1000 pradinių duomenų. Papildomai išnagrinėtos imtys: 20, 50, 100, 200 ir 500. Jos sudarytos atsitiktiniu būdu išrenkant reikalingą kiekį duomenų iš 1000 duomenų imties išlaikant proporcijas tarp galimų pasirinkimų (rankinio, automatinio ir nesvarbu). Kiekvienas eksperimentas atliktas 30 kartų. Tyrimo rezultatų vidurkiai pateikti 16 lentelėje, o apmokymo kreivės - 32 pav.

Vertinant gautus rezultatus, kai turimas skirtingas kiekis mokomųjų duomenų, aktualios teisingo suklasifikavimo, pradinių duomenų išmokimo ir naujų duomenų atspėjimo savybės. Jas palyginus tarpusavyje galima įvertinti, kad ne visada didesnis mokomųjų įrašų skaičius lemia tikslesnę parinkimo prognozę. Eksperimentų metu pasiektas tikslumas sumažėja didinant pradinių duomenų skaičių, nes jis gaunamas sumuojant kvadratines paklaidas. Tikslumas yra labiau vertinamas tarp to paties pradinių duomenų kiekio. Neuroninis tinklas apmokytas su 500 ar 1000 pradinių duomenų pateikia aukštesnius įvertinimus už mažesnių

imčių. Naujus duomenis tinklas nuspėja prasčiausiai apmokytas mažiausiu duomenų kiekiu. Bet toks tinklas geriau suklasifikuoja ir išmoksta duotus duomenis nei 50 ar 100 duomenų apmokytas tinklas. Kai tinklas apmokomas su 200 duomenų, jis prasčiau nuspėja naujus duomenis nors ir turi geresnį suklasifikavimo procentą nei 50 ar 100 duomenų. Vertinant gautus rezultatus grafiškai pagal apmokymo kreivę dėl suminės kvadratinės paklaidos gautas tolydus vaizdas, kai kreivės išsidėstę nagrinėjamų duomenų didėjimo tvarka. Atsižvelgiant į gautus rezultatus, buvo pasirinktas optimalus mokomųjų įrašų kiekis vykdant neuroninio tinklo struktūros nustatymą.

16 lentelė. Eksperimentų duomenys kintant imties dydžiui

Imtis / Savybė	20	50	100	200	500	1000
Išmokta duomenų	14.8 (74%)	36.5 (73%)	71.4 (71.4%)	164.7 (82.35%)	427.0 (85.4%)	839.0 (83.9%)
Atspėta naujų	57.78%	67.22%	67.22%	65.56%	77.22%	80.00%
Pasiektas tikslumas	1.58	3.73	6.66	9.80	19.09	38.75
Teisingai suklasifikuota	75.17%	73.07%	71.37%	82.33%	85.41%	83.90%
Laikas, sek.	0.8274	0.8220	0.8771	1.1705	1.9323	2.4939



32 pav. Eksperimentų apmokymo kreivės pagal duomenų imties skaičių

6.4. Sprendimo taikymo rekomendacijos ir galimybės

Automatinio ar rankinio testavimo metodo parinkimas yra pagrindinė sukurto metodo taikymo sritis. Įmonėms ar suinteresuotiems asmenims, kurie pradinių duomenų yra sukaupę

mažiau nei 500, siūloma apdairiau įvertinti metodo teikiamą sprendimą ir kaupiant duomenis stebėti teisingo prognozavimo pasitvirtinimą. Tai padės ateityje priimant sprendimus dėl vieno ar kito testavimo metodo panaudojimo. Pasiūlytas metodas gali būti panaudojamas ir kitiems parinkimo uždaviniams spręsti. Tuomet jį reikia pritaikyti pagal aktualaus uždavinio specifiką, parinkti kriterijus.

Straipsniuose [33], [34] atsižvelgiant į dėstomas mintis, kad kelių skirtingų metodikų apjungimas padeda gauti geresnius rezultatus ir suteikia sistemoms adaptyvumo, apjungus neuroninius tinklus kartu su spiečiaus ar kitu skaitinio intelekto metodu, galima tikėtis patrauklesnių pateikto metodo rezultatų.

7. Išvados

1. Atlikus analizę buvo nuspręsta kurti testavimo tipo parinkimo metodą paremtą dirbtiniais neuroniniais tinklais, skirtą testavimo metodo (rankinis, automatinis) parinkimui.
2. Atlikta metodo funkcinių reikalavimų detali specifikacija, kuri padeda išskirti panaudojimo atvejus, kuriems įgyvendinti pakanka metodo vartotojo ir kuriems yra reikalinga pritaikyta sistema.
3. Sudarytas dalykinės srities modelis, atsižvelgiant į metodo apmokymo ir naudojimo paskirtis. Išskirtos esybės pritaikomos tiek savarankiškai nustatinėjant parametrus, tiek pasinaudojant *Matlab* įrankio specializuotomis galimybėmis.
4. Detalizuotas metodo taikymas patvirtina, kad reikia skirti vienkartines papildomas pastangas, kai metodas nėra apmokytas, o turint jau apmokytą metodą lieka tik išgauti kriterijus iš sprendžiamos parinkimo problemos.
5. Įvertinus metodui realizuoti reikalingas priemones ir programinės įrangos priemones nustatyta, kad *Mathworks* kompanijos *Matlab* produktas leidžia realizuoti įvairius neuroninių tinklų sprendžiamus uždavinius.
6. Remiantis atlikta literatūrine analize buvo nustatyta, kad neuroninį tinklą turi sudaryti 17 įėjimų ir 3 išėjimai. 17 skirtingų kriterijų įtakoja rankinių, automatinių testų parinkimą.
7. Atlikus eksperimentinius tyrimus nustatyta, kad optimalūs sprendimo rezultatai gaunami, kai neuroninio tinklo struktūrą sudaro daugiasluoksnis neuroninis tinklas su dviem paslėptais sluoksniais, 6 neuronais pirmame ir 5 antrame sluoksniuose, tikslumo funkcija *SSE*, tinklui apmokyti naudojamas *Conjugate Gradient with Powell/Beale Restarts* algoritmas.

8. Realizuotas siūlomo metodo sprendimas ir atlikti eksperimentiniai tyrimai patvirtino teorinį metodo tinkamumą sprendžiant testavimo metodo parinkimo problemą.
9. Siekiant pagerinti metodo efektyvumą, ateityje naudinga išbandyti neuroninį tinklą apjungti su kitu skaitinio intelekto metodu, kad metodo vartotojai galėtų tiksliau ir pagrįsčiau priiminėti testavimo sprendimus.

8. Literatūra

1. Danna Henderson. *Making The Decision To Automate Your Software Testing*. [Žiūrėta: 2010-01-14]. Prieiga per internetą: http://www.testingbrain.com/ARTICLES/102.html
2. Ravesoft Solutions - IT services, IT solutions, IT consulting, IT Outsourcing. *Quality Assurance & Testing*. [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://www.ravesoftsolutions.com/qa_testing.html
3. Tech Wizards, Inc. - Services: Testing. *Testing, Validation and Verification.* [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://www.tech-wizards.com/services.test.html
4. KTU Informacijos sistemų katedra. [Žiūrėta: 2009-12-06]. Prieiga per internetą: ftp://isd.ktu.lt/Isd/Ceponiene/T120M123/12%20paskaita.pdf
5. Ron Patton. *Software Testing*. (2nd Edition) Sams Punblishing, 2005. – 408 p. ISBN 0-672-32798-8.
6. OTS Solutions. *Manual Testing V/S Automated Testing*. [Žiūrėta: 2010-01-03]. Prieiga per internet: http://www.otssolutions.com/doc/whitepapers/manual-testing-vs-automated-testing.pdf
7. Mark Fewster, Dorothy Graham. *Software Test Automation: Effective use of test execution tools*. New York: ACM Press/Addison-Wesley Publishing Co., 1999. – 574 p. ISBN 0-201-33140-3.
8. ARTIFICIAL NEURAL NETWORKS - A neural network tutorial. [Žiūrėta: 2009-10-12]. Prieiga per internetą: http://www.learnartificialneuralnetworks.com/
9. Deepam Agarwal. *A Comparative Study Of Artificial Neural Networks And Info Fuzzy Networks On Their Use In Software Testing*. 2004 m. vasara. [Žiūrėta: 2009-10-10]. Prieiga per internetą: http://etd.fcla.edu/SF/SFE0000445/FinalThesis_Deepam.pdf
10. Info-Fuzzy Network (IFN). [Žiūrėta: 2009-11-09]. Prieiga per internetą: http://www.ise.bgu.ac.il/faculty/mlast/ifn.htm
11. Scott Dick, Abraham Kandel. *Computational Intelligence in Software Quality Assurance*. (Series in Machine Perception and Artificial Intelligence - Vol. 63) – Singapore: World Scientific Publishing Company, 2005. – 200 p. ISBN 981-256-172-2.
12. Genetic Algorithm (GA) | Metaheuristics for Optimal Transfer of P2P Information in VANETs. *Genetic Algorithm (GA)*. [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://neo.lcc.uma.es/staff/jamal/portal/?q=content/genetic-algorithm-ga

13. Balsas.lt. *Išlieka tie, kurie yra altruistiški*. [Žiūrėta: 2009-12-07]. Prieiga per internetą: http://www.balsas.lt/naujiena/237642/islieka-tie-kurie-yra-altruistiski/rubrika:naujienos-mokslasirit-mokslas
14. Eric Bonabeau, Marco Dorigo, Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. (Santa Fe Institute Studies in the Sciences of Complexity Proceedings). New York: Oxford University Press, 1999. – 307 p. ISBN 0195131592
15. The Tech FAQ. *What is Swarm Intelligence?* [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://www.tech-faq.com/swarm-intelligence.shtml
16. National Geographic Magazine. *Swarm Theory*. [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://ngm.nationalgeographic.com/2007/07/swarms/miller-text
17. LIACS Natural Computing Group Leiden University. *Swarm Intelligence*. [Žiūrėta: 2009-12-05]. Prieiga per internetą: http://natcomp.liacs.nl/NC/slides/si.pdf
18. Sabu M. Thampi. Swarm Intelligence. [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://arxiv.org/ftp/arxiv/papers/0910/0910.4116.pdf
19. Wikipedia, the free encyclopedia. *Aco branches*. [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://en.wikipedia.org/wiki/File:Aco_branches.svg
20. Russ Eberhart's Home Page. [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://www.engr.iupui.edu/~eberhart/
21. Particle Swarm Optimization. *Introduction.* [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://www.swarmintelligence.org/
22. A.P. Engelbrecht, *Computational intelligence (an introduction)* – West Sussex: John Wiley and Sons Inc., 2002. – 208 p. ISBN 0-470-84870-7
23. Jovita Nenortaitė. *Akcijų prekybos sistema, paremta spiečiaus intelektu ir dirbtiniais neuroniniais tinklais*. Daktaro disertacija. Vilniaus Universiteto leidykla, 2006. – 121 p.
24. Elisa Valentina Oneț. *Particle Swarm Optimization and Genetic Algorithms*. [Žiūrėta: 2009-12-07]. Prieiga per internetą: http://electroinf.uoradea.ro/reviste%20CSCS/documente/JCSCS_2009/Articole_pdf_JCSCS_C_nr_2/JCSCS_2009_Nr_2_CS_Onet_Particle.pdf
25. Dr. Karl O. Jones. *Comparison of Genetic Algorithm and Particle Swarm Optimisation*. // International Conference on Computer Systems and Technologies - CompSysTech'2005. [Žiūrėta: 2009-12-07]. Prieiga per internetą: http://ecet.ecs.ru.acad.bg/cst05/Docs/cp/SIII/IIIA.1.pdf

26. Thom Garrett. *Useful Automated Software Testing Metrics*. [Žiūrėta: 2010-01-30]. Prieiga per internetą: http://www.innovativedefense.com/img/UsefulAutomatedTestingMetrics.pdf
27. Nenortaitė J., Butleris R. *Improving business rules management through the application of adaptive business intelligence technique*. Information Technology and Control, Vol. 36, No. 1, p. 21-28, 2009.
28. The MathWorks. *Neural Network Toolbox. Product Description.* [Žiūrėta: 2010-06-03]. Prieiga per internetą: http://www.mathworks.com/products/neuralnet/description1.html
29. The MathWorks. *Neural Network Toolbox. Funkcions by Class.* [Žiūrėta: 2010-06-18] Prieiga per internetą: http://www.mathworks.com/access/helpdesk_r13/help/toolbox/nnet/tabls12a.html
30. Christopher M. Bishop. *Neural Networks for Pattern Recognition.* New York: Oxford University Press, 1995. – 482 p. ISBN 0198538642
31. The MathWorks. *Systems Requirements – Release 2009a.* [Žiūrėta: 2010-12-16].Prieiga per internetą: http://www.mathworks.com/support/sysreq/release2009a/index.html
32. The MathWorks. *Systems Requirements – Neural Network Toolbox.* [Žiūrėta: 2010-12-16]. Prieiga per internetą: http://www.mathworks.com/products/neuralnet/requirements.html
33. Corchado E., Abraham A., Carvalho A.: Hybrid Intelligent Algorithms and Applications. Information Sciences, 2633–2634 (2010)
34. Gabrys B.: Do Smart Adaptive Systems Exist? Hybrid Intelligent Systems Perspective. In: Corchado, E., Abraham, A., Pedrycz, W. (eds.) HAIS 2008. LNCS (LNAI), vol. 5271, pp. 2–3. Springer, Heidelberg (2008)
35. E. Corchado, M. Kurzynski, M. Wozniak (Eds.): HAIS 2011, Part I, LNAI 6678, pp. 247-254. Springer, Heidelberg (2011)

9. Santrumpų ir terminų žodynas

Santrumpa	Angliškas terminas	Lietuviškas terminas
AB	*Achieving Budget*	Testavimo finansinių lėšų išnaudojimas
ACO	*Ant Colony Optimization*	Skruzdžių kolonijos optimizavimo algoritmas
ANN	*Artificial Neural Networks*	Dirbtiniai neuroniniai tinklai
CE	*Coverage Extension*	Testavimo apimties padidinimas
DD	*Defect Density*	Defektų tankis
DDP	*Defect Detection Percentage*	Procentinis defektų aptikimas
DDT	*Defect Detection in Testing*	Testavimo metu aptiktų defektų skaičius
FP	*Function Point*	Funkcinis taškas
GA	*Genetic Algorithms*	Genetiniai algoritmai
IFN	*Information Fuzzy Network*	Miglotos informacijos tinklas
KLOC	*Kilo lines of code*	Tūkstantis kodo eilučių
MLP	*Multilayer Perceptron*	Daugiasluoksnis perceptronas
MSE	*Mean Squared Error*	Vidutinė kvadratinė paklaida
MTBF	*Mean Time Between Failure*	Vidutinis laikas tarp sistemos lūžimų
MTTF	*Mean Time to Failure*	Vidutinis sistemos lūžimo laikas
PĮ	*Software*	Programinė įranga
PSO	*Particle Swarm Optimization*	Dalelių spiečiaus optimizavimo algoritmas
ROI	*Return of Investment*	Investicijų grąža
SI	*Swarm Intelligence*	Spiečiaus intelektas
SSE	*Sum Squared Error*	Suminė vidutinė kvadratinė paklaida

10. Priedai

1 priedas. Straipsnis leidinyje „Informacinė visuomenė ir universitetinės studijos IVUS 2010“

Dirbtiniais neuroniniais tinklais paremtas testavimo tipo pasirinkimo metodas

Kristina Smilgytė,
Informacijos sistemų katedra
Kauno technologijos universitetas
Kaunas, Lietuva
El. paštas: kristina.smilgyte@gmail.com

Jovita Nenortaitė
Informacijos sistemų katedra
Kauno technologijos universitetas
Kaunas, Lietuva
El. paštas: jovita.nenortaite@ktu.lt

Anotacija. **Esant konkurencingai IT kompanijų rinkai, didelis dėmesys teikiamas kuriamų sistemų kokybei. Kuriant informacines sistemas, svarbūs ne tik analizavimo, projektavimo, kūrimo etapai, bet ir kuriamo produkto testavimo procesas. Tiksliai, laiku bei kokybiškai atliktas testavimas leidžia užtikrinti kuriamo produkto kokybę, tai turi tiesioginę įtaką viso projekto sėkmei bei įmonės įvaizdžio kūrimui. Dažnu atveju sudėtinga nustatyti, kokio tipo testavimas turėtų būti taikomas vienam ar kitam komponentui testuoti, koks testavimo tipas būtų efektyvesnis ir leistų sutaupyti įmonės resursų. Šiame straipsnyje nagrinėjama testavimo tipo pasirinkimo problema. Tyrimo objektas: testavimo tipo pasirikimo metodas, pagrįstas dirbtinio intelekto metodų taikymu. Straipsnyje pateiktas automatinio ir rankinio testavimo tipų palyginimas, siekiant išskirti vieno ar kito testavimo tipo privalumus bei trūkumus. Apžvelgiamas dirbtinio intelekto metodų taikymas testavimo procesuose bei pristatomas autorių siūlomas testavimo tipo pasirinkimo metodas, paremtas dirbtinių neuroninių tinklų taikymu.**

Raktiniai žodžiai: testavimas, dirbtiniai neuroniniai tinklai, projektų vadovas, informacinės sistemos.

I. ĮVADAS

Vykdant įvairius informacinių technologijų projektus, vis aktualesnis tampa klausimas, kaip efektyviau panaudoti turimus resursus kokybiškam sistemų testavimui. Didėjant vartotojų poreikiui, kuriamos sistemos tampa vis didesnės, sudėtingesnės, kas įtakoja ir sudėtingesnį jų testavimą. Įvairūs su sistemų testavimu susiję klausimai aktualūs ne tik testuotojams, bet ir projektų vadovams bei įmonėms.

Didelę reikšmę vykdant didelius projektus turi atliekami tikslūs ir motyvuoti sprendimai. Testuojant sistemą, projektų vadovui neretai sudėtinga nuspręsti, ar sistemos komponentų testavimui reikalingi automatiniai testai. Jei yra galimybė įvertinti automatinių testų poreikį, iš jų gaunamą naudą bei jų atsipirkimo lygį, projektų vadovams sudaromos sąlygos efektyviau paskirstyti resursus bei sumažinti projekto kaštus. Pagrindinė pasirinkimo problema tarp rankinio ir automatinio testavimo – jie negali vienas kito visiškai pakeisti. Literatūroje minimi įvairūs automatinių ir rankinių testų privalumai ir trūkumai, bet konkrečiu atveju išlieka pasirinkimo pagrįstumo problema.

Aspire systems kompanija yra sukūrusi automatinių testų investicijų grąžos skaičiuoklę [1]. Jos esmė – parodyti, kiek pinigų ir laiko kiekvienais metais padeda sutaupyti automatiniai testai [2]. Straipsnyje [3] pristatoma automatinių testų investicijų grąžos skaičiuoklė, bet šio straipsnio autoriai nesprendžia testavimo pasirinkimo problemos. Sistemoms testuoti reikalingas kombinuotas testavimas, tai reiškia, kad nepakanka vien žinoti, kiek ir kuriais metais galima sutaupyti pinigų ir laiko visam vykdomam projektui. Reikia ir aiškiai žinoti, koks testavimo tipas bus taikomas tam tikram sistemos komponentui. Atlikus esamų sprendimų analizę, buvo nuspręsta pasiūlyti metodą, kuris padėtų įvertinti, kiek verta automatizuoti testavimą arba kuris testavimas (rankinis ar automatinis) yra naudingesnis konkrečiose situacijoje. Kartu įvertinama ne vien investicijų grąža, bet ir grynoji dabartinė vertė, testavimo aprėptis, efektyvumas, tikslumas ir kiti parametrai.

Straipsnyje pristatomas metodas skirtas palengvinti ir pagrįsti testavimo metodo pasirinkimą. Metodas paremtas dirbtinio intelekto metodų taikymu. Remiantis dirbtinio intelekto metodų analizės rezultatais, buvo nuspręsta naudoti dirbtinius neuroninius tinklus (DNT). Siūlomas metodas padės nuspręsti, ar reikalingi automatiniai testai. Norint naudotis metodu, reikia pateikti įėjimo duomenis, kurie apdorojami dirbtiniu neuroniniu tinklu. Atlikus apmokymus bei paskaičiavus išėjimo parametrus yra gaunamos rekomendacijos, kurį testavimo tipą (automatinį ar rankinį) pasirinkti. Įvertinus gautas rekomendacijas galima priimti tikslesnius sprendimus, nes pasiremiama ne tik testavimo tipo privalumais, trūkumais ar investicijų grąža, bet ir kuriamo metodo gaunamais rezultatais. Kuriant testavimo metodą siekiama išlaikyti tikslumą, kad šio metodo taikymas testavimo procese procesui galėtų garantuoti testavimo efektyvumą.

Antroji straipsnio dalis skirta problemai pristatyti ir susijusiems darbams apžvelgti. Trečioje straipsnio dalyje aptariami testavimo tipų privalumai ir trūkumai. Ketvirtoje dalyje pateikiama dirbtinio intelekto metodų bei jų taikymo testavimo procesuose apžvalga. Penkta straipsnio dalis skirta siūlomo testavimo tipo pasirinkimo metodui pristatyti. Straipsnio išvados bei numatomi darbai pateikti paskutinėje straipsnio dalyje.

II. SUSIJUSIŲ DARBŲ APŽVALGA

Įvairių programų testavimas turi didelę įtaką galutinio produkto bendrai kokybei ir patikimumui. Kūrimo procese

aptiktos įvairios klaidos kainuoja daug mažiau, nei pastebėtos atidavus produktą vartotojui(-ams). Laiku ištaisytos klaidos reikalauja mažiau pastangų, nes priklausomai nuo klaidos tipo galima išvengti esminių sistemos pakeitimų. Įdiegus sistemą vartotojui ir tik tada aptikus klaidą, reikia ne tik ją ištaisyti, bet ir dar kartą atnaujinti ar įdiegti sistemą. Produkcinėje sistemų versijoje likusios klaidos gali atnešti didelių nuostolių tiek sistemos užsakovo įmonei, tiek sistemą kuriančiai įmonei. Gerinant produkto kokybę ir turint fiksuotus, ribotus išteklius, svarbu pasirinkti tinkamą testavimo tipą (automatinį ar rankinį) konkrečiam atvejui. Ne visada reikia ir apsimoka automatizuoti testavimą [4], todėl svarbu tinkamai įvertinti testavimo tipo pasirinkimą.

Vis dažniau kuriamos programinės įrangos testavimui – pasirenkami automatiniai testai. Rengiant automatinius testus, siekiama užtikrinti testavimo tikslumą (automatiniai testai padengia visus aprašytus scenarijus, išvengiama žmogiškųjų klaidų pamiršus pratestuoti vieną ar kitą scenarijų), taip pat siekiama sutaupyti testavimui skiriamą laiką (automatiniai testai patikrina visus aprašytus scenarijus greičiau, nei tai padarytų testuotojas; automatinio testo vykdymo metu testuotojas gali atlikti kitus jam priskirtus darbus). Bet taip pat svarbu žinoti, kad parengti automatinius testus – daug laiko reikalaujantis darbas. Todėl sprendžiant, ar tam tikro komponento testavimui bus kuriamas automatinis testas, būtina įvertinti laiko sąnaudas, kaštus ir automatinių testų teikiamą naudą. Pagrindinis asmuo, turintis planuoti atliekamus darbus, nustatyti darbų trukmę bei sekti, kaip vykdomi darbai, – projekto vadovas. Darbų planavimas, jų terminų nustatymas yra atliekamas konsultuojantis su tam tikros srities specialistais, kurie geriau išmano tam tikrą dalykinę sritį ir gali aiškiai nusakyti darbų sudėtį, jų trukmę ir panašius dalykus. Tokiu pačiu principu nustatoma ir testavimo darbų trukmė. Bet visais atvejais projekto vadovas turi pateikti galutinį sprendimą dėl vienų ar kitų darbų atlikimo, įvertinęs projekto apimtį, kaštus ir laiką. Testavimo etape, siekdami tinkamai koordinuoti viso projekto vykdymą, projektų vadovai turi atitinkamai įvertinti testavimo tipo pasirinkimą. Pagrindinė problema – ne visada projektų vadovai turi užtektinai žinių, leidžiančių teisingai pasirinkti testavimo tipą. Net jei žinių ir pakanka, testavimo tipo pasirinkimas nėra vienareikšmis ir turi būti įvertinta daug faktorių, norint nutarti, kad geriau naudoti rankinį ar automatinį testavimą, ar jų derinius konkrečiu atveju.

Vienas pagalbininkų projektų vadovams gali būti minėta automatinių testų investicijų grąžos skaičiuoklė. Suvedus reikalingus duomenis [1], gaunama ataskaita, kurioje pateikiama, kiek konkrečiais projekto vykdymo metais laiko ir pinigų reikalauja automatinis ir rankinis testavimas. Apibendrinant šį palyginimą pateikiama, kiek konkrečiai sutaupoma naudojant automatinius testus [2]. Skaičiuoklę sukūrusi kompanija ją vadina praktiniu gidu, padedančiu geriau suvokti investicijų grąžą pasirinkus automatinius testus. Sprendžiant, ar naudoti šį testavimo tipą, svarbu žinoti, kiek laiko ir pinigų galima tikėtis sutaupyti [3].

Neretai automatinių testų pasirinkimas priklauso nuo testuotojų mastymo ir kompetencijos, nes kiekvienas jų turi nusistovėjusį kriterijų, klausimų sąrašą, pagal kuriuos pasirenka, ar reikia automatinių testų. Vienas populiariausių klausimų: kaip dažnai bus naudojamas kuriamas testas. Nagrinėjant reikia išskirti du atvejus: kai testas yra tinkamas be pakeitimų ir kai reikia atnaujinti nedidelę dalį. Geriausias – pirmas atvejis, nes jis nereikalauja jokių papildomų pastangų ir palaikymo. Antru atveju reikia įvertinti naudą, ar atnaujinto testo naudojimas atsvers pastangas jam atnaujinti. Kitas populiarus klausimas: ar testuotinas atvejis yra varginantis ir turi polinkį į klaidas. Testuojant reikalingas atidumas ir nepasimetimas, pavyzdžiui, reikia patikrinti, ar sistema, atidarius 1 000 vartotojo dokumentų, nenulūžta. Panašiems testams atlikti efektyvesnis yra automatinis būdas, nes 1 000 dokumentų sistema atidarys ir greitai, ir tiksliai. Svarbu per daug nesusižavėti automatinių testų teikiama nauda neįvertinus sukūrimo pastangų. Jeigu 1 000 dokumentų atidarymo testas įvykdomas per 1 minutę, o jam sukurti užtrunkame 30 minučių, automatinį testą reikia vykdyti bent 30 kartų, kad atsipirktų automatizavimo pastangos [5]. Vyrauja dvi populiarios nuomonės žinant, kad testas bus vykdomas ne vieną kartą. Pirmoji: reikia automatizuoti, jeigu testas naudojamas daugiau nei vieną kartą. Sėkmės atveju didės grįžtamoji nauda, nesėkmės – teks nuolat atlikti atnaujinimus arba testas nespės atsipirkti ir bus nebenaudojamas. Antroji: reikia automatizuoti tik tada, kai daug kartų rankiniu būdu naudojamas testas tampa stabilus ir nusistovėjęs. Sėkmės atveju nereikės atnaujinti ir jis bus reikalingas, nesėkmės – nespės atsipirkti. Kuriant automatinius testus svarbu įvertinti, kada automatinis testas gali aptikti klaidą, kiek reikės įdėti pastangų norint ją išnagrinėti. Pavyzdžiui, turime automatinį testą, kuris trunka 30 minučių, jį vykdant aptinkama klaida 29-ą minutę. Kiekvieną kartą norint patikrinti, ar klaida ištaisyta, reikia ilgai laukti. Jeigu nusprendžiama kurti automatinius testus, jie turi būti specifiniai ir kompaktiški.

Analizuojant automatinių testų poreikį pagal klausimus įvertinami testavimo tipų (automatinis, rankinis) privalumai ir trūkumai. Atliktas testavimo tipų įvertinimas padeda tiksliau pasirinkti turint konkrečius atsakymus į klausimus. Vienas iš automatinių testų privalumų anksčiau minėtame 1 000 dokumentų atidarymo pavyzdyje – išlaikomas tikslumas, testuotojas kartais gali pamesti skaičių, kai testas vykdomas daug kartų. Plačiau apie privalumus ir trūkumus – testavimo tipų palyginimo skyriuje.

III. TESTAVIMO TIPŲ PALYGINIMAS

Rankinis testavimas – tai testavimo būdas, kai reikalingas testuotojas testo duomenims įvesti, analizuoti, įvertinti [6]. Testuotojas sukuria visus testavimo atvejus ir juos įvykdo testuojamai programai rankiniu būdu (neautomatiškai). Žingsnis po žingsnio nustatoma, ar tam tikras žingsnis buvo įvykdytas sėkmingai ar klaidingai. Automatinis testavimas – tai testavimo būdas, kai naudojamos programos bei įrankiai, skirti kitų programų ar modulių darbui patikrinti. Automatinis testavimas naudoja programas testavimo atvejams įvykdyti, rezultatams palyginti ir surastoms klaidoms registruoti nesikišant testuotojui. Testuoti sistemą pagal automatizuotų įrankių scenarijus ypač sudėtinga pradedantiesiems testuotojams, nes testuotojas turi turėti geras programavimo žinias ir sugebėti parašyti gerą scenarijų bet kokiam testavimo atvejui [7]. Siekiant tinkamai pasirinkti, kada kokį metodą geriau rinktis, galima pasinaudoti jų privalumų ir trūkumų palyginimu [7], kuris pateiktas 1 lentelėje.

1 LENTELĖ. TESTAVIMO TIPŲ PALYGINIMAS

Kriterijus	Rankinis	Automatinis
Kaina	(+) Pigesnis	(–) Brangesnis
Laikas	(–) Vykdomi lėčiau	(+) Vykdomi greičiau
	—	(–) Testų kodo parengimas lėtas
Reikalavimų pasikeitimai	(+) Nesudėtingai įgyvendinami	(–) Reikalingas testų kodo pakeitimas
Testavimo aprėptis	(+) Platesnė	(–) Fiksuota
Teksto skaitymas	(+) Skaito	(–) Neskaito, nesuvokia prasmių
Spalvų skyrimas	(–) Priklauso nuo jų kombinacijų	(+) Gerai skiria
Atgalinis testavimas	(–) Sudėtingas, laiko reikalaujantis	(+) Puikiai tinka
Išmokstamumas	(+) Nesudėtingas	(–) Sudėtingas, ypač pradedantiem
Klaidų aptikimas žvalgantis, *Ad-hoc*	(+) Daugiau klaidų randama	(–) Mažiau

1 lentelėje pateikti privalumai ir trūkumai yra bendro obūdžio, todėl konkrečiam projektui reikia detalesnės ıformacijos, norint pasirinkti testavimo tipą. Pavyzdžiui, ertinant pagal testavimo aprėptį, jeigu reikia ištestuoti latesnę sritį, geriau rinktis rankinį testavimą. Jeigu svarbu ksliai peržiūrėti tą pačią sritį, ar giliau išanalizuoti siauresnę ritį, geriau rinktis automatinį. Automatiniai testai turi didelį ranašumą, palyginti su rankiniais, ypač jei testai nuolat audojami pakartotinai daug kartų, jų priežiūra, palaikymas ereikalauja didelių pastangų. Praktikoje automatinis stavimas negali visiškai pakeisti rankinio, todėl pasirinkimo roblema lieka aktuali.

IV. DIRBTINIO INTELEKTO METODŲ TAIKYMAS TESTAVIMO PROCESUI

Plačiai naudojami dirbtinio intelekto metodai neaplenkia ir stavimo srities. Straipsnyje nagrinėjami keturi dirbtinio ıtelekto metodai, išskiriami į dvi grupes: pateiktas dirbtinių euroninių ir miglotos informacijos tinklų palyginimas ir ristatomi nagrinėti genetinių algoritmų ir spiečiaus intelekto ptimizavimo algoritmai, jų palyginimas.

Populiariausi dirbtinio intelekto metodai, naudojami stavimui, – dirbtiniai neuroniniai (DNT, angl. *Artificial eural networks*) ir miglotos informacijos tinklai (MIT, ngl. *Information Fuzzy Network*). DNT idėja kilo iš eurobiologijos. Neuroniniai tinklai sudaryti iš tarpusavyje usijungusių neuronų, suskirstytų į sluoksnius. Į kiekvieną inklo neuroną įterpiama perdavimo funkcija. Kiekvieno luoksnio neuronai sujungti su gretimų sluoksnių neuronais. To aties sluoksnio neuronai dažniausiai nėra sujungti. Tokia euronų ir jungčių visuma vadinama dirbtiniu neuroniniu inklu. Neuroniniai tinklai turi įėjimo, išėjimo ir tarpinius luoksnius [8]. MIT – metodas, sukurtas žinioms atskleisti ir uomenims gauti. Metodą sukūrė M. Lastas kartu su O. Maimonu ir A. Kandelu [9]. Jie taikomi renkantis ir įvertinant stavimo atvejus, automatizuotam orakului gauti. Remiantis NT ir MIT metodų lyginamuoju tyrimu sudarant automatinį rakulą [10], abu būdai yra efektyvūs, kai klaidingų atvejų rocentas yra ganėtinai didelis. Dirbtinis neuroninis tinklas – eresnis nuolat vertinamų išėjimų klasifikatorius. Ypač, kai mokomų įrašų yra mažai. Pagrindinis miglotos informacijos tinklo pranašumas prieš dirbtinį neuroninį tinklą – jis yra daug greitesnis. O dėl savo struktūros abu tinklai tinka intelektualiam testavimui vykdyti.

Vieni populiariausių optimizavimo algoritmų, priklausančių dirbtinio intelekto metodams, – genetiniai algoritmai ir spiečiaus intelektas, kurio populiariausi optimizavimo algoritmai – skruzdžių kolonijos ir dalelių spiečius [11].

Genetinių algoritmų (GA, angl. *Genetic Algorithms*) pradininku laikomas J. Hollandas [12]. 1975 m. jis pasiūlė matematinį optimizavimo metodą, paremtą natūraliais gamtoje vykstančiais procesais: natūraliąja individų atranka, kryžminimu ir mutacija. GA – vienas populiariausių metaeuristinių algoritmų – stochastinis optimizavimo metodas, nes vienas iš procesų yra atsitiktinė, bet į konkretų tikslą nukreipta individų atranka. Pasinaudojus evoliucijos mechanizmais, galima sukurti programas, kurios sprendžia problemas net tada, kai nėra visiškai aiškus sprendimo būdas [12]. Genetiniai algoritmai dažniausiai naudojami intelektualiai paieškai, optimizavimui ir kompiuterių apmokymui. GA dažnai naudojamas su įvairiais dirbtinio intelekto metodais. Šiuo metu GA naudojami kartu su neuroniniais tinklais ir miglota logika sudėtingoms problemoms spręsti. Dėl jungtinio jų panaudojimo daugeliui problemų spręsti neuroniniai tinklai ir miglota logika vadinama *soft-computing* [8].

Spiečiaus intelektas (SI, angl. *Swarm Intelligence*) apibūdina decentralizuotų, save organizuojančių, natūralių ar dirbtinių sistemų kolektyvinį elgesį. Vienas spiečiaus intelekto privalumų – jis gali pasiūlyti sprendimus įvairioms problemų rūšims spręsti. Sistemos yra labai tvirtos ir lanksčios, atsparios aplinkos pasikeitimams. Vientisos sistemos elgesys viršija vieno individo turimus elgesio gebėjimus [13].

Skruzdžių kolonijos optimizavimo algoritmą (SKO, angl. *Ant Colony Optimization*) 1992 m. pasiūlė M. Dorigo. Skruzdžių kolonijos elgesys – vienas populiariausių spiečiaus elgesio modelių. SKO algoritmo pagrindinė idėja kilo stebint skruzdžių maisto atsargų paieškas [11]. Pavienės skruzdėlės elgiasi atsitiktinai ir be pastebimo tikslo, bet atsiradus kolektyvinei sąveikai tarp skruzdžių, galima stebėti jų spiečiaus intelektą ir elgesį, kuris gali išspręsti daug problemų. Skruzdžių spiečius gali nustatyti trumpiausią kelią iki maisto šaltinio, pamaitinti visą koloniją, pastatyti didelę struktūrą ir prisitaikyti prie įvairių situacijų. Tinka spręsti įvairias problemas, kurias galima pavaizduoti grafu.

Dalelių spiečiaus optimizavimo algoritmą (DSO, angl. *Particle Swarm Optimization*) 1995 m. pasiūlė R. Eberhartas ir J. Kennedy [14]. Algoritmas sukurtas remiantis paukščių bandos elgesio stebėjimais [15], panašus į GA. Sistema sužadinama su atsitiktinių sprendimų populiacija ir ieškoma optimalumo kartoms atnaujinti. Bet priešingai nei GA, DSO nenaudoja kryžminimo ir mutacijų. DSO algoritme potencialūs sprendimai skrieja per probleminę erdvę sekant palankiausias sąlygas. Kiekvienas atskiras sprendimas paieškos erdvėje yra tarsi paukštis, pavadintas dalele. Visos dalelės turi tinkamumo reikšmes, kurios, siekiant optimizuoti, nustatomos tinkamumo funkcija. Dalelės taip pat turi kryptį ir greitį, kuris nukreipia jų skriejimą. Dalelės skrieja per probleminę erdvę

sekdamos dabartinę optimalią dalelę, vadinamą gidu [11]. Mokymo metu neeliminuojami prasčiausiai pasirodę individai, todėl DSO dažniau naudojamas dirbant su sparčiai, kritiškai kintančiomis aplinkomis, nes yra svarbu adekvačiai sureaguoti į skirtingas situacijas, kurios dažnai gali būti visiškai priešingos esamai situacijai.

Tiek GA, tiek SI yra optimizavimo algoritmai. GA ir DSO yra panašios struktūros, naudoja tinkamumo, tikslo funkcijas. Dalelė DSO algoritme panaši į chromosomą, kuri yra GA populiacijos narys. Tiek chromosoma, tiek dalelė atstovauja galimiems problemos sprendimams [16]. DSO neturi kryžminimų ir mutacijų, nes remiasi paukščių bandos elgesiu. Tačiau DSO algoritme dalelės atnaujina save vidiniu greičiu, jos taip pat turi atmintį, kuri yra svarbi algoritmui [17]. GA remiasi natūralia individų atranka, kurioje neprisitaikę individai miršta – DSO algoritme to nėra. Dėl šios priežasties GA yra greitesnis, bet dirbant su pokyčiams jautria ir nuolat besikeičiančia sistema geriau tinka DSO [18]. Šiuo atveju GA trūkumas dirbant su tokiomis sistemomis – kritiškai pasikeitus situacijai, galima prarasti reikalingus duomenis.

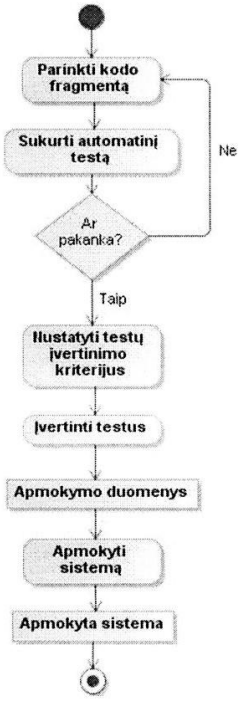
Kiekvieno metodo savybėms, naudojimo paskirčiai, kuriamo metodo sprendžiamai problemai įvertinti siūlomame metode pasirinkta naudoti dirbtinius neuroninius tinklus, nes jie paprastai naudojami klasifikavimo, prognozavimo uždaviniams spręsti. Sistemai apmokyti reikia mažiau mokomųjų įrašų nei MIT, mokomųjų įrašų skaičius yra svarbesnis už veikimo laiką, nagrinėjamo testavimo tipo parinkimo aplinka nėra greitai, chaotiškai besikeičianti. Miglotos informacijos tinklai pagal struktūrą labiau tinka sprendimų medžio uždavinių tipui, genetiniai algoritmai – paieškai, kompiuterių apmokymams, optimizavimui, skruzdžių kolonijos optimizavimo algoritmas – uždaviniams, kuriuos galima pavaizduoti grafu, dalelių spiečiaus optimizavimo algoritmas – chaotiškoms, greitai besikeičiančioms aplinkoms. Kuriamo metodo tikslas – pasiūlyti, prognozuoti, numatyti automatinių testų reikalingumą, todėl neuroniniai tinklai yra priimtinesni už miglotos informacijos tinklus ir optimizavimo algoritmus.

V. SIŪLOMAS SPRENDIMAS

Reikalingas metodas (modelis), kuris būtų skirtas testavimo metodui parinkti ir nurodytų, kokį testavimo metodą naudoti tam tikro testavimo atveju. Renkantis testavimo metodą reikia įvertinti šiuos esminius kriterijus: testavimo efektyvumą ir tikslumą. Atlikus tyrimo objektų, vartotojų, esamų spendimų analizę pastebėta, kad dažniausiai testavimo tipai pasirenkami pagal privalumus, trūkumus ar testuotojo, vadovo intuiciją. Daugelis testavimo atvejų papildo ar iš dalies aprėpia vienas kitą, todėl projektų vadovui gali būti sudėtinga nuspręsti, kurį testavimo metodą (automatinį ar rankinį) geriau naudoti, nes smulkesnius testavimo metodus pasirenka patys testuotojai. Testuotojų komanda gali tik patarti projektų vadovui, ar apsimoka kurti, ar pirkti konkretų automatizuotą testavimo įrankį, ar užtenka rankinio testavimo – juk vadovas atsakingas už projektą. Jis žino, kiek galima išleisti papildomų pinigų ir skirti papildomų resursų testavimui, įvertina ir kitus vykdomo projekto etapus bei jiems atlikti reikalingas sąnaudas.

Siūlomas sprendimas galėtų padėti projektų vadovui lengviau pasirinkti ir atsakingiau įvertinti, kurį testavimo metodą (automatinį ar rankinį) geriau naudoti vienu ar kitu atveju. Šis sprendimas ypač aktualus, jeigu automatizuojančią priemonę reikia kurti, koreguoti ar pirkti. Jeigu organizacija jau turi ją, papildomų rūpesčių nesudaro sėkmingai ją adaptuoti ir naudoti kaip papildomą priemonę klaidoms aptikti. Bet ir tokiu atveju reikia įvertinti testavimo metodo pasirinkimą, nes priemonės adaptavimas tam tikram testavimo atvejui taip pat gali pareikalauti papildomų kaštų bei resursų.

Kuriamas metodas remiasi dirbtinio intelekto pritaikymu siekiant nustatyti, ar konkrečiu atveju geriau naudoti automatinį, rankinį ar abu testavimo metodus. Tokiam metodui pirmiausiai reikia apmokyti išmokyti sistemą, kad būtų galima gauti norimus rezultatus.



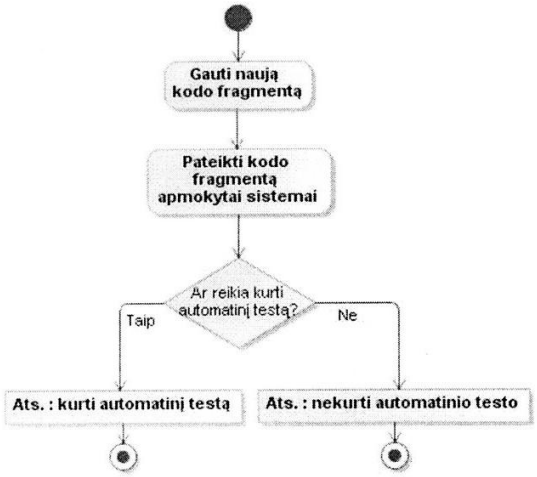
1 pav. Sistemos apmokymas

1 pav. pavaizduotas sistemos apmokymo procesas:

1. parenkamas tam tikras skaičius kodo fragmentų ir jiems sukuriami automatiniai testai;
2. nustatomi testų įvertinimo kriterijai;
3. įvertinami visi turimi testai ir kaupiami gauti duomenys sistemos apmokymui;
4. atliekamas metodo (sistemos) apmokymas.

Sistemos apmokymui pasirinkta naudoti dirbtinį neuroninį tinklą. Testavimo metodui apmokyti DNT pasirinktas remiantis analizės gautomis išvadomis [10], nes jis yra geresnis klasifikatorius nuolat vertinant išėjimus nei MIT. Ypač, kai mokomų įrašų yra mažai.

Apmokius metodą, galima gauti įvertinimą, svarbų sprendžiamai problemai išspręsti, kurį testavimo metodą (automatinį ar rankinį) geriau naudoti. Kuriamo metodo principinė schema pateikta 2 pav.: apmokytai sistemai pateikus naują kodo fragmentą, atitinkamai pagal kodo parametrus sistema rekomenduoja, ar reikia kurti automatinį testą, ar ne. Taip gaunamas automatizavimo lygis sistemai testuoti.



1 pav. Kuriamo metodo principinė schema

Numatoma siūlomo metodo efektyvumą įvertinti ne tik pagal investicijų grąžą, bet ir pagal grynąją dabartinę vertę, testavimo aprėptį, defektų tankį, automatinių testų metu pateiktą klaidingų klaidų skaičių, defektų skaičių.

Detalizuojamos naudojamos metrikos:

- DT (angl. *Defect Density*) – defektų tankis. Defektų kiekio matu laikomas per tam tikrą laikotarpį surastų defektų kiekis, pavyzdžiui, surastų defektų skaičius nuo modulio sukūrimo iki esamos datos. DT naudojamas defektų tankiams skirtinguose programinės įrangos komponentuose palyginti. Tai padeda identifikuoti kandidatus detalesnei peržiūrai, testavimui, struktūros pertvarkymui (angl. *re-engineering*) ar pakeitimui. Kuo defektų tankis mažesnis, tuo geresnė kokybė. $DT = \frac{N_D}{S}$, čia N_D – žinomų defektų kiekis, S – dydis, paprastai matuojamas *KLOC* (angl. *Kilo Lines Of Code*) tūkstančiais kodo eilučių, bet gali būti naudojami ir funkciniai taškai.
- Klaidingų klaidų skaičius, pateiktas automatinių testų metu. Tai klaidos, kurios iš tikro nėra klaidos. Toks įvertinimas gali padėti nustatyti, ar automatiniai testai linkę labiau padėti ar klaidinti.
- TA (angl. *Coverage extension*) – testavimo apimties padidinimas. Rankiniu testavimu surastos klaidos lyginamos su papildomomis klaidomis, kurias padėjo aptikti automatiniai testai (rankiniu testavimu jos buvo praleistos, neaptiktos). Šis įvertinimas naudingas tuo, kad galima palyginti automatinių testų pridėtinę vertę klaidoms aptikti.
- FLI (angl. *Achieving Budget*) – testavimui skirtų finansinių lėšų išnaudojimas. Ši metrika padeda įvertinti testavimui skirtų lėšų išnaudojimą. Naudojant FLI, galima sužinoti, kuris testas – rankinis ar automatinis – sunaudoja mažiau pinigų ir neviršija nustatyto limito. $FLI = \frac{C_A}{C_B}$, čia C_A – reali kaina, išnaudota testavimui, C_B – testavimui numatytas biudžetas.
- Laikas, skirtas analizuoti automatinių testų pateiktoms klaidoms. Rankiniais testais aptiktos klaidos nereikalauja papildomos analizės, nes iškart žinoma, ką vykdant ir kokia klaida buvo aptikta.

Lyginant testavimo metodus, reikia remtis keliais vertinimo kriterijais – taip galima gauti tikslesnį atsakymą, kiek verta automatizuoti testavimą arba kuris testavimas (rankinis ar automatinis) yra naudingesnis konkrečiai situacijai.

VI. IŠVADOS

Atlikta testavimo metodų analizė parodė, kad egzistuoja testavimo tipo pasirinkimo problema tarp automatinių ir rankinių testų. Įvertinus, kad reikia prognozuoti ir siūlyti, koks testavimo tipas geresnis konkrečiu atveju, buvo nuspręsta metodui apmokyti naudoti dirbtinius neuroninius tinklus. Kuriamas metodas skirtas projektų vadovams, testuotojams. Svarbiausia, kad metodas padėtų lengviau pasirinkti ir atsakingiau įvertinti, kuris testavimo metodas (automatinis ar rankinis) naudingesnis vienu ar kitu atveju. Kad metodas būtų tikslesnis, numatoma jo efektyvumą įvertinti ne tik pagal investicijų grąžą, bet ir testavimo aprėptį, defektų tankį, klaidų skaičių, pateiktą automatiniais testais.

Ateityje planuojama nustatyti pasiūlyto metodo naudotinus parametrus (neuroninių tinklų įėjimų skaičių, parinkti struktūrą ir pan.). Veikimui demonstruoti numatoma sukurti pasiūlyto metodo prototipą. Sukūrus prototipą – atlikti eksperimentus, kurie padėtų detaliau ištirti kuriamo metodo efektyvumą ir veikimą.

LITERATŪROS SĄRAŠAS

[1] Aspire Systems. "Test automation ROI calculator". [Žiūrėta: 2010-02-10]. Prieiga per internetą: http://www.aspiresys.com/testautomationroi/.

[2] Aspire Systems. "Test automation ROI report sample". [Žiūrėta: 2010-02-10]. Prieiga per internetą: http://www.aspiresys.com/testautomationroi/TA_ROI_Report.pdf.

[3] A. Narayanan, Aspire Systems. "Test automation ROI calculator". [Žiūrėta: 2010-02-10]. Prieiga per internetą: http://www.aspiresys.com/Register.php?des=WhitePapers/whitepaper_Test_Automation_ROI_Calculator.pdf.

[4] D. Henderson. "Making the decision to automate your software testing". [Žiūrėta: 2010-01-14]. Prieiga per internetą: http://www.testingbrain.com/ARTICLES/102.html.

[5] D. Weiss. "When to Automate Testing?" [Žiūrėta: 2010-01-06]. Prieiga per internetą: http://davidweiss.blogspot.com/2006/08/when-to-automate-testing.html.

[6] Software Testing - Tutorials, QTP, Manual Testing Automation Testing, Load Runner. "Tricky software testing terms". [Žiūrėta: 2010-01-25].

Prieiga per internetą: http://www.onestopsoftwaretesting.com/2009/11/tricky-software-testing-terms.html.

[7] OTS Solutions. "Manual testing v/s automated testing". [Žiūrėta: 2010-01-03]. Prieiga per internetą: http://www.otssolutions.com/doc/whitepapers/manual-testing-vs-automated-testing.pdf.

[8] ARTIFICIAL NEURAL NETWORKS. "A neural network tutorial". [Žiūrėta: 2009-10-12]. Prieiga per internetą: http://www.learnartificialneuralnetworks.com/

[9] Info-Fuzzy Network (IFN). [Žiūrėta: 2009-11-09]. Prieiga per internetą: http://www.ise.bgu.ac.il/faculty/mlast/ifn.htm.

[10] D. Agarwal. "A comparative study of artificial neural networks and info fuzzy networks on their use in software testing". 2004 m. vasara, [Žiūrėta: 2009-10-10]. Prieiga per internetą: http://etd.fcla.edu/SF/SFE0000445/FinalThesis_Deepam.pdf.

[11] S. M. Thampi. "Swarm intelligence". [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://arxiv.org/ftp/arxiv/papers/0910/0910.4116.pdf.

[12] S. Dick, A. Kandel. "Computational intelligence in software quality assurance". (Series in Machine Perception and Artificial Intelligence, vol. 63) Singapore: World Scientific Publishing Company, 2005, 200 p. ISBN 981-256-172-2.

[13] LIACS Natural Computing Group Leiden University. "Swarm intelligence". [Žiūrėta: 2009-12-05]. Prieiga per internetą: http://natcomp.liacs.nl/NC/slides/si.pdf.

[14] Russ Eberhart's Home Page. [Žiūrėta: 2009-12-06]. Prieiga per internetą: http://www.engr.iupui.edu/~eberhart/.

[15] The Tech FAQ. "What is swarm intelligence?" [Žiūrėta: 2009-12-04]. Prieiga per internetą: http://www.tech-faq.com/swarm-intelligence.shtml.

[16] E. V. Oneț. "Particle swarm optimization and genetic algorithms". [Žiūrėta: 2009-12-07]. Prieiga per internetą: http://electroinf.uoradea.ro/reviste%20CSCS/documente/JCSCS_2009/Articole_pdf_JCSCS_C_nr_2/JCSCS_2009_Nr_2_CS_Onet_Particle.pdf.

[17] Dr. K. O. Jones. "Comparison of genetic algorithm and particle swarm optimisation". // International Conference on Computer Systems and Technologies - CompSysTech'2005. [Žiūrėta: 2009-12-07]. Prieiga per internetą: http://ecet.ecs.ru.acad.bg/cst05/Docs/cp/SIII/IIIA.1.pdf.

[18] J. Nenortaitė, R. Butleris. "Improving business rules management through the application of adaptive business intelligence technique". Information Technology and Control, vol. 36, No. 1, p. 21–28, 2009.

Software Testing Type Selection Method, Based on Artificial Neural Networks

Abstract. **Considering the competitive market of the IT companies, a great attention is given to the quality of systems under composition. While creating information systems the stages of analysis, design, and development are as much important as the testing process of the product. The aims and testing done qualitatively and on time let to secure the quality of the product under composition, which has a direct influence to the success of the whole project and to the creation of the company's image in the market. Very often it is difficult to determine what type of testing should be applied for particular component testing, what type of testing would be more effective and would help to save the resources of the company. In this article the problem of the choice of the testing type is analyzed.**

The object of the research: the method of choosing the testing type, based on the application of the artificial intelligence methods. The article gives the comparison of the automatic and manual types of testing in order to distinguish the advantages and disadvantages of the particular testing type. The application of methods of the artificial intelligence in testing processes is reviewed. The suggested method, which is used for choosing type of testing, and is based on application of artificial neural networks is introduced in this article.

Keywords: software testing, artificial neural networks, project manager, information systems.

2 priedas. Pažyma dėl priimto publikuoti mokslinio straipsnio (IVUS 2010)

Tarptautinė magistrantų ir doktorantų konferencija

Informacinė visuomenė ir universitetinės studijos

IVUS 2010

PAŽYMA

DĖL PRIIMTO PUBLIKUOTI STRAIPSNIO

2010 09 08

Kristinos Smilgytės straipsnis *Dirbtiniais neuroniniais tinklais paremtas testavimo tipo pasirinkimo metodas* priimtas publikuoti konferencijos Informacinė visuomenė ir universitetinės studijos medžiagoje „Informacinė technologijos", ISSN 2029-249X.

Konferencijos organizacinio komiteto pirmininkas Tomas Krilavičius

VYTAUTO DIDŽIOJO UNIVERSITETAS
INFORMATIKOS FAKULTETAS
UNIVERSITAS VYTAUTI MAGNI

3 priedas. Straipsnis leidinyje „Proceedings of the 6th International Conference on Hybrid Artificial Intelligence Systems HAIS 2011“

Straipsnio „Artificial Neural Networks Application in SoftwareTesting Selection Method“ pilna versija (8 lapai) pasiekiami www.springerlink.com puslapyje. Detali nuoroda: http://www.springerlink.com/content/978-3-642-21218-5#section=898819&page=1&locus=0

4 priedas. *ANN* apmokymas ir sprendimo prognozės

```
clc;
close all;
clear all;

svoriu_failas = 'svoriai_1HL_6.csv';
FLN = 6;                                % pirmo paslepto sluoksnio neuronu sk.
SLN = 5;                                % antro paslepto sluoksnio neuronu sk.
eks_nr = 1;
k = FLN-1;  % k reiksme skirta svoriu ribai nustatyti
mokymo_f = 'traincgb';

% DUOMENU PASIRUOSIMAS
duom=csvread('duomenys_1000.csv',0,0,[0,0,999,16]);
rez=csvread('duomenys_1000.csv',0,17);
input_weights = csvread(svoriu_failas,0,0,[0,0,k,16]);
input_bias = csvread(svoriu_failas,0,17);

% nustatoma kiek kurios grupes duomenu naudojama, pagal nuskaitoma faila
m = size(find(rez==-1),1); % vnt = -1 rankinis
n = size(find(rez==0),1); % vnt = 0 nesvarbu
k = size(find(rez==1),1); % vnt = 1 automatinis
P = duom;
Tm = [zeros(m,1) zeros(m,1) ones(m,1)]; % rankinis
Tn = [zeros(n,1) ones(n,1) zeros(n,1)]; % nesvarbu
Tk = [ones(k,1) zeros(k,1) zeros(k,1)]; % automatinis
T = [Tm; Tn; Tk];

% SUKURIAMAS DAUGIASLUOKSNIS ANN TINKLAS
% NumberOfHiddenLayerNeurons = FLN;
% NeuronsOfSecondLayer = SLN;
net = newff(P', T',[FLN, SLN],{'tansig', 'tansig', 'purelin'},mokymo_f);
%sukuriamas tinklas
net.divideFcn = 'dividerand';
net.divideParam.trainRatio = 0.7; % nustatoma kokia duomenu dalis naudojama
train
net.divideParam.valRatio = 0.15;  % nustatoma kokia duomenu dalis naudojama
validate
net.divideParam.testRatio = 0.15; % nustatoma kokia duomenu dalis naudojama
test
net.trainParam.lr = 0.05; % nustatomas apmokymo zingsnis (angl. learning
rate)
net.trainParam.epochs = 100; % nustatomas max epochu skaicius
net.trainParam.goal = 0.00001; % nustatomas tikslumas
net.iw{1,1} = input_weights; % priskiriami svoriai nuskaityti is failo
net.b{1} = input_bias;       % priskiriami bias nuskaityti is failo
net.performFcn = 'sse';

% APMOKOMAS ANN
[net, tr, Y, E] = train(net,P',T'); %vykdomas apmokymas
final_weight= net.iw{1,1}; % isimenami svoriai po apmokymo
final_bias = net.b{1};     % isimenami bias po apmokymo
out = sim(net, P'); % imituojamas tinklas
out1 = [zeros(m+n+k,1) zeros(m+n+k,1) zeros(m+n+k,1)];
out2 = zeros(m+n+k,1);
[maks,vieta] = max(out);
% gauti duomenys atverciami i pradini duomenu formata
for i = 1:n+m+k
    if vieta(i) == 1
            out1(i, :) = [1 0 0];
            out2(i)= 1;
```

```
    elseif vieta(i) == 2
            out1(i, :) = [0 1 0];
            out2(i) = 0;
    else
            out1(i, :) = [0 0 1];
            out2(i) = -1;
    end
end

% MOKYMO REZULTATAI
ismoko_sk = 0;
neismoko_sk = 0;
for i = 1:n+m+k
    if rez(i) == out2(i)
        ismoko_sk = ismoko_sk+1;
    else
        neismoko_sk = neismoko_sk+1;
    end
end

fprintf('Naudota duomenu: %d, rankinio %d, nesvarbu %d, automatinis
%d',n+m+k, m, n, k);
fprintf('\nApmokymo  metu ismokta: %d',ismoko_sk);
if neismoko_sk ~= 0
    fprintf('\nApmokymo  metu neismoko: %d\n',neismoko_sk);
end

naujas=csvread('duomenys_tikrinimui.csv',0,0,[0,0,5,16]);
fprintf('\nTikrinamas rinkinys turi buti: rankinis, nesvarbu,
automatinis');
fprintf('\nAksciau buve: rankinis, nesvarbu, automatinis\n');
fprintf('\nTikrinamo duomenu rinkinio prognoze:\n');
outnaujas = sim(net, naujas');
[maksNaujas, vietaNaujas] = max(outnaujas);
stebejimui = zeros(6,1);
for i = 1:6
    if vietaNaujas(i) == 3
        fprintf('%d: rankinis\n',i);
        stebejimui(i)= -1;
    elseif vietaNaujas(i) == 1
        fprintf('%d: automatinis\n',i);
        stebejimui(i) = 1;
    else
        fprintf('%d: nesvarbu\n',i);
        stebejimui(i) = 0;
    end
end
```