VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

STUDIJŲ PROGRAMA: INFORMATIKA

Įrodymo strategijų kūrimas naudojant genetinius algoritmus

Developing proof strategies using genetic algorithms

Baigiamasis magistro darbas

Atliko: Agnė Daujotaitė

VU e-paštas: agne.daujotaite@mif.stud.vu.lt

Vadovas: doc. dr. Haroldas Giedra

Recenzentas: dr. Julius Andrikonis

Vilnius – 2025

Santrauka

Naudojant genetinius algoritmus tyrime siekiama surasti kuo geresnes išvedimo paieškos strategijas automatinio įrodymo sistemai „E“. Populiacijos individai koduojami kaip paieškos strategijos, o jų įvertinimui naudojama TPTP formulių (problemų) bazė. Individų kryžminimas atliekamas vaikams perduodant tėvuose esančias disjunktų įvertinimo funkcijas, o mutavimas – individe atsitiktinai pakeičiant svorio funkcijas arba jų parametrus. Eksperimentų metu paaiškėjo, kad genetiniu algoritmu gauta strategija yra geresnė už sistemos „E“ automatinę strategiją. Geresnis individas gaunamas vykdant evoliucijas su 50, 100 ir 200 individų iki 200 kartų (generacijų). Taip pat buvo pastebėtas dėsningumas, jog didėjant svorio funkcijų kiekiui strategijoje, didėja išspręstų problemų kiekis. Eksperimentai parodė, kad geriausi rezultatai pasiekiami, kai strategijoje sujungiamos keturios ar daugiau svorio funkcijų.

Raktiniai žodžiai: automatinio įrodymo sistema, išvedimo paieškos strategija, disjunkto įvertinimo funkcija, rezoliucijų metodas, genetinis algoritmas.

Summary

Using genetic algorithms, the study aims to find the best possible search strategies for the automatic theorem proving system "E". Individuals of population are coded as search strategies, and they are evaluated using a base of TPTP formulas (problems). Crossing of individuals is performed by transferring clause evaluation functions from parents to children, and mutation is performed by randomly changing weight functions or their parameters in an individual. The experiments revealed that the strategy obtained by the genetic algorithm is better than the automatic strategy of the system "E". A better individual is obtained by performing evolutions with 50, 100 and 200 individuals up to 200 generations. It was also observed that the higher number of weight functions in the strategy increases the number of solved problems. Experiments have shown that the best results are achieved when four or more weight functions are combined in the strategy.

Keywords: prover, proof search heuristics, clause evaluation function, resolution method, genetic algorithm.

Turinys

Įvadas

Logikos mokslo pradžia galėtume laikyti žymaus graikų filosofo Aristotelio darbus. Žmonija nuo senų laikų bando išsiaiškinti ir apibrėžti, kaip žmogus mąsto. Matematinė logika apibrėžia taisykles, kaip iš prielaidų gaunamos išvados ir kaip būtų galima įrodyti, ar formalia kalba aprašytos žinios tam tikroje žinių bazėje yra teisingos ar klaidingos. Nuo 1956 m., kuomet buvo sukurta pirmoji automatinio įrodymo sistema „Logic Theorist“, atsirado galimybė įrodyti žinias, šiuo konkrečiu atveju matematines teoremas, automatiniu būdu, kai kompiuterinė programa imituoja žmogaus samprotavimus [Nor07].

Egzistuoja įvairios logikos, tokios kaip teiginių, pirmos eilės, modalumo, laiko, ir joms yra taikomi įvairūs metodai išvedimo paieškai atlikti. Išvedimo paieška suprantama kaip būdas ar skaičiavimas formulių tapačiam teisingumui nustatyti, t. y. ar formulės teisingos su visomis galimomis interpretacijomis. Pavyzdžiui, tiek teiginių, tiek sudėtingesnėse logikose naudojamas Hilberto tipo skaičiavimas, sekvencinis skaičiavimas, rezoliucijų metodas. Pastarasis skaičiavimas, kurį sugalvojo J. A. Robinson [Rob65], gerokai supaprastina išvedimo paiešką, nes pritaikoma viena rezoliucijos taisyklė vietoj didesnio kiekio taisyklių. Šis metodo bruožas leidžia pilniau automatizuoti įrodymo paiešką. Daug automatinio įrodymo sistemų (angl. provers), tokių kaip „E“, „Vampire“, būtent ir naudoja rezoliucijų metodą. Derėtų atskirti teorijoje įvardijamus įrodymo asistentus (angl. proof assistant), kurie priskiriami interaktyvioms įrodymo sistemoms (angl. interactive theorem proving). Šių asistentų išvedimo paieškoje aktyviai dalyvauja tos sistemos naudotojas, parinkdamas paieškos strategiją bei kitus paieškos parametrus, pavyzdžiui, kiek disjunktų pasirinkti iš aibės. Darbe bus orientuojamasi į automatinio įrodymo sistemas, kurios yra pilniau automatizuotos ir sugebančios autonomiškai priimti sprendimus. Viena iš automatinio įrodymo sistemų yra „E“ programa [Eth]. Ji tinka pirmos eilės predikatų logikai, sukurta vokiečių mokslininko S. Schulz‘o ir yra atvirojo kodo programa. „E“ programa yra naudojama kaip platforma kitoms automatinio įrodymo programoms kurti. Taip pat ji yra laimėjusi ne vieną aukštą apdovanojimą išvedimo paieškos konkursuose, tokiuose kaip „The CADE ATP System Competition“ [Eth]. Remiantis šiais kriterijais „E“ programa bus pasirinkta darbo tyrimui atlikti.

Predikatų logikos formulės gali būti tapačiai teisingos, įvykdomos arba tapačiai klaidingos. Naudojant rezoliucijų metodą galima įrodyti, kad formulė yra tapačiai klaidinga, arba jei nagrinėtume formulę su neigimu, kad ji yra tapačiai teisinga. Atliekant išvedimo paiešką disjunktų aibei yra taikoma rezoliucijos taisyklė. Jos taikymo metu nėra aišku, kokius disjunktus, kokias literas ir kokius termus parinkti. Galima taikyti įvairias parinkimo strategijas.

Automatinio įrodymo sistemoje „E“ naudojamas disjunktų atrankos algoritmas. Disjunktų aibė yra padalijama į apdorotų disjunktų aibę P ir neapdorotų disjunktų aibę U. Pradžioje visi disjunktai yra neapdoroti. Išvedimo paieškos metu disjunktai yra perkeliami iš neapdorotų

disjunktų aibės *U* į apdorotų disjunktų aibę *P*, įvertinant *P* aibėje esančių disjunktų ryšius ir galimas kombinacijas.

Siekiant rasti efektyvią išvedimo paieškos strategiją, būtų galima pritaikyti genetinius algoritmus. Išvedimo strategiją galima užkoduoti kaip populiacijos individą. Taikant kryžminimo ir mutavimo operatorius gaunamos naujos individų (strategijų) kartos. Tokiu evoliucijos principu galima surasti geriausią išvedimo paieškos strategiją [SS15]. Genetiniai algoritmai suteikia plačias galimybes naujų individų kartų sukūrimui. Pavyzdžiui, galima rinktis individų atrankos strategiją, kai individai parenkami atsitiktiniu būdu ar pagal priskirtą reitingą, pasirinkti kryžminimo operatorių, kuomet gali būti atliekamas chromosomų kryžminimas viename ar keliuose taškuose, bei pasirinkti mutacijos operatorių, kai vienas su kitu sukeičiami pasirinkti du ar daugiau genų chromosomoje arba kai genai sukeičiami atsitiktiniu būdu.

Automatinio įrodymo sistemos ir jų panaudojimas yra aktualus šiandienos technologijų vystymuisi. Akivaizdžiausias šių sistemų pritaikymas matematikos srityje, kai tikrinamas matematikos teorijų neprieštaringumas. Sistemos plačiai naudojamos aparatinės ir programinės įrangos verifikavimui, tinklo saugumui užtikrinti bei dirbtinio intelekto pritaikymui robotikoje. Pavyzdžiui, kuriant mikroprocesorius būtina tikrinti jų architektūrą, kadangi praleistos klaidos gali lemti sunkias finansines pasekmes. Raketos „Ariane 5“ sprogimo praėjus 40 sekundžių po paleidimo priežastis buvo programinės įrangos klaida, kuri būtų pašalinta, jei būtų panaudotos verifikavimo sistemos [Dou00]. Pastaruoju metu automatinio įrodymo sistemos pritaikomos ir generatyvinėse dirbtinio intelekto sistemose, tokiose kaip Amazon Code Guru1, AlphaProof2, AlphaGeometry išvesties verifikavimui ir paaiškinimui [Mck24]. Ateityje automatinio įrodymo sistemos turės svarbų vaidmenį verifikuojant statistinių dirbtinio intelekto metodų išvadas.

Darbo tikslas

Pritaikyti genetinius algoritmus išvedimo paieškos strategijų kūrimui automatinio įrodymo sistemai „E“.

Darbo uždaviniai

Tyrimo tikslui pasiekti keliami uždaviniai:

- Sukurti individų kodavimo, mutavimo, kryžminimo ir atrankos genetiniame algoritme modelį.
- Sudaryti skirtingo sudėtingumo problemų (formulių) imtis, kurios bus naudojamos individų tinkamumo įvertinimui.
- Realizuoti individo įvertinimo funkciją, kuri naudoja problemų imtį ir automatinio įrodymo sistemą „E“.
- Realizuoti genetinio algoritmo kryžminimo, mutavimo ir atrankos operatorius.

- Realizuoti programą, kuri modeliuoja evoliuciją naudojant genetinį algoritmą.
- Atlikti evoliucijų eksperimentus ir nustatyti geriausias strategijas.
- Surasti ir aprašyti kitų autorių išskirtas geriausias strategijas.
- Palyginti genetiniu algoritmu gautą geriausią strategiją su kitų autorių strategijomis.

1. Predikatų logika

Šiame skyriuje apžvelgsime predikatų logikos sintaksę ir semantiką bei detaliau susipažinsime su rezoliucijų metodu. Šis skyrius aktualus norint suprasti, kas yra išvedimo paieška ir kaip ji veikia.

1.1 Sintaksė

Logikos mokslo teorijoje apibrėžta daug logikos teorijų. Jei teiginių logika nagrinėja teiginius su loginėmis operacijomis ir iš jų sudarius formules galima nustatyti samprotavimų teisingumą arba klaidingumą, tai predikatų logikoje atsiranda predikatai – teiginių požymiai arba teiginiai su vienu arba keliais parametrais.

1.1.1 Apibrėžimas. N-viečiu predikatu aibėje A vadiname vienareikšmę n argumentų funkciją, kurios apibrėžimo sritis yra aibė A, o reikšmių sritis – aibė, sudaryta tik iš dviejų elementų – tiesa ir klaida $\{t, k\}$ [Nor07].

Predikato parametrai arba argumentai vadinami individiniais kintamaisiais (x, y, z, ..., x_1, x_2, x_3, ...), predikatiniai kintamieji (kai predikato funkcija nekonkretizuota) žymimi didžiosiomis raidėmis $P_i(x_1, x_2, x_3, ...)$.

Predikatų logikoje kaip ir teiginių logikoje naudojami loginiai operatoriai – neigimas, konjunkcija, disjunkcija ir implikacija (žymima $\neg$, $\wedge$, $\vee$, $\rightarrow$), tačiau papildomai atsiranda dar du operatoriai – kvantoriai, t.y. loginiai veiksmai, kiekybiškai apibūdinantys objektų sritį ir naudojami kartu su individiniais kintamaisiais. Vienas jų – bendrumo kvantorius $\forall$, pavyzdžiui, formulėje $\forall x(P(x) \vee Q(x))$ būtų skaitomas kaip „su kiekvienu x formulė $(P(x) \vee Q(x))$ yra teisinga“. Bei egzistavimo kvantorius $\exists$, pavyzdžiui, formulėje $\exists x(P(x) \vee Q(x))$ būtų skaitomas kaip „egzistuoja x, su kuriuo formulė $(P(x) \vee Q(x))$ teisinga“.

Predikatinis kintamasis $P(x)$ gali simbolizuoti bet kurį teiginį su parametrais, kai funkcija nėra apibrėžta. Predikatų logikos formulėse įeitimi vadiname individinio kintamojo aptikimą formulėje.

1.1.2 Apibrėžimas. Kintamasis x formulėje yra laisvas, jei formulėje F yra bent viena laisva x įeitis, t. y. ji nepatenka į kvantoriaus ($\forall x$ arba $\exists x$) veikimo sritį.

Formulės predikatų logikoje gali būti sudėtingesnės į jas įtraukiant funkcijas. Funkcijos žymimos $f_i(x_1, ..., x_n)$. Predikatų logikoje naudojamas ir terminas termas.

1.1.3 Apibrėžimas. Termu gali būti individinė konstanta (*a, b, Tomas, Viktorija, 1, 5, ...*), individinis kintamasis (x, y, z, ...) ir jei $t_1, ..., t_n$ yra termai, o $f(x_1, ..., x_n)$ yra funkcinis simbolis, tai $f(t_1, ..., t_n)$ taip pat yra termas [Nor07].

1.1.4 Apibrėžimas. Predikatų logikos formulių aibė $\mathscr{F}$ yra tokia pati mažiausia aibė, kad:

- Jei $P(x_1, ..., x_n)$ yra predikatinis kintamasis, o $t_1, ..., t_n$ yra termai, tai $P(t_1, ..., t_n)$ priklauso aibei $\mathscr{F}$. Ši formulė dar vadinama atomine formule.
- Jei $F, G \in \mathscr{F}$, tai $\neg F \in \mathscr{F}$, $F \wedge G \in \mathscr{F}$, $F \vee G \in \mathscr{F}$, $F \rightarrow G \in \mathscr{F}$.
- Jei $F \in \mathscr{F}$, o x – formulės F laisvasis kintamasis, tai $\forall xF \in \mathscr{F}$, $\exists xF \in \mathscr{F}$.

Formulių pavyzdžiai: $\forall x(P(f(b)) \vee Q(x))$, $\forall x \exists y \forall z(P(x, y, z) \vee Q(y, x, x))$.

1.2 Semantika

Norint nustatyti kokios nors predikatų logikos formulės teisingumo vertę, reikia konkretizuoti tos formulės individinių kintamųjų apibrėžimo aibę, predikatinius kintamuosius, laisvuosius individinius kintamuosius ir funkcinius simbolius. Šie elementai konkretizuojami naudojant struktūras.

1.2.1 Apibrėžimas. Sąrašas <M, R, C, G> vadinamas formulę F atitinkančia struktūra S, kur:

- M – kuri nors netuščia aibė (individinių kintamųjų apibrėžimo aibė).
- R – predikatų aibė, atitinkanti formulės F predikatinius kintamuosius.
- C – konstantų aibė (elementai iš aibės M), atitinkanti formulės F laisvuosius individinius kintamuosius.
- G – funkcijų aibė, atitinkanti formulės F funkcinius simbolius.

Formulė F yra teisinga struktūroje S, jei pakeitę formulėje F predikatinius kintamuosius juos atitinkančiais predikatais, o laisvuosius kintamuosius – juos atitinkančiais elementais iš struktūros S, gauname teisingą formulę. Pavyzdžiui, turime struktūrą S=<N, R, C, G>, kur N – natūraliųjų skaičių aibė, R={$Q(x, y)$}, C= ∅, G=∅, $Q(x, y)$=t tada ir tik tada, kai $x=y$. Norėdami sužinoti, ar formulė $\forall x \forall y \forall z((P(x, y) \wedge P(y, z)) \rightarrow P(x, z))$ yra teisinga struktūroje S, turime jos elementus pakeisti struktūros S atitinkamais elementais. Įsistatę gauname $\forall x \forall y \forall z(((x=y) \wedge (y=z)) \rightarrow (x=z)))$, vadinasi formulė yra teisinga struktūroje S, nes kiekvienam x, y ir z, jei x lygus y ir y lygus z, tai x lygus z.

Formulė gali būti įvykdoma, tapačiai teisinga ar tapačiai klaidinga.

1.2.2 Apibrėžimas. Formulė vadinama įvykdoma, jei egzistuoja struktūra, kurioje ji yra teisinga.

1.2.3 Apibrėžimas. Formulė vadinama tapačiai teisinga, jei ji teisinga bet kurioje struktūroje.

1.2.4 Apibrėžimas. Formulė vadinama tapačiai klaidinga, jei ji klaidinga bet kurioje struktūroje.

1.3 Skaičiavimai

Paprastais nesudėtingų formulių atvejais gan lengva nustatyti, ar formulė yra tapačiai teisinga semantiškai jas analizuojant. Sudėtingesniems atvejams mokslininkai yra sukūrę metodus arba skaičiavimus, kurių pagalba galima nustatyti formulių tapatų teisingumą. Būtų galima paminėti tokius metodus kaip Hilberto tipo skaičiavimas, sekvencinis skaičiavimas, semantiniai

medžiai, lentelių metodas ir rezoliucijų metodas. Tai nėra baigtinis metodų sąrašas. Rezoliucijų metodas išsiskiria tuo, kad yra dažnai naudojamas automatinio įrodymo sistemose.

1.3.1 Rezoliucijų metodas

Rezoliucijų metodas skirtas įrodyti, kad formulė yra tapačiai klaidinga. Šį metodą sugalvojo J. A. Robinson 1965 m. [Rob65]. Kad būtų galima taikyti rezoliucijų metodą, reika sudaryti formulės disjunktų aibę. Pirmiausia formulė transformuojama į normaliąją priešdėlinę formą (toliau NPF). Tai reiškia, kad kiekviena formulė gali būti pertvarkoma taip, jog kvantoriai būtų tik formulės priekyje, o tai atliekama pritaikius ekvivalentumo pertvarkymus. Toliau formulė skolemizuojama, t. y. eliminuojami egzistavimo kvantoriai į egzistavimo kvantoriumi suvaržytus kintamuosius įstatant funkcinius simbolius. Tada išbraukiami bendrumo kvantoriai. Formulė transformuojama į normaliąją konjunkcinę formą (NKF) ir paskutiniu veiksmu sudaroma disjunktų aibė.

1.3.1.1 Apibrėžimas. Litera vadinamas predikatinis kintamasis $P(t_1, ..., t_n)$ arba jo neigimas $\neg P(t_1, ..., t_n)$, kur t_i yra termai.

1.3.1.2 Apibrėžimas. Disjunktu vadinama literų disjunkcija $l_1 \vee l_2 \vee ... \vee l_n$.

1.3.1.3 Apibrėžimas. Formulės F normaliąja konjunkcine forma vadinama jai ekvivalenti formulė $\wedge^{m}_{i=1} D_i$, kur D_i yra disjunktai.

Rezoliucijų metode naudojamas keitinys, t. y. reiškinys $(t_1/x_1, t_2/x_2, ..., t_n/x_n)$, kur t_i yra termai, o x_i individiniai kintamieji. Keitinys σ vadinamas reiškinių R ir Z unifikatoriumi, jei $R\sigma=Z\sigma$.

Taikant rezoliucijos metodą naudojama pagrindinė rezoliucijos taisyklė:

$$\frac{Di\sigma \quad Dj\sigma}{D\sigma}$$

Šioje taisyklėje D_i, D_j ir D yra disjunktai. Taisyklė taikoma prielaidoms, kuriose galima rasti tokias dvi literas $P(t'_1, ..., t'_n)$ ir $\neg P(t''_1, ..., t''_n)$, kad σ yra $P(t'_1, ... t'_n)$ ir $P(t''_1, ... t''_n)$ unifikatorius. $D\sigma$ gaunama išbraukus $P(t'_1, ... t'_n)\sigma$ ir $\neg P(t''_1, ... t''_n)\sigma$ iš prielaidų ir apjungus gautąsias formules disjunkcija. Jei išvadoje yra dvi vienodos literos ar daugiau, paliekama tik viena. Jei iš formulės F disjunktų aibės S išvedamas tuščias disjunktas, tai formulė F yra tapačiai klaidinga.

Formulės išvedimas naudojant rezoliucijų metodą. Tarkime norime įrodyti, kad formulė $\neg((\forall x(Z(x) \rightarrow \neg V(x)) \wedge \exists x A(x) \wedge \forall x(A(x) \rightarrow \neg\neg V(x))) \rightarrow \exists x(A(x) \wedge \neg Z(x)))$ yra tapačiai klaidinga. Atlikus formulės pakeitimus pagal aukščiau pateiktus veiksmus, gauname disjunktų aibę $S=\{\neg Z(x) \vee \neg V(x), A(a), \neg A(x) \vee V(x), \neg A(x) \vee Z(x)\}$. Turėdami disjunktų aibę atiekame išvedimą.

$$\frac{A(a) \quad \neg A(x) \vee V(x)}{V(a)} \sigma = \{a/x\}$$

$$\frac{V(a) \quad \neg Z(x) \vee \neg V(x)}{\neg Z(a)} \sigma = \{a/x\}$$

$$\frac{\neg Z(a) \quad \neg A(x) \vee Z(x)}{\neg A(a)}\ \sigma = \{a/x\}$$

$$\frac{A(a) \quad \neg A(a)}{\square}\ \sigma = \emptyset$$

Renkamės disjunktus, kuriems galėtume pritaikyti keitinį ir pašalinti literas, pavyzdžiui, $A(a)$, $\neg A(x) \vee V(x)$. Pritaikę rezoliucijos taisyklę su keitiniu $\sigma=\{a/x\}$, gauname $V(a)$. Turėdami naują disjunktą $V(a)$, žiūrime kokį kitą disjunktą galėtume parinkti, kad gautume $\neg Z(a)$. Tad naudojame tą patį keitinį $\sigma=\{a/x\}$ ir pritaikome disjunktams $V(a)$ ir $\neg Z(x) \vee \neg V(x)$. Pritaikius taisyklę gaunamas naujas disjunktas $\neg Z(a)$. Vėlgi turėdami naują disjunktą $\neg Z(a)$ imame kitą disjunktą $\neg A(x) \vee Z(x)$ ir su keitiniu $\sigma=\{a/x\}$, gauname $\neg A(a)$. Paskutiniame išvedimo žingsnyje turime disjunktus: naująjį $\neg A(a)$ ir iš disjunktų aibės $A(a)$, pritaikę taisyklę gauname tuščią disjunktą (unifikatorius šiuo atveju būtų tuščias keitinys). Šioje išvedimo paieškoje buvo ieškomi disjunktai ir pasirinkta tokia strategija, kaip būtų galima gauti tuščią disjunktą. Pirmiausia buvo pasirinkti disjunktai su viena litera. Atliekant išvedimą galima pasirinkti, kaip valdyti paiešką – kokius disjunktus pasirinkti pirmiausia, į kokias literas atkreipti dėmesį, kokius termus pasirinkti keitiniams, kokiu eiliškumu rinktis disjunktus atlikus kiekvieną išvedimo žingsnį ir dar yra daugybė kitų pasirinkimo variantų. Sudėtingesnėse ir didesnėse disjunktų aibėse strategijų, kurį disjunktą pasirinkti pirmiausia, padaugėja, be to svarbu ir kokį disjunktą pasirinkus išvedimo paieška truktų kuo trumpiau, t. y. reikėtų atlikti kuo mažiau išvedimo žingsnių.

Šiame tyrime naudojama automatinio išvedimo sistema „E“, kuri remiasi praplėstu rezoliucijų metodu [Sch24]. Praplėstas rezoliucijų metodas dar įvardijamas kaip sotusis (angl. saturating) skaičiavimas. Taikant išvedimo taisykles, disjunktų aibė yra prisotinama iki pertekliaus, kai nebegalima išvesti naujo neperteklinio disjunkto ir procesas sustoja. Arba procesas sustoja, kai išvedamas tuščias disjunktas. Skaičiavimas taip pat apima supaprastinimo taisykles, kurios leidžia kai kuriuos disjunktus pakeisti paprastesniais, mažesniais disjunktais arba pašalinti perteklinius disjunktus. Pavyzdžiui, perrašymo (angl. rewriting) taisyklė pakeičia termus į mažesnius termus, sutraukimo (angl. subsumption) taisyklė pašalina disjunktą, kuris išvestas iš bendresnio pobūdžio disjunkto [SM16].

2. Automatinio įrodymo sistema „E“

Rezoliucijų metodas taikomas įvairiose automatinio įrodymo sistemose (AĮS). Viena tokių sistemų yra programa „E“, sukurta 1999 vokiečių mokslininko Stephan Schulz [Eth]. AĮS „E“ skirta formulės (problemos) įvertinimui: ar formulė yra tapačiai teisinga (angl. theorem), tapačiai klaidinga (angl. unsatisfiable) ar įvykdoma (angl. satisfiable). Sistema kartais sugeba įvertinti, kartais ne, ir tas patikrinimo rezultatas didžiąja dalimi priklauso nuo įrodymo strategijos. Naudotojui yra suteikiama galimybė konfigūruoti strategijos parametrus – išvedimo paiešką [Sch24].

2.1 Algoritmas

Sistema „E“ naudoja „Given-clause algorithm“, kurio suspaprastintas variantas pateiktas 1 paveiksle [Sch06].

```
# Input:  Axioms in U, P is empty
while U ≠ ∅ begin
    # First check for propositional unsat (Section 3.3.1)
    if prop_trigger(U,P)
        if prop_unsat_check(U,P)
            SUCCESS, Proof found
    c := select(U)
    U := U \ {c}
    # Apply (RN), (RP), (NS), (PS), (CLC), (DR), (DD), (DE)
    simplify(c,P)
    # Apply (CS), (ES), (TD)
    if c is trivial or subsumed by P then
        # Delete/ignore c
    else if c is the empty clause then
        # Success:  Proof found
        stop
    else
        T := ∅ # Temporary clause set
        foreach p ∈ P do
            if p can be simplified with c
                P := P \ {p}
                U := U \ {d|d  is direct descendant of  p}
                T := T ∪ {p}
            done
        end
        P := P ∪ {c}
        T := T ∪ e-resolvents(c) # (ER)
        T := T ∪ e-factors(c) # (EF)
        T := T ∪ paramodulants(c,P) # (SN), (SP)
        T' := {}
        foreach p ∈ T do
            # Apply efficiently implemented subset of (RN),
            # (RP), (NS), (PS), (CLC), (DR), (DD), (DE)
            p := cheap_simplify(p, P)
            # Apply (TD) or efficient approximation of it
            if p is trivial
                # Delete/ignore p
            else
                T' := T' ∪ cheap_simplify(p, P)
            fi
        end
        U := U ∪ T'
    fi
end
# Failure:  Initial U is satisfiable, P describes model
```

1 pav. *„Given-clause algorithm“ supaprastintas variantas*

https://wwwlehre.dhbw-stuttgart.de/~sschulz/WORK/E_DOWNLOAD/V_3.1/eprover.pdf

Išvedimo paieška pradedama disjunktų aibę padalijant į du poaibius – apdorotų disjunktų aibę *P* (angl. processed) ir neapdorotų disjunktų aibę *U* (angl. unprocessed). Išvedimo pradžioje aibė *P* yra tuščia, o visi nagrinėjamos formulės (problemos) disjunktai yra aibėje *U*. Algoritmas parenka naują disjunktą iš aibės *U* ir toliau atlieka visus galimus išvedimus su šiuo ir aibės *P* disjunktais. Naujai gauti disjunktai pridedami prie aibės *U*, o nagrinėtas disjunktas (iš aibės *U*) perkeliamas į aibę *P*. Procesas kartojamas ir sustoja, kuomet išvedamas tuščias disjunktas arba nėra galimybės toliau atlikti išvedimą – kai pasibaigia disjunktai aibėje *U* (nerandamas įrodymas) arba pasiektas laiko ar atminties išteklių limitas.

Algoritmas „Given-clause“ turi du variantus: „Discount loop“ ir „Otter loop“ [SM16]. Programoje „E“ naudojamas „Discount loop“ variantas, kuriame supaprastinimas (angl. back-simplification) taikomas tik apdorojimui parinktiems disjunktams, o neapdoroti disjunktai yra pasyvioje būsenoje. „Otter loop“ variante supaprastinimui naudojami tiek apdoroti, tiek neapdoroti disjunktai. Algoritmo variantai skiriasi algoritmo iteracijų kiekiu ir trukme. Su „Otter loop“ variantu išvedimo paieškos kiekviena iteracija užtrunka ilgiau, tačiau vykdoma mažiau kartų, o su „Discount loop“ variantu kiekviena pagrindinio ciklo iteracija trunka trumpiau, tačiau gali užtrukti, kol bus pasirinkti reikiami disjunktai apdorojimui iš neapdorotų disjunktų aibės. „E“ programoje naudojamas „Discount loop“, nes juo galima labiau kontroliuoti disjunkto parinkimą – kiekviena iteracija apima mažesnę išvedimo paieškos dalį.

2.2 Išvedimo paieškos strategijos

Išvedimo paiešką galima kontroliuoti pagal tris svarbiausius parametrus: termų, literų ir disjunktų parinkimo eiliškumą. Šis darbas apima išvedimo paieškos strategijų analizę pagal trečiąjį iš išvardintų parametrų – disjunkto parinkimo būdą. Išvedimo paieškos proceso metu disjunktai parenkami naudojant atrankos schemą „round robin scheme“ [Sch24] ir disjunktų įvertinimo funkciją, sudarytą iš prioriteto ir svorio funkcijų.

Prioriteto funkcijos apibrėžia neapdorotų disjunktų aibės padalijimą į kelias aibes. Kiekviena prioriteto funkcija dalina disjunktus neapdorotų disjunktų aibėje taip, kad tam tikros klasės disjunktams yra skiriamas prioritetas nepaisant svorio funkcijos suteikto įverčio. Prioritetų funkcijų sąrašas nėra baigtinis ir jį sistemoje „E“ galima pildyti naujomis funkcijomis. 1 lentelėje pateikiama sistemos „E“ keletas naudojamų pagrindinių prioriteto funkcijų [Sch24].

1 lentelė. *Sistemos „E“ pagrindinės prioriteto funkcijos*

Prioriteto funkcijos pavadinimas	**Aprašas**
PreferGround	Pirmenybė teikiama disjunktams be kintamųjų (su konstantomis ir funkcijomis).

PreferNonGround	Pirmenybė teikiama disjunktams su kintamaisiais.
PreferGoals	Pirmenybė teikiama disjunktams, kuriuose visos literos paneigtos.
PreferNonGoals	Pirmenybė teikiama disjunktams, kuriuose yra bent viena litera be neigimo.
PreferUnits	Pirmenybė teikiama disjunktams, sudarytiems tik iš vienos literos.
PreferNonUnits	Pirmenybė teikiama disjunktams, sudarytiems iš daugiau nei vienos literos.
PreferGroundGoals	Pirmenybė teikiama disjunktams be kintamųjų, kuriuose visos literos paneigtos.
PreferUnitGroundGoals	Pirmenybė teikiama disjunktams, sudarytiems tik iš vienos paneigtos literos be kintamųjų.
PreferProcessed	Pirmenybė teikiama disjunktams, kurie jau buvo panaudoti ir buvo pašalinti iš panaudotų disjunktų aibės.
PreferNew	Pirmenybė teikiama naujiems disjunktams.
PreferHorn	Pirmenybė teikiama disjunktams, kuriuose yra ne daugiau nei viena litera be neigimo.
PreferNonHorn	Pirmenybė teikiama disjunktams, kuriuose yra daugiau nei viena litera be neigimo.
ConstPrio	Visiems disjunktams suteikiamas vienodas prioritetas.
ByLiteralNumber	Prioritetas suteikiamas pagal literų kiekį, pavyzdžiui, pirmenybė teikiama disjunktui su mažesniu literų kiekiu.
ByNegLitDist	Maišyta funkcija, kurioje pirmenybė teikiama disjunktams be kintamųjų. Tarp disjunktų, kurių visos literos paneigtos, pirmenybė teikiama tiems disjunktams, kurie turi mažiau literų ir yra be kintamųjų. Pavyzdžiui, disjunktui už literą be kintamojo prioritetas padidinamas vienetu, o už literą su kintamuoju – trimis. Disjunktai su mažesne prioriteto reikšme yra parenkami pirmiau nei disjunktai su didesne prioriteto reikšme.
ByGoalDifficulty	Maišyta funkcija, kurioje pirmenybė teikiama disjunktams be kintamųjų ir atsižvelgiama į jų sudėtingumą. Pirmiausia pasirenkami disjunktai su viena litera be kintamųjų ir su paneigtomis literomis, po

	to su viena litera be kintamųjų, tada be kintamųjų, ir tuomet visos likusios.
ByCreationDate	Pagal disjunkto sukūrimo laiką disjunktui yra priskiriamas prioritetas. Ši funkcija atitinka FIFO (First In First Out) metodą.

Disjunkto įvertinimo funkciją sudaro svorio funkcija, kurios sudėtyje yra prioriteto funkcija ir keletas parametrų [MS25]. Svorio funkcija įvertina disjunktą – t.y. atsižvelgia į simbolius ir jų tipą disjunktuose. Svorio funkcija kaip argumentą paima disjunktą ir grąžina jo svorio skaitinę reikšmę: disjunktas su mažesniu svoriu turės pirmenybę prieš disjunktą su didesniu svoriu. Kiekviena svorio funkcija turi unikalių tai svorio funkcijai parametrų aibę, kurie pateikiami po prioriteto funkcijos. Skaitinės parametrų reikšmės nurodo tų parametrų svorius. Pavyzdžiui, antrame paveikslėlyje pateiktame disjunkto įvertinimo funkcijos pavyzdyje, pirmo parametro – funkcinių simbolių svorio – reikšmė yra 3, o antro parametro – kintamųjų svorio – 2:

2 pav. *Disjunkto įvertinimo funkcijos pavyzdys*
https://easychair.org/publications/paper/PXgt/open

Sistemoje „E“ yra apibrėžtos aštuonios svorio funkcijos. Tačiau šis sąrašas vėlgi nėra baigtinis ir gali būti pildomas. Toliau pateikiama svorio funkcijos sintaksė.

<prio-fun> *::=* *PreferGround* ||
PreferNonGround ||
PreferGoals ||
PreferNonGoals ||
PreferUnits ||
PreferNonUnits ||
PreferGroundGoals ||
PreferUnitGroundGoals ||
PreferProcessed ||
PreferNew ||
PreferHorn ||
PreferNonHorn ||

ConstPrio ||
ByLiteralNumber ||
ByNegLitDist ||
ByGoalDifficulty ||
ByCreationDate

<*weight-fun*> ::= *Clauseweight ‘(‘<prio-fun>‘, <int>, <int>, <float>‘)‘* ||
Refinedweight ‘(‘<prio-fun>‘, <int>, <int>, <float>, <float> , <float> ‘)‘ ||
Orientweight ‘(‘<prio-fun>‘, <int>, <int>, <float>, <float> , <float> ‘)‘ ||
Simweight ‘(‘<prio-fun>‘, <float>, <float> , <float>, <float> ‘)‘ ||
FIFOweight ‘(‘<prio-fun>‘)‘ ||
FIFOweight ‘(‘<prio-fun>‘)‘ ||
Funweight ‘(‘<prio-fun>‘, <int>, <int>, <float>, <float> , <float>, <fun>, <posint>) ‘)‘* ||
SymOffsetWeight ‘(‘<prio-fun>‘, <int>, <int>, <float>, <float> , <float>, <fun>, <int>) ‘)‘ ‘)‘*

2 lentelėje pateikiami svorio funkcijų aprašai.

2 lentelė. *Sistemos „E“ pagrindinės svorio funkcijos*

Svorio funkcija	**Aprašas**
Clauseweight(prio, fweight, vweigth, pos_mult)	Funkcija remiasi simbolių skaičiavimu. Kintamieji yra skaičiuojami su svoriu *vweight*, funkciniai simboliai su svoriu *fweight*. Literų be neigimo svoris dauginamas iš parametro *pos_mult* prieš susumuojant su galutiniu svoriu.
Refinedweight(prio, fweigth, vweight, term_pen, lit_pen, pos_mult)	Funkcija yra panaši į Clauseweight funkciją, tačiau skiriasi tuo, jog papildomai atsižvelgia į termų įvertinimą. Termo svoris, kuris yra maksimalus literoje, padauginamas iš *term_pen*, o maksimalių literų svoris yra padauginamas iš *lit_pen*.
Simweight(prio, equal_weigth, vv_clash, vt_clash, tt_clash)	Funkcija suteikia mažą svorį literoms, kurios turi panašius termus.
FIFOWeight(prio)	Funkcijos priskiriamas svoris disjunktams monotoniškai didėja. Taikoma FIFO (First In First Out) strategija.
LIFOWeight(prio)	Funkcijos priskiriamas svoris disjunktams monotoniškai mažėja. Taikoma LIFO (Last In First Out) strategija.

FunWeight(prio, fweigth, vweight, term_pen, lit_pen, pos_mult, fun:fweight ...)	Funkcija yra panaši į Refinedweight funkciją, tačiau skiriasi tuo, kad papildomi svoriai (neneigiami sveikieji skaičiai) suteikiami konkrečiai įvardintiems funkciniams simboliams *fun: fweigth.*
SymOffsetWeight(prio, fweigth, vweight, term_pen, lit_pen, pos_mult, fun:fweight ...)	Funkcija yra panaši į FunWeight funkciją, tačiau skiriasi tuo, kad papildomi svoriai (sveikieji skaičiai) suteikiami konkrečiai įvardintiems funkciniams simboliams *fun: fweigth.*

Išvedimo paieškos metu disjunktų įvertinimo funkcijos naudojamos disjunktų atrinkimui remiantis išvedimo strategija.

2.2.1 Apibrėžimas. Strategija suprantama kaip tam tikra tvarka išdėstytas disjunktų įvertinimo funkcijų sąrašas.

Kiekviena disjunkto įvertinimo funkcija turi skaitinę (angl. integer) svorio reikšmę, kuri nurodo, kiek disjunktų turi būti pasirinkta pagal šią disjunktų įvertinimo funkciją prieš pereinant prie kitos disjunktų įvertinimo funkcijos (arba grįžtant prie pirmos disjunktų įvertinimo funkcijos po paskutinės disjunktų įvertinimo funkcijos strategijoje). Žemiau pateiktame programos „E“ išvedimo strategijos pavyzdyje prieš kiekvieną disjunktų įvertinimo funkciją eilutės pradžioje yra jos skaitinis svoris: pirmoji disjunktų įvertinimo funkcija *FunWeight* būtų panaudota du kartus parenkant disjunktą, antroji disjunktų įvertinimo funkcija *FIFOWeight* būtų panaudota vieną kartą, trečioji *Clauseweight* būtų panaudota aštuonis kartus, ketvirtoji *Refinedweight* – tris kartus. Po 2 + 1 + 8 + 3 = 14 kartų disjunktų atrinkimo, viskas pasikartotų vėl iš naujo tokiu pačiu eiliškumu.

Išvedimo strategijos pavyzdys:

```
- H‘ (
    2*FunWeight(ConstPrio,2,1,1,1,1,f:2,g:5,j:3,a:0,b:0)
    1*FIFOWeight(PreferProcessed),
    8*Clauseweight(PreferUnitGroundGoals,1,1,0.49),
    3*Refinedweight(PreferGoals,2,4,7,4.94,6.55),
 )‘
```

Sistema „E“ leidžia lanksčiai konfigūruoti išvedimo strategijas nurodant disjunktų parinkimo prioritetų ir svorio funkcijas. Šis tyrimas skirtas būtent šių strategijų kūrimui ir analizei.

2.3 Kitų autorių atlikti išvedimo paieškos strategijų tyrimai

Šiame skyriuje siekiama apžvelgti išvedimo paieškos strategijų sistemai „E“ tyrimus – išsiaiškinti, kokia išvedimų paieškos ir automatinio įrodymo sistemos „E“ situacija mokslinių tyrimų lauke, kokie darbai atlikti išvedimo paieškos strategijų atžvilgiu, su kokiomis problemomis susiduriama, ir kaip iškilusios problemos yra sprendžiamos.

Sistema „E“ išvedimo paieškai leidžia taikyti ir konstruoti strategijas. Daugiausiai jų aprašo ir tyrimų su jomis atlieka pats sistemos kūrėjas S. Schulz. Sistemoje „E“ yra automatinis režimas, kuris atrenka tinkamiausias strategijas. Dauguma mokslininkų naudodamiesi programos „E“ automatiniu režimu ir taikydami mašininio mokymosi metodus tobulina išvedimo paieškos efektyvumą atrenkant kuo tinkamesnius disjunktus.

Apžvelgtoje literatūroje aptiktas disjunktų parinkimo strategijų klasifikavimas ir jų tarpusavio palyginimas efektyvumo atžvilgiu, kuris aptariamas kitame poskyryje. Labai daug tyrimų atliekama programos „E“ disjunktų atrinkimo strategijų paieškai, taikant mašininį mokymąsi – įvairius metodus, neuroninius tinklus, sustiprintą mokymąsi.

2.3.1 Išvedimo strategijų klasifikavimas

S. Schulz kartu su M. Mohrmann atliko tyrimą [SM16], kuriame palygino skirtingas disjunktų parinkimo strategijas. Šiuo tyrimu autoriai pirmą kartą bandė sistemiškai įvertinti skirtingas strategijas disjunktų atrinkimo atžvilgiu. Programa „E“ turi daugiau nei 200 strategijų („E“ 1.6 versijoje).

S. Schulz skirsto strategijas į klasikines (paprastas) ir modernias (sujungtas sudėtingesnes strategijas). Autorius paprastomis įvardina tokias strategijas, kurios apima vieną arba dvi disjunktų įvertinimo funkcijas. Sudėtingos strategijos apima daugiau nei dvi disjunktų įvertinimo funkcijas, turi daugiau parametrų ir jose yra į išvadą orientuotos funkcijos. S. Schulz išskiria šias pagrindines klasikines disjunktų įvertinimo strategijas:

- FIFO (angl. first in first out),
- simbolių skaičiavimo SC (angl. symbol counting),
- apimančias rikiavimą (angl. ordering-aware),
- į išvadą orientuotas funkcijas GD (angl. goal-directed).

FIFO strategijos visada pirmenybę teikia seniausiam neapdorotam disjunktui. SC strategijos skaičiuoja disjunkto simbolius ir pirmenybę teikia mažiausiai simbolių turintiems disjunktams. Ordering-aware tipo strategijos – tai strategijos panašios į SC strategijas, kuriose pirmenybė teikiama disjunktams su maksimaliais termais ir maksimaliomis literomis. Ordering-aware stategijos pavyzdys yra RW (angl. refined weight) funkcija, kuri sudaugina maksimalių termų ir maksimalių literų svorį su sistemos naudotojo parinktais nekintančiais dydžiais. GD tipo strategijos suteikia mažesnius svorius simboliams, esantiems išvadoje, ir didesnį svorį kitiems simboliams – tokiu būdu pirmenybė teikiama disjunktams, kurie labiau tikėtina bus panaudoti išvedimui. Moderniosios strategijos apjungia mažiausio svorio ir seniausio disjunkto išvedimo paiešką.

Autorių atliktame tyrime [SM16] buvo ištestuota 40 strategijų ir iš jų atrinkta 14, kurios apima visas pagrindines disjunktų įvertinimo funkcijas bei jų kombinacijas su FIFO strategija. Strategijos buvo įvertintos pagal šiuos kriterijus:

- kiek strategija atliko sėkmingų išvedimų per tam tikrą laiko limitą;
- kiek išvedimų atliko konkreti strategija, kurių nesugebėjo įvykdyti kitos strategijos;
- kiek strategija surado sprendimų per 1 sekundę.

Autoriai apibendrinę gautus rezultatus teigia, jog visos strategijos randa didžiąją dalį išvedimų per pirmąsias sekundes. Netgi dauguma strategijų sėkmingai suranda 80% išvedimų per pirmą sekundę. FIFO strategija yra silpniausia iš visų 14-os strategijų. Netgi SC11, geriausia paprasta simbolių skaičiavimo strategija, išveda apie 10% daugiau už FIFO strategijas. Lyginant tarpusavyje simbolių skaičiavimo ir rikiavimo (angl. ordering aware, RW212) strategijas, rikiavimo strategijų našumas mažesnis.

Visos strategijos, kurios tarpusavyje derino paprastų simbolių skaičiavimo ir FIFO disjunktų įvertinimo funkcijas, rado daugiau išvedimų už paprastas simbolių skaičiavimo strategijas. Taip pat goal-directed strategijos kartu su simbolių skaičiavimo ar FIFO strategijomis yra daug efektyvesnės. SC11 ir FIFO strategijų sąjunga aplenkia individualias strategijas. Goal-directed strategija pati viena nėra efektyvi ir prilygsta tik simbolių skaičiavimo strategijai. Tačiau šios strategijos veikimas kartu su FIFO sąjunga žymiai geresnis. Be to, pirminis pradinių disjunktų apdorojimas strategijoje priduoda didesnį efektyvumą.

Atlikto tyrimo [SM16] galutinė išvada – tarpusavyje derinamos simbolių skaičiavimo ir FIFO strategijos atlieka išvedimo paiešką žymiai geriau nei kiekviena atskirai. Taip pat nustatyta, jog goal-directed strategija geriausiai veikia kartu su kitomis strategijomis.

3 lentelėje pateikiamos tyrime [SM16] aptartos geriausios strategijos.

3 lentelė. *S. Schulz atlikto tyrimo strategijos*

Pavadinimas	**Strategija**
FIFO	heuristic="-H(1.FIFOWeight(ConstPrio))"
SC11	heuristic="-H(1.Clauseweight(ConstPrio,1,1,1))"
SC21	heuristic="-H(1.Clauseweight(ConstPrio,2,1,1))"
2SC11/FIFO	heuristic="-H(2.Clauseweight(ConstPrio,1,1,1),1.FIFOWeight(ConstPrio))"
5SC21/FIFO	heuristic="-H(5.Clauseweight(ConstPrio,2,1,1),1.FIFOWeight(ConstPrio))"

5SC11/FIFO	heuristic="-H(5.Clauseweight(ConstPrio,1,1,1),1.FIFOWeight(ConstPrio))"
10SC21/FIFO	heuristic="-H(10.Clauseweight(ConstPrio,2,1,1),1.FIFOWeight(ConstPrio))"
10SC11/FIFO	heuristic="-H(10.Clauseweight(ConstPrio,1,1,1),1.FIFOWeight(ConstPrio))"
15SC21/FIFO	heuristic="-H(15.Clauseweight(ConstPrio,2,1,1),1.FIFOWeight(ConstPrio))"
15SC11/FIFO	heuristic="-H(15.Clauseweight(ConstPrio,1,1,1),1.FIFOWeight(ConstPrio))"
RW212	heuristic="-H(1.Refinedweight(ConstPrio,2,1,2,1,1))"

2.3.2 Sistemos „E“ automatinis režimas

Kadangi sistemoje „E“ jau yra įdiegtos tam tikros parametrizuotos išvedimo paieškos galimybės, sistema turi automatinį režimą. Šis režimas nustato disjunkto parinkimo strategiją, literos parinkimo strategiją bei termų rikiavimo būdą, taip pat ir tam tikrus kitus parametrus. „E“ automatinis režimas išsiskiria tuo, jog šis funkcionalumas nėra suprogramuotas kūrėjų, o automatiškai sugeneruojamas remiantis apibrėžtu problemų klasifikavimu ir testų rezultatais [Sch02]. Problemos skirstomos į klases pagal apibrėžtus kriterijus, kurie nuolat keičiasi. Kiekvienai iš problemų klasių priskiriamos atskiros išvedimo paieškos strategijos. Visų pirma strategijos yra testuojamos ir surikiuojamos pagal gautus rezultatus, t.y. pagal tai, kiek rasta išvedimų ir koks vidutinis sėkmingo išvedimo trukmės laikas. Toliau visos strategijos rikiuojamos mažėjimo tvarka ir kiekvienai problemų klasei priskiriama pirmoji iš strategijų, kuri išsprendžia maksimalų tos klasės problemų kiekį. S. Schulz teigimu toks klasifikavimas gali pernelyg apriboti problemų ir strategijų specializaciją, tačiau automatinis režimas veikia daug geriau nei bet kuri atsitiktinai parinkta startegija.

Sistema geba automatiškai analizuoti išvedimo paieškos protokolus, t.y. orientuotus grafus, kurių viršūnės atitinka disjunktus kiekvieno išvedimo žingsnyje, o briaunos eina iš prielaidų į išvadą [Sch01]. Kelias nuo pradinio disjunkto iki tuščio disjunkto vadinamas išvedimo keliu, disjunktai šiame kelyje vadinami padedančiaisiais disjunktas ir pasirenkami kaip apmokymų pavyzdžiai, visi kiti – vadinami nereikalingais disjunktais. Dažniausiai nereikalingų disjunktų būna žymiai daugiau nei padedančiųjų. Tačiau artimiausiai esantys nereikalingi disjunktai paimami kaip mokymo pavyzdžiai. Atmetami disjunktai, kurie niekada nebuvo pasirinkti išvedimo paieškoje. Mokymų pavyzdžiais atrinkti disjunktai transformuojami į šablonus, be to

kiekvieną disjunktą apibūdina anotacija, t.y. kokį vaidmenį disjunktai atlieka ankstesniuose išvedimuose. Šablonų aibė kaupiama žinių bazėje ir jie indeksuojami pagal tam tikrus kriterijus. Pritaikymo fazėje vadovaujantis indeksais nusprendžiama, kuri strategija tiktų naujai išvedimo paieškai, t.y. sistema atrenka disjunktus remdamasi įgyta patirtimi. Atitinkami šablonai paimami ir paduodami mašininio mokymosi algoritmui. Išvedimo paieškos eigoje atsiradę nauji disjunktai yra palyginami su šablonais ir atitinkamai įvertinami. Būtina pabrėžti, jog sistemos „E“ mokymosi procesas yra pilnai automatizuotas. Žmogaus įsikišimas būtinas tik keliais atvejais parenkant tam tikrus parametrus.

S. Schulz siekdamas įsitikinti, jog mokymusi paremta strategija prisideda prie išvedimo paieškos gerinimo, atliko tyrimą, kuriame mokymusi paremta strategija buvo palyginta su standartine simbolių skaičiavimo euristika. Rezultatai atksleidė, jog naujoji strategija surado daugiau įrodymų nei standartinė strategija, ypač sprendžiant sudėtingas problemas. Taip pat tyrimas leidžia daryti išvadą, jog didesnis mokymosi pavyzdžių kiekis lemia geresnę neišsprestų problemų išvedimo paiešką.

2.3.3 Strategijų parinkimas taikant mašininį mokymąsi

Mašininis mokymasis – tai dirbtinio intelekto sritis, kuria keliamas ir sprendžiamas klausimas, kaip kompiuteriai ir kompiuterinė įranga galėtų automatizuotu būdu tobulėti remdamasi žiniomis ir patirtimi [Mit97]. Tad tai apima algoritmų/metodų ir programų kūrimą, kurie leistų kompiuteriams mokytis ir spręsti tam tikrus uždavinius (mašininis mokymasis turi du tikslus – klasifikuoti duomenis remiantis sukurtais modeliais, antrasis – remiantis šiais modeliais prognozuoti būsimus rezultatus) bei tobulėti apimant didesnius duomenų kiekius. Mašininis mokymasis gali būti prižiūrimas (angl. supervised, programai pateikiama įvestis su pavyzdžiais ir gaunamais rezultatais, tikslas – išmokti pagrindinę taisyklę, kaip susieti įvestį su išvestimi), neprižiūrimas (angl. unsupervised, programai pavyzdžiai nepateikiami, tikslas – surasti duomenų modelius ar struktūras), sustiprintas mokymasis (angl. reinforced learning, programa sąveikauja su dinamiška aplinka, kurioje ji turi atlikti tam tikrą užduotį). Mašininio mokymosi modelis tai matematinis modelis, kuris vieną kartą apmokytas su duomenimis, gali atlikti jam paskirtas užduotis.

Išskiriami tokie mašininio mokymosi modeliai:

- dirbtiniai neuroniniai tinklai,
- sprendimų medžiai,
- SVM (angl. support vector machines),
- regresijos analizė,
- Gauso procesai.

2.3.3.1. SVM ir Gauso metodų taikymai

Mokslininkas James P. Bridge [Bri10] tinkamos išvedimo paieškos strategijos parinkimo klausimą sprendžia pasitelkdami mašininį mokymąsi. Autoriai siekė tikslo, jog strategijos parinkimas problemų sprendimui veiktų be jokio papildomo sistemos naudotojo įsikišimo, o sistema pati galėtų pilnai nuspręsti, kokią strategiją pasirinkti. Be to, programa „E“ turi virš 200 apibrėžtų strategijų, tad reikėtų labai daug laiko ir tinkamų resursų siekiant išmokti pasirinkti tinkamiausią iš visų strategijų. Mašininio mokymosi metodams apsimokyti buvo pateiktos žinomos 5 programos „E“ strategijos. Jos buvo atrinktos remiantis S. Schulz atliktais įvairiais eksperimentais su strategijomis sprendžiant TPTP problemas. Programos „E“ automatinis režimas klasifikuoja problemas remdamasis gautais rezultatais ir pagal tai parenka strategiją – kiekvienai klasei priskiriama geriausia strategija. Strategijos surikiuotos pagal išspręstų TPTP problemų skaičių. Pirmoji iš penkių strategijų turi didžiausią skaičių išvedimų. Autoriai taikydami mašininį mokymąsi siekė sukurti geresnę „E“ programos automatinio režimo versiją. Tyrime buvo naudojami SVM ir Gauso procesų mašininio mokymosi modeliai. Rezultatai parodė, jog kiekvienai problemai parenkant atskirą strategiją išvedimų buvo rasta daugiau nei visas problemas sprendžiant su bet kuria viena strategija.

4 lentelėje pateikiamos tyrime [Bri10] naudotos strategijos.

4 lentelė. *James P. Bridge atlikto tyrimo strategijos*

Pavadinimas	**Strategija**
Rank1	4*ConjectureGeneralSymbolWeight(SimulateSOS,100,100,100,50,50,10,10,1.5,1.5,1), 3*ConjectureGeneralSymbolWeight(PreferNonGoals,200,100,200,50,50,1,100,1.5,1.5,2), 1*Clauseweight(PreferProcessed,1,1,1), 1*FIFOWeight(PreferProcessed)
Rank2	8*Refinedweight(PreferGoals,1,2,2,2,2), 8*Refinedweight(PreferNonGoals,2,1,2,2,0.5), 1*Clauseweight(PreferUnitGroundGoals,1,1,1), 1*FIFOWeight(ConstPrio)
Rank3	10*PNRefinedweight(PreferGoals,1,1,1,2,2,2,0.5), 10*PNRefinedweight(PreferNonGoals,2,1,1,1,2,2,2), 5*OrientLMaxWeight(ConstPrio,2,1,2,1,1), 1*FIFOWeight(ConstPrio)

Rank4	10*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5, 1.5,1.5), 1*FIFOWeight(ConstPrio)

2.3.3.2. E-MaLeS sistema

Autorių kolektyvas kartu su S. Schulz [KSU13] įgyvendino E-MaLeS (angl. E Machine Learning of Strategies) strategijų parinkimo sistemą, kuri naudoja mašininio mokymosi metodus geriausios strategijos parinkimui. Pagal įvairias problemos savybes sistema E-MaLeS sugeba nuspėti, kiek laiko truks problemos sprendimas taikant konkrečią „E“ sistemos strategiją.

2.3.3.3. BliStr sistema

Taip pat buvo sukurtas automatizuotas programos „E“ strategijų sukūrimo įrankis BliStr (angl. Blind Strategymaker). Jo autorius J. Urban [Urb13] iškėlė tikslą, kad strategijų atrinkimo procesas galėtų būtų kiek įmanoma daugiau automatizuotas („aklas“) ir remtųsi kuo mažesniu duomenų kiekiu apie programos „E“ strategijas. Autorius siekia atrinkti kuo mažesnę 20-ies reikšmingų parametrų aibę. Mažesnės parametrų aibės sukūrimui buvo panaudotos jau egzistuojančios programos „E“ automatinio režimo strategijos, kurios buvo efektyviausios sprendžiant problemas.

Tyrime [Urb13] buvo sprendžiamos 200 TPTP problemos (iš 1000 mokymuisi skirtų problemų) su 5 sekundžių laiko limitu naudojant 280 automatinio režimo strategijų. Išsprendus 117 problemų (iš 200) buvo išskirtos šešios strategijos:

G-E--_008_B31_F1_PI_AE_S4_CS_SP_S2S

G-E--_008_K18_F1_PI_AE_CS_SP_S0Y

G-E--_010_B02_F1_PI_AE_S4_CS_SP_S0Y

G-E--_024_B07_F1_PI_AE_Q4_CS_SP_S0Y

G-E--_045_B31_F1_PI_AE_S4_CS_SP_S0Y

G-E--_045_K18_F1_PI_AE_CS_OS_S0S

Šios strategijos vėl buvo panaudotos sprendžiant visas 1000 mokymuisi skirtų problemų su 60 sekundžių laiko limitu. Pavyko įrodyti 597 problemas, 403 problemos liko neįrodytos. Siekdamas išspręsti likusias 403 problemas autorius naudoja ParamILS sistemą. Ši sistema ieško gerų strategijos parametrų remdamasi iteracine vietos paieška (angl. iterative local search). Autoriaus kuriamas įrankis BliStr (angl. Blind Strategymaker) apima ParamILS sistemą, išspręstų problemų aibę, geriausių rezultatų matricą ir tinkamų strategijų rinkinį. BliStr programa veikia tokiu būdu: kuomet ParamILS sukuria naują strategiją, ji per tam tikrą laiką yra įvertinama su Mizar@Turing mokymuisi skirtu problemų rinkiniu, tuomet atnaujinamos rezultatų matricos ir surikiuojamas tinkamų strategijų sąrašas. Toliau vykdoma kita ParamILS iteracija su geriausia

atnaujinta strategija ir atnaujintu problemų rinkiniu, jei ParamILS sistemoje tokia strategija dar nebuvo panaudota. BliStr programa sustoja, kai nebėra tinkamų strategijų arba kai visos tinkamos strategijos jau buvo ištestuotos su problemų rinkiniu. J. Urban atlikti tyrimai [Urb13] rodo, jog naujos strategijos veikia efektyviau nei programos „E“ automatinis režimas (E versija 1.6).

J. Jakubuv ir J. Urban [JU16] straipsnyje pristatė BliStr sistemos plėtinį BliStrTune, kuris kurdamas naujas strategijas remiasi naujomis į išvadą orientuotomis disjunkto svorio funkcijomis. Įrankiu siekiama disjunktų įvertinimo funkcijų parinkimą padaryti dar labiau „aklu“ tiriant didesnę startegijų parametrų erdvę. Pagrindinė idėja – didelė aibė optimizuotų parametrų yra suskirtyta į du sluoksnius: vienas jų lieka fiksuotas, kitas – naudojamas strategijos kūrimui. Rezultatai paskleidžiami tarp sluoksnių, o sluoksnių derinimas ir paskleidimas yra kartojami. Jei BliStr naudoja fiksuotą parametrų rinkinį ParamILS programos veikimui, tai BliStrTune – kiekvienam ParamILS veikimui pateikia skirtingą funkcijų kolekciją. Keičiami tik pačių funkcijų viduje esantys individualūs disjunktų įvertinimo funkcijų parametrai, visi kiti lieka fiksuoti. Įvertinus BliStrTune veikimą, ji išsprendė daugiau problemų nei paprasta BliStr sistema.

2.3.3.4. ENIGMA sistema

Kaip teigia autorių kolektyvas [JU17] automatizuoti disjunktų atrankos metodai vertina disjunktus izoliuotai, t.y. neatsižvelgia į kitus disjunktus atliekant išvedimo paiešką. Šios problemos sprendimui autoriai sukūrė „E“ sistemos papildymą – ENIGMA (angl. Efficient learNing-based Internal Guidance Machine), kuri pagrįsta mašininiu mokymusi. Išvedimo paieškoje disjunktai yra klasifikuojami į teigiamus ir neigiamus priklausomai nuo to, ar jie dalyvauja sėkmingose išvedimo paieškose. Iš naujo atliekant išvedimo paiešką su jau turimais duomenimis apie disjunktus (teigiami jie ar ne), pirmenybė teikiama teigiamiems disjunktams. Išvedimo paieška tokiu atveju turėtų būti trumpesnė.

Buvo sukurtas ir pastarosios sistemos plėtinys ENIGMA-NG [CJS+19], kurioje vietoj naudoto klasifikatoriaus naudojami kiti mašininio mokymosi metodai: gradiento sustiprinti medžiai ir rekursyvieji neuroniniai tinklai. Disjunktai traktuojami kaip literų sekos. Rekursyvieji neuroniniai tinklai (RNN) yra skirti apdoroti vektorių sekas. Todėl turėdami disjunkto literų seką galime apmokyti RNN, kad jis pateiktų, ar disjunkstas yra tinkamas išvedimo paieškai.

Tyrime [CJO+21] autoriai kontekstiniam disjunktų įvertinimui kūrė ENIGMA programą su GNN (angl. graph neural network) tinklu. GNN-ENIGMA sistemoje sugeneruoti disjunktai nėra iš karto sureitinguojami ir yra nepriklausomi nuo kitų disjunktų. Jie yra įvertinami GNN didesnėmis partijomis atsižvelgiant į jau atrinktus disjunktus – t.y. į kontekstą. GNN bendrai įvertina naudingiausią konteksto poaibį ir naujus disjunktus atlikdamas kelis pranešimų perdavimo etapus. Pranešimų perdavimo algoritmas atsižvelgia į ryšius tarp simbolių, termų, subtermų,

atomų, literų ir disjunktų. Jis apmokomas daugeliu anksčiau atliktų išvedimo paieškų ir įvertina, kurie dijsunktai geriausiai gali padėti išvedimo paieškai.

ENIGMA Anonymous plėtinys disjunktų atranką vykdo nepriklausomai nuo specifinių simbolių vardų. Iš sėkmingos išvedimo paieškos paimami apdoroti disjunktai ir pažymimi kaip teigiami disjunktai, likusieji – kaip neigiami. Šie mokymosi duomenys paverčiami į tensorių (vieno ar dviejų dimensijų kintamojo ilgio vektorių) ir kiekvienas jų reprezentuoja disjunktų aibę kaip grafą su trijų tipų mazgais. Išvados disjunktai įtraukiami į grafą. Kai GNN klasifikatorius apmokomas, jis naudojamas disjunktų įvertinimui naujoje išvedimo paieškoje. Disjunktai įvertinami ne kiekvienas atskirai, o suformuoja tam tikro ilgio užklausą. Autoriai atliko keletą algoritmo pakeitimų patobulinančių kontekstinį įvertinimą – kontekstą papildant vis naujai sugeneruotais disjunktais. Atlikus eksperimentus su Mizar bibliotekos problemomis gauti geri rezultatai – sistema rado įrodymus daliai neišspręstų problemų.

2.3.3.5. Giliojo mokymosi taikymai

Tyrime [LIS+17] Google Research komandos nariai pristatė giliuoju mokymusi pagrįstą išvedimo paiešką. Šiuo eksperimentu giliųjų neuroninių tinklų modeliai pirmą kartą įvertinti kaip vedliai pritaikant juos automatinio įrodymo sistemos viduje. Keli gilieji neuroniniai tinklai buvo apmokyti ir panaudoti atrenkant apdorotus disjunktus išvedimo paieškos procese. Eksperimentais buvo įrodyta, jog giliuoju mokymusi paremtas disjunktų atrinkimas gali ženkliai sumažinti vidutinį išvedimo paieškos žingsnių skaičių bei padidinti įrodytų teoremų skaičių. Modernių automatinio įrodymo sistemų vieksmingumas vis dar mažas bandant apdoroti dideles problemų bibliotekas. Automatinėms įrodymo sistemoms galima padėti pritaikant mašinio mokymosi metodus. Gilieji konvoliuciniai neuroniniai tinklai pasižymi puikiais rezultatais kalbos atpažinimo ir natūralios kalbos apdorojimo, objektų atpažinimo srityse. Gilieji hierarchiniai konvoliuciniai tinklai įtakojo pažangą kalbos ir garso generavime. Tačiau giliųjų neuroninių tinklų taikymas yra brangus, nes vykdymas užtrunka daug laiko – kol neuroninis tinklas įvertina vieną disjunktą, automatinio įrodymo sistema gali atlikti keletą šimtų ar tūkstančių žingsnių. Tad šių neuroninių tinklų pritaikymas turi duoti gerus rezultatus. Autoriai [LIS+17] pritaikė kelis techninius sprendimus, kurie pagerino išvedimų paiešką. Vienas jų – neuroniniai tinklai naudojami kartu su strategijomis pateikė geresnius rezultatus nei neuroniniai tinklai naudojami vieni be strategijų. Kitas – kelių etapų paieškos vykdymas, kuomet neuroninio tinklo veikimą kitame etape keičia strategijos naudojimas.

Autoriai eksperimente neuroninius tinklus apmoko ir įvertina su Mizaro matematinės bibliotekos įrodymais. Problemų aibė padalinama į mokymosi ir validavimo aibes. Autorių pasirinkti neuroninių tinklų modeliai turi dvi įvestis: paneigtą išvadą, kurią reikia įrodyti, ir neapdorotą disjunktą. Disjunktas, kuris buvo panaudotas išvedimo paieškoje, gauna didesnį

išvesties įvertį (tikimybė artimensė „1“) nei disjunktas, kuris nebuvo panaudotas paieškoje (tikimybė artimesnė „0“). Išvedimo paieškoje pasirenkamas neapdorotas disjunktas su didžiausia tikimybe. Neuroniniai tinklai apmokomi su programos „E“ sugeneruotais įrodymais taikant Auto208 strategiją. Įterpimo tinkle (angl. embedding network) naudotos tokios architektūros: paprastas konvoliucinis tinklas, WaveNet tinklas ir rekurentinis neuroninis tinklas. Į žodyną surenkami visi simboliai (konstantos, funkcijų vardai, kintamųjų vardai, loginės operacijos, skliaustai) ir įvesties sluoksnis konvertuoja kiekvieną jų į vektorių. Įterpimai inicijuojami atsitiktiniu būdu ir išmokstami mokymosi proceso metu. Konvoliucinis ir WaveNet modeliai gauna disjunktą ir paneigtą išvadą kaip įterpinių seką, rekurentiniai tinklai naudoja įterpinius CNF medžio lapuose esantiems vardams.

Po apmokymo gilieji neuroniniai tinklai yra įvertinami naudojant tris priemones. Pirmoji priemonė įvertina, kiek tiksliai apmokytas modelis gali nuspėti, jog disjunktas bus panaudotas galutiniame išvedime. Antroji – patikrina, ar neuroninis tinklas nedaro daugiau žalos pridedant jį prie automatinio režimo strategijos. Trečioji – tikrina, ar naudojant giliuosius neuroninius tinklus galima įrodyti Mizaro bibliotekos teiginius, kurie dar neturi jokio automatinės įrodymo sistemos įrodymo. Naudojami tik geriausius tikslumo įverčius pateikę neuroniniai tinklai. Šie modeliai suteikia įvertį kiekvienam neapdorotam disjunktui. Sistema „E“ naudoja šį įvertį neapdorotų dijsunktų reitingavimui. Tada kiekviename žingsnyje apdoroja disjunktą, kuris turi geriausią reitingą.

Eksperimentų rezultatai rodo, jog gilieji neuroniniai tinklai yra efektyviausi, kai naudojami kartu su jau egzistuojančiomis greitomis strategijomis – tai yra gilieji neuroniniai tinklai 20 minučių naudojami pirmame etape, o laiko limitui besibaigiant perjungiama į 10 minučių automatinį režimą – tradicinį paieškos užbaigimo etapą, kuris gali apdoroti daugybę disjunktų be neuroninio tinklo pagalbos. Automatinis režimas leidžia neprarasti visų laiko resursų. Tinklų vedama paieška leidžia išvedimo paieškai atrinkti pačius svarbiausius disjunktus, tačiau dėl savo lėtumo gali paieškos neužbaigti. O perjungimas į automatinį režimą leidžia šio trūkumo išvengti.

Didžiausią tikslumą pateikė WaveNet ir du CNN tinklai. Eksperimentuose CNN tinklai aplenkė WaveNet tinklą išvedimo paieškose su teiginiais, kurie jau turi automatinės įrodymo sistemos įrodymus. Neuroninio tinklo tikslumas nenulemia didesnio įrodymų skaičiaus. Daugiau išvedimų atlieka paprastesni ir mažiau kokybiški neuroniniai tinklai.

5 lentelėje pateikiamos tyrime [LIS+17] naudotos strategijos.

5 lentelė. *Giliojo mokymosi tyrimo strategijos*

Pavadinimas	**Strategija**

Auto208	1*ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1), 4*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5), 1*FIFOWeight(PreferProcessed), 1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1), 4*Refinedweight(SimulateSOS,3,2,2,1.5,2).
Auto200	1*ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1), 6*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5), 2*FIFOWeight(PreferProcessed), 1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1), 8*Refinedweight(SimulateSOS,1,1,2,1.5,2).

2.3.3.6. Sustiprinto mokymosi taikymai

G. Sutcliffe ir J. McKeown atliko tyrimą [MS25], kuriame iškėlė idėją, jog galima sukurti vieną strategiją tam tikram problemų duomenų rinkiniui, kurią sudarytų programos „E“ automatinio režimo strategijos. Automatinis režimas analizuoja programai pateiktą problemą, kad galėtų rasti efektyvią strategiją. Strategiją sudaro termų, literų rikiavimas, disjunkto parinkimo strategija ir kiti parametrai. Šiuo metu programos „E“ automatinio režimo strategija sudaro 108 įvairūs parametrai. Autoriai kuriamoje strategijoje 107 parametrus užpildė individualiose strategijose dažniausiai naudotomis parametrų reikšmėmis, o paskutinį disjunktų atrinkimo strategijos parametrą *heuristic_def* jungė keliais būdais:

- sujungiant visas disjunkto įvertinimo funkcijas, naudojamas visose strategijose, ir joms priskiriant tą patį svorį lygų vienam;
- kiekvienai disjunkto įvertinimo funkcijai priskiriant svorį, kuris buvo gautas susumuojant visose strategijose naudotus tos funkcijos svorius;
- įvertinant kiekvienos individualios strategijos pasikartojimą;
- įvertinant, ar strategija sėkmingai atliko išvedimą.

Autoriai įvairiais metodais sujungtą strategiją įvertino su MPTPTP2078, VBT, ir SLH-29 duomenų rinkiniais. Tyrimas parodė, kad apjungtoji strategija aplenkė automatinį programos „E“ režimą. Tačiau apjungta strategija yra efektyvi tik homogeniškam duomenų rinkiniui.

Tie patys autoriai J. McKeown ir G. Sutcliffe kitame tyrime [MS23] panaudojo sustiprinto mokymosi metodą. Eksperimento tikslas – išmokti metastrategiją, kuri nuspręstų, kokią konkrečią

strategiją panaudoti kiekviename programos „E“ išvedimo paieškos žingsnyje. Išvedimo paieškos efektyvumas priklauso nuo įrodymo žingsnių paieškos lauko dydžio. Mažėjant atrinktų disjunktų kiekiui, išvedimo paieškos laikas trumpėja.

Sustiprintas mokymasis – tai mašinino mokymosi metodas, kuriame dalyvauja agentai ir jie gauna atlygį. Autoriai tyrime [MS23] naudojo penkis agentą apibūdinančius parametrus:

- atrinktų disjunktų skaičius,
- apdorotų disjunktų skaičius,
- neapdorotų disjunktų skaičius,
- vidutinis simbolių skaičius neapdorotuose disjunkutose,
- vidutinis simbolių skaičius apdorotuose disjunktuose.

Agentų veiksmas tyrime apibrėžiamas kaip funkcijos pasirinkimas iš disjunktų įvertinimo funkcijų aibės. Apdovanojimai agentams skirstomi po išvedimo paieškos bandymo. Jei išvedimas buvo sėkmingas, agentas apdovanojamas už kiekvieną disjunkto, dalyvaujančio galutiniame išvedime, parinkimą. Neuroninio tinklo apmokymui ir įvertinimui buvo pasirinkta 20 disjunktų įvertinimo funkcijų, kurios buvo dažniausiai naudotos programos „E“ automatiniame režime su MPTPTP2078 duomenų rinkiniu. Autorių atlikto tyrimo [MS23] rezultatai rodo, jog išvedimo paieška yra efektyvesnė su metastrategija nei naudojant programos „E“ automatinį režimą.

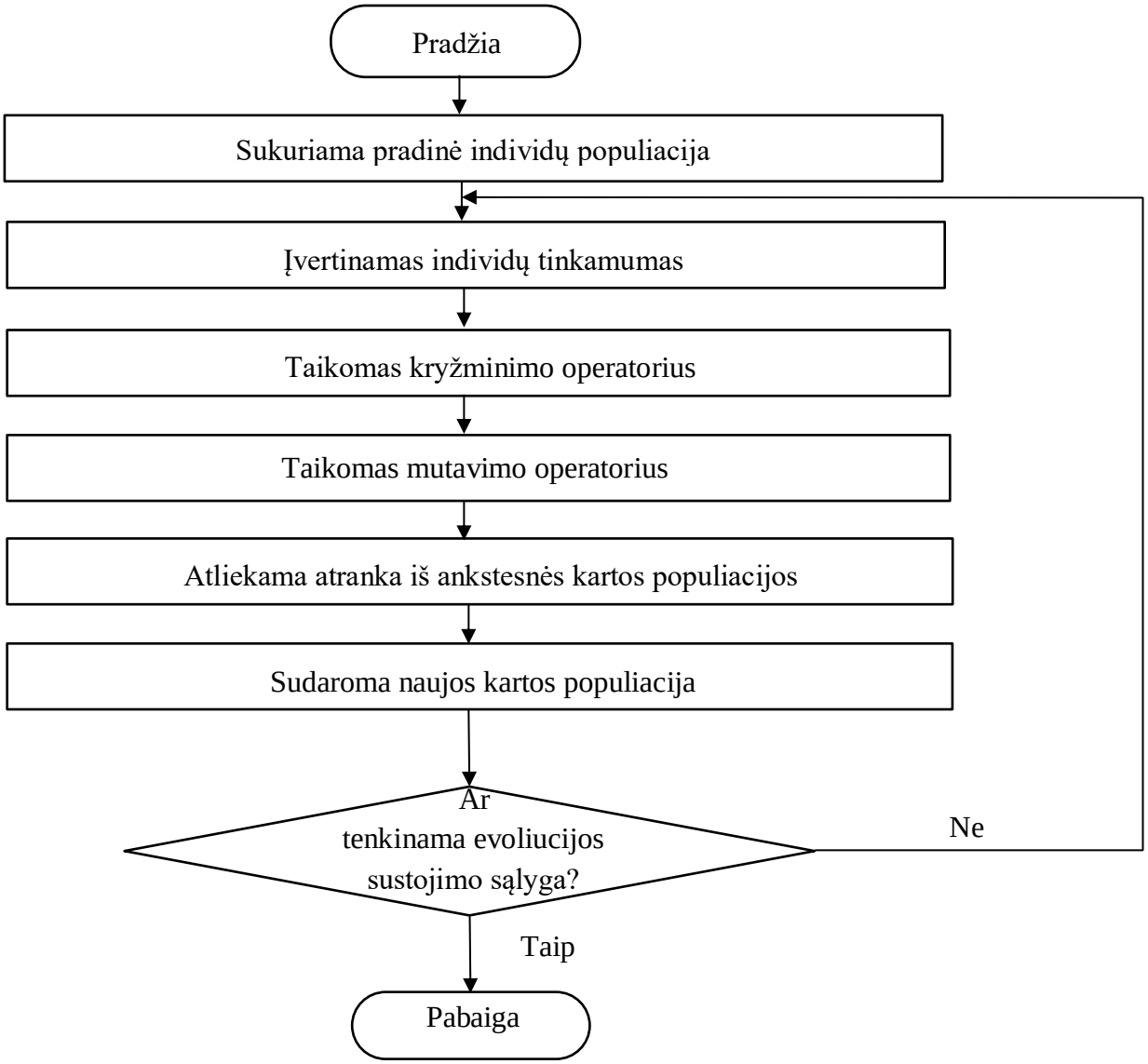
3. Formulių bazė

AĮS „E“ sugeba atpažinti kelis įvesties formatus – LOP ir TPTP. Tyrime bus naudojama TPTP (angl. Thousands of Problems for Theorem Provers) formulių (problemų) bazė (versija 6.10) [TPT]. TPTP bibliotekos kūrėjai Geoff Sutcliffe ir Christian Suttner formulių bazę apibrėžia kaip biblioteką, skirtą predikatų logikos sistemoms. Visos problemos pateikiamos vieningu formatu ir yra suteikiama galimybė šį formatą konvertuoti į kitus AĮS formatus. Biblioteką sudarančios problemos apima įvairias sritis. Remiantis oficialaus TPTP tinklalapio informacija [TPT] šią biblioteką sudaro 7 sričių ir 53 posričių problemos, kaip matematika, informatika, inžinerija, socialiniai ir humanitariniai mokslai.

Taip pat problemos skiriasi savo sudėtingumu – yra labai paprastų, lengvai išsprendžiamų ir labai sudėtingų problemų. Problemų reitingai parodo, kiek AĮS yra pajėgi išspręsti konkrečią problemą [Sut10]. TPTP problemos yra reitinguojamos suteikiant joms įvertį nuo 0 iki 1.0. Žemiausias reitingas reiškia, jog dauguma AĮS gali išspręsti problemą, ir problema įvardinama kaip lengva. Aukščiausias reitingas 1.0 reiškia, jog nei viena iš AĮS iki šiol neišsprendė problemos, ir problema įvardinama kaip labai sunki.

4. Genetinis algoritmas

Genetiniai algoritmai buvo sukurti mokslininko John Holland su kolegomis Michigan universitete 8-tame praeito amžiaus dešimtmetyje [Gol89]. Šie algoritmai naudoja simuliacinę evoliuciją, kad būtų galima surasti sprendimus sudėtingoms realaus pasaulio problemoms [JPK19]. Gamtoje individai turi prisitaikyti prie gyvenamos aplinkos tam, kad galėtų išgyventi. Šis procesas žinomas kaip evoliucija. Reprodukcijos metu bruožai ar ypatybės, kurios individą daro gebiu konkuruoti su kitais, yra paliekamos, o silpninančios ypatybės pašalinamos. Genai yra šiuos bruožus kontroliuojantys veiksniai, o genų rinkiniai suformuoja chromosomas. Tik geriausi individai išlieka tolimesnėse kartose ir jų genai perduodami palikuonims. Genetiniai algoritmai naudoja biologinės evoliucijos įkvėptus mechanizmus, kurie grindžiami Darvino evoliucine atrankos teorija, tokius kaip reprodukcija, ir tris pagrindines operacijas – mutaciją, rekombinaciją arba kryžminimą ir atranką. Genetinio algoritmo schemą galima peržiūrėti 3 paveiksliuke.



3 pav. *Genetinio algoritmo schema*

Pradžioje atsitiktiniu būdu sukuriama individų aibė. Naudojant tinkamumo funkciją įvertinamas individų tinkamumas (gerumas). Daliai individų (vadinamų tėvais) yra taikomas dvinaris kryžminimo operatorius – gaunami nauji individai (vaikai). Daliai individų pritaikomas unarinis mutavimo operatorius – gaunami nauji mutuoti individai. Iš ankstesnės populiacijos atrenkama dalis individų taikant pasirinktą atrankos metodą. Kryžminimo ir mutavimo metu gautų individų aibė apjungiama su atrinktų individų aibe ir tokiu būdu gaunama naujos kartos populiacija, kuri toliau vertinama naudojant tinkamumo funkciją ir procesas kartojamas nuo pradžių, kol randamas pakankamai geras individas arba išpildoma kita evoliucijos nutraukimo sąlyga.

Šiame tyrime individas yra išvedimo paieškos strategija, kurią apibūdina ją sudarančios svorio ir prioriteto funkcijos. Populiacija suprantama kaip visi šiuo metu esantys individai – išvedimo strategijos, o generacija arba karta – tai populiacija tam tikru laiko momentu. Laiko tarpas tarp generacijų yra diskretus, todėl galime kiekvienai generacijai skirti vis didėjantį sveikąjį skaičių.

4.1 Individų kodavimas

Šiame tyrime genetinio algoritmo ir evoliucijos realizavimui yra naudojama *Python* programavimo kalba. Individai – paieškos strategijos – koduojami naudojant *Python* duomenų struktūrą – sąrašų sąrašą (angl. list of lists). Sąrašo elementas atitinka svorio funkciją, kuri pati savyje turi sąrašą – unikalų svorio funkcijos pavadinimą, šiai funkcijai skirtą svorio įvertį ir nuo 1 iki *n* parametrus.

Paieškos strategijos pavyzdys:

```
heuristic = [
        ['ConjectureGeneralSymbolWeight' , 1, 'SimulateSOS' , 488, 104, 105, 32, 173, 0,
        327, 3.63, 1.43 ,1],
        ['FIFOWeight' , 1, 'PreferProcessed'],
        ['Clauseweight' , 8, 'PreferUnitGroundGoals' , 1, 1, 0.49]
]
```

4.2 Pradinės populiacijos kūrimas

Pradinei populiacijai sudaryti yra sukuriami individai, atsitiktinai parenkant paieškos strategijų svorio funkcijas ir jų parametrus. Galimas parametrų reikšmes apibrėžia žodynas, pateiktas AĮS „E“ dokumentacijoje [Sch24]. Pavyzdžiui, svorio funkcijos *Refinedweight* galimos reikšmės:

```
"Refinedweight":
[
```

["string", ["PreferNGoals", "SimulateSOS", "PreferGoals"]],
["int", 1, 8],
["int", 1, 8],
["int", 1, 8],
["float", 0.1, 5.0],
["float", 0.1, 5.0],
]

Parametrai gali būti simbolių eilutės (angl. string), slankiojo kablelio skaičiai (angl. floating number) ar sveikieji skaičiai (angl. integer). Simbolių eilutėms reikia nurodyti galimas reikšmes, o skaitiniams parametrams – minimalią ir maksimalią reikšmes.

Kuriant naują individą atsitiktiniu būdu parenkamas skaičius *n*, kuris reiškia, kiek svorio funkcijų sudarys individą. Toliau *n* kartų atsitiktiniu būdu yra parenkamos svorio funkcijos ir jų parametrų reikšmės. Kitaip tariant, skirtingi individai gali turėti skirtingą kiekį (*n*) svorio funkcijų. Procesas kartojamas, kol užpildoma visa populiacija.

4.3 Individų įvertinimas

Gamtoje individo tinkamumas priklauso nuo daugelio faktorių, tokių kaip gebėjimas apsiginti nuo priešų, susirasti maisto ar atbaidyti kenkėjus. Evoliucinio skaičiavimo teorijoje tinkamumas yra funkcija, kuri apibūdina, kaip gerai individas išsprendžia jam paskirtą apibrėžtą problemą [Gol89].

Šiame tyrime sistemai „E“ yra paduodama išvedimo paieškos strategija (individas) ir problema (formulė), kurią sistema bando išspręsti. Po formulės (problemos) įvertinimo sistema „E“ gali grąžinti:

- „Proof found“ (įrodymas rastas) – jei formulė yra tapačiai teisinga arba tapačiai klaidinga.
- „No proof found“ (įrodymas nerastas) – jei formulė yra įvykdoma.
- „Resource limit exceeded (time)“ (pasibaigė laiko limitas) – jei per nurodytą laiką sistemai „E“ nepavyko įvertinti formulės (eksperimentų metu buvo naudojamas 3 sekundžių laiko limitas).

Su kiekvienu individu (išvedimo paieškos strategija) yra bandoma išspręsti iš anksto nurodytą kiekį problemų (formulių). Individo tinkamumas (gerumas) yra išspręstų problemų (formulių), kurioms sistema „E“ grąžino „Proof found“ arba „No proof found“, kiekis. Kuo daugiau išspręstų problemų, tuo individas (išvedimo paieškos strategija) yra geresnis.

Pavyzdinis sistemos „E“ iškvietimas vertinant vieną individą su viena problema:

```
eprover -
H'(8*Refinedweight(SimulateSOS,1,6,1,1.05841109696,3.82294625984),1*FIFOWeight(
ConstPrio),3*StaggeredWeight(ConstPrio,16.261760366),1*FIFOWeight(PreferProcess
ed),9*PNRefinedweight(PreferNonGoals,7,3,6,4,6,7,0.939051607931),2*StaggeredWeig
ht(ConstPrio,4.87198036924),3*RelevanceLevelWeight2(ConstPrio,0,4,2,2,88,240,172,5
8,0.220014733351,0.683154772914,0),1*StaggeredWeight(ConstPrio,2.39661745124))'
--cpu-limit=3 --memory-limit=3000 ALG195+1.p
```

Pavyzdyje parametru *H* yra nurodoma strategija (individas), *cpu-limit* – procesoriaus laiko limitas sekundėmis, *memory-limit* – atminties limitas megabaitais, o pavyzdinė problema (formulė) yra pateikta faile *ALG195+1.p*.

Bandant išspręsti tam tikrą kiekį problemų (formulių), sistemos „E“ iškvietimas ir įvykdymas yra atliekamas lygiagrečiai kiekvienai problemai, naudojant *Python Subprocess* biblioteką. Kiekvienos problemos sprendimas yra kaip unikali gija. Šių gijų planavimą atlieka operacinė sistema. Eksperimente buvo naudojama *Linux Debian 12* operacinė sistema.

Individų vertinimui eksperimente buvo naudojama 50 arba 100 atsitiktinių būdu atrinktų problemų iš TPTP bibliotekos [TPT]. Vienai problemai išspręsti buvo skirtas maksimalus 3 sekundžių laiko limitas. Jei individo vertinimui naudojama 100 problemų, tai bendras individo vertinimo laikas yra ne didesnis nei 3 s * 100 = 300 s = 5 min.

4.4 Kryžminimas

Atliekant kryžminimą dviejų individų tėvų genetinė informacija yra sujungiama tam, kad būtų sukurtas naujas individas. Genetiniuose algoritmuose šis kryžminimo operatorius yra analogiškas gamtoje vykstančiai reprodukcijai [US15].

Šiame tyrime individus sudaro svorio funkcijos su savo parametrais. Kryžminant individus (tėvus) gaunami nauji individai (vaikai) parenkant svorio funkcijas iš tėvų. Toliau pateikiamas tyrime naudojamas kryžminimo algoritmas *Python* programavimo kalba.

```
# Input: ind1, ind2, favor1
# Output: newInd
def cxParams(w1, w2, favor1):
    newInd = []
    for w1, w2 in zip(w1, w2):
        if random.uniform(0,1) < favor1:
                newInd.append(w1)
        else:
                newInd.append(w2)
```

```
        return  newInd
    newInd = []
    matches = []
    for i1, e1 in enumerate(ind1):
        for i2, e2 in enumerate(ind2):
            if e1[0] == e2[0]:
                matches.append([e1,e2])
                ind1.remove(e1)
                ind2.remove(e2)
              break
    nonmatches = ind1
    nonmatches.extend(ind2)
    for m in matches:
        newInd.append(cxParams(m[0], m[1], favor1))
    minNum = random.randint(MINW, MAXW)
    while len(newInd) < minNum and len(nonmatches) > 0:
        new = random.choice(nonmatches)
        newInd.apend(new)
        nonmatches.remove(new)
    return newInd
```

Algoritmas pradedamas ieškant sutampančių svorio funkcijų tėvų individuose (*ind1*, *ind2*). Visos svorio funkcijos, kurių pavadinimai tėvuose sutampa, yra perkeliamos į sąrašą *matches*, o iš tėvų tos funkcijos pašalinamos. Iš nesutampančių tėvų funkcijų sudaromas kitas sąrašas *nonmatches*. Pradžioje vaiko individui (*newInd*) priskiriamos sutampančios svorio funkcijos. Tėvuose esančių funkcijų pavadinimai gali sutapti, bet skirtis jų parametrai. Svorio funkcijos parametrai vaikui parenkami iš kažkurio vieno iš tėvų naudojant funkciją *cxParams*, kurioje palankumas pirmam individui *ind1* nurodomas parametru *favor1*. Likusios svorio funkcijos iš tėvų vaikui priskiriamos atsitiktiniu būdu.

4.5 Mutacija

Mutacija naudoja vieną iš tėvų ir sukuria vieną palikuonį pritaikydama tam tikrą atsitiktinį pokytį genams. Realioje gamtos evoliucijoje genetinė medžiaga gali būti pakeista atsitiktinai dėl klaidingo dauginimosi ar kitų genų deformacijų, pavyzdžiui, dėl radiacijos. Genetiniame algoritme mutacija suprantama kaip atsitiktinė genų deformacija su tam tikra tikimybe [Men13].

Mutacija yra reikalinga, nes pritaikius kryžminimo operatorių gali būti prarandama potencialiai reikšminga genetinė informacija, tad mutacija apsaugo nuo negrįžtamo praradimo

[Gol89] ir išsaugo genetinę chromosomų įvairovę populiacijoje, perkeldama ją į naujai formuojamą populiaciją. Taip siekiama apsaugoti populiaciją nuo individų chromosomų supanašėjimo.

Toliau pateikiamas šiame tyrime naudojamas mutavimo operatoriaus algoritmas:

```
# Input: individual, probWeightMutates, probParamMutates
# Output: individual
  for weight in individual:
        if random(0,1) < probWeightMutates:
  for p in weight:
      if random(0,1) < probParamMutates:
          if parameter.type == string:
              paramNew = weightsDictionary.getRandomString()
          elseif parameter.type == int or float:
              paramNew = random.gauss(mu=p, sigma=(max-min)/4)
              if paramNew < 0 : paramNew = 0
          p = paramNew
```

Individe gali būti pakeičiama tiek svorio funkcija, tiek jos parametrai. Tikimybė, kad pasikeis svorio funkcija, nurodoma parametru *probWeightMutates*, o kad pasikeis argumentai joje – parametru *probParamMutates*. Individas mutavimui taip pat parenkamas su tikimybe. Jei individo mutavimo tikimybė yra 0.1, svorio funkcijos mutavimo tikimybė – 0.5, o parametro joje – 0.3, tai bendra tikimybė, kad mutuos to individo svorio funkcijos parametras yra 0.1 * 0.5 * 0.3 = 0.015 = 1.5%. Žema parametro mutavimo tikimybė yra svarbi ir lemia, kad naujas mutuotas palikuonis nebus nepanašus į savo tėvą.

4.6 Atranka

Atranka yra individų grupės atrinkimas iš populiacijos. Dalis individų yra atrenkami, kad liktų nepakitę jų genai naujoje generacijoje. Šiame tyrime yra atrenkamas iš anksto numatytas kiekis geriausių individų ir iš anksto numatytas kiekis atsitiktinių individų.

4.7 Naujos kartos populiacijos sudarymas

Nauja populiacija sudaroma apjungiant kryžminimo, mutavimo ir atrankos metu gautus individus į vieną aibę.

5. Išvedimo strategijų paieška naudojant genetinį algoritmą

5.1 Infrastruktūra

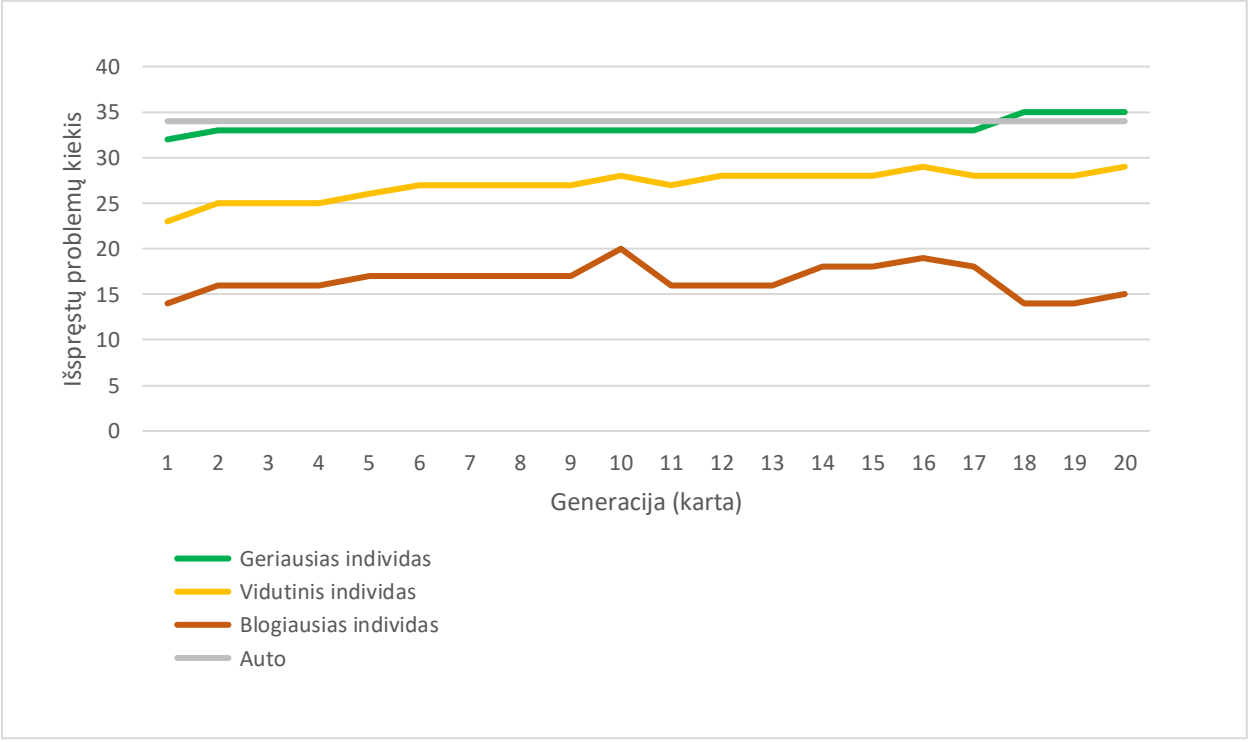
Tyrimui buvo naudojama VU MIF infrastruktūra – virtuali mašina su *Linux Debian 12* operacine sistema ir techniniais parametrais: 2-iem realaus procesoriaus branduoliais, 4-iais virtualaus procesoriaus branduoliais, 10 GB operatyvios atminties ir 43 GB kietojo disko atminties.

5.2 Eksperimentas nr. 1

Evoliucija buvo atliekama nurodant jai tokius parametrus:

- generacijų kiekis: 20;
- populiacijos dydis: 100;
- sprendžiamų problemų kiekis: 50;
- individo parinkimo kryžminimui tikimybė: 0.15;
- individo mutavimo tikimybė: 0.1;
- svorio funkcijos mutavimo tikimybė: 0.5;
- svorio funkcijos parametro mutavimo tikimybė: 0.3;
- atrenkamų geriausių individų kiekis populiacijoje: 0.1;
- minimalus svorio funkcijų kiekis individe: 3;
- maksimalus svorio funkcijų kiekis individe: 10.

Individų tinkamumo kitimo kreives galima peržiūrėti 4 paveiksliuke.



4 pav. *Pirmo eksperimento individų tinkamumo kitimo kreivės*

AĮS „E“ sprendžiant problemą (formulę) leidžia nurodyti *auto* režimą – pritaikyti geriausią sistemai žinomą strategiją problemoms spręsti. Iš pateiktų 50 problemų *auto* režimu sistema išsprendė 34. Evoliucijos metu 18-oje kartoje buvo gautas individas, kuris išsprendė 35 problemas (3-iame paveiksliuke geriausio individo kreivė pažymėta žalia spalva). Šio individo strategija:

[['Clauseweight', 8, 'ByCreationDate', 1, 1, 1.9007780960436236], ['Clauseweight', 6, 'ConstPrio', 1, 1, 0.8455270733242315], ['ClauseCMaxWeight', 10, 'PreferUnitGroundGoals', 1, 1, 0.3346462626538935], ['ClauseWeightAge', 4, 'ConstPrio', 5, 19, 13, 11], ['ClauseWeightAge', 4, 'ConstPrio', 7, 19, 19, 16], ['RelevanceLevelWeight2', 3, 'SimulateSOS', 1, 3, 1, 2, 458, 528, 282, 27, 2.942175799978793, 1.238297862090133, 2], ['ConjectureSymbolWeight', 9, 'ConstPrio', 43, 37, 19, 49, 15, 1.2100454037318302, 3.0030960296545666, 4.937428765640723]].

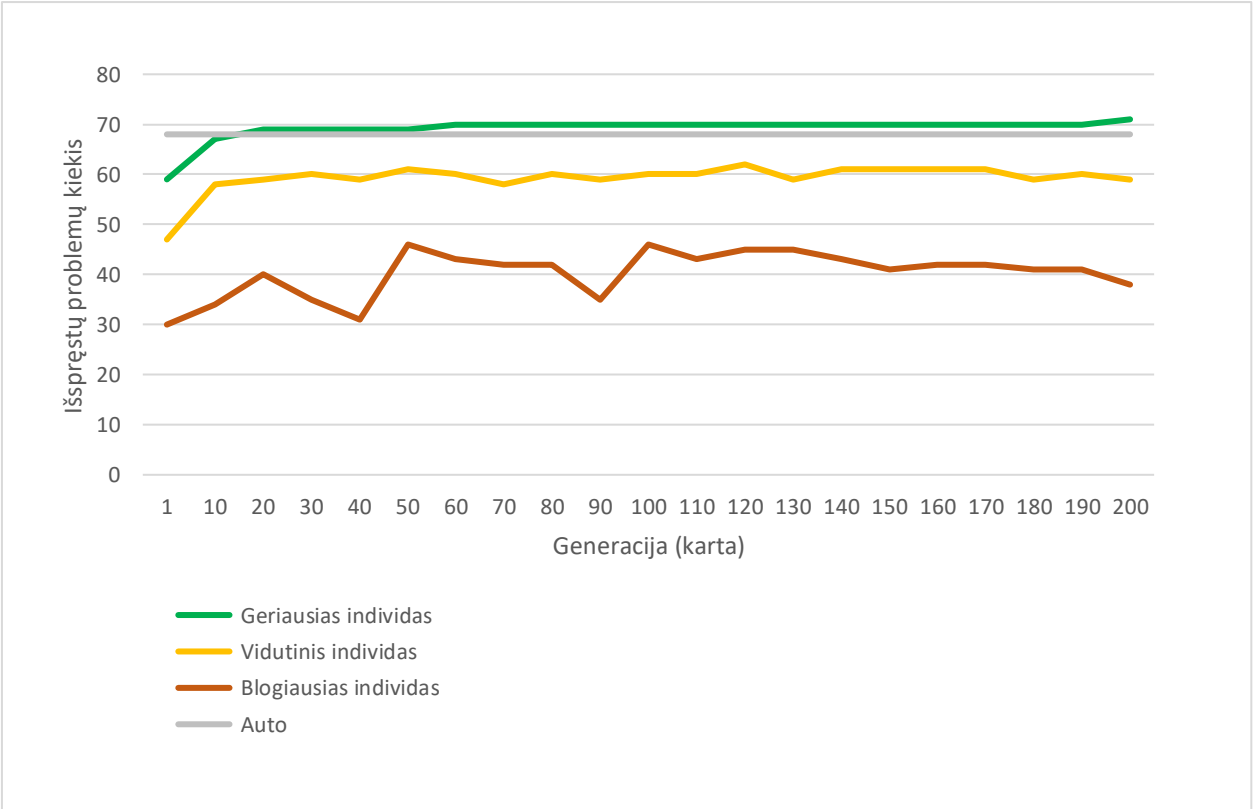
Evoliucijos vykdymas užtruko 5 valandas 22 minutes su anksčiau minėtais resursais.

5.3 Eksperimentas nr. 2

Antrame eksperimente evoliucijos parametrai yra panašūs į pirmojo eksperimento, tačiau skiriasi problemų kiekis ir generacijų kiekis:

- generacijų kiekis: 200;
- sprendžiamų problemų kiekis: 100.

Individų tinkamumo kitimo kreives galima peržiūrėti 5 paveiksliuke.



5 pav. *Antro eksperimento individų tinkamumo kitimo kreivės*

Antrame eksperimente geriausias individas (strategija), kuris aplenkia *auto* režimą, buvo gautas 16-oje kartoje. Jis išsprendė 69 problemas, kai tuo tarpu *auto* režimas 68 problemas. Tęsiant evoliuciją 199-oje kartoje buvo gautas pats geriausias individas, kuris išsprendė 71 problemą. Toliau pateikiama jo strategija:

[['PNRefinedweight', 1, 'SimulateSOS', 3, 4, 1, 4, 5, 0, 0.1], ['ConjectureRelativeSymbolWeight', 3, 'SimulateSOS', 0.9074636809836559, 86, 406, 290, 83, 3.7464135609796276, 2.682528097307579, 2], ['PNRefinedweight', 1, 'PreferGoals', 1, 12, 7, 7, 8, 1, 10.397783251349011], ['ClauseLMaxWeight', 7, 'ConstPrio', 0, 0, 0.0], ['ConjectureRelativeSymbolWeight', 3, 'ConstPrio', 0.0, 98, 212, 346, 4, 3.9857759227730765, 1.2735717934828743, 2], ['Clauseweight', 11, 'PreferProcessed', 1, 1, 0.1382281048166647], ['ConjectureRelativeSymbolWeight', 1, 'PreferNonGoals', 0.02624944708389443, 196, 241, 215, 23, 4.770433378869118, 1.2735717934828743, 2], ['ConjectureRelativeSymbolWeight', 5, 'PreferNonGoals', 0.3577731487614044, 128, 57, 400, 48, 7.008613891247519, 1.5952901280634364, 1], ['StaggeredWeight', 11, 'ConstPrio', 0]].

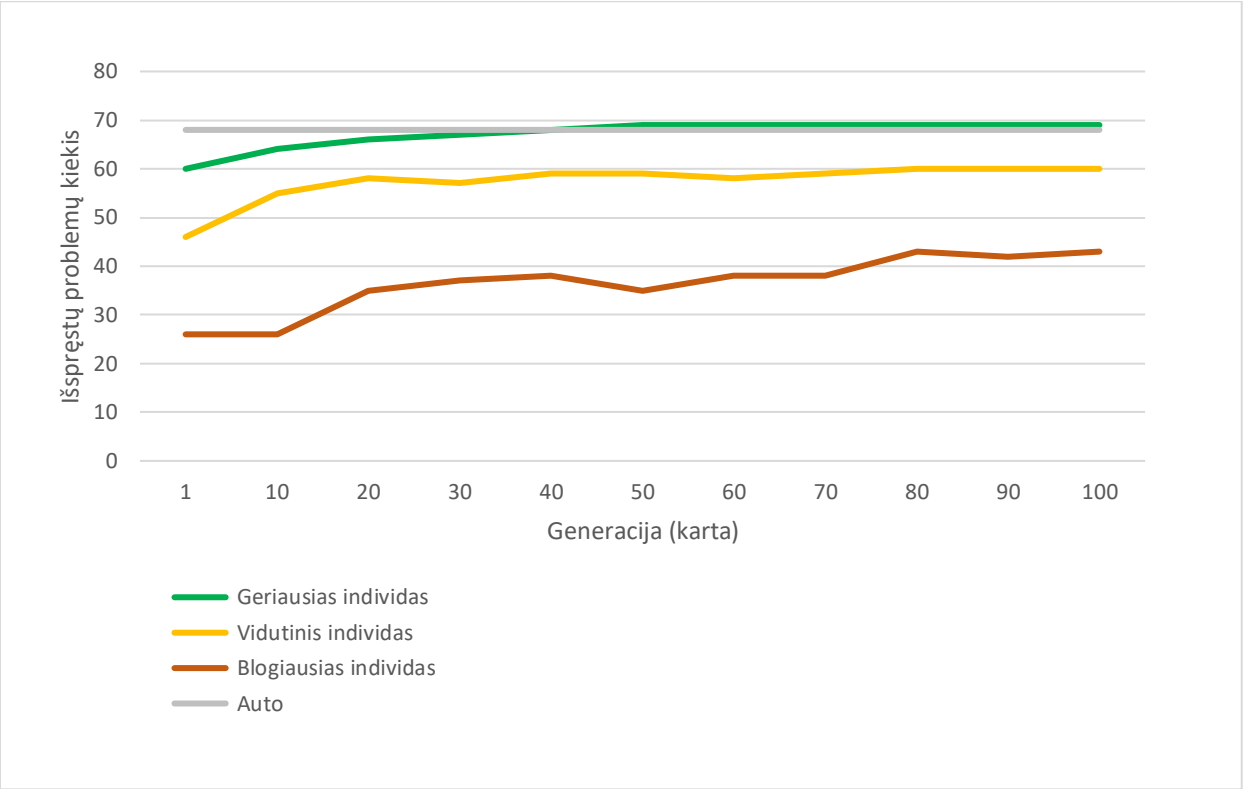
Evoliucijos vykdymas užtruko 86 valandas (3 paras ir 14 valandų).

5.4 Eksperimentas nr. 3

Trečiame eksperimente evoliucijos parametrai yra panašūs į antrojo eksperimento, tačiau skiriasi populiacijos dydis ir generacijų kiekis:

- generacijų kiekis: 100;
- populiacijos dydis: 200.

Individų tinkamumo kitimo kreives galima peržiūrėti 6 paveiksliuke.



6 pav. *Trečio eksperimento individų tinkamumo kitimo kreivės*

Trečiame eksperimente geriausias individas (strategija), kuris aplenkė *auto* režimą, buvo gautas 46-oje kartoje. Jis išsprendė 69 problemas, kai tuo tarpu *auto* režimas 68 problemas (*auto* režimu išspręstų problemų kiekis toks pat kaip ir antrame eksperimente, nes buvo naudojamas tas pats problemų sąrašas). Tęsiant evoliuciją ir sukūrus 100 kartų (generacijų) geresnis individas nebuvo gautas. Toliau pateikiama geriausio individo strategija:

[['FIFOWeight', 3, 'PreferProcessed'], ['Refinedweight', 9, 'PreferNonGoals', 7, 3, 3, 2.608738681882051, 1.2009749844449762], ['RelevanceLevelWeight2', 9, 'SimulateSOS', 0, 3, 1, 5, 194, 377, 470, 375, 1.612795485327435, 2.552245181447268, 1], ['ConjectureGeneralSymbolWeight', 2, 'SimulateSOS', 134, 177, 360, 42, 123, 1, 127, 2.5325545016594893, 1.6673963682314896, 1], ['Refinedweight', 1, 'PreferGoals', 7, 5, 1, 3.126398476153193, 1.1332669394712431], ['ConjectureSymbolWeight', 2, 'ConstPrio', 89, 102, 2, 32, 18, 4.18477204660184, 0.23700115817158474, 0.3283548769961686], ['PNRefinedweight', 2, 'PreferNonGoals', 5, 7, 7, 2, 4, 11, 5.263783016813847], ['RelevanceLevelWeight2', 9, 'SimulateSOS', 1, 3, 0, 4, 270, 253, 105, 278, 2.3539740560211797, 2.552245181447268, 1]].

Evoliucijos vykdymas užtruko 95 valandas ir 31 minutę (3 paras, 23 valandas ir 31 minutę).

6. Strategijų palyginimas

Šio darbo 2.3.3 skyriuje yra aptariamos strategijos, kurios buvo naudojamos kitų autorių įvairiuose mašininio mokymo taikymo tyrimuose. Šiame skyriuje kitų autorių išskirtos strategijos yra lyginamos su genetiniu algoritmu gauta geriausia strategija „GA strategija“. Visos strategijos pateikiamos 6 lentelėje.

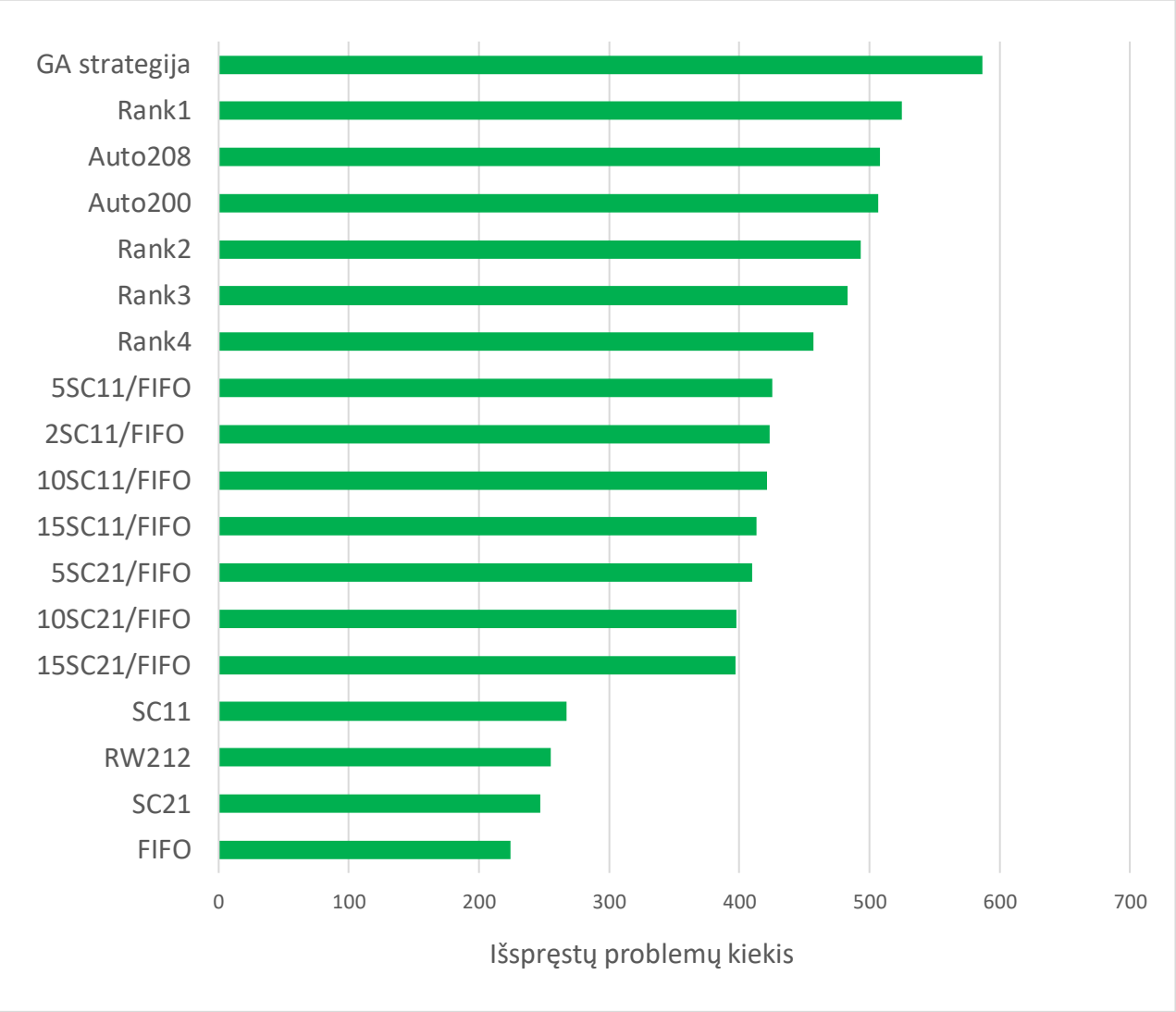
6 lentelė. *Lyginamos strategijos*

Pavadinimas	**Strategija**	**Šaltinis**
Auto208	1*ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1), 4*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5), 1*FIFOWeight(PreferProcessed), 1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1), 4*Refinedweight(SimulateSOS,3,2,2,1.5,2)	[LIS+17]
Auto200	1*ConjectureRelativeSymbolWeight(SimulateSOS,0.5,100,100,100,100,1.5,1.5,1), 6*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100,100,1.5,1.5,1.5), 2*FIFOWeight(PreferProcessed), 1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.5,100,100,100,100,1.5,1.5,1), 8*Refinedweight(SimulateSOS,1,1,2,1.5,2)	[LIS+17]
Rank1	4*ConjectureGeneralSymbolWeight(SimulateSOS,100,100,100,50,50,10,10,1.5,1.5,1), 3*ConjectureGeneralSymbolWeight(PreferNonGoals,200,100,200,50,50,1,100,1.5,1.5,1), 1*Clauseweight(PreferProcessed,1,1,1), 1*FIFOWeight(PreferProcessed)	[Bri10]
Rank2	8*Refinedweight(PreferGoals,1,2,2,2,2), 8*Refinedweight(PreferNonGoals,2,1,2,2,0.5), 1*Clauseweight(PreferUnitGroundGoals,1,1,1), 1*FIFOWeight(ConstPrio)	[Bri10]

Rank3	10*PNRefinedweight(PreferGoals,1,1,1,2,2,2,0.5), 10*PNRefinedweight(PreferNonGoals,2,1,1,1,2,2,2), 5*OrientLMaxWeight(ConstPrio,2,1,2,1,1), 1*FIFOWeight(ConstPrio)	[Bri10]
Rank4	10*ConjectureRelativeSymbolWeight(ConstPrio,0.1,100,100,100, 100,1.5,1.5,1.5), 1*FIFOWeight(ConstPrio)	[Bri10]
FIFO	1*FIFOWeight(ConstPrio)	[SM16]
SC11	1*Clauseweight(ConstPrio,1,1,1)	[SM16]
SC21	1*Clauseweight(ConstPrio,2,1,1)	[SM16]
2SC11/FIFO	2*Clauseweight(ConstPrio,1,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
5SC21/FIFO	5*Clauseweight(ConstPrio,2,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
5SC11/FIFO	5*Clauseweight(ConstPrio,1,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
10SC21/FIFO	10*Clauseweight(ConstPrio,2,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
10SC11/FIFO	10*Clauseweight(ConstPrio,1,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
15SC21/FIFO	15*Clauseweight(ConstPrio,2,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
15SC11/FIFO	15*Clauseweight(ConstPrio,1,1,1), 1*FIFOWeight(ConstPrio)	[SM16]
RW212	1*Refinedweight(ConstPrio,2,1,2,1,1)	[SM16]
GA strategija	1*PNRefinedweight(SimulateSOS,3,4,1,4,5,0,0.1), 3*ConjectureRelativeSymbolWeight(SimulateSOS,0.9074636809 836559,86,406,290,83,3.7464135609796276,2.682528097307579, 2), 1*PNRefinedweight(PreferGoals,1,12,7,7,8,1,10.39778325134901 1), 7*ClauseLMaxWeight(ConstPrio,0,0,0.0),	Gauta šiame tyrime

	3*ConjectureRelativeSymbolWeight(ConstPrio,0.0,98,212,346,4,3.9857759227730765,1.2735717934828743,2), 11*Clauseweight(PreferProcessed,1,1,0.1382281048166647), 1*ConjectureRelativeSymbolWeight(PreferNonGoals,0.02624944708389443,196,241,215,23,4.770433378869118,1.2735717934828743,2), 5*ConjectureRelativeSymbolWeight(PreferNonGoals,0.3577731487614044,128,57,400,48,7.008613891247519,1.5952901280634364,1), 11*StaggeredWeight(ConstPrio,0)	

Strategijų palyginimui atsitiktiniu būdu buvo parinkta 1000 TPTP problemų. Eksperimentų metu gauti rezultatai pateikiami 7 paveiksliuke.



7 pav. *Strategijų palyginimo diagrama*

Eksperimentų rezultatai parodė, kad daugiausiai problemų (587 iš 1000) sugebėjo išspręsti genetiniu algoritmu gauta strategija „GA strategija“. Taip pat buvo pastebėtas dėsningumas – didėjant svorio funkcijų kiekiui strategijoje, didėja išspręstų problemų kiekis. Mažiausiai problemų išsprendė strategijos (FIFO, SC21, RW212 ir SC11), kuriose naudojama po vieną svorio funkciją. Jas naudojant vidutiniškai išsprendžiama 248 iš 1000 problemų. Jei strategijoje naudojamos dvi svorio funkcijos, rezultatai reikšmingai pagerėja – išsprendžiama 418 problemų. Geriausi rezultatai pasiekiami, kai strategijoje sujungiamos keturios ar daugiau funkcijų. Tuomet vidutiniškai išsprendžiama 517 problemų.

Rezultatai ir išvados

Tyrimo metu buvo sukurta programa, kuri leidžia modeliuoti evoliucijas, naudojant genetinį algoritmą. Taip pat buvo pasirinkta TPTP problemų (formulių) bazė, kuri leido įvertinti individų tinkamumą. Atlikus evoliucijų eksperimentus, kurių trukmė nuo 5 iki 95 valandų, paaiškėjo, kad geriausias individas (strategija), aplenkiantis automatinės įrodymo sistemos „E“ automatinę strategiją, gaunamas:

- 18-oje kartoje, jei individų populiacijos dydis yra 100, o sprendžiamų problemų kiekis – 50;
- 16-oje kartoje, jei individų populiacijos dydis yra 100, o sprendžiamų problemų kiekis – 100;
- 46-oje kartoje, jei individų populiacijos dydis yra 200, o sprendžiamų problemų kiekis – 100.

Pats geriausias individas tarp visų eksperimentų sugebėjo išspręsti 71 problemą iš 100, kai tuo tarpu sistemos „E“ automatinė strategija išsprendė 68 problemas. Toliau pateikiama geriausio individo strategija:

[['PNRefinedweight', 1, 'SimulateSOS', 3, 4, 1, 4, 5, 0, 0.1], ['ConjectureRelativeSymbolWeight', 3, 'SimulateSOS', 0.9074636809836559, 86, 406, 290, 83, 3.7464135609796276, 2.682528097307579, 2], ['PNRefinedweight', 1, 'PreferGoals', 1, 12, 7, 7, 8, 1, 10.397783251349011], ['ClauseLMaxWeight', 7, 'ConstPrio', 0, 0, 0.0], ['ConjectureRelativeSymbolWeight', 3, 'ConstPrio', 0.0, 98, 212, 346, 4, 3.9857759227730765, 1.2735717934828743, 2], ['Clauseweight', 11, 'PreferProcessed', 1, 1, 0.1382281048166647], ['ConjectureRelativeSymbolWeight', 1, 'PreferNonGoals', 0.02624944708389443, 196, 241, 215, 23, 4.770433378869118, 1.2735717934828743, 2], ['ConjectureRelativeSymbolWeight', 5, 'PreferNonGoals', 0.3577731487614044, 128, 57, 400, 48, 7.008613891247519, 1.5952901280634364, 1], ['StaggeredWeight', 11, 'ConstPrio', 0]].

Lyginant genetiniu algoritmu gautą geriausią strategiją su kitų autorių išskirtomis strategijomis, genetiniu algoritmu gauta strategija sugebėjo išspręsti daugiausiai problemų (587 iš 1000). Taip pat buvo pastebėtas dėsningumas – didėjant svorio funkcijų kiekiui strategijoje, didėja išspręstų problemų kiekis. Eksperimentai parodė, kad geriausi rezultatai pasiekiami, kai strategijoje sujungiamos keturios ar daugiau funkcijų.

Šiame tyrime buvo ieškoma vienos strategijos, kuri išspręstų daugiausiai problemų. Ir genetiniu algoritmu gauta strategija pasirodė esanti pakankamai gera. Tačiau apžvelgtuose kitų autorių tyrimuose yra bandoma surasti ne tik geriausią strategiją, bet ir sukurti sistemą, kuri kiekvienai problemai individualiai parinktų jos sprendimui tinkamiausią strategiją. Tokios

sistemos yra efektyvesnės, sugeba išspręsti daugiau problemų nei bet kuri viena strategija atskirai. Todėl tolimesni tyrimai galėtų būti nukreipti į tokių sistemų kūrimą.

Literatūra ir šaltiniai

[Bri10] J. P. Bridge. Machine learning and automated theorem proving. Technical Report, number 792, University of Cambridge, computer laboratory. [prieiga tikrinta 2025-05-25].
URL: <https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-792.pdf>

[CJS+19] K. Chvalovsky, J. Jakubuv, M. Suda, J. Urban. ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E. arXiv:1903.03182v1, 2019. [prieiga tikrinta 2025-05-25].
URL: < https://arxiv.org/pdf/1903.03182v1>

[CJO+21] K. Chvalovsky, J. Jakubuv, M. Olšak, J. Urban. Learning Theorem Proving Components. arXiv:2107.10034, 2021. [prieiga tikrinta 2025-05-25].
URL: < https://arxiv.org/pdf/2107.10034>

[Dou00] N. A. Douglas. The explosion of Ariane 5. [prieiga tikrinta 2025-05-25].
URL: <http://www-users.math.umn.edu/~arnold/disasters/ariane.html>

[Eth] Internetinis puslapis. The E Theorem Prover. [prieiga tikrinta 2025-05-25].
URL: <https://wwwlehre.dhbw-stuttgart.de/~sschulz/E/E.html>

[Gol89] D. E. Goldber. Genetic Algorithms in Search, Optimizations and Machine Learning. Addison-Wesley Publishing Company, Inc, 1989. [prieiga tikrinta 2025-05-25].
URL:<http://www2.fiit.stuba.sk/~kvasnicka/Free%20books/Goldberg_Genetic_Algorithms_in_Search.pdf>

[JU16] J. Jakubuv, J. Urban. BliStrTune: Hierarchical Invention of Theorem Proving Strategies. arXiv:1611.08733, 2016. [prieiga tikrinta 2025-05-25].
URL: <https://arxiv.org/pdf/1611.08733>

[JU17] J. Jakubuv, J. Urban. ENIGMA: Efficient Learning-based Inference Guiding Machine. arXiv:1611.08733, 2017. [prieiga tikrinta 2025-05-25].
URL: <https://arxiv.org/pdf/1701.06532>

[JPK19] Sanjay Jangid, Ravinder Puri, Thilak Kumar. Evolutionary Algorithms: A Critical Review and its Future Prospects. Iš *Conference Proceeding Issue Published in International Journal of Trend in Research and Development (IJTRD)*, Bangalore, 23 March, 2019. [prieiga tikrinta 2025-05-25].
URL: <https://www.ijtrd.com/papers/IJTRD20409.pdf>

[KSU13] D. Kuhlwein, S. Schulz, J. Urban. E-MaLeS 1.1. Iš *Proceedings of the 24th international conference on Automated Deduction, Lecture Notes in Computer Science* 7898:407-413, 2013. [prieiga tikrinta 2025-05-25].

URL: <https://wwwlehre.dhbw-stuttgart.de/~sschulz/PAPERS/KUS-CADE-2013.pdf>

[LIS+17] S. Loos, G. Irving, C. Szegedy, C. Kaliszyk. Deep Network Guided Proof Search. arXiv:1701.06972, 2017. [prieiga tikrinta 2025-05-25].
URL: <https://arxiv.org/pdf/1701.06972>

[Mck24] J. McKeown. Search strategy selection for automated theorem proving. A dissertation. 2024. [prieiga tikrinta 2025-05-25].
URL: <https://scholarship.miami.edu/esploro/outputs/doctoral/Search-Strategy-Selection-for-Automated-Theorem/991032485170602976#file-0>

[Men13] J. Magalhaes-Mendes. A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem. Iš *WSEAS TRANSACTIONS on COMPUTERS*, Issue 4, Volume 12, April 2013. [prieiga tikrinta 2025-05-25].
URL: <https://www.wseas.org/multimedia/journals/computers/2013/5705-156.pdf>

[Mit97] T. M. Mitchell. Machine Learning. McGraw-Hill Science/Engineering/Math, 432 p., 1997. [prieiga tikrinta 2025-05-25].
URL: <https://www.cs.cmu.edu/~tom/files/MachineLearningTomMitchell.pdf>

[MS23] J. McKeown, G. Sutcliffe. Reinforcement Learning for Guiding the E Theorem Prover. Iš *The International FLAIRS Conference Proceedings*, Vol. 36(1), 2023. [prieiga tikrinta 2025-05-25].
URL: <https://journals.flvc.org/FLAIRS/article/view/133334/137663>

[MS25] J. McKeown, G. Sutcliffe. Dataset-Specific Strategies for the E Theorem Prover. Iš *Proceedings of the 14th and 15th International Workshops on the Implementation of Logics*, Volume 21, 2025, Pages 34–44. [prieiga tikrinta 2025-05-25].
URL: <https://easychair.org/publications/paper/PXgt/open>

[Nor07] S. Norgėla. *Logika ir dirbtinis intelektas*. TEV, Vilnius, 2007, p. 8-9, 118-131, 158-163.

[Rob65] J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. Iš *Journal of the Association for Computing Machinery*, Vol. 12, No. 1 (January, 1965), pp. 23-41. [prieiga tikrinta 2025-05-25].
URL: <https://dl.acm.org/doi/pdf/10.1145/321250.321253>

[Sch01] S. Schulz. Learning Search Control Knowledge for Equational Theorem Proving. Iš *Proc. of the Joint German/Austrian Conference on Artificial Intelligence* (KI-2001), Vienna, Austria, LNAI 2174, 2001, p. 320-334, p. 370-375, Springer.

[Sch02] S. Schulz. E – A Brainiac Theorem Prover. Iš *Journal of AI communications*, Volume 15(2/3), 2002, p. 111-126.

[Sch06] S. Schulz. Algorithms and Data Structures for First-Order Equational Deduction. [prieiga tikrinta 2025-05-25].
URL: <https://www.researchgate.net/publication/228735705>

[Sch24] S. Schulz. E 3.1 User Manual. [prieiga tikrinta 2025-05-25].
URL: <https://wwwlehre.dhbw-stuttgart.de/~sschulz/WORK/E_DOWNLOAD/V_3.1/eprover.pdf>

[SS15] S. Schafer, S. Schulz. Breeding Theorem Proving Heuristics with Genetic Algorithms. Iš *EpiC Series in Computer Science*, Volume 36, 2015, p. 263-274. [prieiga tikrinta 2025-05-25].
URL: <http://wwwlehre.dhbw-stuttgart.de/~sschulz/PAPERS/ss_gcai2015.pdf>

[SM16] S. Schultz, M. Mohrmann. Performance of Clause Selection Heuristics for Saturation-Based Theorem Proving. Iš *Proceedings of the 8th IJCAR*, Coimbra, volume 9706 of LNAI. Springer, 2016. [prieiga tikrinta 2025-05-25].
URL: <https://wwwlehre.dhbw-stuttgart.de/~sschulz/PAPERS/sm_ijcar-2016.pdf>

[Sut10] G. Sutcliffe. The TPTP World – Infrastucure for Automated Reasoning. Iš *Logic for Programming, Artificial Inteligence, and Reasoning*, LNAI 6355, 2010, p. 1-10, Springer.

[TPT] Internetinis puslapis. The TPTP Problem Library for Automated Theorem Proving. [prieiga tikrinta 2025-05-25].
URL: <https://tptp.org/TPTP/>

[US15] A. J. Umbarkar, P. D. Sheth. Crossover operators in genetic algorithms: a review. Iš *ICTACT Journal on Soft Computing*, October 2015, Volume: 06, Issue: 01. [prieiga tikrinta 2025-05-25].
URL:<https://pdfs.semanticscholar.org/6888/7a8e96440f9e69c015c923a83455478f0290.pdf>

[Urb13] J. Urban. BliStr: The Blind Strategymaker. arXiv:1611.08733, 2013. [prieiga tikrinta 2025-05-25].
URL: <https://arxiv.org/pdf/1301.2683>