VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Kursinis projektas

Neuro-evoliucija pritaikyta valdikliui

(Neuro-evolution for evolving a controller)

Atliko: 4 kurso 1 grupės studentas
Laimonas Beniušis (parašas)

Darbo vadovas:
Linas Litvinas (parašas)

Vilnius – 2018

Turinys

Įvadas

Dirbtinis neuronų tinklas (angl. *Artificial Neural Network*) gali būti pritaikomas įvairiems uždaviniams spręsti, kaip simbolių atpažinimas, paveikslėlių kategorizavimas ar dirbtinio intelekto mokymas. Didžioji tokių uždavinių dalis yra struktūrų atpažinimas (angl. *pattern recognition*), ką žmonės geba atlikti natūraliai.

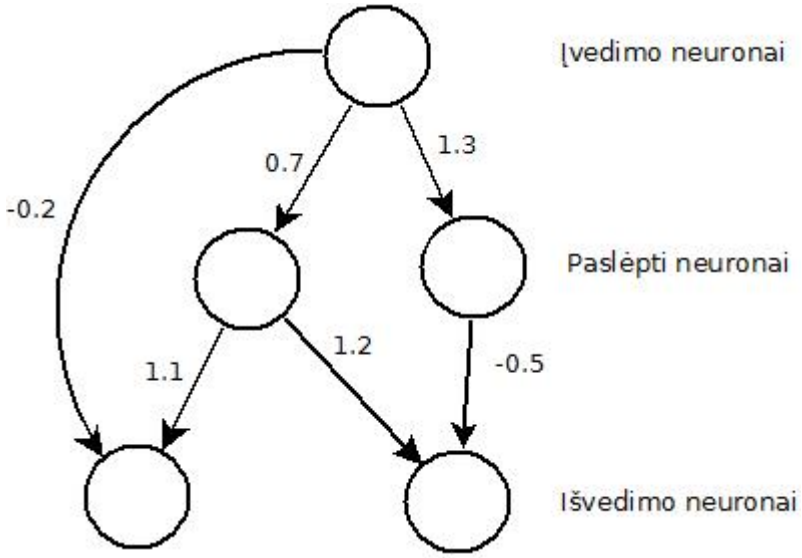
Populiarus „YouTube“ vaizdo įrašas pavadinimu „MarI/O - Machine Learning for Video Games“ [Set15], pritraukė didžiulį susidomėjimą genetiniais algoritmais ir dirbtiniais neuronų tinklais. Šiame vaizdo įraše dirbtinis neuronų tinklas genetinės evoliucijos būdu apmokomas greitai įveikti pirmąjį „Super Mario World“ žaidimo lygį.

Šiame darbe yra plačiau aptariamas ir panaudojamas kursinio darbo metu pristatytas genetinis algoritmas dirbtiniams neuronų tinklams, bei sudėtingesnės jo variacijos. Sprendžiama dirbtinio neuronų tinklo pritaikymo kaip valdiklio (angl. *Controller*) problema.

1 Dirbtiniai neuronų tinklai (DNT)

Tinklą sudaro neuronai (grafo viršūnės) ir jungtys. DNT tinklo struktūra:

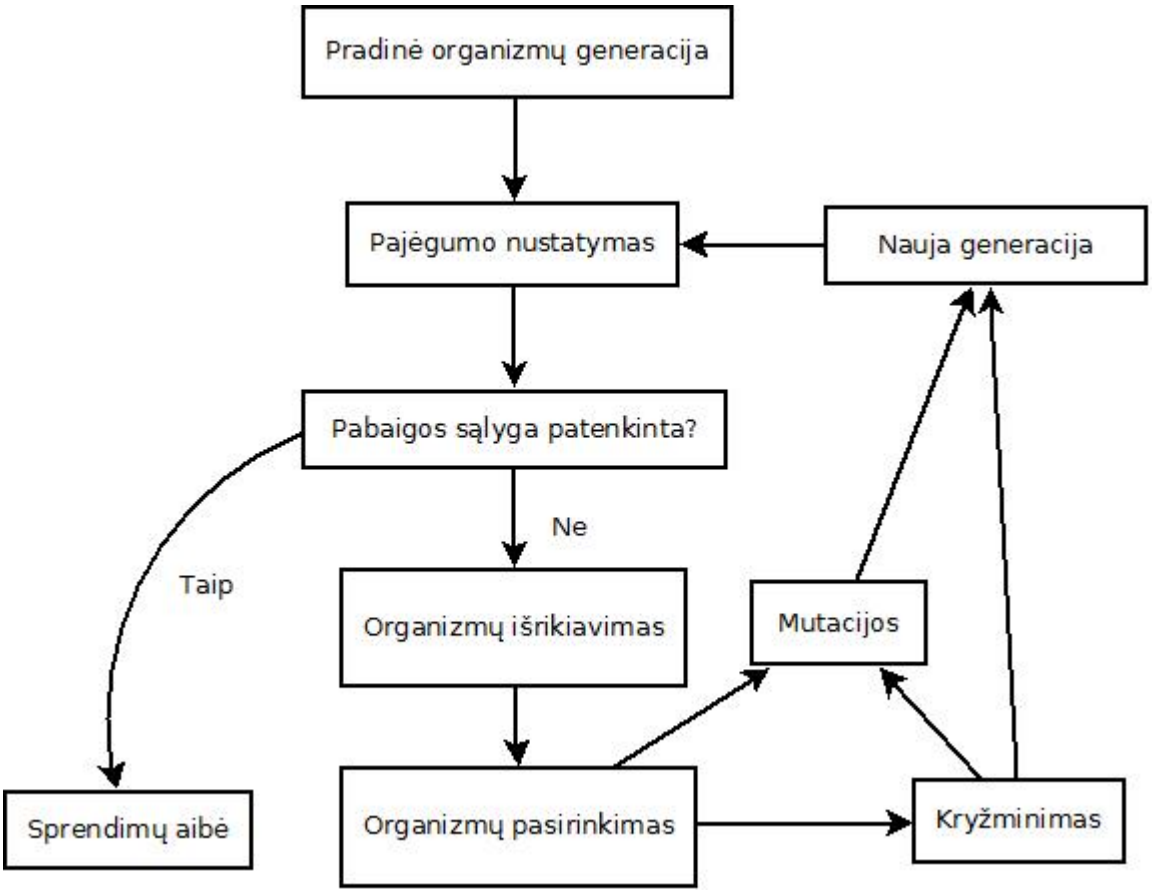
- Nekintantis kiekis įvedimo neuronų
- Nekintantis kiekis išvedimo neuronų
- Dinaminis arba nekintantis kiekis paslėptų neuronų
- Dinaminis arba nekintantis kiekis paslėptų neuronų sluoksnių
- Kiekvienas neuronas turi aktyvacijos funkciją ir (nebūtina) slenksčio reikšmę
- Dinaminis arba nekintantis kiekis jungčių, kurios apjungia neuronus
- Kiekviena jungtis turi svorio reikšmę



1 pav. Tiesioginio išvedimo dirbtinis neuronų tinklas.

2 Genetiniai algoritmai (GA)

Genetiniai algoritmai (toliau GA) yra evoliucinių algoritmų poklasis. Evoliuciniai algoritmai simuliuoja organizmų evoliuciją ir panašiai kaip gamtoje, išlieka stipriausi. Tai yra metaeuristinis optimizavimo metodas. Generuojant naują organizmų aibę yra naudojami įvairūs metodai kaip kryžminimas (angl. crossover) ar paprasčiausias klonavimas. Evoliucijoje dalyvaujantys organizmus galima vadinti genomais (genotipais). Genotipas yra genų sąrašas, o vizuali genotipo forma arba genų realizacija yra fenotipas [Phi94, 11][KW16, 3].



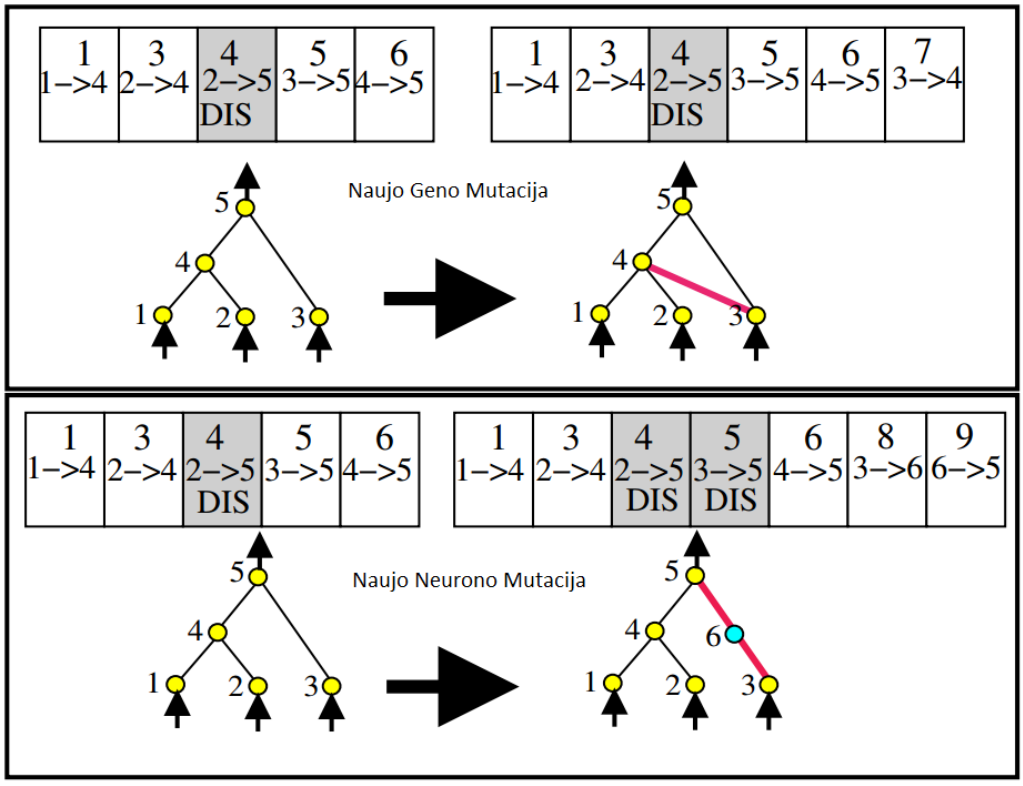
2 pav. Genetinio algoritmo veiksmų seka.

3 Neuro-Evoliucija Augančioms Topologijoms (NEAT)

Augančių topologijų neuro-evoliucija (angl. *Neuro Evolution of Augmenting Topologies*) yra neuro-evoliucinis algoritmas, sukurtas Kenneth O. Stanley ir Risto Miikkulainen [KR02]. Šis algoritmas minimizuoja topologijas evoliucijos metu, o ne tik jos pabaigoje, nenaudojant specialios funkcijos, kuri matuoja DNT sudėtingumą. Taip pat DNT struktūros sudėtingėjimas koreliuoja su jo sprendinio korektiškumu, t. y. nėra nereikalingų neuronų ar jungčių.

3.1 Mutacijos

Kieviena naujo geno sukūrimo mutacija turi savo inovacijos žymę (viršuje). Vienodos genų mutacijos skirtinguose genomuose įgauna vienodą inovacijos žymę. Taip pat yra naudojamos standartinės neuro-evoliucijos mutacijos, kurios keičia lankų svorius.

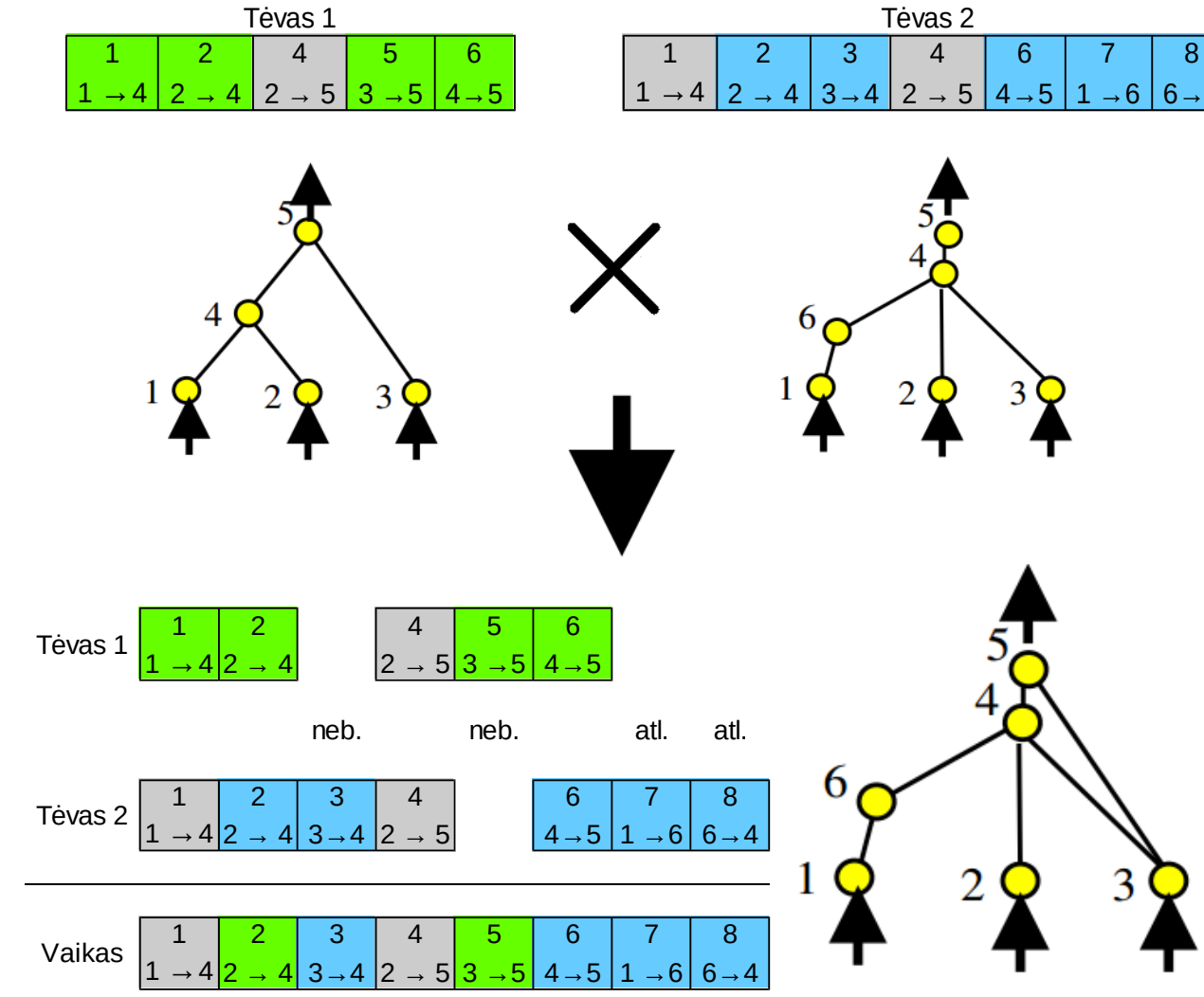


3 pav. NEAT algoritme naudojamos mutacijos [Ken04, 36]. Pilki genai yra išjungti. Naujo neurono atveju senas genas išjungiamas, o jo vietoje atsiranda 2 nauji genai, sujungti naujo neurono. Mutacijų metu genai gali būti išjungiami.

3.2 Kryžminimas

Nors ir tėvų fenotipai skiriasi, inovacijos žymės (geno viršuje) parodo sutampančius genus, kuriuos suporavus galima vykdyti kryžminimą [Ken04, 37]. Naudoti žymėjimai:

- neb. (nebendri) genai yra tie, kurių inovacijos žymės neviršija mažesniojo tėvo didžiausiosios.
- atl. (atliekami) genai yra tie, kurie viršija mažesniojo tėvo didžiausiąją inovacijos žymę.
- pilki genai yra išjungti



4 pav. Skirtingų DNT kryžminimas pagal inovacijos žymes.

3.3 Konkuravimo paskirtymas rasėmis

Dažniausiai, įterpiant naują atsitiktinį geną į genomą sumažėja jo pajėgumas, todėl globalioje populiacijoje modifikuotas genomas ilgai neišliks. Dėl šios priežasties yra įvedama lokali populiacija (angl. *species*), kurios genomai yra homologiškai panašūs ir konkuriuoja tarpusavyje, o ne globaliai.Tačiau kaip atskirti ar genomai yra iš tos pačios rasės? Inovacijos žymės suteikia paprastą būdą šiai problemai spręsti. Genomų panašumo funkcija:

(1) $$d = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 W$$

E - atliekamų (angl. *excess*) genų skaičius, D - nebendrų (angl. *disjoint*) genų skaičius, W - sutampančių genų svorių skirtumų vidurkis, N - didesniojo genomo genų skaičius, c_1, c_2, c_3 nustato šių daugiklių įtaką. „Kuo mažiau yra atliekamų ir nebendrų genų genomuose, tuo jie yra artimesni evoliucinės istorijos prasme“ [KR02, 112].

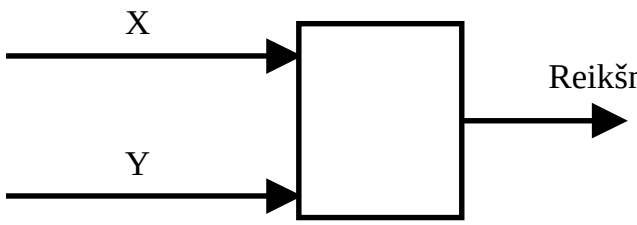
3.4 Informacijos abstrakcijos problema

Visa evoliucinių algoritmų idėja daugiau ar mažiau remiasi procesu, kuris vyko ir tebevyksta gamtoje. Stebinantis gamtos evoliucijos rezultatas paskatina algoritmus kurti modeliuojant jos procesus. Viena iš svaresnių problemų genetiniuose algoritmuose yra kodavimas. Kaip informacinėmis priemonėmis įgyvendinti tai, ką sugeba gyvuose organizmuose esantis DNR? T.y. užkoduojamos visos organizmo detalės palyginant mažame informacijos kiekyje.

4 Kompoziciniai ornamentus konstruojantys tinklai (CPPN)

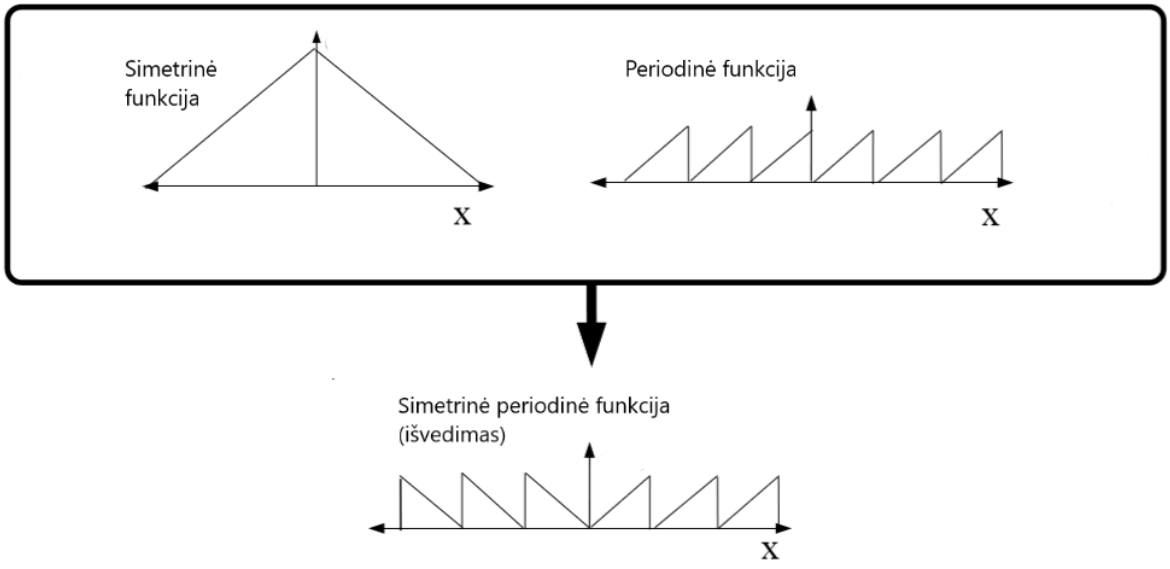
Tinkamos informacijos kodavimo abstrakcijos ieškojmas atvedė prie kompozicinių ornamentus konstruojančių tinklų (angl. *Compositional Patern Producing Networks*) išradimo. CPPN yra tiesioginio išvedimo dirbtinių neuronų tinklų klasė, kuri veikia kaip funkcija suteikianti erdvės taškui reikšmę. Tokiu būdu galima išsaugoti neribotos rezoliucijos paveikslėlį:

- Paduodamos koordinatės yra normalizuojamos [-1, 1], ar kitokiame intervale (centras yra 0)
- Galima įtraukti atstumo nuo centro reikšmę (lengviau atvaizduoti skritulius)
- Galima turėti kelias išvedimo reikšmes (ryškumui, spalvai, tamsumui)



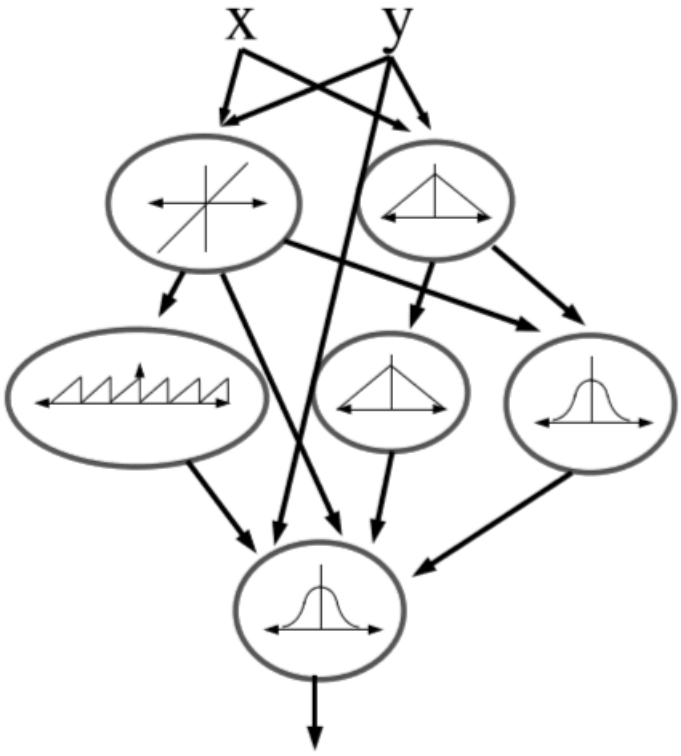
5. pav. Paprastas erdvės taško kodavimas.

Viena iš svarbiausių CPPN savybių yra aktyvacijos funkcijos. Skirtingos aktyvacijos funkcijos pagamina skirtingus dėsningumus. Kadangi tai yra tiesioginio išvedimo tinklas, skirtingos aktyvacijos funkcijų kompozicijos leidžia gauti dar sudėtingesnius ornamentus ar dėsningumus.



6. pav. Skirtingų funkcijų kompozicija.

Pagrindinė CPPN idėja yra aktyvacijos funkcijų eiliškumas. Tų pačių funkcijų skirtingas išsidėstymas kardinaliai keičia rezultatą [Ken07,11]. Natūraliai galima tokią kompoziciją įgyvendinti dirbtiniu neuronu tinklu. Taigi, dabar turime įrankį, kuris yra informacijos erdvėje abstrakcija. Tačiau kaip tai padeda su DNT?



7. pav. CPPN eskizas šaltinis:[Ken07,12]

5 Hiperkubinė Neuro-Evoliucija augančioms topologijoms (HyperNEAT)

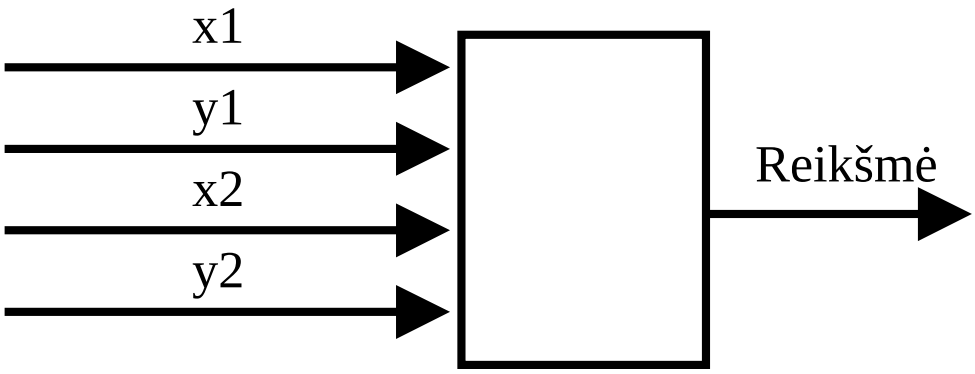
Ši NEAT algoritmo versija yra sukurta Kenneth O. Stanley, David D'Ambrosio ir Jason Gauci [KDJ09,NFAQ]. HyperNEAT (angl. *Hypercube-based NeuroEvolution of Augmenting Topologies*) algoritmo siekis yra išspręsti dvi DNT problemas:

- Efektyvi informacijos abstrakcija
- Erdvės suvokimas

Dauguma DNT neatsižvelgia į užduoties erdvę. Pvz.: Įvedimo aibė būna masyvas, suformuotas iš matomos lentos. Pats DNT mato tik pačias reikšmes, tačiau neturi suvokimo kaip arti viena kitos jos yra ar kai panašios reikšmės yra šalia. Dažniausiai tokie dėsningumai yra atrandami iš naujo tinklui sudėtingėjant ir besimokant, jeigu iš viso yra atrandami.

5.1 Erdvės dėsningumų transformavimas į DNT lankus

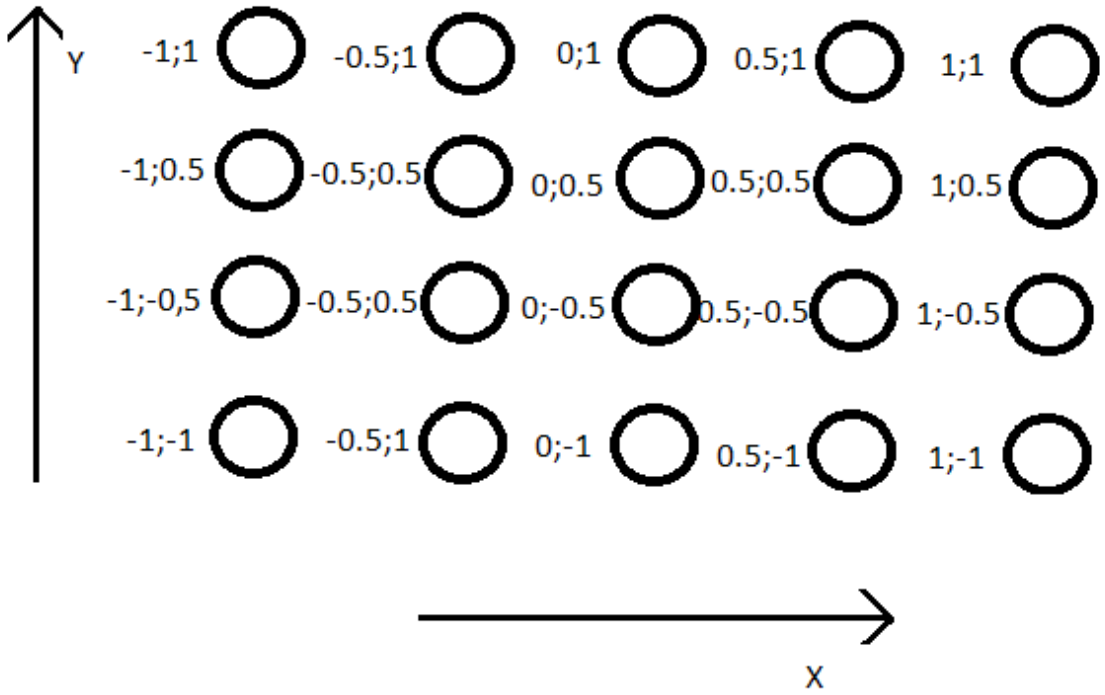
HyperNEAT algoritmas pasitelkia jau minėtu informacijos abstrakcijos įrankiu erdvėje: CPPN. Šiuo atveju, CPPN yra pagrindinis tinklas, kuris dalyvauja originaliam NEAT algoritme ir yra naudojamas kito tinklo generavimui. Žinoma, norint įgyvendinti kažką panašaus į 7 pav. būtina naujo tipo mutacija. Taigi, mutacijų sąrašą papildo aktyvacijos funkcijos tipo mutacija. Taip pat reikia CPPN pritaikyti naujo DNT generavimui. Pavyzdžiui, paprastas sluoksniuotas dviejų dimensijų DNT gali būti generuojamas 4 įvedimų CPPN pagalba.



8. pav. CPPN pritaikytas 2 dimensijų DNT generavimui

Tokiu atveju, DNT įgyja stačiakampio formą (pav. 9):

- Sluoksnių kiekis tampa stačiakampio ilgiu bei generuojamos figūros tikslumu (kuo daugiau sluoksnių, tuo didesnis tikslumas)
- Įvedimo/išvedimo kiekis tampa stačiakampio pločiu.



9. pav. 2 dimensijų stačiakampio metafora pritaikyta DNT su atitinkamomis koordinatėmis.

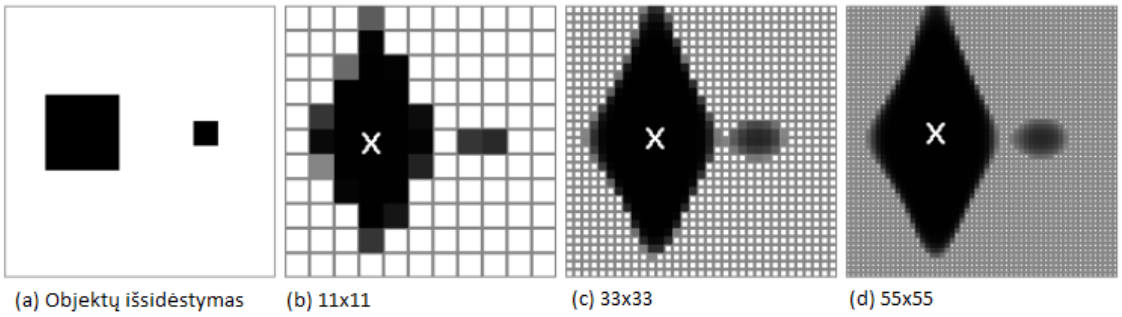
Taigi, dabar generuojamo DNT neuronų struktūra yra statinė, tik kinta jungčių reikšmės. Kiekvienas sluoksnis yra pilnai jungus su šalia esančiais. Kaip 9 pav. parodyta, kiekvienas neuronas turi lokalias koordinates, kurios ir yra naudojamos jungties svoriui nustatyti. Norint sumažinti nereikšmingų jungčių kiekį, siekiant pagreitinti sugeneruoto DNT įvertinimą, galima neįterpti jungčių, kurios turi mažą (moduliu) reikšmę. Siekiant išlaikyti tradicinį sluoksniuotą DNT modelį, jungtys yra įterpiamos tik tarp loginių sluoksnių, tačiau galima jungti bet kurio sluoksnio neuroną su bet kurio kito sluoksnio nesudarant ciklų.

5.2 HyperNEAT masiškumas

Galima pastebėti, kad šis algoritmas yra nesunkiai išplečiamas į didesnes dimensijas – reikia CPPN įvedimą adaptuoti iki neuronų koordinačių atitikimo. Jeigu įvedimo aibė būtų lenta, tada generuojamas DNT taptų stačiakampio gretasienio formos, jeigu įvedimo aibė yra trimatė erdvė, tada taptų 4 dimensijų hiperstačiakampiu arba hiperkubu (nuo pastarojo ir kilo pavadinimas) ir t.t..

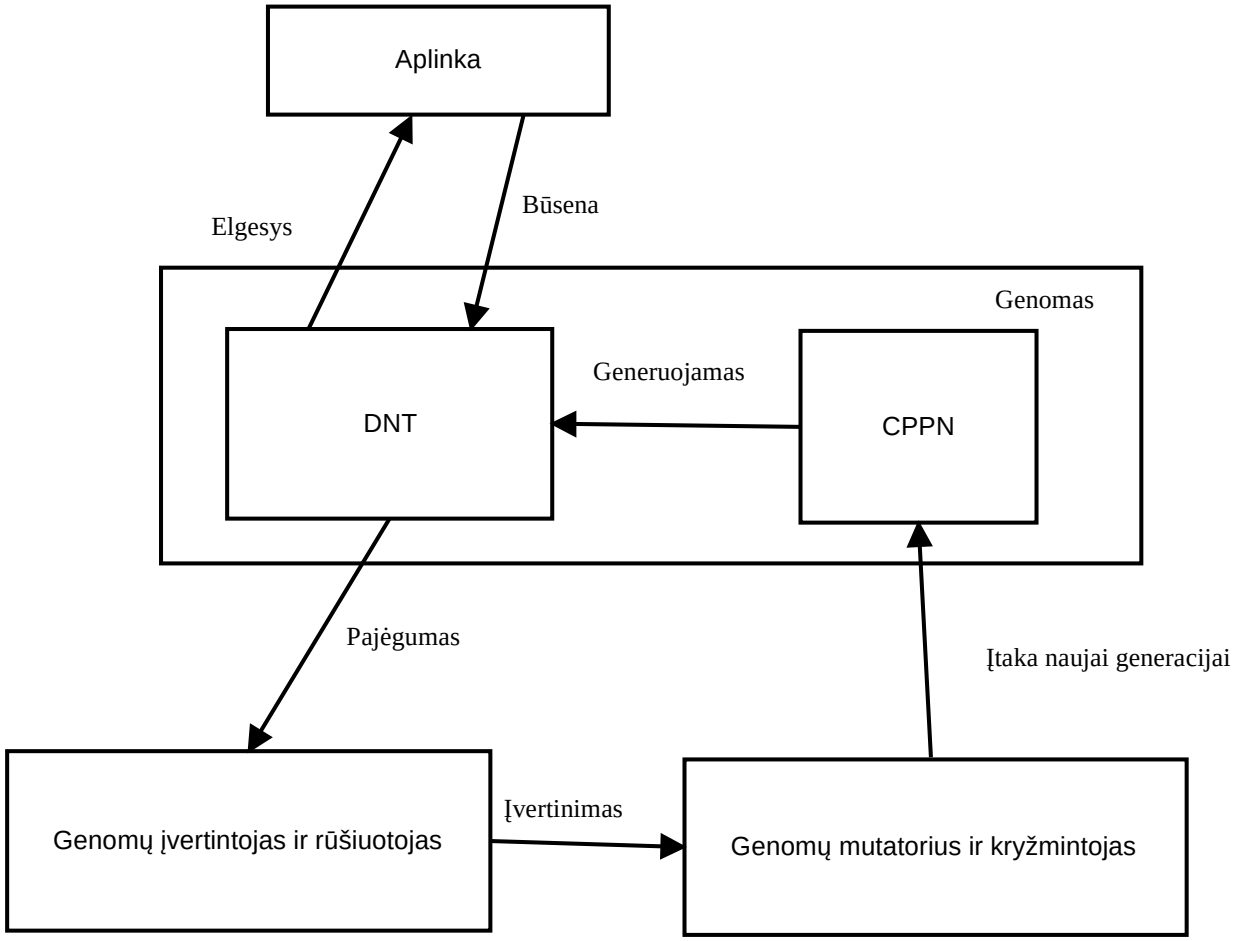
Kadangi HyperNEAT algoritmo esmė yra vidinis CPPN, kuris ir išsaugo informaciją apie DNT jungtis, tą patį CPPN galima panaudoti su skirtingų kiekių įvedimo/išvedimo DNT (10 pav.). Taip pat galima tinklą apmokyti su didelio masto sluoksniais ir po to pritaikyti mažesnio masto DNT generavimui, bei atvirkščiai.

Taip pat nebūtina apsiriboti stačiakampiais ir kvadratais, nes tai tik vienas iš galimų HyperNEAT modelių. Šį algoritmą galima adaptuoti apskritimo formai naudojant polines koordinates, jeigu to reikalautų problemos sritis pvz.: jūrų žvaigždės formos genomo elgesiui modeliuoti [KDJ09,11].



10. pav. To paties CPPN pritaikymas skirtingų dydžių DNT. Šaltinis:[KJ07]

5.3 HyperNEAT algoritmo ciklas



11. pav. HyperNEAT algoritmo ciklo modelis

HyperNEAT algoritmo ciklas panašus kaip ir NEAT, tik yra vienas papildomas žingsnis. Evoliucijoje dalyvauja vidinis CPPN, tačiau pajėgumas įvertinimas pagal sugeneruotą išorinį DNT, kuris ir yra naudojamas duotai užduočiai spręsti.

6 NEAT ir HyperNEAT pritaikymas valdikliui

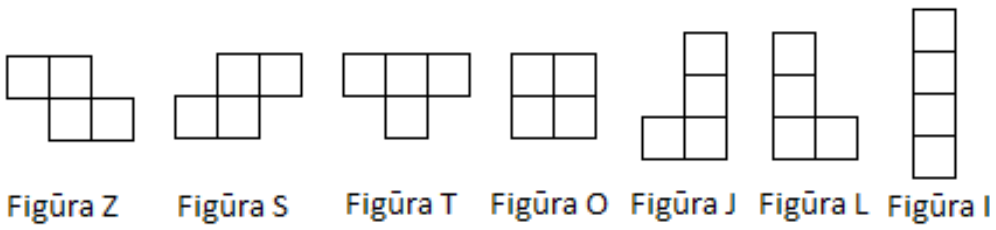
6.1 Užduoties formulavimas

Užduoties tikslas – neuro-evoliucijos būdu suformuoti valdiklį (angl. *Controller*), kuris gebėtų išspręsti duotą užduotį. Informacijos apie ją valdikliui duodama mažai bei taisykles valdiklis turi suprasti/atrasti pats, iš jo pajėgumo įverčio (angl. *Fitness)*.

6.2 Valdiklis video žaidimui „Tetris“

Valdikliui natūrali ir lengvai pritaikoma sritis – video žaidimai. Visame pasaulyje žinomas, daugiausiai kartų (daugiau negu 100 milijonų) parduotas video žaidimas – Tetris. Žaidimo taisyklės:

- Žaidimas vyksta ribotos erdvės „šulinyje“
- Po vieną krenta įvairios kampuotos figūros, sudarytos iš 4 kvadratėlių
- Pasiekus dugną arba jau gulinčią figūrą, atsiranda kita figūra
- Reikia iš jų sudaryti horizontalias linijas
- Sudarius liniją (-as), ji (-os) išnyksta ir viskas kas buvo virš jos, nukrenta per išvalytų linijų skaičių
- Gaunami taškai už išvalytas linijas (kuo daugiau išvaloma per 1 ėjimą, tuo daugiau taškų)
- Gaunami taškai už padarytą ėjimą ir už paspartintą figūros nuleidimą
- Yra 4 galimi ėjimai, kurie valdo krentančią figūrą:
  - Figūros pajudinimas per vieną langelį į kairę/dešinę
  - Figūros pasukimas
  - Figūros paspartintas kritimas
- Žaidimas baigiasi kada, kada nauja figūra negali atsirasti, nes ją blokuoja kita



12. pav. „Tetris“ žaidimo figūros.

6.2.1 Konkretus valdiklio įgyvendinimas

Tradiciniame video žaidime „Tetris“, figūros krito numatytu laiko intervalu (greitėjančiu), tačiau paprastumo dėlei figūra krenta kaip tik valdiklis padaro veiksmą. Naudotos konkrečios „Tetris“ žaidimo implementacijos „šulinio“ gylis yra 22 langeliai, o plotis 10. Taigi, valdiklio įvedimo neuronų yra 220. Kadangi galimi ėjimai yra 4, tiek ir yra išvedimo neuronų. Ėjimas nusprendžiamas pagal tai, kuris išvedimo neuronas turi aukščiausią reikšmę. Įvedimas formuojamas taip:

- 0, jeigu langelis tuščias
- 1, jeigu langelis nėra tuščias
- -1, jeigu langelis yra „skylė“

Langelis yra „skylė“, jeigu jis yra tuščias, tačiau yra netuščių nejudančių langelių virš jo. Pajėgumui matuojamas pagal žaidimo taškų skaičių:

- Padarytas ėjimas +1
- Paspartintas kritimas +10
- Išvalyta eilė +1000
- Išvalytos 2 eilės + 3000
- Išvalytos 3 eilės + 5000
- Išvalytos 4 eilės + 8000

Įvedama papildoma pabaigos sąlyga (laimėjimo sąlyga), kada taškų kiekis pasiekia 500000. Taigi NEAT algoritmui įgyvendinti tiek ir tereikia žinoti.

6.2.2 Adaptavimas HyperNEAT

Kadangi įvedimą siekiama išsaugoti, natūralu naudoti 3 dimensijų HyperNEAT variaciją, kai sluoksniai yra 2 dimensijų. Sluoksnis šiuo atveju tampa stačiakampis kurio ilgis 10, o aukštis 22 (pagal žaidimo išmatavimus). Kuo daugiau sluoksnių, tuo tikslesnio galima tikėtis sprendimo, tačiau smarkiai padidėja DNT generavimui skirtas laikas. Galiausiai, kadangi išvedimas tapo nebe 4, o 220 neuronų, reikia naujo ėjimo interpretacijos metodo. Vienas iš galimų – suskirstyti išvedimo sluoksnį į 4 regionus, vieną regioną atitinka 55 neuronai. Kai turime 4 regionus (vienas regionas atitinka vieną ėjimo tipą), aukščiausio vidurkio regionas ir bus padarytas ėjimas.

6.3 Rezultatai

Pilnas žaidimas „Tetris“ buvo per sunkus NEAT ir HyperNEAT algoritmams, tačiau paprastesnės jo versijos pasirodė įveikiamos. Kadangi pajėgumas matuojamas pagal gautus taškus, pirmiausia abu algoritmai suoptimizavo figūrų greitą kritimą. Vėliau „šulinio“ užpildymą. Kaip matome, lengviausią pajėgumo padidinimą pasiekė lengvai. Eilučių išvalymas yra sudėtingesnis veiksmas, iš kelių tarpinių žingsnių. Šie žingsniai (sudaryti eilutę) atskirai nėra teigiamai įvertinami (nėra skiriama taškų), todėl ir sunkiau atsitiktinai tokį veiksmą atlikti. Kadangi figūros „O“ ir „I“ yra gludžios (dviejų ašių simetrija), eilutės pakartotinis išvalymas buvo aptiktas, todėl gebėjo pasiekti pergalės būseną.

Pati paprasčiausia žaidimo variacija – naudojant tik „O” figūrą. Tada figūros pasukimas efekto neturi ir lieka tik 3 prasmingi ėjimai. Sudėtingesnė žaidimo versija – naudojant tik „I” figūrą. Šiuo atveju reikalingi visi 4 ėjimai. Kitos žaidimo versijos (su kitomis figūromis) pasirodė neįveikiamos per praktišką laiko kiekį.

NEAT algoritmo versija buvo leidžiama 200 generacijų, HyperNEAT 100 generacijų. Su kiekviena figūra leidžiami abu algoritmai po 5 kartus.

1. lentelė. Apibendrinti rezultatai.

Figūra	Pergalės būsenos pasiekimas NEAT (200 generacijų)	Pergalės būsenos pasiekimas NEAT (pirmos 100 generacijų)	Pergalės būsenos pasiekimas HyperNEAT (100 generacijų)
I	80%	40%	60%
L	0%	0%	0%
J	0%	0%	0%
O	100%	100%	100%
S	0%	0%	0%
T	0%	0%	0%
Z	0%	0%	0%

Parametrai ir techninė informacija:

- Pageidautinas rasių kiekis 5
- Kryžminimo tikimybė 70%
- Rasės išgyvenančioji dalis 30%
- Genomų panašumo santykiai:
  - atliekamų genų svoris 1
  - nebendrų genų svoris 1
  - sutampančių genų svoris 0.4
- Mutacijos:
  - Tikimybė keisti svorį 100%
  - Svorio keitimo žingsnio dydis 0.2
  - Svorio atstatymo į pradinį tikimybė 10%
  - Svorio įjungimo tikimybė 60%
  - Svorio išjungimo tikimybė 40%
  - Neurono pradinės reišmės (angl. *Bias*) keitimo tikimybė 80%
  - Neurono pradinės reišmės keitimo žingsnio dydis 0.2
  - Neurno pradinės reikšmės atstatymas į pradinė tikimybė 10%
  - Naujo lanko atsiradimas 200% (įterpiami bent po 2 naujus lankus)
  - Naujo neurono atsiradimo tikimybė 60%
  - Maksimali absoliuti lanko svorio vertė 20
  - Keisti aktyvacijos funkciją tikimybė 30%
  - Aktyvacijos funkcijų aibė: Sigmoidinė, Sin, Sinh, Tapatybės($x \rightarrow x$), Absoliučios vertės($x \rightarrow |x|$), modulio($x \rightarrow x \bmod 1$), Gauso
- HyperNEAT erdvės konfiguracija buvo iš 2 sluoksnių (įvedimo ir išvedimo).
- HyperNEAT absoliuti minimali svorio reikšmė yra 0.2
- HyperNEAT generuotame DNT aktyvacijos funkcija yra sigmoidinė

100 generacijų HyperNEAT užtruko ~30 min. 200 generacijų NEAT užtruko ~25 min.

Valdiklio naudojimas vyksta lygiagrečiai, pagal tai kiek gijų procesorius palaiko. Programa reikalauja Java 8 aplinkos.

7 Išvados

Neuro-Evoliucija yra galingas įrankis įvairaus mąsto ir spektro problemoms spręsti. Konkretus jos algoritmas NEAT, kurio pagrindinės idėjos yra inovacijos žymės ir suskaidymas rasėmis, gali būti sėkmingai pritaikytas valdikliui evoliucionuoti. Jeigu normas rezultatas buvo evoliucionuoti valdiklį žaisti žaidimą „Tetris“, tuomet rezultatas nepasiektas, tačiau jeigu buvo norima rasti būdą/įrankį, kuris leistu kompiuteriui aptikti žaidimo taisykles vien tik iš elgesio įvertinimų, rezultatas buvo pasiektas. Nereikia papildomų pokyčių algoritmui pritaikyti žaisti skirtingas „Tetris“ versijas. Jeigu valdiklio logiką reikėtų įgyvendinti rankiniu būdu, ji būtų kitokia skirtingoms „Tetris“ versijoms.

CPPN yra galingas informacijos abstrakcijos įrankis, leidžiantis išsaugoti struktūras erdvėje ir gali būti nesunkiai plečiamas į aukštesnes dimensijas. Pasitelkus CPPN yra įmanomas HyperNEAT algoritmas, kuris bando išspręsti informacijos abstrakcijos ir erdvės suvokimo problemą vystant DNT. Žinoma, HyperNEAT yra prasmės naudoti tik ten, kur yra geometriniai dėsniai. Taip pat didelių DNT generavimas kainuoja daug resursų, todėl matuojant realiu laiku NEAT gali būti greitesnis, tačiau matuojant generacijų skaičiumi, HyperNEAT dažniausiai bus greitesnis ar bent toks pat greitas kaip ir NEAT.

Vienas iš HyperNEAT trūkumų yra stacionari struktūra. Pagrindinis NEAT privalumas – nėra pradinės struktūros, ją suformuoja algoritmas. HyperNEAT versijos su augančia struktūra yra ES-HyperNEAT (angl. *Evolvable Subsrate* HyperNEAT*)* algoritmas, kurį galima būtų panagrinėti ateityje, kaip ir skirtingų struktūrų HyperNEAT konfiguracijas (pvz apskritimo). Siekiant imituoti tam tikrą gamtinė aplinką, vertėtų pasinaudoti rtNEAT(angl. *Real Time* NEAT) variacija.

8 Priedai

Java programos kodas yra GitHub tinklapyje

Pagrindiniai algoritmai: https://github.com/laim0nas100/LBML

„Tetris“ žaidimo pritaikymas su algoritmu: https://github.com/laim0nas100/tetrisNEAT

Pagalbinių klasių rinkinys: https://github.com/laim0nas100/LibraryLB

Google JSON biblioteka: https://github.com/google/gson

2 lentelė. Detalūs rezutatai.

Figūra	Algoritmas	Bandymas	Galutinė generacija	Daugiausia taškų
I	NEAT	1	127	500000+
I	NEAT	2	25	500000+
I	NEAT	3	47	500000+
I	NEAT	4	200	6385
I	NEAT	5	190	500000+
I	HyperNEAT	1	96	500000+
I	HyperNEAT	2	62	500000+
I	HyperNEAT	3	92	500000+
I	HyperNEAT	4	100	4642
I	HyperNEAT	5	100	9870
L	NEAT	1	200	2686
L	NEAT	2	200	2686
L	NEAT	3	200	2053
L	NEAT	4	200	2726
L	NEAT	5	200	2077
L	HyperNEAT	1	100	1304
L	HyperNEAT	2	100	1300
L	HyperNEAT	3	100	1305
L	HyperNEAT	4	100	1303
L	HyperNEAT	5	100	1300
J	NEAT	1	200	2576
J	NEAT	2	200	2360
J	NEAT	3	200	3050
J	NEAT	4	200	2913
J	NEAT	5	200	2673
J	HyperNEAT	1	100	1960
J	HyperNEAT	2	100	2028
J	HyperNEAT	3	100	1957
J	HyperNEAT	4	100	2515
J	HyperNEAT	5	100	2884
O	NEAT	1	71	500000+
O	NEAT	2	17	500000+
O	NEAT	3	31	500000+

O	NEAT	4	44	500000+
O	NEAT	5	7	500000+
O	HyperNEAT	1	17	500000+
O	HyperNEAT	2	49	500000+
O	HyperNEAT	3	18	500000+
O	HyperNEAT	4	66	500000+
O	HyperNEAT	5	52	500000+
S	NEAT	1	200	2038
S	NEAT	2	200	2012
S	NEAT	3	200	2064
S	NEAT	4	200	1940
S	NEAT	5	200	1731
S	HyperNEAT	1	100	1989
S	HyperNEAT	2	100	2230
S	HyperNEAT	3	100	2123
S	HyperNEAT	4	100	1867
S	HyperNEAT	5	100	1986
T	NEAT	1	200	3129
T	NEAT	2	200	2800
T	NEAT	3	200	3215
T	NEAT	4	200	2589
T	NEAT	5	200	2125
T	HyperNEAT	1	100	3365
T	HyperNEAT	2	100	2913
T	HyperNEAT	3	100	3216
T	HyperNEAT	4	100	2613
T	HyperNEAT	5	100	2874
Z	NEAT	1	200	2078
Z	NEAT	2	200	1904
Z	NEAT	3	200	2073
Z	NEAT	4	200	1995
Z	NEAT	5	200	2044
Z	HyperNEAT	1	100	2332
Z	HyperNEAT	2	100	2002
Z	HyperNEAT	3	100	2213
Z	HyperNEAT	4	100	2156

Z	HyperNEAT	5	100	2189

Literatūra

[KR02] Kenneth O. Stanley, Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary Computation*, 10(2):99, 127, 2002.

[Ken04] Kenneth O. Stanley. Efficient Evoliution of Neural Networks through Complexification. PhD thesis, The University of Texas, 2004

[Ken07] Kenneth. O. Stanley. Compositional pattern producing networks: A novel abstraction of development. Genetic Programming and Evolvable Machines, 8(2):131–162, 2007.

[KDJ09] Kenneth. O. Stanley, D. D'Ambrosio, J. Gauci. A Hypercube-Based encoding for evolving Large-Scale neural networks. Artificial Life, 15(2):185–212, 2009

[KJ07] Kenneth. O. Stanley, J. Gauci. Generating Large-Scale Neural Networks Through Discovering Geometric Regularities, GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation 997-1004, 2007

[KW16] Kearney, William T., Using Genetic Algorithms to Evolve Artifcial Neural Networks (2016).Honors Theses. Paper 818. htp://digitalcommons.colby.edu/honorstheses/818

[NFAQ] Kenneth O. Stanley, The NeuroEvolution of Augmenting Topologies (NEAT) Users Page, FAQ, https://www.cs.ucf.edu/~kstanley/neat.html

[Phi94] Phillipp Koehn (1994) Combining Genetic Algorithms and Neural Networks: The Encoding Problem. Master Thesis. University of Tennessee, Knoxville

[Set15] SethBling. MarI/O - Machine Learning for Video Games. (2015). https://youtu.be/qv6UVOQ0F44