Katedra Aplikovanej Informatiky
Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Konštrukcia Stavových Automatov Evolučnými Algoritmami

(Bakalárska práca)

Péter Dobsa

Vedúci: Mgr. Pavel Petrovič, PhD.

Bratislava, 2014

Fakulta Matematiky, Fyziky a Informatiky
Univerzita Komenského, Bratislava

Konštrukcia Stavových Automatov Evolučnými Algoritmami

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanéj informatiky
Školitel:	Mgr. Pavel Petrovič, PhD.

Péter Dobsa

Bratislava, 2014

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Péter Dobsa
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Konštrukcia stavových automatov evolučnými algoritmami / *Construction of State Automata by Means of Evolutionary Algorithms*

Cieľ: Cieľom výskumno-experimentálnej práce je preskúmať metódy konštrukcie stavových automatov pomocou evolučných algoritmov. Porovnať metódy a aspekty evolučných algoritmov s ohľadom aplikovania myšlienok evolučného algoritmu na evolúciu rekurentných neurónových sietí NEAT na evolúciu stavových automatov: jednoliatu populáciu a populáciu rozdelenú na druhy, inicializáciu minimálnymi jedincami, ochranu inovácií, topologické kríženie globálnym identifikátorom génov. Práca porovná jednotlivé aspekty na zvolenej/navrhnutej aplikácii a vlastnej využívajúcej naevolvovaný stavový automat a výsledky vyhodnotí a prediskutuje.

Literatúra:
K.Stanley, R.Miikkulainen: Evolving Neural Networks through Augmenting Topologies, Evolutionary Computation 10(2): 99-127, 2002.
Petrovic P. (2009) Evolúcia stavových automatov inšpirovaná metódou NEAT, Kognícia a umelý život, Stará Lesná, 2009.
A.Eiben: Introduction to Evolutionary Computing, Springer, 2010.
Stanislav Hreha: Evolúcia stavových diagramov, diplomová práca, FMFI UK, Bratislava, 2011.

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 22.10.2013

Dátum schválenia: 22.10.2013

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

Dobsa
.....
študent

Pavel Petrovič
.....
vedúci práce

Čestne vyhlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím citovaných zdrojov.

.....

Poďakovanie

Touto cestou by som chcel poďakovať vedúcemu práce Mgr. Pavlovi Petrovičovi, PhD. za konzultácie, obetavý čas, cenné rady a pripomienky počas tvorby práce.

Abstrakt

Dobsa, Péter: Konštrukcia stavových automatov evolučnými algoritmami (Bakalárska práca)
Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky a informatiky. Katedra aplikovanej informatiky.
Školitel': Mgr. Pavel Petrovič, PhD.

Cieľom bakalárskej práce je implementácia rámcového prostredia na evolúciu stavových automatov pomocou aplikovania aspektov evolučného algoritmu na evolúciu rekurentných neurónových sietí NEAT. Základom rámcového prostredia je konfigurovateľný algoritmus, ktorý umožňuje experimentovanie s rôznymi nastaveniami s cieľom nájdenia optimálnych parametrov pre rozličné typy úloh. Popritom práca porovnáva jednotlivé aspekty NEAT v rôznych testovacích scenároch, výsledky vyhodnotí a prediskutuje.

Kľúčové slová: evolučné algoritmy, NEAT, stavové automaty

Abstract

Dobsa, Péter: Construction of state automata by means of evolutionary algorithms (Bachelor thesis)
Comenius University in Bratislava. Faculty of Mathematics, Physics and Informatics. Department of Applied Informatics.
Supervisor: Mgr. Pavel Petrovič, PhD.

The goal of this bachelor thesis is to implement a framework for state automata evolution by applying aspects of the evolutionary algorithm for the evolution of recurrent neural networks NEAT. The base of this framework is a configurable algorithm, which allows experimentation with different setups with the goal of finding optimal parameters for different tasks. In addition, the paper compares aspects of NEAT in various test scenarios, evaluates and discusess the results.

Keywords: evolutionary algorithms, NEAT, state automata

Obsah

Kapitola 1

Úvod

Ľudia už odjakživa skúsili nájsť riešenie na svoje problémy inšpirovaním sa javmi a zákonitosťami v prírode - prvé, jednoduché zbrane na lov zvierat, lietadlá, fotoaparáty a mnoho ďalších vynálezov napodbňujú čo príroda vytvorila počas miliónov rokov.
Informatika tiež nie je výnimkou, mnohé postupy aplikujú princípy prevzatých z prírody. Dobrým príkladom na to je presadenie mechanizmu evolúcie do binárnej sústavy počítačov, účelom riešenia úloh, ktoré by boli neriešiteľné s tradičnými metódami.
Nazývajú sa to evolučné algoritmy a ich známym predstaviteľom je NEAT. Cieľom tejto bakalárskej práce je používanie myšlienky NEAT na evolúciu stavových automatov, s ohľadom na túto skutočnosť implementovať evolučný algoritmus a prostredie, ktoré umožňuje s ním experimentovať. Overiť funkčnosť tohto prostredia vykonaním niekoľkých experimentov a následne dosiahnuté výsledky analyzovať.

Štruktúra práce

Kapitola 2 zahŕňa všetky dôležité informácie z ktorých sme vychádzali počas realizácie práce, teda potrebnú teóriu a opis podobných prác v minulosti.
Návrh výslednej aplikácie, algoritmu, použité metódy a experimenty obsahuje kapitola 3. Podrobne prediskutuje rôzne aspekty a parametre algoritmu, ich účel a využitie; ďalej prezentuje nastavenia testovaných algoritmov.
Výsledky experimentov s komentármi sú úvedné v kapitole 4.
V závere (kapitola 5) stručne zhrnieme vlastnosti a prínosy práce, spolu s možnými vylepšeniami.

Kapitola 2

Východiská práce

Táto kapitola slúži na uvedenie čitateľa do problematiky bakalárskej práce, na vysvetlenie základných pojmov a techník z oblasti evolučných výpočtov a stavových automatov. Pokúsime si načrtnúť základné teoretické pozadie, uviesť si terminológiu; ďalej preskúmať a prezentovať podobné práce z minulosti.

2.1 Evolučné výpočty

Darwinová teória o evolúcii je dobre známa. Proces, počas ktorého jednotlivci populácie daného druhu sa postupne vyvíjajú na genetickej úrovni. Nakoľko tieto genetické informácie rodičia odovzdávajú novým potomkom, evolúcia má silný vplyv aj na zloženie populácie: vývojové vetvy, ktoré nespôsobia žiadne zdokonalenia vedú k jedincom (príp. celému druhu), ktorí sa nevedia prispôsobiť, nemôžu konkurovať silnejším a zmenám v prostredí, nakoniec vyhynú. Počas tohto procesu často z počiatočných jednoduchých genetických štruktúr vznikajú čoraz komplexnejšie.

Z tohto princípu vychádzajú techniky evolučných výpočtov, ich základnými kameňmi sú *evolučné algoritmy* (*evolutionary algorithms, EA*).

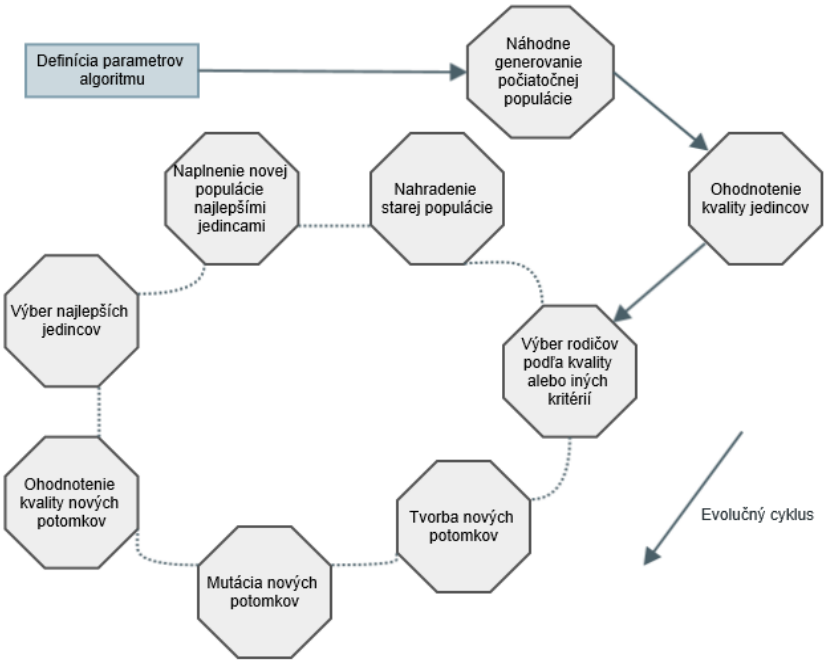
2.1.1 Evolučné algoritmy

Pojem 'evolučný algoritmus' zahŕňa pomerne široké spektrum optimalizačných a heuristických metód, ale ich základná idea je spoločná: prehľadávanie priestoru riešení, kde každý bod priestoru reprezentuje riešenie pre daný problém. Takýto priestor potenciálnych riešení môže byť až tak obrovský, že klasické deterministické prehľadávacie algoritmy a analytické metódy optimálne riešenie nedokážu nájsť v polynomiálnom čase. Evolučné algoritmy sa používajú práve na riešenie takých problémov.

Na takýto priestor riešení môžeme pozerať ako na populáciu jedincov, kde každý jedinec danej puplácie je kandidát na riešenie problému. Počiatočná populácia je generovaná náhodne. Populácia sa časom mení, nevhodné jednice (tj. najhoršie riešenie) sú odstránené z populácie aby sa uvoľnilo miesto pre nových, lepších jedincov. Tieto procesy sa iterujú v *evolúčnom cykle* po tzv. *generáciách*, kým sa nenájde dostatočne kvalitné riešenie - príp. nedosiahneme prednastavenú hranicu počtu iterácií.

2.1.2 Komponenty evolučných algoritmov

- **reprezentácia** - prvý krok algoritmu je vyriešiť mapovanie medzi priestorom problému a priestorom algoritmu. Jedinec v kontexte pôvodného problému sa volá *fenotyp* a jeho kódovanie v EA sa volá *genotyp*. Tieto dve reprezentácie môžu byť veľmi odlišné, nakoľko verzia s ktorou pracuje algoritmus musí byť ľahko modifikovateľná aj v závislosti od iných genotypov (viď kríženie a mutácia) - napr. binarný reťazec. Hotové riešenie dostaneme jeho dekódovaním na fenotyp, na ktorom sa testuje aj kvalita riešenia.



Obr. 2.1: Všeobecný cyklus evolučného algoritmu - podľa [11]

- **vyhodnocovacia funkcia (fitness funkcia)** - slúži na vyhodnotenie kvality daného riešenia podľa genotypu. Každý jedinec má tzv. *fitness hodnotu* , čím kvalitnejšie je riešenie, tým väčšia je hodnota.

- **selekcia** - proces na začiatku každej generácie, počas ktorého sú vybratí rodičia generácie, zvyčajne pomocou pravdepodobnosti - lepšia fitness hodnota znamená väčšiu pravdepodobnosť. Keďže výber výlučne najlepších jedincov (greedy search) by viedol k *predčasnej konvergencii* (*premature convergence*), čiže stagnácii na lokálnom optime, EA s nejakou nenulovou pravdepodobnosťou vyberajú aj jedince s nižšou fitness, čím umožnia populácii objaviť vzdialenejšie oblasti prehľadávacieho priestoru.

  Typické selekčné mechanizmy [1]:

- *ruletová selekcia (roulette wheel selection / fitness proportionate selection)* - individuá sú namapované na spojitý segment nejakej čiary, kde veľkosť tohto segmentu je úmerná jeho fitness hodnote. Potom sa generuje náhodné číslo podľa veľkosti čiary, a bude vybratý jedinec, do ktorému segmentu padne toto číslo - takže veľkosť pravdepodobnosti selekcie jedinca závisí od veľkosti jeho fitness.
- *turnajová selekcia (tournament selection)* - z populácie je náhodne vybratá množina k jedincov (obyčajne $k = 2$) a následne z tejto množiny je vybratý jedinec s najlepšou fitness. Je to jedna z najpopulárnejších metód, jednak kvôli udržaniu selekčného tlaku nezávisle na rozdieloch fitness jedincov a na druhej strane jedince s menšou fitness hodnotou tiež dostávajú (na rozdiel napríklad od ruletovej selekcie) nezanedbateľne veľkú šancu na vybratie, čo zvyšuje obranu proti predčasnej konvergencii.
- *selekcia s orezávaním (truncation selection)* - elitistická selekcia, počas ktorého populácia je utriedené podľa fitness, slabšie jedince sú orezané (podľa nejakej prahovej hodnoty) a nedostanú žiadnú šancu na prežitie, ostatné sú vybraté.
- *elitizmus (elitism)* - nie je to samostatná metóda, zabezpečuje iba to, aby najlepší jedinec (príp. množina najlepších jedincov) bol v každom prípade prenesený bez zmien do ďalšej generácie, zvyšok je už vybratý iným mechanizmom. Väčšina selekčných metód to totiž negarantuje a v každom kroku máme šancu na stratenie najlepšieho jedinca.

- **kríženie** - alebo rekombinácia je operácia, počas ktorého sa kombinujú

dva jedince (rodičia) podľa pevne daných pravidiel a výsledkom toho je nový jedinec (potomok). Operácia je stochastická, dvojice rodičov sú náhodne vybrané, tak ako aj ich časti na kríženie. Je to najčastejšie binárna operácia, hoci kríženie viac než dvoch rodičov je matematicky zrealizovateľné.

- **mutácia** - ďalšia operácia, ktorá sa aplikuje len na jedného jedinca a výsledkom je jeden nový, modifikovaný jedinec. Taktiež stochastická operácia.

- **nahradenie** - mechanizmus, ktorý sa spúšťa na konci každej generácie a rozhoduje sa, ktoré jedince môžu dostať do ďalšej generácie. Veľkosť populácie počas celej evolúcie je často konštantná, a vyberanie vhodných jedincov na rozdiel od ostatných komponentov algoritmu je deterministické. Obyčajne ostanú individuá s najlepšou fitness hodnotou, ale často aj vek je rozhodovací faktor.

2.1.3 Druhy evolučných algoritmov

Evolučné algoritmy môžeme rozdeliť podľa rôznych kritérií a aspektov, z pohľadu tejto práce je ale zaujímavé základné rozdelenie podľa mechanizmu nahradenia [10]:

- **generačné algoritmy** *(generational algoritms)* - aktuálna populácia kompletne vyhynie, populácia ďalšej generácie z nich neobsahuje ani jedného jedinca, namiesto toho je vytvorená jedine z potomkov

- **inkrementálne algoritmy** *(incremental/steady-state algorithms)* - počas každej generácie je nahradený iba jeden jedinec populácie

- kombinácia predošlých dvoch prístupov (napr. iba polovica populácie je nahradená potomkami)

2.1.4 Genetický algoritmus

Genetický algoritmus [6] je jeden z najznámejších a najvýznamnejších typov evolučných algoritmov. Je to výsledkom práce J.Hollanda z USA, ktorého cieľom bol čo najpresnejšie presadiť fenomén prirodzenej evolúcie do počítača. Jedince sú binárne reťazce bez akýchkoľvek dodatočných informácií pevne danej dĺžky a reprodukujú sa krížením a mutáciou blízke prirodzeným. Operátory sú ale aplikované na pravdepodobnostnom princípe, potomok vôbec nemusí obsahovať nové informácie od rodičov.

2.1.5 Neurónové siete

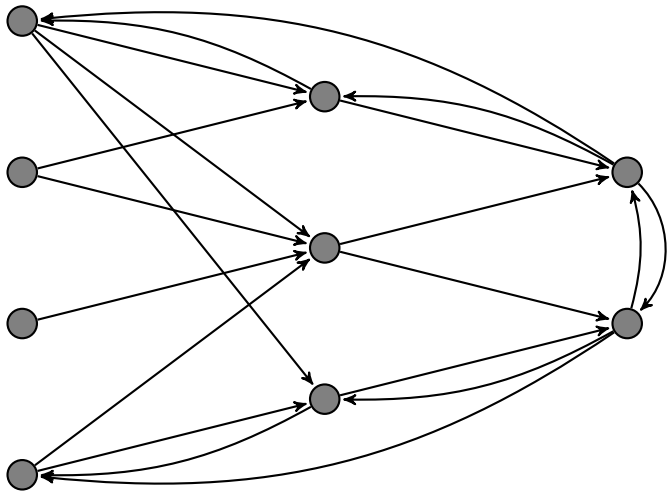
Neurónová sieť (NS) je paralelný výpočtový systém [8], ktorý bol silne inšpirovaný ľudským mozgom vo viacerých aspektoch. Ako aj v mozgu, základným prvkom NS sú neuróny a v medzineurónových spojeniach sú uložené znalosti systému. Všetky tieto znalosti sú pozbierané počas učenia. Štruktúru NS môžeme si predstaviť ako orientovaný graf, kde vrcholy grafu sú neuróny a hrany sú prepojenia medzi nimi. Podľa toho akým spôsobom dostávajú informáciu a ako ju šíria, rozlišujeme:

- **vstupné neuróny** - dostávajú vstup z okolitého prostredia a výstup pokračuje k ďalším neurónom
- **skryté neuróny** - dostávajú vstup z ostatných neurónov a výstup pokračuje k ďalším neurónom
- **výstupné neuróny** - ako skryté, lenže výstup pokračuje do okolitého prostredia

Neuróny sú často zoskupené do vrstiev, z ktorých analogicky rozpoznávame *vstupné* , *skryté* a *výstupné*.

Neurónové siete môžeme rozdeliť do dvoch skupín v závislosti od toho ako sa v nich šíria informácie:

- **Dopredné NS (feed-forward)** - signál šíri len jedným smerom: dopredu.
- **Rekurentné NS** - signál môže šíriť oboma smermi, práve preto pre niektoré neuróny nemôžeme jednoznačne povedať, či sú vstupné, výstupné alebo skryté. Takéto siete sú často vytvorené podľa dopredu určených pravidiel - vtedy hovoríme o *čiastočne rekurentných neurónových sietiach*.



Obr. 2.2: Štruktúra rekurentnej NS

2.1.6 Evolúcia neurónových sietí

Evolvovanie neurónových sietí - alebo **neuroevolúcia** (NE) - je populárny spôsob na optimalizovanie štruktúr NS alebo zvýšenie efektívnosti riešenia

danej úlohy. Neuroevolučné metódy, poväčšine pomocou genetických algoritmov, prehľadávajú priestor správaní za účelom nájsť vhodné parametre siete. Bolo aj ukázané, že tento spôsob trénovania NS môže byť kvalitnejší ako metódy fungujúce na princípe *reinforcement learning* (učenie na základe odmien a trestov).

NE metódy sú väčšinou aplikované na neurónové siete, ktoré už majú finálnu štruktúru a evolúcia ovplyvňuje len váhy prepojení medzi neurónmi. Argumentácie o tom, že evolúcia topológie môže prispieť ku kvalitneniu NE metód už existujú od roku 1992 (Fullmer a Miikkulainen), nakoľko topológia tak isto má vplyv na funkcionalitu siete a manuálne hľadanie optimalnej topológie je niekedy realizovateľne len metódou pokusov a omylov (trial-and-error), pritom ale otestovanie všetkých permutácií topológií siete by výrazne zvýšilo komplexnosť a čas behu algoritmu. Pokusy samozrejme boli aj na teoretickej aj na aplikovanej úrovni (*TWEANN*- Topology and Weight Evolving Artificial Neural Networks), metóda, ktorá ale bezpochybne dokázala, že evolúcia štruktúry môže zvýšiť výkon neuroevolúcie bola publikovaná až v roku 2002: *NEAT*.

Neuroevolučné metódy na správne fungovanie musia prekonať viac problémov, ako sú efektívne kódovanie, Competing conventions, správnu inicializáciu populácie a ochranu inovácie. V ďalšej podkapitole ukážeme ako ich vyrieši NEAT v porovnaní s TWEANN-om.

2.2 NEAT

NEAT (NeuroEvolution of Augmenting Topologies) je pomerne nová neuroevolučná metóda, ktorá prináša idey ako vylepšiť TWEANN. Jednak z hľadiska evolúcie váh prepojení aj z hľadiska evolúcie topológií neurónových

sietí. Autori Kenneth O. Stanley a Risto Miikkulainen ju publikovali v roku 2002 a dokázali, že NEAT môže byť viackrát rýchlejší ako metódy s fixnou topológiou.

Pokúsime si teraz vymenovať hlavné charakteristiky NEAT-u, ako rieši najvážnejšie problémy neuroevolučných algoritmov, čo a prečo robí inak ako predchádzajúce metódy.

Informácie a použité obrázky boli čerpané z [9].

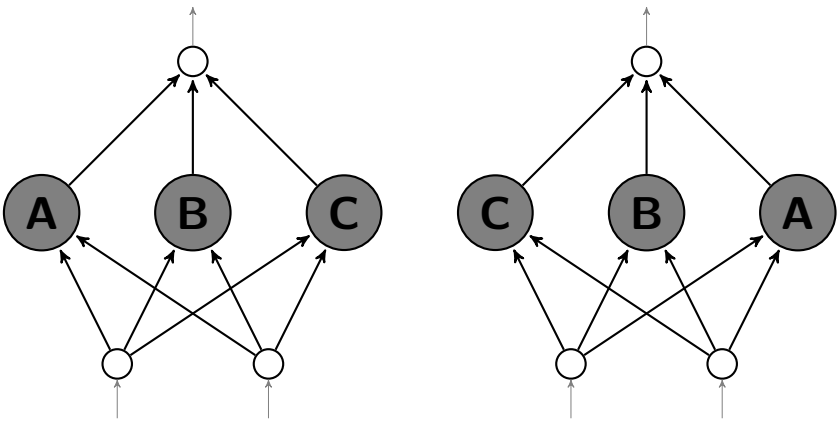
2.2.1 Competing Conventions

Jeden z hlavných problémov neuroevolúcie je *Competing Conventions Problem* alebo *Permutačný problém* (*Permutations Problem*). Pokrýva to situáciu, keď optimalizáciu váh môžeme vyjadriť viacerými spôsobmi a kríženie genómov môže vyprodukovať nekvalitných potomkov, ktorým rodičia neodovzdávajú dôležité informácie. V konečnom dôsledku to vedie ku strate týchto informácií.

Jednoduchý príklad permutačného problému je na obrázku 3: 3 skyté neuróny A, B a C v oboch sieťach znamenajú $3! = 6$ možných permutácií, z ktorých $[C, B, C]$ a $[A, B, A]$ sú potomkovia, ktorí stratili tretinu informácií, ktorú rodičia poznali.

Situácia je oveľa horšia, ak informácia v rodičoch nie je rovnaká (napr. $[A, B, C]$ a $[X, Y, C]$) alebo majú odlišnú topológiu (príp. veľkosť genómov je odlišná) - čo je bežná vec pri evolúcii topológie siete, hlavne ak na výslednú štruktúru nekladieme žiadne obmedzenia. Väčšina TWEANN metód problém adresuje jednoducho: predpokladajú, že podštruktúry každej NS sa skladajú z takých genómov, ktoré sa dajú rekombinovať bez straty dôležitých údajov a pritom rôzne topológie nemôžu obsahovať tie isté podštruktúry.

NEAT ale také obmedzenia nemá a permutačný problém rieší podľa prírody. Genómy v prírode tiež nemajú fixnú dĺžku a rovnaké štruktúry, a keby počas



[A, B, C]

X [C, B, A]

Možné kombinácie: [A, B, A] [C, B, C]

Obr. 2.3: Competing conventions. Dve neurónové siete ponúkajú riešenie pre ten istý problém, akurát skryté neuróny sú inak prepojené. Po krížení z 6 možných potomkov 2 stratia informáciu.

prirodzenej evolúcie nové gény boli umiestené náhodne, život by pravdepodobne neexistoval.

Riešením je *homológia*: gény sú homologické ak majú rovnakú štruktúru a charakteristiky, slúžia na rovnaké zámery. V NEAT-e každý gén obsahuje informácie o historickom pôvode, podľa čoho jednoznačne môžeme povedať, ktoré gény majú spoločného predchodcu, tj. ktoré sú kompatibilné na kríženie.

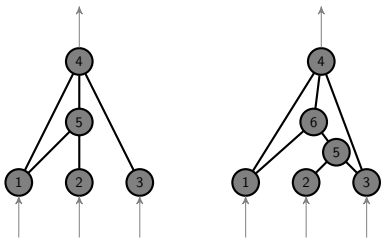
2.2.2 Inovačné číslo

Sledovanie historického pôvodu génu v NEAT-e nevyžaduje žiadnu výpočtovú silu: každý gén obsahuje tzv. *inovačné číslo* (*innovation number*), ak sa tieto čísla zhodujú, gény boli odvodené od spoločného predka a preto majú rov-

nakú štruktúru (alebo aspoň podobnú - váha prepojení neurónov môže byť odlišná). Keď pribudne nový gén, zvýšime *globálne počítadlo inovačných čísel* a priradíme jeho hodnotu príslušnému génu.
Mechanizmus je nesmierne priamočiary a elegantný, má ale jednu nevýhodu: ak by sme po každej zmene štruktúry novému génu priradili novú hodnotu, tie isté zmeny (napr. tá istá mutácia jednotlivca) by mohli dostať odlišné čísla. Práve preto v rámci každej generácie treba viesť zoznam všetkých inovácií a podľa toho priradiť inovačné čísla.
Keď dojde ku kríženiu dvoch genómov, z génov, ktoré majú rovnaké inovačné čísla - volajú sa *matching genes* - náhodne vyberieme ten, ktorý sa dostane do potomka. Všetky ostatné gény zrejme reprezentujú štruktúru, ktorá v druhom rodičovi neexistuje - sú to *excess* alebo *disjoint genes* , podľa toho, či ich číslo padne do rozsahu inovačných čísel v druhom rodičovi - a vždy sa dostanú do potomka z vhodnejšieho rodiča.

2.2.3 Genetická reprezentácia

Genetická reprezentácia opisuje spôsob, ako sú genotypy ukladané v pamäti počítača. TWEANN metódy problém riešia rôznymi spôsobmi (binárne reťazce, grafy, nepriame reprezentácie, ...), NEAT má lineárnu reprezentáciu všetkých prepojení. Každý genóm je reprezentovaný zoznamom génov, ktoré slúžia ako prepojenia (*connection genes*) medzi 'vrcholmi' genómu (*node genes*). Prepojovacie gény obsahujú váhu, inovačné číslo a informáciu o incidentných vrcholoch, vrcholy poznajú svoj typ (vstupné, výstupné alebo skryté).
Počas mutácie genómu môžu pribudnúť hociktoré verzie z týchto dvoch génov. V prípade prepojovacích génov pribudne nová hrana s náhodnou váhou medzi vrcholmi, ktoré predtým neboli prepojené, a nové vrcholy sa pridávajú medzi



Rodič 1	1 1-4	2 2-4	3 3-4	4 2-5	5 5-4			8 1-5		
Rodič 2	1 1-4	2 2-4	3 3-4	4 2-5	5 5-4	6 5-6	7 6-4		9 3-5	10 1-6
Potomok	1 1-4	2 2-4	3 3-4	4 2-5	5 5-4	6 5-6	7 6-4	8 1-5	9 3-5	10 1-6

Obr. 2.4: Ukážka kríženia pomocou inovačných čísel. Topológia rodičov je odlišná ale podľa inovačného čisla (hore v tabuľke) je jednoznačné, ktoré gény sa zhodujú.

dvomi, už existujúcimi vrcholmi - tým pádom štruktúra ostane konzistentná po zmene a ľahšie sa evolvuje.

2.2.4 Ochrana inovácie

Inovácia je proces, počas ktorého pridáme do siete nové štruktúry pomocou mutácie. Nové štruktúry ale pravdepodobne hneď po pridaní nebudú mať žiadne užitočné funckie - čo ľahko môže viesť ku zníženiu fitness celej štruktúry. Potrebujú teda čas na optimalizáciu - niekoľko generácií -, čo za bežných okolností nemajú, vďaka strate fitness by tieto inovácie toľko času neprežili. Potrebujú teda ochranu.

TWEANNy ochranu inovácie často zabezpečujú pridaním bezfunkčných

štruktúr ktoré ale nie sú prepojené do genómov, tým pádom fitness nezhoršujú. Stanú sa súčasťou siete až v okamihu, keď sa dostatočne optimalizujú, čo sa ale nemusí nikdy stať a tieto štruktúry zbytočne skomplikujú prehľadávanie. Idea v NEAT-e je nesledovná: populáciu rozdelíme do druhov takým spôsobom, že štruktúry s podobnou topológiou sa dostanú do jedného druhu. Takto jednotlivci budú súperiť len v rámci svojho druhu a dostanú čas na optimalizáciu svojich štruktúr.

Ostane teda otázka: ako sa rozhodnúť, ktoré genómy sa môžu dostať do spoločného druhu? Kompatibilitu genómov v NEAT-e môžeme merať počtom *excess* a *disjoint* génov - čím viac, tým menej sú kompatibilné. Vzorec kompatibility δ:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W},$$

kde N je počet génov vo väčšom genóme, E a D sú počty excess a disjoint génov a $\overline{W}$ je priemerný rozdiel váh matching génov. Koeficienty c_1, c_2 a c_3 slúžia na nastavenie dôležitosti jednotlivých faktorov.

Ak rozličnosť dvoch genómov neprekročí nejakú prahovú hodnotu δ_t, považujeme ich za kompatibilné.

V každej generácii všetky nové genómy sú priradené do nejakého druhu - druhy sú prebraté z predošlej generácie a každý druh je reprezentovaný nejakým genómom. Genóm g sa dostane do daného druhu ak je kompatibilný s genómom, ktorý tento druh reprezentuje. Ak taký druh neexistuje, vytvorí sa nový s reprezentatívným genómom g.

Dôležitá vlastnosť NEAT-u je aj *explicit fitness sharing*, čo je mechanizmus, ktorý vynúti aby jednotlivci v rámci druhov zdieľali fitness hodnotu - slúži na vyrovnanie síl druhov, aby výnimočné jedince neobsadili celú populáciu. Upravená fitness jedinca i je určená nasledovne:

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))},$$

kde funkcia sh vráti 0, ak vzdialenosť $\delta(i,j)$ prekročí prahovú hodnotu δ_t, inak vráti 1.

2.2.5 Inicializácia topológie

Vo väčšiane TWEANN-ov, počiatočná populácia sa skladá z náhodne vygenerovaných topológií, tým pádom topologická rozmanitosť je zabezpečená. Na druhej strane ale istá časť týchto náhodných štruktúr je bezfunkčná, nedostanú sa ani do druhej generácie, ich vytvorenie nemá žiadny význam. Lepšie riešenie je, ak neuroevolúcia uprednostňuje čo najmenšiu počiatočnú populáciu, populácia neobsahuje žiadne skryté neuróny, tie sú vytvorené až počas evolúcie takým spôosobom, akú vyžaduje daný problém. Takáto minimalizácia populácie pozitívne ovplyvňuje aj čas prehľadávania, nakoľko priestor riešení bude tiež minimálny. Podľa tohto princípu funguje aj NEAT.

2.3 Stavové automaty

2.3.1 Konečno-stavový automat

Konečno-stavový automat (*finite-state automaton - FSA*, alebo *finite-state machine - FSM*) je jednoduchý výpočtový model [7], formálnejšie je to pätica $(K, \Sigma, \delta, q_0, F)$, kde K je konečná, neprázdna množina stavov, Σ je konečná vstupná abeceda (konečná množina vstupných symbolov), $q_0 \in K$ je začiatočný stav, $F \subseteq K$ je množina koncových (akceptačných) stavov, a q je zobrazenie $\delta : K \times \Sigma \mapsto K$.

Konečno-stavový automat akceptuje slovo $w = w_0 w_1 \dots w_N$ zložené zo sym-

bolov $w_i \in \Sigma$, ak $\delta(\ldots\delta(\delta(q_0, w_0), w_1), \ldots, w_N) = p$ pre stav $p \in F$. Jazyk akceptovaný automatom A je množina $L(A) = \{w \mid \delta(q_0, w) \in F\}$, čiže slovo w je akceptované automatom A, ak postupnosť stavových prechodov generovaná symbolmi slova w vedie z počiatočného stavu do nejakého koncového stavu. Každý jazyk, ktorého rozpoznáva nejaký FSA, sa nazýva *regulárny jazyk*.

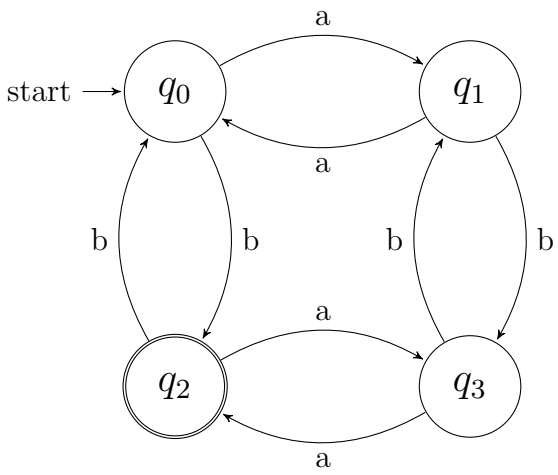
2.3.2 Rozdelenie FSA

Konečno-stavové automaty sa najčastejšie rozdeľujú podľa správania prechodovej funkcie δ na *deterministické konečné automaty (DFA)* a na *nedeterministické konečné automaty (NFA)* . Deterministický automat bol popísaný v predošlej časti, NFA v porovnaní nepriďeluje každému vstupnému symbolu jeden stav z F, ale podmnožinu stavov z F, tj. $\delta : K \times \Sigma \mapsto \mathcal{P}(K)$ [1]. Hoci výpočtová sila oboch modelov je rovnaká, táto vlastnosť umožní automatu rozhodovať sa, tj. robiť viacero rôznych výpočtov na jednom slove.
V tejto práci budeme zaoberať sa jedine s deterministickými konečnými automatmi.

2.3.3 Reprezentácia FSA

Konečné automaty (presnejšie prechodové funkcie automatov) zvyčajne zapisujeme dvomi spôsobmi: prechodovou tabuľkou alebo prechodovým diagramom. Nakoľko automaty so svojou štruktúrou sa veľmi podobajú na ohodnotené grafy vychádza z toho aj ich diagramová reprezentácia: jednotlivé stavy sú vrcholy v grafe, hrany medzi nimi reprezentujú existenciu prechodu medzi stavmi a hrany sú ohodnotené prechodovými podmienkami. Akceptačné stavy

[1] $\mathcal{P}(X)$ je množina všetkých podmnožín množiny X



Obr. 2.5: Ukážka prechodového diagramu jednoduchého konečno-stavového automatu akceptujúceho jazyk $L = \{w \mid |w|_a \, mod \, 2 = 0, \, |w|_b \, mod \, 2 = 1\}$, tj. slová obsahujúce párny počet znakov 'a' a nepárny počet znakov 'b' nad abecedou $\Sigma = \{a, b\}$

sú zvyčajne kreslené dvojitým krúžkom.

Tabuľka 2.1: Prechodová tabuľka k predošlému automatu

	a	b
q_0	q_1	q_2
q_1	q_0	q_3
q_2	q_3	q_0
q_3	q_2	q_1

2.3.4 Konečno-stavové prekladače

Ak automatu dovolíme, aby nielen rozpoznával slová, ale pritom aj vytvoril nejakú výstupnú postupnosť symbolov, tak dostávame *konečno-stavový*

prekladač (*finite-state transducer - FST*). Formálna definícia FST je šestica $(K, \Sigma, \delta, \Gamma, q_0, F)$, kde pribudol nový symbol Γ, označujúca konečnú abecedu pre výstup (konečnú množinu výstupných symbolov) a prechodová funkcia sa zmenila na $\delta : K \times \Sigma \mapsto K \times \Gamma$.

Výstupne symboly generované takýmto prekladačom môžu označovať volania príslušných procedúr. Takýto automat sa teda stáva riadiacou architektúrou napr. robotického systému, ktorá pracuje v reálnom čase, alebo inteligentného agenta, ktorý vykonáva akcie v nejakom prostredí.

2.3.5 Evolúcia FSA

Ako už bolo uvedené v sekcii 2.1.5, topológia neurónových sietí (hlavne rekurentných NS - vďaka tomu, že nemajú obmedzenia na smer výpočtu) pripomína štruktúru ohodnoteného, orientovaného grafu, čo sa vzťahuje aj na konečno-stavové automaty (2.3.3) - rozdiely sú len v ohodnotení hrán a v spôsobe priebehu výpočtu (NS majú aktívnych všetky neuróny - vrcholy, kým FSA len práve jeden). Tieto rozdiely nám vôbec nebránia v tom, aby sme rovnakým spôsobom ako u NS nemohli konečno-stavové automaty alebo prekladače evolvovať rôznymi neuroevolučnými metódami - napríklad NEAT-om. Idea samozrejme nie je nová, korene evolvovania stavových automatov siahajú späť až do roku 1962 (Fogel L J - Autonomous Automata), keď už na reprezentáciu genotypu v evolučnom programovaní používali konečno-stavové automaty.

Chellapilla a Czarnecki dokázali, že reprezantácia problému s modulárnym FSM môže byť, a často je efektívnejší spôsob, ako s nemodulárnym FSM (napr. bitovými reťazcami), ako demonštráciu vybrali problém mravca (Chellapilla, Czarnecki - A Preliminary Investigation into Evolving Modular Finite StateMachines, 1999).

Frey a Laugering prechodovú funkciu reprezentovali stromovou štruktúrou, vo vyšších uzloch sú vetviace podmienky a v nižších jednotlivé prechody a výstupy (Frey, Laugering - Evolving Strategies for Global Optimization - A Finite State Machine Approach, 2001).

Lucas [5] už v roku 2003 evolvoval konečno-stavové prekladače, ktoré transformovali reťazové obrazové kódy.

Horihan a Lu dokázali evolvovať FSM, ktorý akceptoval slová jazyka generovaného rôznymi regulárnymi gramatikami (Horihan J, Lu Y-H - Improving FSM Evolution with Progressive Fitness Functions, 2004).

Petrovič v roku 2009 [4] úspešne aplikoval prostriedky NEAT-u na evolúciu konečno-stavových prekladačov. Vo svojej práci ukázal nielen to, že adaptovanie NEAT-u na evolúciu FSM je naozaj realizovateľné, ale aj to, že dá sa takýmto spôsobom dosiahnuť podobné, alebo aj lepšie výsledky ako s klasickými evolučnými algoritmami. Jeho experiment spočíval v evolvovaní automatu, ktorý vo vstupnej postupnosti symbolov prepíše všetky nulové symboly podľa najbližsieho nenulového symbolu smerom vľavo. Evolúcia NEAT-om v každom behu našla riešenie a to v lepšom čase ako v prípade klasického spôsobu.

Hreha v roku 2011 v rámci svojej diplomovej práce [3] navrhol svoj vlastný algoritmus na evolvovanie stavových automatov, inšpirovaný metódou NEAT, a jej efektívnosť porovnal s existujúcou implementáciou pre daný problém (tiež realizovaný pomocou NEAT-u). Efektívnosť novej metódy ukázal na dvoch experimentoch, jeden z nich bol problém Minesweeper, ktorého pôvodné riešenie bolo predstavené spolu s NEAT-om (Stanley, Miikkulainen, 2002). Cieľom Minesweeper-a je prejsť v ohraničenom čase čo najväčšiu plochu priestoru, pričom v priestore sa nachádzajú prekážky a Minesweeper pomocou senzorov môže zistiť či pred ním je nejaká prekážka alebo už prejdená cesta.

Hrehové modifikácie zahŕňali zmenu konfiguračných parametrov a topológie neurónovej siete, aby NS hneď na začiatku obsahoval rekurentné linky naspäť, aby bolo možné dostať sa z jedného stavu do druhého a opačne; ďalej predefinoval operáciu mutácie a kríženia - mutoval nielen prechodové podmienky ale aj akcie linkov a tieto podmienky navzájom kombinoval. V druhom experimente sa jedinci naučili navštevovať rohy priestoru v používateľom zadanom poradí. Táto nová metóda pri správnej inicializácii dosiahla lepšie priemerné výsledky ako pôvodné riešenia.

Kapitola 3

Návrh

Hlavným výstupom práce je aplikácia, ktorá umožňuje spustenie konfigurovateľného evolučného algoritmu v štýle NEAT na rôznych experimentoch riadených stavovými automatmi, čiže prostredie na evolúciu FSA. Pomocou nej sme vytvorili naše testy a ich výsledky sú uvedené v ďalšej kapitole, dosiahnuté meraním výkonu evolučného algoritmu v rôznych situáciach s rôznymi parametrami a nastaveniami. Účelom tejto kapitoly je predstaviť štruktúru tohto algoritmu, aké postupy a princípy sú v ňom aplikované, ďalej popisať použité parametre, experimenty a našu aplikáciu.

3.1 Aplikácia

Ako už bolo spomenuté, základom celej aplikácie je evolučný algoritmus, ktorý dovolí študovať vplyv jednotlivých nastavení s cieľom najdenia optimálnych parametrov a verzie algoritmu pre rozličné typy úloh. Z tohto pohľadu funguje ako menšie rámcové prostredie (*framework*), pretože dovolí používateľovi-výskumníkovi, aby zadefinoval svoje vlastné verzie algoritmu a svoje vlastné experimenty.

Nové algoritmy sú vytvorené dedením od triedy *NEATAlgorithm*, každý dôležitý aspekt evolúcie je zapuzdrený do príslušnej virtuálnej metódy, pripravenej na prepísanie.
Experimenty musia implementovať interface *IExperiment*, ktorý vynúti naprogramovanie hlavných vlastností experimentu (ako napr. množinu všetkých možných udalostí, akcií a prekladov) a mechanizmus vyhodnotenia automatov. Okrem toho každý experiment môže mať svoje parametre, ktoré budú načítané a zmeniteľné v grafickom režime a spôsob simulácie.
Aplikácia podporuje dva režimy: grafický a batch.
Grafický je spustený hlavným *exe* súborom (*BenchmarkDepot.exe*), parametre sú vtedy nastavené v grafickom okne, priebeh evolúcie je vizualizovaný v grafe podľa stavu aktuálneho najlepšieho jedinca a dôležité údaje sú logované, následne umožňuje simuláciu výsledného automatu v konzolovom okne.
Batch režim slúži na jednoduché spustenie experimentov. Štartuje sa súborom *QuickStart.bat* a inicializuje parametre a nastavania podľa priloženého *xml* súboru (meno súboru je druhým parametrom ktorý dostane aplikácia pri štarte, prvý je reťazec *console*). Na rozdiel od grafického režimu umožňuje spustiť evolúciu n-krát po sebe, pre každý beh vygeneruje textový súbor údajov a pred ukončením ich spája do jedného *csv* súboru pre jednoduchšie spracovanie.

3.2 Algoritmus

3.2.1 Inicializácia populácie

Každá dôležitá vlastnosť evolúcie je nastaviteľná cez parametre okrem spôsobu inicializácie. Kvôli tomu sú vytvorené tri rôzne verzie algoritmu:

- *Full NEAT*- je dodržaný princíp mininalizácie, každý počiatočný jedinec je vytvorený s dvomi stavmi a s náhodným prechodom medzi nimi
- *Initial random NEAT*- štruktúra každého jedinca je stochastická, vytvorená z náhodných počtov stavov a prechodov. Počet stavov ja náhodne vybratý z intervalu $[MaxIndividualSize/2, MaxIndividualSize]$, počet prechodov z intervalu $[x, 2x)$, kde x je počet stavov
- *Complete structure NEAT*- jedince sú inicializované ako kompletné grafy s maximálnym počtom vrcholov (t.j. parameter *MaxIndividualSize*), každý prepojený s každým. Vlastnosti prechodov sú samozrejme náhodné

3.2.2 Selekcia

Pri návrhu, jedným z naších cieľov bolo, aby algoritmus bol dostatočne flexibilný, aby jednotlivé časti boli čo najjednoduchšie zmeniteľné. Vďaka tomu je realizovateľné použitie rôznych selekčných operátorov, prípadne ich kombinácia.
Sú implementované nasledujúce metódy selekcie:

- *elitizmus*
- *turnajová selekcia* - s nastaviteľnou veľkosťou turnaja
- *selekcia s orezávaním* - je to skôr modifikácia, je nastaviteľná veľkosť orezanej časti, z prežitých bude náhodne vybratý jedinec (jedince) na rozmnožovanie

3.2.3 Druhy

Boli vykonané menšie zmeny pri realizovaní rozdelenia populácie do druhov oproti článku. [2.2.4]

Nezdalo sa nám vhodné použiť konštantnú prahovú hodnotu kompatibility, nakoľko jednou z hlavných vlastností NEAT je evolvovanie štrukúry jedincov. Vďaka tomu hodnota použitá na začiatku nemusí byť vhodná aj na konci, hlavne, ak sa štruktúra jedincov radikálne zmenila. Kvôli tomu prahová hodnota sa dynamicky zmení, podľa počtu druhov. Bol zavedený parameter *CriticalSpeciesCount* (kritický počet druhov), a ak reálny počet druhov presiahne toto číslo, prahová hodnota je inkrementovaná s deltou (*CompatibilityThresholdDelta*). Na druhej strane, ak počet druhov je menší ako 1% veľkosti celej populácie, prah je znížený. Tým pádom vždy bude nájdené nastavenie, ktoré udržuje ideálny počet druhov.

Proces zmeny reprezentanta bol tiež modifikovaný. Pôvodne, na konci každej generácie je vybratý jedinec s najvyššou fitness hodnotou ako nový reprezentant. Túto možnosť sme nechali (*UseNormalizedRepresentant*), bola ale pridaná alternatívna verzia, keď nového reprezentanta nevyberieme z populácie daného druhu, ale skonštruujeme podľa najčastejšie sa vyskytujúcich prechodov. Prechody su rozdelené do skupín podľa inovačného čísla a prechodovej podmienky - prechody s rovnakým inovačným čislom, ale s rôznou podmienkou môžu reprezentovať rôznu funkcionalitu jedinca, takže nestačí selektovať podľa inovačných čísel - a je pridaný práve jeden, najčastejšie sa vyskytujúci, z každej skupiny. Naša hypotéza je taká, že použitie týchto "normalizovaných"reprezentantov vedie k lepšiemu pokrytiu distribúcie jedincov, prípadne spájaním druhov s priveľmi podobnými reprezentantami redukuje ich číslo.

Posledná zmena, ktorú sme aplikovali vyplýva z podstaty stavových automatov: vo vzorci kompatibility je použitie priemerného rozdielu váh matching génov - v prípade FSA ale väčšinou nemáme spojité hodnoty na prechodoch. Namiesto toho premenná $\overline{W}$ je vypočítaná z rovnosti prechodovej akcie a

prechodového spúšťača. V prípade úplnej rovnosti hodnota premennej bude $MatchingWeightDifferenceValue$, jeho polovica ak sa zhoduje iba jedna zložka, inak 0.

3.2.4 Vynulovanie zoznamu inovácií

Pôvodný článok obsahoval jedno kontraproduktívne stanovenie o vedení zoznamu inovácií. Bolo totiž uvedené [2.2.2], že zoznam všetkých inovácií na konci každej generácie je zahodený, čo v konečnom dôsledku môže viesť k tomu, aby tie isté štruktúrové zmeny v každej generácii dostali iné čísla, teda historický pôvod nebude presne zistiteľný a operácia kríženia sa stane nespoľahlivou.
Bol to pravdepodobne preklep, pre zaujímavosť sme ale zaviedli parameter na prepínanie zmazania tohto zoznamu.

3.2.5 Testované verzie algoritmu

Nasledujúce nastavenia budu použité pri spúšťaní experimentov (ďalej sa na ne budeme odvolávať pomocou uvedených skratiek):

- **SN** - štandard NEAT [3.3.2]
- **VD** - vypnuté druhy
- **DEL** - rozšírené mutačné oprátory - NEAT podporuje len vyvíjajúcu evolúciu, sú zadefinované len oprátory pridávania nových štuktúr a zmena existujúcich. Sme zvedaví, ako evolúciu ovplyvňuje zavedenie operátorov odstránenia prechodov a stavov
- **VE** - vypnutý elitizmus
- **VT** - vypnutá turnajová selekcia

Ďalej budú testované aj všetky tri inicializačné metódy pri rôznych veľkostiach populácie.

3.3 Parametre

Konfigurovateľné parametre sú dôležitou súčasťou každého algoritmu, ktorého vnútorný stav tak silne závisí od hodnôt početných premenných a konštánt, a pravdepodobnostných výpočtov ako je to v prípade evolučných algoritmov, obvzlášť, ak ide o testovanie efektívnosti jednotlivých aspektov algoritmu. Možnosť jednoduchej zmeny parametrov je kritická, ak chceme experimentovať s rôznymi nastaveniami algoritmu.
Kvôli prehľadnosti, množninu naších parametrov sme rozdelili do dvoch skupín podľa ich použitia: všeobecné parametre evolučných algoritmov a parametre ktoré sú špecifické pre NEAT.

3.3.1 Všeobecné parametre EA

- **InitialPopulationSize** - veľkosť počiatočnej populácie
- **MaxPopulationSize** - maximálna veľkosť populácie, zároveň je to aj bežná veľkosť populácie, nakoľko počas každej generácie je vytvorených toľko jedincov, aby celá populácia bola naplnená
- **MaxIndividualSize** - maximálna veľkosť jedinca, čo v tomto prípade znamená maximálny počet stavov automatu. Ovplyvňuje teda aj operátory kríženia a mutácie, keďže nikdy nemôže byť vytvorený väčší jedinec ako je povolené
- **GenerationThreshold** - maximálny povolený počet generácií, ak počet generácií dosiahne túto hodnotu, algoritmus skončí

- **SelectionProportion** - nastavuje, koľko percent najlepších jedincov môže byť vybratých na rozmnožovanie. Slúži teda ako prahová hodnota pre selekciu s orezávaním, ak je nastavené na 100% (hodnota 1.0), táto verzia selekcie je vypnutá
- **TournamentSize** - reprezentuje veľkosť turnaja v turnajovej selekcii, t.j. aká veľká je množina jedincov, z ktorej berieme nového rodiča. Logicky, hodnota 1 vypne túto selekciu
- **Elitism** - prepinač elitizmu
- **ReplacementProportion** - koľko percent aktuálnej populácie bude nahradených v ďalšej generácii, namiesto nenahradených sa kopírujú nejlepšie jedince aktuálnej populácie
- **CrossoverProbability** - pravdepodobnosť, že nový jedinec bude vytvorený krížením. V opačnom prípade kopíruje štruktúru rodiča.
- **StateDeletionMutationProbability** - pravdepodobnosť aplikovania mutácie odstránenia náhodného stavu jedinca
- **TransitionDeletionMutationProbability** - pravdepodobnosť aplikovania mutácie odstránenia náhodného prechodu jedinca
- **TransitionActionMutationProbability** - pravdepodobnosť aplikovania mutácie zmeny prechodovej akcie jedinca
- **TransitionTranslationMutationProbability** - pravdepodobnosť aplikovania mutácie zmeny prechodového prekladu jedinca
- **TransitionTriggerMutationProbability** - pravdepodobnosť aplikovania mutácie zmeny prechodovej podmienky jedinca

- **MutationCount** - počet po sebe aplikovaných, potenciálnych mutácií na novovytvorenom jedinci

3.3.2 Parametre špecifické pre NEAT

- **SpeciesAllowed** - prepínač, či je povolené použitie druhov. Ak je vypnuté, všetky parametre súvisiace s druhmi sú ignorované
- **CriticalSpeciesCount** - kritický počet druhov je hraničná hodnota, od ktorého viac druhov nechceme mať. Algoritmus ale negarantuje, že v danom okamihu výpočtu nebude naraz prítomných viac druhov. Viď 3.2.3
- **AllowedSpeciesStagnatedGenerationCount** - druh stagnuje, ak počet jedincov po rozdelení je menej ako 1% veľkosti populácie. Parameter nastavuje koľko generácií môže druh stagnovať v sérii, kým bude odstránený.
- **CompatibilityThreshold** - prahová hodnota kompatibility [2.2.4]
- **MinCompatibilityThreshold** - minimálna prahová hodnota kompatibility - potrebné kvôli dynamickej zmene tejto hodnoty
- **CompatibilityThresholdDelta** - delta zmeny prahovej hodnoty kompatibility. Viď 3.2.3
- **CoefExcessGeneFactor** - prvá konštanta vo vzorci kompatibility [2.2.4]
- **CoefDisjointGeneFactor** - druhá konštanta vo vzorci kompatibility [2.2.4]
- **CoefMatchingWeightDifferenceFactor** - tretia konštanta vo vzorci kompatibility [2.2.4]

- **MatchingWeightDifferenceValue** - aká hodnota je použitá namiesto priemerného rozdielu váh matching génov. Viď 3.2.3
- **UseNormalizedRepresentant** - prepínač, či algoritmus má používať ako reprezentanta normalizovaného alebo najlepšieho jedinca. Viď 3.2.3
- **AddNodeMutationProbability** - pravdepodobnosť aplikovania mutácie pridania nového stavu do jedinca medzi dvomi náhodnými, ale prepojenými stavmi
- **AddTransitionMutationProbability** - pravdepodobnosť aplikovania mutácie pridania nového prechodu do jedinca medzi dvomi náhodnými stavmi. Všetky vlastnoti prechodu sú stochastické, ak už vedie prechod z prvého stavu do druhého s rovnakou podmienkou, tak je nahradený
- **InnovationResetPerGeneration** - prepínač, či zoznam inovácií by mal vynulovať na konci každej generácie. Viď 3.2.4

3.3.3 Štandardné nastavenia

Aplikácia, ktorá bola vytvorená na spustenie experimentov umožňuje vytvorenie a použitie sád preddefinovaných hodnôt (tzv. *preset*). Detaily sady, ktorú používame ako štandard pre NEAT, sú zachytené v tabuľke 3.1.

3.4 Experimenty

Bolo vytvorených 5 jednoduchých testovacích scenárov, jednak kvôli rozmanitosti, na druhej strane na demonštráciu, aké funkčnostné výhody majú stavové automaty oproti neurónovým sietiam.
Dva experimenty sa sústredia na prekladanie. Prvý z nich evolvuje automat,

Tabuľka 3.1: Štandardné hodnoty parametrov

InitialPopulationSize	1500	SpeciesAllowed	*true*
MaxPopulationSize	1500	CriticalSpeciesCount	35
MaxIndividualSize	10	AllowedSpeciesStagGenCount	3
GenerationThreshold	2000	CompatibilityThreshold	0.8
SelectionProportion	1	MinCompatibilityThreshold	0.1
TournamentSize	2	CompatibilityThresholdDelta	0.02
ReplacementProportion	1	CoefExcessGeneFactor	1
CrossoverProbability	0.5	CoefDisjointGeneFactor	1
StateDeletionMutProb	0	CoefMatchWeightDiffFactor	1
TransitionDeletionMutProb	0	MatchingWeightDiffValue	1
Elitism	*true*	UseNormalizedRepresentant	*false*
TransitionActionMutProb	0.9	AddNodeMutationProb	0.75
TransitionTranslMutProb	0.9	AddTransitionMutationProb	0.9
TransitionTriggerMutProb	0.9	InnovationResetPerGen	*false*
MutationCount	1		

ktorý dostane postupnosť pozostávajúci zo znakov ’0’ a ’1’, a vráti ďalší reťazec, v ktorom tieto znaky sú prevrátené - napr. pre vstup ’0110’ vráti ’1001’. Druhý bol inšpirovaný prácou Petroviča [4] - prekladač, ktorý vo vstupnom reťazci číslic prepíše každý výskyt nuly na poslednú nenulovú číslicu zľava (príklad: ’0102300’ $\mapsto$ ’0112333’). Počet možných číslic je konfigurovateľný. Tretí slúži na ukážku podmienečného prechodu automatu, t.j. vykonávanie prechodu nezávisí od aktivácie spúšťača, ale od ďalšej podmienky (napr. hodnota nejakého senzora je menšia ako stanovený limit) - v našom prípade je to termostat, ktorý pozná minimálnu a maximálnu teplotu a evolvujeme dolnú a hornú hranicu ideálnej teploty. Na vstupe dostane čísla a podľa toho

či dané číslo padne do tohto rozsahu vráti boolovskú hodnotu. Okrem dvoch čísel algoritmus musí nájsť aj optimálny operátor - v našom prípade boli definované nasledujúce: menší ako x, väčší ako x, rovná sa x, leží/neleží v intervale $< x, y >$.

Vo štvrtom experimente hľadáme FSA, ktorý sleduje preplnenie miestnosti. Automat dostane signál keď prichádza a odchádza osoba a reaguje na to generovaním príslušného znaku v závislosti od toho, či daná akcia je realizovateľná - t.j. nemôže nikto odísť ak miestnosť je prázdna a nemôže nikto prísť ak počet osôb dosiahol stanovený limit. Počet možných prechodov sa dá zdvojnásobniť rozpoznaním žien od mužov.

V týchto štyroch experimentoch fitness jedinca sa určuje ako percentuálny podiel správnych odpovedí. Vstupy sú v každom prípade generované náhodne.

Posledný experiment evolvuje automat na riadenie agenta, ktorý zbiera pohybujúcich sa objektov na ploche s rozmerom $N \times N$. Na ploche sa vždy nachádza práve jeden objekt a v každom kroku výpočtu môže (ale nemusí) urobiť aj agent, aj daný objekt vertikálne alebo horizontálne jeden krok. Objekt má svoje vnútorné počítadlo, ktorého hodnota sa dekrementuje v každom kroku a ak dosiahne nulu, objekt sa zmizne. Ďalej, v každom kroku sa objaví s nastaviteľnou pravdepodobnosťou nepriateľ, a kým je prítomný, agent sa nemôže hýbať (objekty ale môžu). Agent pomocou svojich senzorov môže zistiť, že v ktorom zo štyroch smerov sa nachádza objekt, a či nepriateľ je prítomný. Výsledná fitness jedinca sa rovná počtu prežitých krokov, simulácia sa končí, ak nepriateľ odhalí agenta, alebo ak agent stratí stanovený počet objektov.

Kapitola 4

Výsledky a diskusia

Teraz by sme predstavili výsledky získané spustením uvedených experimentov [3.4] s rôznymi nastaveniami v rámci vytvoreného prostredia. Úspešnosť evolúcie je zvyčajne reprezentovaná počtom potrebných generácií alebo počtom vykonaných ohodnotení jedincov na najdenie dostatočne kvalitného jedinca - naša aplikácia podporuje oba prístupy, pre lepšiu prehľadnosť ale budeme uvádzať len počet generácií.
Ako už bolo spomenuté, evolučné algoritmy vo veľkej miere spoliehajú na náhodnosť, teda rovnaké nastavenia ľahko môžu viesť k značne rôzným záverom. Práve preto každý uvedený výsledok je dosiahnutý spustením 10 behov.

Použité skratky:

- *Alg* - verzia algoritmu [3.2.5]
- *Init* - verzia inicializácie [3.2.1]
- *PopSize* - veľkosť testovanej populácie
- *AvgReqGen* - aritemtický priemer potrebných generácií

- *MinReqGen* - najmenší z potrebných generácií
- *FailCount* - počet neúspešných behov

4.1 Experiment 1

Prvý experiment je najjednoduchší, priestor riešení je dostatočne malý na to, aby pri veľkých populáciach ideálne riešenie sa dalo nájsť aj počas inicializácie. Populáciu sme teda limitovali na 100 jedincov, riešenia boli aj tak nájdené rýchlo bez ohľadu na zvolený algoritmus. Na druhej strane náhodná inicializácia v každom prípade trvala značne dlhšie, bude to pravdepodobne tým, že riešenie sa dá zostrojiť aj z jedného stavu s dvoma prechodmi, čiže minimalizácia má výhodu, a kompletná štruktúra má väčšiu šancu hneď na začiatku vygenerovať jedinca blízkeho ideálnemu.

Tabuľka 4.1: Výsledky prvého experimentu

Alg	*Init*	*PopSize*	*AvgReqGen*	*MinReqGen*	*FailCount*
SN	FN	100	6.2	4	0
SN	IRN	100	44.3	15	0
SN	CSN	100	3.4	1	0
VD	FN	100	15.1	3	0
VD	IRN	100	31.2	10	0
VD	CSN	100	13.9	1	0
DEL	FN	100	7.6	4	0
DEL	IRN	100	50.5	15	1
DEL	CSN	100	6.5	1	0
VE	FN	100	8.9	5	0

VE	IRN	100	47.5	11	1
VE	CSN	100	7.7	2	0
VT	FN	100	20.4	4	0
VT	IRN	100	51.3	29	0
VT	CSN	100	18	2	0

4.2 Experiment 2

Druhý experiment sme spustili v dvoch verziách: rozdiel je len v počte možných čísel (okrem 0), ktoré môžu vyskytovať v testovacích reťazcoch. V tabuľke to označuje stĺpec *StrNums*, a testovali sme s číslami 2 a 3. Predbežné testy nám ukázali, že ak necháme maximálnu veľkosť jedincov na štandardnom 10, tak verzia algoritmu s kompletným grafom nestačila na nájdenie riešenie. Kvôli tomu sme tento parameter nastavili na $n+2$, kde n je najmenšia možná veľkosť ideálneho automatu (ak $StrNums = x$, potom najmenšia veľkosť je $x + 1$, viď obr. 4.2). Zjavne, táto verzia algoritmu môže byť efektivne použitá len vtedy, ak priestor možných kandidátov je dostatočne malý, alebo ak dokážeme dobre odhadnúť horný limit počtu stavov.

Pri testovaní boli použité náhodne vygenerované reťazce s 75% pravdepodobnosťou výskytu charakteru 0, s veľkosťou populácie 800 v prípade 2, 800 a 1500 v prípade 3 čísel.

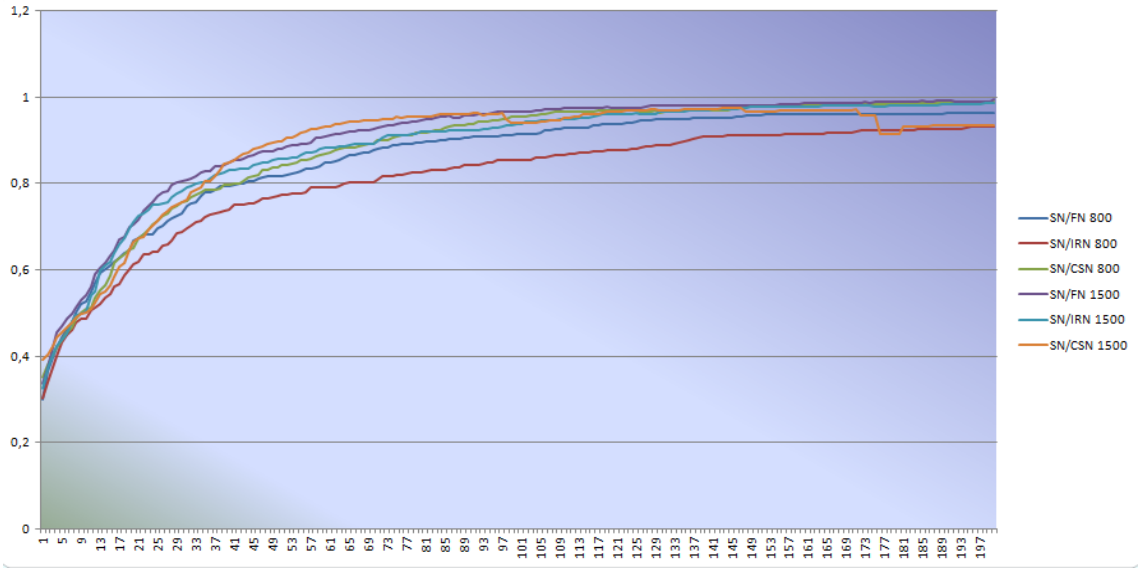
Tabuľka 4.2: Výsledky druhého experimentu

StrNums	*Alg*	*Init*	*PopSize*	*AvgReqGen*	*MinReqGen*	*FailCount*
2	SN	FN	800	59.3	38	0
2	SN	IRN	800	55.9	34	0

2	SN	CSN	800	59.4	22	0
2	VD	FN	800	108.8	39	1
2	VD	IRN	800	76.8	53	0
2	VD	CSN	800	51.3	28	0
2	DEL	FN	800	83.1	66	1
2	DEL	IRN	800	79.9	43	1
2	DEL	CSN	800	75	26	1
2	VE	FN	800	50.7	59	0
2	VE	IRN	800	60	42	0
2	VE	CSN	800	47.4	37	0
2	VT	FN	800	190.7	119	0
2	VT	IRN	800	341	179	0
2	VT	CSN	800	221.8	76	0
3	SN	FN	800	484	264	0
3	SN	IRN	800	671.6	361	0
3	SN	CSN	800	271.5	125	0
3	SN	FN	1500	208.8	87	0
3	SN	IRN	1500	267	125	0
3	SN	CSN	1500	142.6	75	0
3	VD	FN	800	465.5	77	1
3	VD	IRN	800	549.8	272	3
3	VD	CSN	800	385.7	146	1
3	VD	FN	1500	338.9	121	2
3	VD	IRN	1500	422	154	3
3	VD	CSN	1500	268.9	97	0
3	DEL	FN	800	522.7	304	1

3	DEL	IRN	800	753.4	543	3
3	DEL	CSN	800	602.6	363	1
3	DEL	FN	1500	358.2	160	1
3	DEL	IRN	1500	472.2	168	0
3	DEL	CSN	1500	356.3	222	0
3	VE	FN	800	463.2	201	0
3	VE	IRN	800	619.2	278	0
3	VE	CSN	800	307.3	88	0
3	VE	FN	1500	215.7	112	0
3	VE	IRN	1500	327	117	0
3	VE	CSN	1500	188.4	67	0
3	VT	FN	800	648.7	453	4
3	VT	IRN	800	755.2	526	5
3	VT	CSN	800	667.5	412	4
3	VT	FN	1500	463.3	388	1
3	VT	IRN	1500	512.8	478	3
3	VT	CSN	1500	388.7	397	0

Pri testovaní s dvoma číslami ani jeden z algoritmov nemal väčšie problémy, výsledky sú podľa očakávania. Vypnutá turnajová selekcia spôsobila výrazne horšie výsledky ako ostatné prístupy, riešenie ale bolo nájdené v každom behu. Zaujímavejšie sú ale výsledky väčšieho automatu. Experimenty sme spustili s dvoma rôznymi veľkosťami populácie a väčšie číslo v niektorých prípadoch to viedol k zníženiu počet potrebných generácií na polovicu. Hlavne v prípadoch, keď druhy boli zapnuté - zjavne potrebujú dostatočne veľa jedincov, aby mohli byť vytvorené druhy, ktoré sa dostanú aj do vzdialenejších bodov prehľadávacieho priestoru.

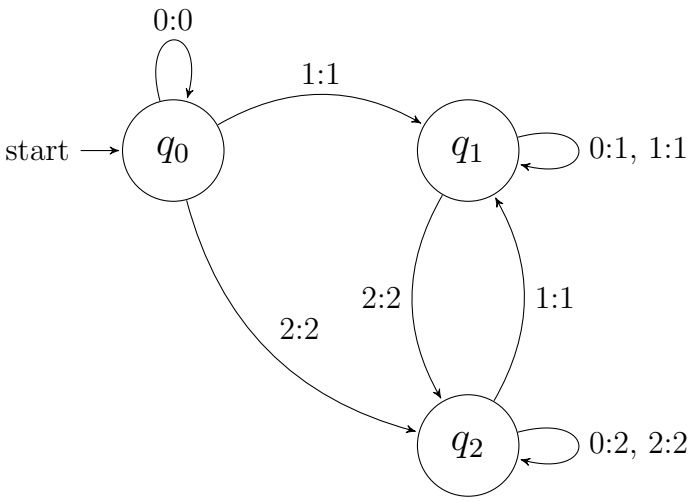


Obr. 4.1: Porovnanie priebehu priemerných hodnôt najlepších fitness v experimente 2 pre rozličné veľkosti populácie

Čo sa týka algoritmov, tak ako predtým najrýchlejšie boli algoritmy so štandardnými hodnotami a najhoršie výsledky spôsobilo vypnutie turnajovej selekcie. Môžeme to vysvetliť tým, že orezávanie bolo vypnuté, takže selekcia fungovala úplne náhodne.
Najúspešnejšia bola inicializácia kompletným grafom, čo nie je až tak prekvapivé, nakoľko parameter maximálneho počtu stavov bol nastavený optimálne a tým pádom ušetril čas skonštruovania samotného automatu. Keď sme ale zapli možnosť odstránenia prechodov a stavov, táto výhoda bola zoslabnená, a priemerný počet generácií bol skoro rovnaký ako pri minimálnej inicializácii.

4.3 Experiment 3

Evolvovali sme FSA, ktorý nájde ideálnu teplotu medzi 50 a 60 stupňov, ak dolný a horný limit sú 0 a 100. Môže to byť v princípe podobne jednoduchý



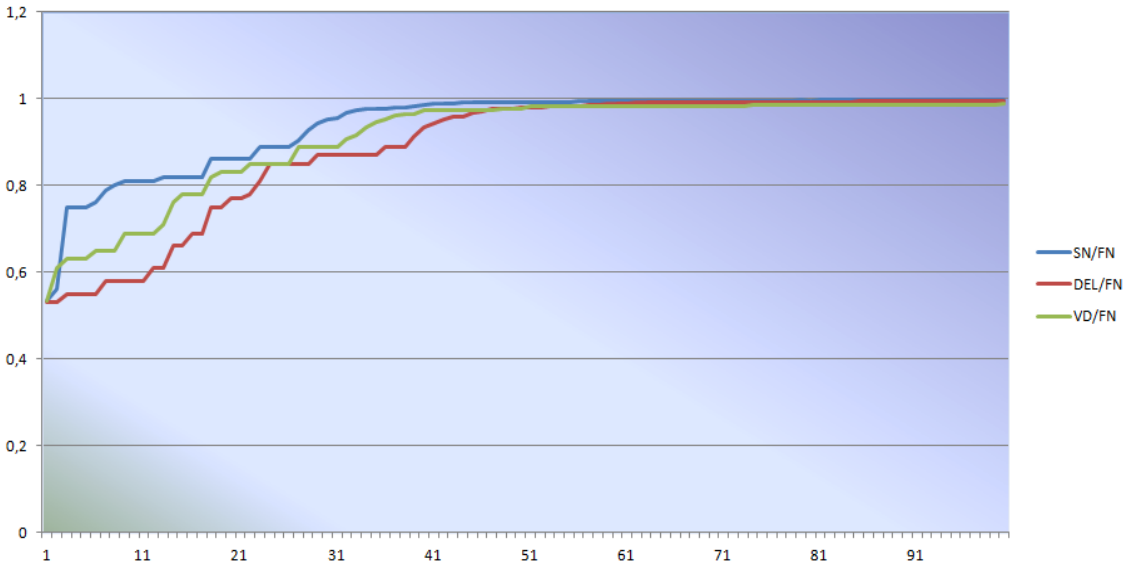
Obr. 4.2: Ukážka najmenšieho možného automatu v druhom experimente, ak $StrNums = 2$

automat ako v prvom experimente, napriek tomu nájdenie trvá oveľa dlhšie - ako sa to dá aj vidieť na výsledkoch. Na obrázku 4.3 je vidno, že algoritmus v priemere po 30 generáciach už našiel, že potrebuje nejaké hranice, ďalej to už bolo len o nájdení týchto čísel, čo potom závisí jedine od náhody.

Tabuľka 4.3: Výsledky tretieho experimentu

Alg	*Init*	*PopSize*	*AvgReqGen*	*MinReqGen*	*FailCount*
SN	FN	500	240.9	54	0
SN	IRN	500	233.9	17	0
SN	CSN	500	213.3	16	0
VD	FN	500	756.6	647	6
VD	IRN	500	337.8	75	2
VD	CSN	500	238.4	12	0
DEL	FN	500	392.7	11	2
DEL	IRN	500	377.1	14	1

DEL	CSN	500	355.5	63	0
VE	FN	500	174.4	17	0
VE	IRN	500	273.5	1	1
VE	CSN	500	345.3	76	1
VT	FN	500	582.2	241	0
VT	IRN	500	289.9	31	0
VT	CSN	500	687.4	344	6



Obr. 4.3: Ukážka priebehu evolúcie v experiment 3

4.4 Experiment 4

Veľkosť miestnosti sme nastavili na 3 s rozpoznaním mužov od žien a s maximálnym dovoleným počtom generácií 1000. Hoci sa nám zdalo, že bude to dostatočne veľa času, z použitých algoritmov iba polovica konvergovala tak rýchlo, aby chybovosť nebola vyššia ako 20%. Pomocou štandardných

NEAT nastavení boli vyhľadávania najrýchlejšie, aj výhody druhov aj selekčné metódy sa ukázali ako užitočné pri riešení tejto úlohy.

Tabuľka 4.4: Výsledky štvrtého experimentu

Alg	*Init*	*PopSize*	*AvgReqGen*	*MinReqGen*	*FailCount*
SN	FN	1000	521.2.1	61	1
SN	IRN	1000	496.6.8	53	1
SN	CSN	1000	569.4.7	246	0
VD	FN	1000	766.8	116	2
VD	IRN	1000	703.5	218	3
VD	CSN	1000	899.1	318	3
DEL	FN	1000	916.4	883	3
DEL	IRN	1000	956.2	917	5
DEL	CSN	1000	685.9	237	2
VE	FN	1000	925.2	737	2
VE	IRN	1000	953.5	846	6
VE	CSN	1000	911.1	721	3
VT	FN	1000	917	966.2	4
VT	IRN	1000	987.5	942	7
VT	CSN	1000	895.6	477	3

4.5 Experiment 5

Výsledky posledného experimentu vôbec neprekvapia, každý beh skončil úspešným riešením s priemerným počtom generácií okolo 50. Veľkosť plochy bolo nastavené na 6×6, s 0.5% pravdepodobnosťou nepriateľa v každom kroku. Maximálnú veľkosť jednicov sme nastavili na 6, toľko stavov ale táto

úloha nepotrebuje, pravdepodobne kvôli tomu algoritmy, ktoré mali nastavené aj mutácie na odstránenie štruktúr skončili lepšie ako ostatné.

Tabuľka 4.5: Výsledky piateho experimentu

Alg	*Init*	*PopSize*	*AvgReqGen*	*MinReqGen*	*FailCount*
SN	FN	1000	45.3	31	0
SN	IRN	1000	52.7	38	0
SN	CSN	1000	49.3	42	0
VD	FN	1000	47.2	37	0
VD	IRN	1000	49.9	41	0
VD	CSN	1000	42.6	32	0
DEL	FN	1000	32.1	23	0
DEL	IRN	1000	30.8	25	0
DEL	CSN	1000	44.4	34	0
VE	FN	1000	39	19	0
VE	IRN	1000	51.3	34	0
VE	CSN	1000	57.3	41	0
VT	FN	1000	66.8	42	0
VT	IRN	1000	63.5	49	0
VT	CSN	1000	56.2	45	0

Kapitola 5

Záver

V tejto práci sme sa oboznámili s rôznymi aspektami evolučných výpočtov: ako môžu byť využité vlastnosti a mechanizmy prírodzenej evolúcie v pokročilých algoritmoch na riešenie optimalizačných problémov; predstavili sme si niekoľko druhov evolučných algoritmov a rôzne princípy v nich použitých. Dôraz bol kladený na jeden špecifický algoritmus: NEAT, pôvodne navrhnutý na evolvovanie neurónových sietí.

Ďalej sme si uviedli definíciu deterministického stavového automatu, ukázali sme, že aj keď ich využitie je značne rozlišné, majú rovnakú štruktúru ako neurónové siete. Práve preto s istými zmenami evolučné algoritmy môžu byť aplikované aj na konštrukciu stavových automatov s účeľom vykonania úloh, ktoré by neboli realizovateľné s neuronovými sieťami. Predstavili sme si dôležitejšie práce, ktoré s touto problematikou už niekedy v minulosti zaoberali.

Cieľom práce bolo navrhnúť a implementovať rámcové prostredie, ktoré umožňuje evolvovanie stavových automatov, umožňuje štúdium dôležitosti jednotlivých aspektov evolúcie pri rôznych typoch úloch a umožňuje experimentovať s nastaveniami algoritmu. Naprogramovať všeobecný a konfigurova-

teľný evolučný algoritmus inšpirovaný myšlienkami NEAT, vytvoriť niekoľko testovacích scenárov vo vyššie uvedenom prostredí a pomocou nich overiť funkčnosť toho algoritmu, a nakoniec prezentovať dosiahnuté výsledky.
Tento cieľ sa nám úspešne podarilo splniť, aplikácia stabilne funguje, je jednoducho rozšíriteľná s novými verziami algoritmu aj s novými experimentami a výsledky nás presvedčili, že algoritmus je korektne realizovaný.

Plány do budúcnosti:

- vizualizácia výsledku - výsledný automat evolúcie je zatiaľ len zalogovaný v textovej reprezentácii, chceli by sme implementovať interaktívnu vizualizáciu
- podpora grafických experimentov - po skončení algoritmu experiment môže byť spustený s naevolvovaným automatom v konzolovom okne, chýba ale možnosť grafickej simulácie
- pravdepodobnostná reprezentácia reprezentanta druhu - počas práce sme uvažovali o reprezentanta, v ktorom namiesto konkrétnych prechodov by bola uložená pravdepodobnosť každej prípustnej hodnoty (v prípade disktrétnych hodnôt, spojité hodnoty by mohli byť nahradené aritmetickým priemerom), vypočítané podľa aktuálnej populácie druhu
- rozšíriť možnosti evolúcie - pridanie nových selekčných metód, spôsobu inicializácie, atď ...
- vytvoriť zložitejšie experimenty - z dôvodu nedostatku času experimenty boli navrhnuté tak, aby ich realizácia nebola časovo náročná, ale máme v pláne rozšírenie aplikácie s komplexnejšími testovacími scenármi

Literatúra

[1] **Back, T** *Selective pressure in evolutionary algorithms: a characterization of selection mechanisms (1994), Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference*

[2] **Eiben A** *Introduction to Evolutionary Computing (2010) Springer, ISBN 978-3-662-05094-1*

[3] **Hreha S** *Evolúcia Stavových Automatov, diplomová práca, FMFI UK, Bratislava (2011)*

[4] **Petrovič P** *Evolúcia stavových automatov inšpirovaná metódou NEAT (2009), Kognícia a umelý život, Stará Lesná, 2009*

[5] **Lucas S M** *Evolving Finite State Transducers: Some Initial Explorations (2003), In European Conference on Genetic Programming, EuroGP'2003, 130-141.*

[6] **Mach M** *Evolucné algoritmy - Prvky a princípy (2009), Košice : Elfa, s.r.o., 2009. 250 s. [cit. 2011-05-01].*

[7] **Rovan B, Forišek M** *Formálne jazyky a automaty, FMFI UK, Bratislava (2013)*

[8] **Sinčák P, Andrejková G** *Neurónové siete - inžinierský prístup 1.diel, Košice (1996)*

[9] **Stanley K O, Miikkulainen R** *Evolving Neural Networks through Augmenting Topologie (2002), In Evolutionary Computation. 2002, vol. 10, no. 2, s 99–127. [cit. 2011-04-25].*

[10] **Vavak F, Fogarty T** *Comparison of Steady State and Generational Genetic Algorithms for Une in Nonstationary Environments (1996), Proc. of the 3rd IEEE Int. Conf. on Evolutionary Computation - ICEC'96, Nagoya Unversity, IEEE Publishing, Inc., pp192-195*

[11] **Zelinka I, Oplatková Z, Šeda M, Ošmera P, Včelař F** *Evoluční výpočetní techniky - Principy a aplikace (2009). Praha : BEN – technická literatura, 2009. 536 s. ISBN 978–80–7300–218–3.*