UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

EVOLÚCIA STAVOVÝCH AUTOMATOV

Diplomová práca

Študijný program: Aplikovaná informatika
Študijný odbor: 9.2.9 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2011 **Bc. Stanislav Hreha**

9958b536-090e-4166-831c-ecd9f2053d11

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Stanislav Hreha
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Evolúcia stavových automatov

Cieľ: Preskúmať spôsoby evolúcie stavových automatov. Navrhnúť vlastnú metódu, implemtnovať a overiť na štandardných príkladoch. Porovnať s inými metódami.

Vedúci: Mgr. Pavel Petrovič, PhD.

Dátum zadania: 13.11.2009

Dátum schválenia: 03.05.2011

za
prof. RNDr. Ivan Kalaš, CSc.
garant študijného programu

student

vedúci

Prehlásenie

Čestne prehlasujem, že som predloženú diplomovú prácu spracoval samostatne s použitím uvedenej literatúry a ďalších informačných zdrojov.

V Bratislave, 6.5.2011

.....

podpis autora práce

Poďakovanie

Chcel by som poďakovať všetkým, ktorí mi akýmkoľvek spôsobom (najmä testovaním) pomohli pri spracovaní tejto diplomovej práce. Moje poďakovanie patrí predovšetkým vedúcemu práce, Mgr. Pavlovi Petrovičovi, PhD., za vedenie a cenné pripomienky pri záverečnom spracovaní práce.

Osobitné poďakovanie patrí mojím rodičom, mojej priateľke a mojím najbližším, ktorí ma počas celého štúdia podporovali.

ABSTRAKT

Stanislav Hreha: Evolúcia stavových automatov.
Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky, Katedra aplikovanej informatiky
Diplomová práca, 59 strán, 2011

Predložená diplomová práca sa zaoberá evolúciou stavových automatov metódou NEAT. V úvodnej časti sa zameriava na potrebné teoretické znalosti o evolučných algoritmoch. V ďalších častiach popisuje neurónové siete a stavové automaty ako možné reprezentácie genotypov pre evolučné algoritmy. Kapitola o metóde NEAT zahŕňa informácie o NEAT a problémoch iných evolučných metód, ktoré táto metóda rieši. Praktická časť je zameraná a implementáciu NEAT metódy s reprezentáciou stavovými automatmi, a porovnanie dosiahnutých výsledkov na vybraných úlohách voči východiskovej implementácii s reprezentáciou neurónovými sieťami.

Kľúčové slová: evolúcia stavových automatov, NEAT, minesweeper

ABSTRACT

Stanislav Hreha: Evolution of state automata.
Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics
Diploma work, 59 pages, 2011

The diploma work is focused on evolution of state automata by method NEAT. Introductory chapter describes needed theoretical knowledge about evolutionary algorithms. In following chapters describes neural nets and state automata as possible representation of genotypes in evolutionary algorithms. Chapter about NEAT includes knowledge about NEAT and problems of other evolutionary methods that this method solves. Practical part is focused on implementing NEAT method with representation of finite state automata, comparing achieved results on selected tasks with source implementation with representation by neural nets.

Key words: evolution of state automata, NEAT, minesweeper

Obsah

Úvod

Evolúcia sama o sebe je veľmi fascinujúci jav. Chaoticky pôsobiaci neustály jav, stojí v pozadí každej živej bytosti na svete, od baktérii po človeka. Kým však človek dokázal prijať fakt, že niečo také vôbec existuje, bolo potrebné nazbierať množstvo priamych či nepriamych dôkazov. Dôvodom tohto odmietania evolúcie bolo najmä jej dlhé trvanie s pomalými a malými prejavmi.

S vývojom počítačov, sa evolúcia postupne presúva z okolitej prírody do binárnej sústavy strojov. Ba, evolúcia v počítačoch je dokonca niekoľkonásobne rýchlejšia ako v prírode, a zdá sa byť ohraničená len fantáziou programátorov a výskumníkov. Evolúcia binárnych jedincov je preto veľmi často skloňovanou témou, a s množstvom vzrušujúcich možnosti prináša ešte viac dôležitých otázok typu: ako reprezentovať takého jedinca v kóde, čo by taký jedinec mal vedieť vykonať?

Niekoľko možných odpovedí na niektoré otázky prinášajú evolučné algoritmy. Sústreďujú niekoľko možných reprezentácii jedincov, a snažia sa verne napodobňovať evolúciu v prírode.

Ďalšiu skupinu možných riešení prinášajú neurónové siete. Na rozdiel od EA sa nesnažia imitovať prírodnú evolúciu, ale relatívne jednoduchými výpočtami sa prispôsobovať na riešenie daných úloh či problémov.

Niekoľko štúdií prichádza s kombináciou NS a EA, pričom na niektorých úlohách zaznamenávajú lepšie výsledky ako tradičné metódy riešenia. Veľmi dobrú pozíciu si medzi týmito postupmi udržiava metóda NEAT formulovaná K. O. Stanleym a R. Miikkulainenom v roku 2002, ktorá rieši niektoré z problémov, s ktorými sa podobné metódy stretli a dosahuje na mnohých úlohách veľmi dobré výsledky.

Primárnym cieľom tejto práce je pokúsiť sa aplikovať evolúciu metódou NEAT na inej reprezentácii ako NS. Tou reprezentáciou sú stavové automaty, ktoré sú z robotického a tiež z hľadiska ľudského pochopenia oveľa vhodnejšie ako neurónové siete, ktoré sa nám javia ako uzavretá čierna skrinka.

Popri primárnom cieli sa oboznámime s teóriou evolučných algoritmov, neurnových sietí, stavových automatov a metódou NEAT.

Ďalej je naším cieľom túto novú reprezentáciu porovnať na vhodnej úlohe s pôvodnou reprezentáciou neurónovými sieťami. Dosiahnuté výsledky chceme štatisticky spracovať a využiť túto metódu aj na inú podobnú úlohu s cieľom porovnania s pôvodným algoritmom.

1 Evolučné algoritmy

Ľudstvo sa už od nepamäti zamýšľalo nad začiatkom sveta a svojou históriou. Za stvoriteľov sveta, jeho usporiadanie a vývoj, sa pokladali bohovia − rôzne nadprirodzené bytosti, ktoré sa postupne menili od civilizácie ku civilizácii. Až začiatok vedeckej revolúcie v 17. storočí a veľký prílev nových vedeckých informácií, ktoré boli v rozpore s existujúcimi dogmatickými predstavami náboženstva, začali tieto názory pomaly rúcať.

Predstavy o nemennom svete vytlačili rôzne vedecké teórie. Pozorovania postupného prispôsobovania sa jednotlivých generácií prostrediu, v ktorom žijú, viedlo vedcov ako J.B. Lamarck a Ch. Darwin k úvahám o evolúcii. Podľa Darwina odlišné druhy môžu byť v skutočnosti rôznymi vývojovými vetvami pochádzajúcimi zo spoločného predchodcu, ktoré sa však počas fylogenézy prispôsobili rôznym podmienkam prostredia.

Tento neustály proces prispôsobovania sa druhov podmienkam prostredia je pritom samovoľný, pôsobí chaoticky a vedie k tomu, že jednoduchšie druhy sa postupne vyvíjajú na zložitejšie a komplexnejšie. Je to prirodzeným následkom faktu, že druhy, ktoré sa nevedia prispôsobiť, vyhynú. Nie je pritom dôležité, či sa tak deje prostredníctvom vzájomného konkurovania si alebo kooperácie, ale to, že ďalším generáciám odovzdajú svoju genetickú informáciu lepšie prispôsobené a silnejšie jedince.

Týmto procesom, nazývaným evolúcia, sú inšpirované evolučné algoritmy(EA). Tento pojem zastrešuje stochastické optimalizačné algoritmy, ktoré zdieľajú základný koncept simulovania evolúcie individuálnych štruktúr procesmi selekcie, mutácie a kríženia. Ich cieľom je nájsť pre zvolenú reprezentáciu v priestore prípustných riešení riešenie maximálnej kvality. Množina prípustných riešení je pritom zvyčajne príliš veľká na to, aby bolo možné efektívne nájsť kvalitné riešenie deterministickým spôsobom. Avšak evolučné algoritmy sú práve jedným z možných spôsobov riešenia týchto problémov.

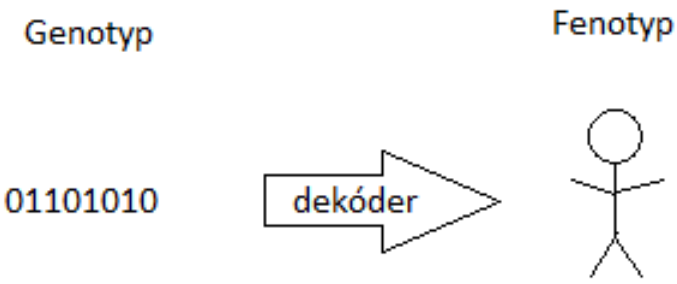
1.1 Vlastnosti evolučných algoritmov

1.1.1 Reprezentácia

Cieľ EA (nájsť v reprezentácii prípustných riešení riešenie maximálnej kvality) ovplyvňuje aj spôsob reprezentácie potenciálnych riešení, jedincov. Tieto rôzne riešenia sú zastúpené ako body v priestore riešení. Riešenie by malo byť možné rozložiť na zložky (atribút − hodnota)[29]:

$$R = (A_1=h_1 \ A_2=h_2 \ ... \ A_n=h_n)$$

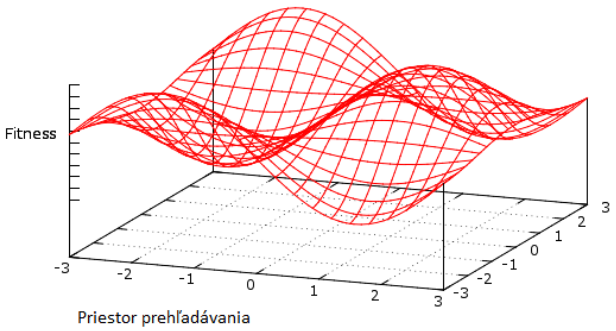
Tieto zložky sú usporiadané do postupnosti nazývanej reťazec alebo chromozóm. Každý z týchto atribútov potom predstavuje jeden rozmer v priestore riešení. Počet atribútov teda určuje dimenziu priestoru riešení. Takéto jedno riešenie označujeme genotyp a objekt predstavujúci toto riešenie v originálnom prostredí nazývame fenotyp. Pritom musí existovať spôsob ako zobraziť genotyp na fenotyp, pretože aj keď EA robia väčšinu operácii na genotype, kvalita tohto riešenia sa určuje na fenotype.



Obr. 1 Vzťah medzi genotypom a fenotypom.

1.1.2 Účelová funkcia (Objective function, Fitness)

Rôzni jedinci môžu mať rôznu kvalitu, preto sa zavádza numerická miera, označovaná ako vhodnosť – fitness, ktorá umožňuje jedincov porovnať a určiť, ktorý z nich je lepším riešením daného problému. Fitness je heuristická metóda, ktorá karteziánsky súčin $A_1 \times A_2 \times ... \times A_n$ mapuje na číslo z množiny všetkých reálnych čísel R. Jednotlivé body v tomto (n+1)-rozmernom priestore vytvárajú hyperplochu označovanú ako plocha vhodnosti, povrch fitness i fitness landscape. V prípade maximalizačnej optimalizácie platí, že čím vyššia je hodnota fitness, tým je jedinec lepší.



Obr. 2 Príklad fitness landscape.

1.1.3 Populácia

EA, ako populačné algoritmy, pracujú s populáciou jedincov, teda jedinci sú zoskupení do populácie pričom kardinalita takejto populácie sa pohybuje v desiatkach až stovkách jedincov. V priebehu času sa populácia mení a adaptuje. Čas je meraný v diskrétnych jednotkách – generáciách. Prvá generácia je generovaná stochasticky, jedinci sú zaradení do jedného prechodu evolučným cyklom, na ktorého konci sa generuje ďalšia generácia jedincov aplikáciou pravidiel daných EA.

1.1.4 Selekcia

Selekcia je proces výberu rodičov z aktuálnej populácie jedincov, ktorí vstúpia do reprodukcie. Počet rodičov záleží najmä od počtu potomkov, ktorých je potrebné vygenerovať a tiež od použitých genetických operátorov. Výber rodičov sa realizuje na základe ich fitness, väčšinou jedince s vyššou fitness a teda vhodnejšie, majú väčšiu pravdepodobnosť stať sa rodičmi než jedince s nižšou fitness. Nie je však nutnou podmienkou EA, aby boli niektoré jedince pri selekcii nejakým spôsobom zvýhodnené. V prípade, že k takejto preferencii niektorých jedincov voči ostatným dôjde, stáva sa selekcia zdrojom selekčného tlaku.

Selekčné metódy môžeme klasifikovať do viacerých skupín. Jedným z možných kritérií je, či daná metóda garantuje pre nejakého jedinca výber do množiny rodičov. Ak má každý jedinec šancu stať sa rodičom, hovoríme o uchovávajúcej selekcii(Mach, 2009). Ak daná metóda deklaruje, že niektorý jedinec sa v množine rodičov nachádzať nebude, hovoríme o vymierajúcej selekcii. Ak naopak daná metóda deklaruje, že najvhodnejší jedinec príp. viac najvhodnejších jedincov, sa stane rodičom, hovoríme o elitistickej selekcii – elitizmus.

1.1.5 Genetické operátory

Genetické operátory slúžia na vytvorenie nových jedincov kombináciou alebo čiastočnou zmenou rodičov. Ich úlohou je zabezpečiť vyhľadávanie nových potenciálnych riešení, teda zabezpečiť prehľadávanie priestoru riešení. Najčastejšie sa rozdeľujú na základe počtu rodičov, z ktorých sa generujú noví jedinci.

- Mutácia – unárny operátor

Pri mutácii dochádza k zmene aspoň jedného génu práve jedného rodiča. Táto zmena je vždy stochastická. Mutácia teda prináša do populácie nové gény, a je preto hlavnou hybnou silou evolúcie.

- Kríženie – binárny operátor

Kríženie (rekombinácia) náhodne kombinuje gény dvoch rodičov do jedného príp. viacerých potomkov. Pri bežnom spôsobe kríženia sa dva rodičovské reťazce rozdelia na jednom alebo viacerých miestach, rovnakých pre oboch rodičov, a potomkovia získajú striedavo každú druhú časť takto oddelených podreťazcov od každého z rodičov. Tento operátor, na rozdiel od mutácie, teda len kombinuje vlastnosti rodičov a neprináša nové gény do populácie, len ich rôzne kombinácie.

- Panmiktické operátory

Pri tomto operátore získava jedinec gény od viacerých rodičov. Používa sa zriedkavo. Tieto genetické operátory sa zvyknú spolu kombinovať. Najprv prebieha kríženie rodičov a následne na získaných jedincov je aplikovaný operátor mutácie.

1.1.6 Nahradenie - Replacement

Na konci evolučného cyklu po aplikácii genetických operátorov sa vytvára nová populácia, ktorá vstupuje do ďalšej generácie. Jedinci sú do tejto generácie vyberaní z predchádzajúcej populácie a z potomkov týchto jedincov. Ich vzájomný pomer v novej generácii je často hlavným atribútom, podľa ktorého sa rozlišujú druhy nahradenia. Častým prípadom je úplne nahradenie starej populácie vygenerovanými potomkami alebo súťaženie potomkov so svojimi rodičmi o pozíciu v ďalšej generácii, prípadne aj stochastický výber z oboch množín.

1.2 Druhy evolučných algoritmov

V druhej polovici dvadsiateho storočia boli podľa vzoru prirodzenej evolúcie na rôznych pracoviskách vo svete na riešenie odlišných problémov vytvorené viaceré, ale v istých črtách podobné prístupy. Všetky takéto smery sa dnes zvyknú zastrešovať pojmom evolučné algoritmy.

1.2.1 Evolučné programovanie

Táto metóda sa od iných EA odlišuje tým, že evolúciu simuluje s dôrazom na podobnosť fenotypov a správania sa fenotypov rodiča a potomka. Preto využíva iba operátor mutácie, ktorý aj keď je strojcom rozmanitosti populácie, zachováva medzi rodičmi a ich potomkami vysokú mieru korelácie ich správania. Vo svojej práci evolvovania jedincov na predikciu postupností ich predstavili v roku 1966 v USA L.J. Fogel, A.J. Owens a M.J. Walsh. Jedinci boli spočiatku reprezentovaní stavovými

automatmi. Počas ich vývoja sa však od tejto reprezentácie upustilo a nahradili ich vektory reálnych čísel.

1.2.2 Genetický algoritmus

Jedným z najvýznamnejších predstaviteľov evolučných algoritmov s veľmi širokým uplatnením sú genetické algoritmy. Za ich vznik sa považujú práce skupiny pod vedením Johna Hollanda v USA v sedemdesiatych rokoch dvadsiateho storočia. Jedinci sú reprezentovaní binárnymi reťazcami danej dĺžky n>0. Populácia pozostáva z N náhodne vygenerovaných binárnych reťazcov.

Na výber populácie rodičov sa používa proporcionálny výber na základe fitness. Pravdepodobnosť výberu každého jedinca je priamo úmerná hodnote fitness tohto jedinca. Nová populácia sa vytvára pomocou operátorov kríženia a mutácie. V prvom kroku prebehne kríženie a na vzniknutého jedinca je následne použitý operátor mutácie.

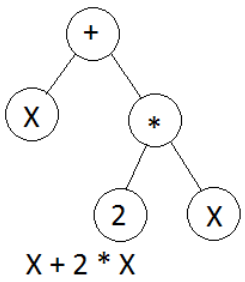
1.2.3 Genetické programovanie

Genetické programovanie je prístup určený najmä na automatizovaný vývoj a optimalizáciu programov, funkcionálnu regresiu alebo strojové učenie. Za ich vznikom koncom osemdesiatych rokov stojí J. Koza.

Základný rozdiel medzi jednoduchým genetickým algoritmom a genetickým programovaním spočíva v reprezentácii jedincov a v následných zmenách operátorov kríženia a mutácie pracujúcich nad jedincami.

Jedinci sú zvyčajne reprezentovaní koreňovými stromami s ohodnotenými vrcholmi. Vnútorné vrcholy obsahujú funkcie z množiny prípustných funkcií. Listy obsahujú terminálne symboly, ktoré zväčša reprezentujú premenné, konštanty alebo funkcie bez parametrov(Obr. 3). Preto je potrebné najprv definovať množinu prípustných funkcií a terminálnych symbolov.

Pri krížení sa náhodne vybratý podstrom vymení s iným náhodne vybraným podstromom z iného jedinca.



Obr. 3 Príklad reprezentácie v genetickom programovaní.

1.2.4 Evolučné stratégie

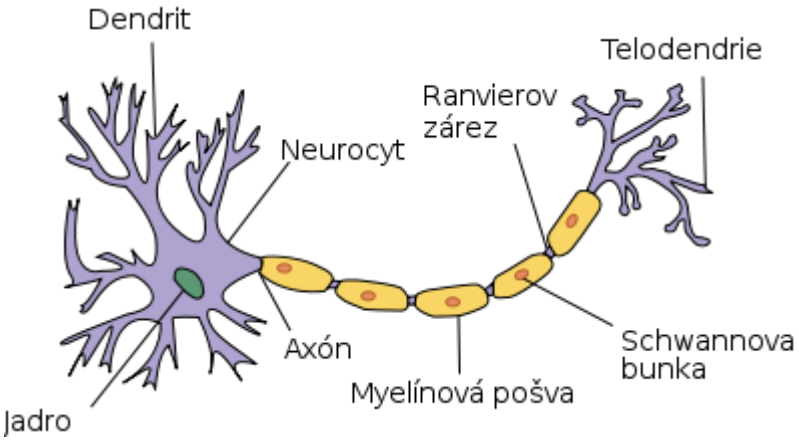
Evolučné stratégie boli vyvíjané od polovice šesťdesiatych rokov pri optimalizácii konštrukčných úloh I. Rechenbergom a H.P. Schwefelom. Najvýraznejšími črtami evolučných stratégií je reprezentácia jedincov reálnymi vektormi a skutočnosť, že pri variáciách sa dôraz kladie najmä na mutáciu. Napriek tomu sa hranice medzi genetickými algoritmami a evolučnými stratégiami neustále stenčujú a ich delenie je skôr výsledkom historického vývoja.

ES rozdeľujeme na plus-stratégie, $(\mu+\lambda)$-ES a čiarka-stratégie, (μ,λ)-ES. Parameter μ reprezentuje počet rodičov, a λ počet potomkov. Symbol „+“ znamená výber najlepších riešení z populácie rodičov a potomkov do ďalšej generácie. Symbol „ ,“ znamená výber μ rodičov len z populácie potomkov za predpokladu $\mu < \lambda$.

2 Neurónové siete

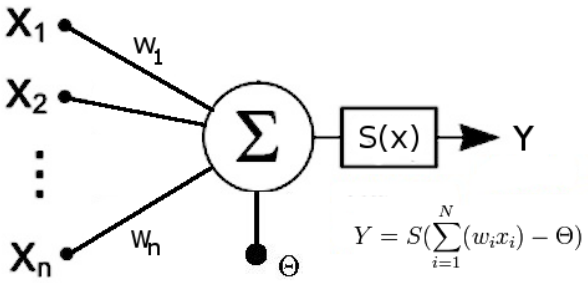
Umelá neurónová sieť(NS) je paralelný výpočtový model inšpirovaný biologickými nervovými systémami. Založený je na základnom postuláte neurovedy, podľa ktorého základnou časťou ľudského mozgu je neurón, ktorý prijíma signály z okolia pomocou dendritov, spracúva ich a posiela tieto spracované vstupné signály iným neurónom pomocou axónu.

V mozgu je približne 10^{12} neurónov, ktoré sú navzájom poprepájané tzv. synapsami, ktorých počet sa pohybuje okolo 10^{14}.



Obr. 4 Stavba biologického neurónu (prevzaté z [31]).

Tento biologický model neurónu slúžil ako predloha umelému neurónu. Z viacerých modelov je najviac používaný model McCullocha a Pittsa, logický neurón:



Obr. 5 McCullochov a Pittsov model logického neurónu s aktivačnou funkciou.

X_i sú vstupy neurónu, w_i sú synaptické váhy, Θ je prah, S(x) prechodová funkcia neurónu a Y je výstup neurónu. Neurón sa nachádza v pasívnom stave, kým suma vstupov násobených k nim prislúchajúcimi váhami neprekročí prah. Preto čím je vyššia hodnota váhy, tým je daný vstup dôležitejší.

2.1 Učenie

NS dokážu abstrahovať zákonitosti a pravidlá medzi vstupnými a výstupnými hodnotami a následne ich aplikovať na akékoľvek (aj iné) vstupné hodnoty. Tento proces abstrakcie pravidiel medzi vstupom a výstupom sa nazýva učenie. Pri učení sa aktualizujú hodnoty váhových spojení. Rozlišujeme učenie s učiteľom a učenie bez učiteľa.

2.1.1 Učenie s učiteľom (Supervised learning)

NS dostane trénovaciu množinu dát, ktorá je zložená zo vstupov a požadovaného výstupu na tieto vstupy. Pri učení sa potom váhy spojení postupne adaptujú tak, aby minimalizovali rozdiel medzi požadovaným výstupom a výstupom, ktorý generujú. Toto učenie sa opakuje až kým nedosiahneme nami stanovenú minimálnu chybu.

Problémom pri tomto učení môže byť preučenie (en. overfitting), ktorý väčšinou nastáva, ak je trénovacia množina príliš malá. Sieť je vtedy príliš prispôsobená trénovacím dátam, na ktorých dosahuje vynikajúce výsledky, ale schopnosť zovšeobecnenia je už príliš malá, a teda na dátach iných ako trénovacích nedáva dobré výsledky. Riešiť sa to dá zväčšením trénovacej množiny alebo tiež rozdelením trénovacej množiny na estimačné a validačné dáta. Sieť sa potom trénuje na estimačných dátach a jej kvalita sa po každej epoche kontroluje na validačných, resp. testovacích dátach. Ak chyba na validačných dráhach začne narastať, učenie siete sa zastaví.

Rozlišujeme tri druhy kontrolovaného učenia: učenie na základe opravy chyby(en. error correction learning), stochastické učenie a učenie na základe hodnotenia činnosti(en. reinforcement learning).

2.1.2 Učenie bez učiteľa (Unsupervised learning)

NS dostane iba množinu vstupov. Tieto vzory si sieť samovoľne utriedi do skupín alebo vstupu prispôsobí svoju topológiu. Keďže sa sieť sama rozhoduje o svojich vlastnostiach, zvykneme nazývať tieto siete samoorganizujúce sa siete(en. self-organizing NN). Učenie siete sa zastavuje keď je už zmena váh dostatočne malá.

Rozlišujeme dva základné typy učenia bez učiteľa – Hebbove učenie a Kooperačné a konkurenčné učenie. Pri Hebbovom učení sa synaptická váha zvýši, ak sú neuróny na jej opačných stranách aktivované súčasne, inak takáto synaptická váha konverguje k nule. Pri konkurenčnom učení (en. winner takes it all) neurón s najvyššou hodnotou sa nastaví na hodnotu 1, ostatné neuróny na hodnotu nulovú.

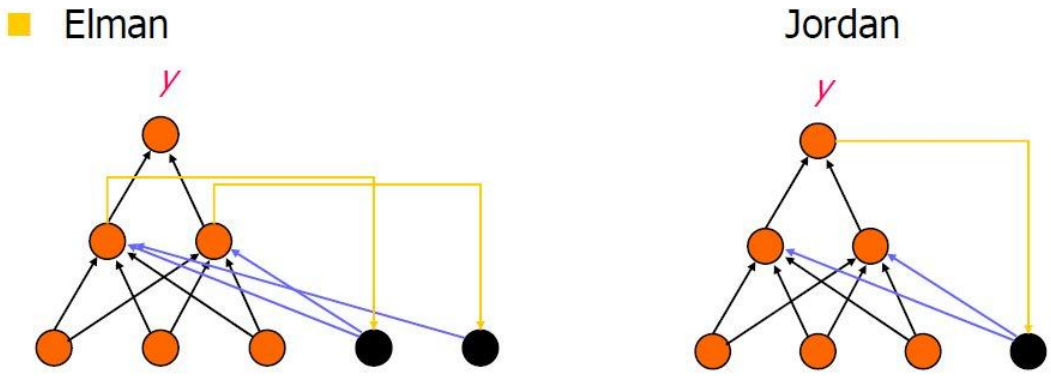
2.2 Rozdelenie NS podľa štruktúry/topológie

2.2.1 Dopredné neurónové siete (Feed-forward)

Jednotlivé neuróny v sieti sú zaradené do vrstiev, rozlišujeme vstupnú, výstupnú a skryté vrstvy. V tejto architektúre sa signál šíri synaptickými spojeniami len jedným smerom. Vstupná vrstva príjme vstup a spracovaný výstup pošle do skrytej vrstvy. Neuróny v skrytých vrstvách prijímajú vstup od neurónov zo vstupnej vrstvy a spracované výstupy posielajú ďalej do siete. Výstupná vrstva prijatý vstup spracovaný vráti do externého sveta. Bolo dokázané, že dopredné NS sú univerzálnym aproximátorom, čo znamená, že sú schopné aproximovať s požadovanou presnosťou ľubovoľnú spojitú funkciu [8].

2.2.2 Rekurentné neurónové siete (Recurrent NN)

Pri rekurentných sieťach (RNS) je zložité deliť neuróny na vstupné a výstupné, pretože jednotlivé neuróny môžu predstavovať oba typy. Signál sa nešíri len dopredu, ale existuje tu aspoň jedna spätná väzba. Vďaka tomu má takáto sieť vnútorný stav, čiže vlastnú pamäť. Najznámejšími architektúrami RNS sú Elmanova a Jordanova architektúra. V Elmanovej architektúre tzv. pamäťové neuróny uchovávajú stav skrytých neurónov z predchádzajúcej epochy. V Jordanovej architektúre uchovávajú stav výstupných neurónov a svojho stavu z predchádzajúcej epochy[16].



Obr. 6 Elmanova a Jordanova architektúra RNS (prevzaté z [16]).

Podľa Siegelmanna a Sontaga (1992), môže byť každý turingov stroj simulovaný plne prepojenou rekurentnou sieťou, zostrojenou z neurónov so sigmoidnou aktivačnou funkciou. To znamená, že rekurentné siete sú výpočtovo ekvivalentné v polynomiálnom čase turingovmu stroju.

2.3 Perceptrón

Perceptrón sa považuje za najjednoduchšiu NS. Navrhol ho Frank Rosenblatt v roku 1958. Do modelu logického neurónu McCullocha a Pittsa, ktorý mal pevne nastavené váhové aj prahové koeficienty, podľa boolovej funkcie, ktorú mal vykonávať, vniesol učenie, a teda váhové a prahové koeficienty sa stali premennými.

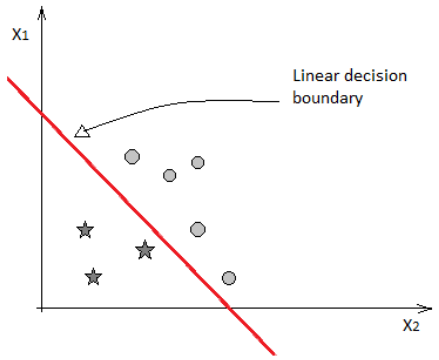
Perceptrón má tri vrstvy. Vstupná vrstva (en. projection area) prijíma hodnoty vstupov a nezmenené posiela do asociatívnej vrstvy. Asociatívna vrstva (en. association area) tieto vstupy spracúva v rovnici (aktivačná funkcia):

$$y = f(\Sigma^{n}_{j=1} w_j x_j - \theta)$$

Tretia vrstva, výstup (en. responses), je potom prahovou hodnotou aktivačnej funkcie podľa vzťahu:

$$ou(t) = \begin{cases} 1 & ak\ in(t) \geq 0 \\ 0 & ak\ in(t) < 0 \end{cases}$$

Perceptrón sa používa na dichotomickú klasifikáciu, teda rozdelenie vstupov do dvoch tried, ak sú lineárne separovateľné. To vyplýva z Fixed-increment convergence teorémy, ktorú sformuloval Rosenblatt: „Nech triedy A a B sú konečné a lineárne separovateľné, potom učiaci algoritmus perceptrónu konverguje (upravuje svoj vektor váh) v konečnom počte krokov“.



Obr. 7 Príklad lineárne separovateľných tried v 2D priestore.

2.4 Spôsoby trénovania sieti

2.4.1 Backpropagation (BP)

Je to základná a najpoužívanejšia metóda trénovania NS. Bez modifikácií funguje len pre dopredné NS. Je to rekurzívna gradientová metóda na nastavovanie váh NS s ohľadom na minimalizáciu učiacej chyby (Bundzel, 1999). Sieť najprv vypočíta svoj

výstup, potom rekurzívne vypočíta delty pre všetky výstupné a skryté neutróny podľa vzorca:

$$\delta_i = (d_i - y_i)\, f_i'$$

$$\delta_k = (\Sigma_i\, w_{ik}\, \delta_i)\, f_k'$$

kde d je požadovaný výstup, y je výstup siete, f' je derivácia aktivačnej funkcie, w je váha. Váhy siete sa potom upravujú vzťahom:

$$v_{kj}(t+1) = v_{kj}(t) + \alpha\delta_k x_j$$

2.4.2 Backpropagation throught time (BPTT)

Je rozšírením BP metódy aplikovaná na trénovanie rekurentných sieti. Zakladá sa na myšlienke rozkladu rekurentnej NS na potenciálne viacvrstvovú doprednú NS, na ktorej sa aplikuje klasický backpropagation algoritmus. Podmienkou je, aby neboli žiadne spojenia medzi vstupnými neurónmi. Medzi nevýhody tohto algoritmu patria veľké časové nároky a dlhšie trénovacie postupnosti.

2.4.3 Rekurentné učenie v reálnom čase RTRL

Gradient sa počíta on-line, v reálnom čase, a preto nie je potrebné určovať hĺbku časového okna ako pri BPTT. Je však časovo a pamäťovo veľmi náročná, so zložitosťou až N^4, pre N neurónov.

2.5 Neuroevolúcia

Neuroevolúcia je metóda menenia váh alebo topológie NS za účelom naučenia siete riešiť špecifickú úlohu. Evolučné algoritmy sú tu používané na trénovanie siete – vyhľadávanie parametrov siete, ktoré maximalizujú úspešnosť siete riešiť určitú úlohu. Je to veľmi užitočná metóda pri úlohách, kde je ľahké merať výkon siete, ale ťažké správne určiť množinu správnych trénovacich dát pri učení s učiteľom. Do tejto situácie sa môžeme dostať, ak nie sú hneď známe optimálne akcie v niektorých bodoch a ich kvalita sa ukáže až neskôr. Neuroevolúcia sa preto často využíva pri hrách, kontrole riadenia vozidiel a robotike.

Niektoré štúdie ukázali, že neuroevolúcia môže byť pri riešení problémov rýchlejšia a efektívnejšia než učenie odmenou a trestom (en. reinforcement learning), napríklad pri problémoch riadenia pohybu robotického ramena alebo vyrovnávania rovnováhy prevráteného kyvadla.

Štandardné neuroevolčné algoritmy upravujú evolučným prehľadávaním len váhy siete. Tento proces je jednoduchý, keďže nie je ťažké zapísať váhy siete do reťazca, ktorý je potom použitý tradične ako reprezentant jedinca a vstup do evolúčneho algoritmu. Topológia siete je v takýchto prípadoch dopredu daná a vopred nie je isté, aká topológia je pre daný problém vhodná, čo znamená, že užívateľ si volí topológiu siete sám. To často vedie k návrhom siete štýlom pokus – omyl alebo časovo náročnej analýze, aká by topológia siete mala byť.

2.5.1 Enforced Subpopulations

Táto metóda pri inicializácii vytvorí náhodnú topológiu, a potom ju už v priebehu výpočtu nemení. Kedykoľvek sa zasekne alebo neuspeje, jednoducho sa reštartuje s iným náhodným počtom skrytých neurónov. Ukázala sa byť na probléme vyrovnania rovnováhy prevráteného kyvadla 5-krát rýchlejšou než tradičné NE metódy, ktoré evolvujú len váhy.

2.5.2 TWEANN

Tento pojem (Topology and Weight Evolving Artificial Neural Networks) zahŕňa mnoho metód na evolvovanie topológie a váh sieti. V ďalšej kapitole tieto problémy popíšeme, a tiež uvedieme ako ich rieši metóda NEAT.

3 NEAT

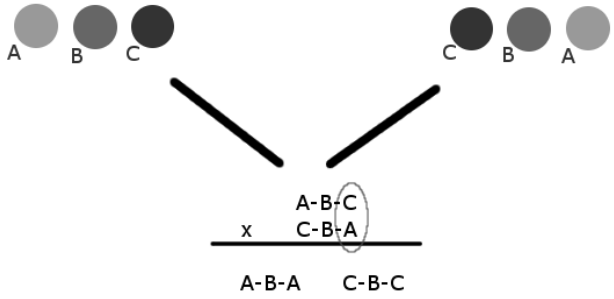
NEAT (NeuroEvolution of Augmenting Topologies) je metóda evolvovania neurónových sietí genetickým algoritmom. Táto metóda bola prvýkrát predstavená v publikácii v roku 2002 jej autormi Kenneth O. Stanley a Risto Miikkulainen (Stanley, Miikkulainen, 2002). Pokusmi sa podarilo ukázať, že NEAT je na probléme pole-balancing 5-krát rýchlejšia ako metóda Enforced Subpopulations, ktorá pri inicializácii vytvorí náhodnú topológiu, a potom ju už v priebehu výpočtu nemení.

Charakteristickými vlastnosťami metódy NEAT je inicializácia NS s minimálnou štruktúrou, aplikácia kríženia, ktoré nenarúša topológiu siete a rozdelenie jedincov na druhy. V ďalších statiach uvedieme základné problémy, s ktorými sa stretávajú tradičné metódy evolvovania topológie NS, a tiež ako ich metóda NEAT pomocou vymenovaných charakteristických vlastností rieši.

3.1 Competing Convention problem

Hlavný problém neuroevolúcie je známy ako Competing Conventions Problem (Montana, Davis, 1989) alebo tiež ako Permutations problem (Radcliffe, 1993). Tento problém nastáva, ak máme viacero spôsobov reprezentácie rovnakého riešenia. V tom prípade kríženie s veľkou pravdepodobnosťou vyprodukuje nekvalitného potomka.

Pre ukážku majme 2 jedincov zakódovaných reťazcami ABC a CBA, ktorí sú rovnako kvalitní. Ich krížením môžu vzniknúť jedinci ABA a CBC, na ktorých je hneď vidieť, že obaja stratili nejakú informáciu (obr. 8).



Obr. 8 Ukážka competing convention problem.

Príroda čelí rovnakému problému. Gény pridávané náhodne na náhodné pozície by zrejme neviedli k úspešnej evolúcii. Preto využíva homológiu, teda dva gény sú

homologické, ak slúžia na rovnaký zámer. Úzko to súvisí aj historickým pôvodom daných génov: ak vznikli z rovnakého predchodcu, sú tieto gény v podstate homologické.

NEAT tento problém rieši tak, že o každom géne eviduje inovačné číslo, ktoré popisuje genetický pôvod. Pri krížení potom môže identifikovať gény s rovnakým genetickým pôvodom. Predpokladá, že gény s rovnakým genetickým pôvodom majú rovnakú alebo podobnú funkciu pri výpočte v sieti.

3.2 Ochrana inovácií

Inovácie pridávajú do sietí nové štruktúry pomocou mutácie. Takéto pridanie inovácie však často spôsobuje zníženie fitness daného jedinca a pozitívny efekt môže nadobudnúť až o niekoľko generácií neskôr, keď sa váhy novej štruktúry dostatočne optimalizujú. Keďže však môže táto optimalizácia trvať dlhší čas, je dosť nepravdepodobné, že táto inovácia dovtedy pretrvá.

NEAT takéto inovácie ochraňuje pomocou zdieľania fitness. Jednotlivých jedincov rozdeľuje do druhov – skupín navzájom podobných jedincov. Jedinci súperia medzi sebou len v rámci svojho druhu a v globálnom meradle súperia medzi sebou jednotlivé druhy.

Rozdelenie do druhov si vyžaduje, aby NEAT mala funkciu na meranie kompatibility jedincov, ktorá roztriedi jedincov do jednotlivých druhov. Je zložité formulovať takúto funkciu pre siete s rozličnou topológiou. Competing convetion robí túto funkciu zvlášť problematickú, lebo funkčne podobné siete môžu byť topologicky značne odlišné. Keďže však NEAT competing convention problém rieši, môže byť populácia jednoducho roztriedená. Jedinci v rámci druhu potom explicitne zdieľajú fitness (en. explicit fitness sharing). Tá sa počíta ako aritmetický priemer ich skutočných fitness. Podľa tejto fitness sa potom určuje pomer jedincov daného druhu v celej populácii. To chráni druhy pred obzvlášť dobrými jedincami, ktorí by potenciálne mohli obsadiť svojimi kópiami celú populáciu.

Kompatibilita jedincov δ sa počíta ako lineárna kombinácia počtu nadbytočných génov E, ktoré majú vyššie inovačné číslo než maximálne inovačné číslo druhého jedinca, počtu rôznych génov D, ktoré sú odlišné v rozmedzí inovačných čísel druhého jedinca a priemerného rozdielu váh spoločných génov W'. Pomocou parametrov c_1, c_2, c_3 môžeme nastaviť predchádzajúcim trom parametrom dôležitosť vo výpočte. Počet neurónov väčšieho genotypu N slúži na normalizáciu pre veľkosť genotypu[17]:

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

3.3 Prvotná populácia a zmena topológie

Väčšina TWEANN metód inicializuje siete náhodnými topológiami. To zabezpečuje diverzitu topológií, ale tiež spôsobuje možné problémy, ako napríklad prípad, že sieť nebude mať cestu z každého vstupu do výstupov. Na odstránenie takýchto sietí z populácie je potrebný istý čas. Náhodné siete tiež môžu zapríčiniť, že sa stanú zbytočne veľkými a ich minimalizácia je potom relatívne komplikovaná.

Niektoré metódy riešia tieto problémy penalizáciou fitness sieti s väčším počtom neurónov. To však tiež nemusí byť najlepší postup, pretože rôzne problémy si môžu vyžadovať rôzne topológie, teda aj väčšie. Tiež je zložité určiť, aká by táto penalizácia mala byť veľká.

Tento problém rieši metóda NEAT tak, že začína s minimálnymi sieťami. To zaručí, že sieť začne hľadať možné riešenie na najnižšom možnom počte neurónov a rozšírenie siete na väčšiu nastáva iba v tom prípade, že dané rozšírenie je prínosom pre sieť. To zaručuje tiež vyššiu výkonnosť, keďže prehľadávame menší priestor riešení.

4 Stavové automaty

4.1 Chomskeho hierarchia

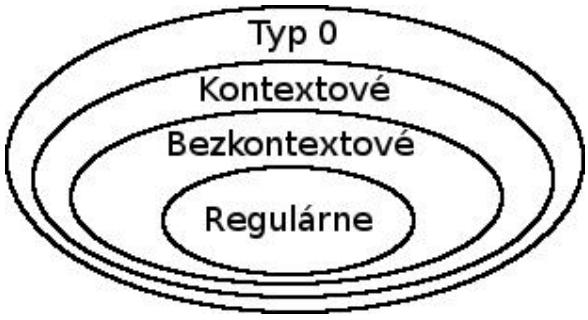
V roku 1956 N. Chomsky vytvoril matematický model gramatiky. Gramatika je štvorica (V_N, V_T, P, S), kde symbol V_N označuje konečnú množinu premenných symbolov, V_T je konečná množina terminálnych symbolov, P je konečná množina pravidiel a S je začiatočný symbol. Pritom predpokladáme, že prienik množín V_N a V_T je prázdna množina [9].

Chomsky zaviedol štyri typy gramatík. Typ 0 alebo aj frázová gramatika, zahrňuje všetky formálne gramatiky generujúce jazyky rozpoznateľné turingovým strojom. Jazyk generovaný touto gramatikou sa nazýva rekurzívny (rekurzívne vypočítateľný). Gramatika typu 0 je ekvivalentná turingovmu stroju.

O type 1, známom aj ako kontextová gramatika, hovoríme, ak pre každé pravidlo $\alpha \rightarrow \beta$ z P platí $|\beta| \geq |\alpha|$. Táto gramatika je ekvivalentná lineárne ohraničenému automatu a generuje kontextové jazyky.

O type 2, známom aj ako bezkontextová gramatika, hovoríme, ak pre každé pravidlo $\alpha \rightarrow \beta$ z P platí, že α je jediný neterminálny symbol a β je nejaký reťazec, rôzny od ε. Táto gramatika je ekvivalentná zásobníkovému automatu a generuje bezkontextové jazyky.

O type 3, známom aj ako regulárna gramatika, hovoríme, ak každé pravidlo z P má tvar $A \rightarrow \alpha B$ alebo $A \rightarrow \alpha$, pričom A, B sú neterminálne symboly a α je terminálny symbol. Táto gramatika je ekvivalentná konečnému automatu. Platí pritom, že regulárne jazyky sú nevlastnou podmnožinou bezkontextových jazykov, ktoré sú nevlastnou podmnožinou kontextových jazykov, ktoré sú napokon nevlastnou podmnožinou rekurzívne vypočítateľných jazykov.



Obr. 9 Chomskeho hierarchia tried jazykov.

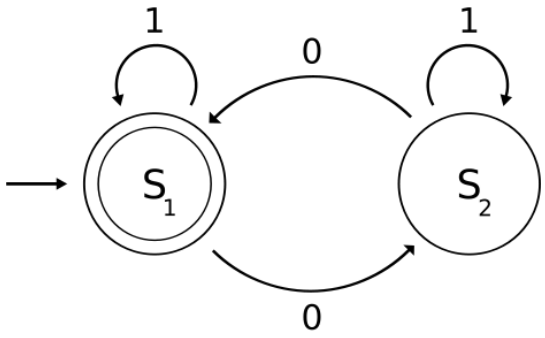
4.2 Konečný stavový automat

Definícia konečného stavového automatu (eng. finite state machine, príp. finite state automaton, FSA) ako formálneho systému podľa Hopcrofta a Ullmana z roku 1978: „Konečný automat M nad abecedou Σ je systém (K, Σ, δ, q_0, F), kde K je konečná neprázdna množina stavov, Σ je konečná vstupná abeceda, δ je zobrazenie K × Σ do K, q_0 z K je začiatočný stav a F ⊆ K je množina koncových stavov“.

Konečný stavový automat akceptuje slovo w = $w_0w_1...w_n$, zložené zo symbolov w_i ∈ Σ ak δ(...δ(δ(q_0, w_0), w_1),...w_n)=δ(q_0, w)=p pre nejaký stav p ∈ F. Množinu všetkých slov w, ktoré akceptuje automat M, označujeme T(M) a hovoríme, že M rozpoznáva T(M):

$$T(M) = \{ x \mid \delta(q_0, w) \in F \}$$

Každá množina slov, ktorú rozpoznáva nejaký FSA, sa nazýva regulárna.



Obr. 10 Konečný stavový automat (prevzaté z [32]).

4.3 Rozdelenie FSA

Je niekoľko rôznych kritérií na rozdelenie FSA. Najčastejšie sa FSA rozdeľujú na deterministické (DFA) a nedeterministické (NFA). Rozdiel je v prechodovej funkcii δ, ktorá v DFA každému vstupnému symbolu z Σ priradí práve jeden stav z K a v NFA priraďuje každému vstupnému symbolu z Σ, podmnožinu stavov z K, ktorá môže byť aj prázdna. Výpočtová sila oboch týchto typov FSA je však rovnaká.

4.3.1 FSA a transducery

Transducery sú FSA, ktoré generujú výstup resp. výstupnú postupnosť znakov. Definícia automatu sa potom rozširuje na šesticu (K, Σ, δ, Γ, q_0, F), kde Γ je konečná množina výstupných symbolov. Podľa toho, či je výstup funkciou len stavov alebo stavov a vstupov rozdeľujeme transducery na Mealyho a Moorove automaty.

Moorov automat priraďuje výstupnú hodnotu každému stavu. Výstup pritom nie je závislý od súčasného vstupu, a preto má takýto automat vždy oneskorenie.

Na druhej strane Mealyho automat priraďuje výstupnú hodnotu každej dvojici Σ×K, teda výstupná hodnota je priradená prechodu do ďalšieho stavu. Každý prechod Mealyho automatu má teda určenú nielen vstupnú hodnotu, ktorá ho aktivuje, ale tiež výstupnú hodnotu, ktorá sa stáva vstupom nasledujúceho stavu.

4.4 Stavové automaty a robotika

Stavové automaty sa veľmi dobre osvedčili ako nástroje ovládania správania sa robotov, čo vyplýva aj z ich vzájomnej podobnosti. Podobne ako FSA, ktorý zostáva v istom stave, kým nie sú splnené podmienky stavového prechodu, aj robot vykonáva istú činnosť, kým sa nezmenia vlastnosti prostredia, ktoré vníma senzormi.

4.4.1 Subsumpčná architektúra

Subsumpčná architektúra bola predstavená v roku 1986 R. Brooksom, určená je pre reaktívnych agentov. Je to vrstevnatá architektúra, pričom v jednej vrstve sa nachádza viacero správaní, ktoré sú reprezentované stavovými automatmi. Vrstvy majú vzostupnú prioritu, teda napríklad ak horná vrstva predstavuje správanie sa robota v podmienkach bez prekážok, po evidovaní prekážky, prechádza sa do spodnejšej vrstvy, v ktorej vybraný automat určuje, ako sa má agent prekážke vyhnúť. Spodnejšie vrstvy teda majú vyššiu prioritu ako vrchné, ktoré existujú ako zabezpečenie komplexnejšieho správania.

4.5 Rekurentné NS a FSA

Topológia NS sa dá popísať ako orientovaný graf, ktorého vrcholy predstavujú jednotlivé neuróny; hrany ohodnotené váhami, predstavujú jednotlivé prepojenia medzi neurónmi. Rovnako topológiu FSA môžeme reprezentovať orientovaným grafom, ktorého vrcholy predstavujú stavy automatu a hrany zodpovedajú stavovým prechodom medzi týmito stavmi. Namiesto váh teda obsahuje podmienky stavových prechodov. Odlišný je aj samotný výpočet – v neurónovej sieti sú aktívne všetky vrcholy, zatiaľ čo FSA má aktívny iba jeden vrchol.

Môžeme teda vidieť, že z hľadiska topológie sú FSA veľmi podobné s NS, najmä rekurentnými NS, ktoré majú aj spätné linky, a teda pohyb medzi neurónmi môže byť rôzny podobne ako aj pri FSA.

4.6 FSA ako reprezentácia pre EA

FSA môžeme reprezentovať orientovaným alebo neorientovaným grafom. Uzlom grafu zodpovedajú stavy automatu a hrany predstavujú stavové prechody medzi týmito stavmi. Stavové automaty boli používané ako reprezentácia genotypu v prvotnom vývoji evolučného programovania. Neskôr boli nahradené vektormi reálnych čísel.

Petrovič [17] porovnáva FSA a programy so stromovou reprezentáciou navrhnuté genetickým programovaním a uvádza, že v závislosti od typu úlohy je vhodnejšia vždy iná, pričom sa avšak dopĺňajú vo svojich vlastnostiach. FSA sú vhodnejšie v úlohách s cyklickými interakciami a tam, kde sa očakáva okamžitá odpoveď na podnet. Programové stromy je zase výhodnejšie používať v úlohách, kde je najvhodnejším riešením dlhá neopakujúca sa postupnosť.

4.7 Evolúcia FSA

Spočiatku sa na evolvovanie FSA využívalo evolučné programovanie. Chelapilla a Czarnecki navrhli variáciu evolučného programovania na riešenie problému syntézy modulárnych FSA, adaptovaný na známy problém mravca. Reprezentácia modulárnymi FSA sa na tomto probléme ukázala byť úspešnejšia než nemodulárnymi FSA (Chelapilla, Czarnecki, 1999). Clelland a Newlands používajú pravdepodobnostné FSA na hľadanie závislostí vo vstupných dátach s dôrazom na nájdenie čo najjednoduchšieho automatu. Rôzne heuristické metódy inkrementálneho pridávania substringov na vytvorenie pravdepodobnostných FSA nevedú ku globálnemu minimu, preto využívajú vo svojej práci evolučné programovanie[5].

Horihan a Lu využívajú genetický algoritmus na evolvovanie FSA akceptujúcich slová jazykov generovaných z jednoduchších gramatík.

V ďalšej časti práce sa budeme snažiť modifikovať NEAT metódu evolvovania NS na evolvovanie stavových automatov.

5 Model

Väčšina prác evolučnej robotiky sa zaoberá evolúciou NS kvôli ich elasticite a adaptívnosti na široké spektrum problémov, úloh. Ich analýza je však zložitá, pretože sa správajú ako čierna skrinka; pri pohľade do vnútra nevieme povedať, ktorá časť vykonáva akú činnosť, resp. prečo sú tam tie a tie časti. Na druhej strane analýza FSA je jednoduchšia, pretože aj ich správanie sa je pre ľudskú myseľ ľahšie pochopiteľné. Ako sme už uviedli v kapitole 4.4 FSA sú veľmi vhodné na použitie v niektorých robotických úlohách.

Napriek tomu evolúcia FSA nie je dostatočne preskúmaná a venuje sa jej málo štúdií. FSA sa spočiatku používali ako reprezentácie v evolučnom programovaní, neskôr však boli nahradené vektormi reálnych čísel. Mnohé štúdie experimentujú s použitím rôznych typov FSA ako reprezentácií EA, pričom dosahujú na niektorých typoch úloh lepšie výsledky.

Kvôli naznačenej podobnosti rekurentných NS a FSA predpokladáme, že metóda NEAT je použiteľná aj na evolúciu FSA, dokonca je pravdepodobné, že na niektorých problémoch budú FSA evolvované touto metódou dokonca efektívnejšie než NS. Táto práca predstavuje pokračovanie práce [17], v ktorej automaty boli reprezentované pomocou NS na základe prekladovej funkcie, ktorá každému FSA priradila NS a naopak.

Cieľom tejto práce je overiť či sú myšlienky metódy NEAT využiteľné aj pre reprezentáciu genotypu stavovými automatmi.

5.1 NEAT implementácia Minesweeper

Na realizáciu našich pokusov sme si vybrali NEAT verziu riešenia problému Minesweeper (Stanley, Miikkulainen, 2002). Minesweeper má za úlohu prejsť čo najväčšiu plochu prostredia. Priestor a čas sú ohraničené, pričom v priestore sa nachádzajú aj prekážky. Každý minesweeper má rovnaký počet senzorov (v našom prípade 5+5), ktorými dokáže zistiť, či do vzdialenosti senzora v jeho smere je nejaká prekážka, prípadne či tam je prejdená cesta. Zistená vzdialenosť je normovaná na intervale <0, 1>. Ak daný senzor žiadnu prekážku nedetegoval, jeho hodnota je – 1.

Zvolená implementácia siete je inicializovaná 11 vstupnými neurónmi (10 vstupov zo senzorov, 11. parameter určuje či bola detegovaná kolízia) a 2 výstupnými neurónmi, ktoré predstavujú rýchlosť otáčok kolies minesweepera.

V nasledujúcich statiach opíšeme niektoré z najdôležitejších tried a štruktúr použitých v tejto implementácii minesweeper.

5.1.1 Konfiguračné parametre

- *NumInputs, NumOutputs* – počet vstupných a výstupných neurónov siete
- *iNumTicks* – určuje počet krokov, ktoré vykonajú jedince počas jednej generácie
- *iPopSize* – určuje veľkosť populácie
- *dSurvivalRate* – percento populácie, ktoré má šancu ísť do selekcie
- *dChanceAddLink* – šanca na pridanie nového linku do siete
- *dChanceAddNeuron* – šance na pridanie nového neurónu do siete
- *dMutationRate* – pravdepodobnosť mutácie génu
- *dMaxWeightPerturbation* – maximálna absolútna hodnota, o ktorú sa môže váha meniť
- *dProbabilityWeightReplaced* – pravdepodobnosť úplnej náhrady génu v rámci mutácie
- *dActivationMutationRate* – pravdepodobnosť zmeny aktivačného prahu siete
- *dMaxActivationPerturbation* - maximálna absolútna hodnota, o ktorú sa môže zmeniť hodnota aktivačného prahu
- *dCompatibilityThreshold* – hodnota funkcie kompatibility, po prekročení ktorej sa dva jedince považujú za dva rôzne druhy
- *iOldAgeThreshold* – počet generácií, po ktorých je druh označený ako starý a jeho fitness podlieha penalizácii
- *dOldAgePenalty* – koeficient penalizácie fitness starých druhov
- *dYoungFitnessBonus* – počet generácií, do kedy je druh označený ako nový, jeho fitness je zvýšené
- *iYoungBonusAgeThreshhold* – koeficient zvýšenia fitness nových druhov
- *dCrossoverRate* – pravdepodobnosť kríženia dvoch jedincov

5.1.2 Trieda CController

- Vytvára prvotnú populáciu
- Funkcia *Update*() je základnou funkciou programu, kontroluje celú simuláciu. Kým neprebehne určitý počet iterácii stanovený parametrom *iNumTicks*, vkladá minesweeperom informácie zo senzorov, tie na ne reagujú a pohybujú sa v priestore. Po dosiahnutí daného počtu iterácii, nastaví tento počet na nulu, a jedincom priraďuje nové fitness v závislosti od počtu políčok, ktoré navštívili,

vymazáva fenotypy a posúva genotypy do reprodukcie. Z novej množiny genotypov potom vytvára novú populáciu, ktorá vojde do novej generácie.

- Niekoľko ďalších funkcií zabezpečuje správne vykreslenie minesweeperov, senzorov a tiež prekážok

5.1.3 Cga

- Konštruktor vytvára populáciu genómov zo vstupných parametrov
- Funkcia *CreatePhenotypes* vytvorí pre všetkých členov populácie fenotypov
- Funkcia Epoch sa volá na konci každej generácie, každému jedincovi priradí jeho fitness, vymaže všetky fenotypy a resetuje niektoré hodnoty jedincov. Potom týchto jedincov usporiada zostupne podľa ich fitness (*SortAndRecord*). Rozdelí populáciu funkciou *SpeciateAndCalculateSpawnLevel*. V rámci každého druhu potom vykonáva selekciu rodičov (*Spawn*) a kríženie (*Crossover*). Vzniknutí jedinci mutujú (*MutationWeight*) a po priradení inovačného čísla, im s istou pravdepodobnosťou pridá / odoberie niektoré gény (*AddNeuron, AddLink*). Nového jedinca potom zaradí do novej populácie.
- Funkcia *SpeciateAndCalculateSpawnLevel* rozdeľuje jedincov do druhov porovnávaním so všetkými jedincami už zahrnutými v daných druhoch. Každému jedincovi potom priradí druhovú fitness a vypočíta pre každý druh koľko potomkov môže mať v ďalšej generácii.
- Funkcia *Crossover* dostane na vstupe dva rodičovské genotypy. Postupne prechádza po všetkých génoch rodičov. Ak majú porovnávané gény rovnaké inovačné čísla, náhodne sa vyberie jeden z nich. Inak sa vyberú gény, ktoré majú nižšie inovačné číslo a zároveň sú lepšie. Ak porovnávané gény má len jeden rodič, prenesú sa na potomka len v prípade, že je daný rodič lepší ako rodič s menším počtom génov.

5.1.4 CNeuralNet

- Táto trieda predstavuje neurónovú sieť
- Funkcia *Update* dostane na vstup vektor prijatých signálov zo senzorov minesweepera. Klasickým postupom výpočtu NS dostaneme zo vstupných údajov 2 výstupné hodnoty z intervalu <0,1>, ktoré vráti ako výstup funkcie, a ktoré určujú rýchlosti otáčok oboch motorov jedinca.

5.1.5 CGenome

- Konštruktor vytvára na začiatku minimálny genotyp, v ktorom je každý vstupný neurón prepojený s každým výstupným neurónom.
- Funkcia *CreatePhenotype* vytvorí z parametrov genotypu nový fenotyp. To znamená, že v novom fenotype vytvorí genotyp neurónovú sieť podľa svojej predlohy.
- Funkcia *AddLink* s pravdepodobnosťou *dChanceAddLink* pridá do siete nový link medzi neuróny, ktoré nie sú navzájom prepojené.
- Funkcia *AddNeuron* s pravdepodobnosťou *dChanceAddNode* pridá do siete nový neurón. Nový neurón pridá medzi dva už spojené neuróny, pričom pôvodný link medzi nimi teraz smeruje do pridaného neurónu a vytvorí sa nový link, ktorý spája pridaný neurón s pôvodným adresátnym neurónom.
- Funkcia *MutateWeights* v cykle s každým linkom s pravdepodobnosťou *dMutationRate* začne mutáciu. S pravdepodobnosťou *dProbabilityWeightReplaced* alebo nahradí danú váhu novou úplne náhodnou z intervalu <-1, 1>, alebo váhu zníži alebo zvýši o náhodné číslo z intervalu <-1, 1> prenásobené parametrom *dMaxWeightPerturbation* < 1.
- Funkcia *MutateActivationResponse* každému neurónu s pravdepodobnosťou *dActivationMutationRate* zmení jeho aktivačný prah o náhodné číslo z intervalu <-1, 1> prenásobené parametrom *dMaxActivationPerturbation* < 1.

5.1.6 CInovation

- Udržuje a upravuje informácie o všetkých inováciách.

5.1.7 CSpecies

- Konštruktor vytvorí nový druh zo vstupného jedinca, ktorý je hneď označený za najlepšie riešenie daného druhu.
- Funkcia *AddNeuron* pridá do druhu nového jedinca, ak je jeho fitness lepšie ako fitness už pridaných jedincov, je označený ako najlepší jedinec daného druhu.
- Funkcia *AdjustFitnesses* najprv fitness jedinca penalizuje, ak je členom druhu staršieho než *iOldAgeThreshold*, a naopak zvýši fitness jedincom v druhoch mladších než *dYoungBonusAgeThreshold*. Takto vyrátanú fitness predelenú počtom jedincov v danom druhu priradí jedincom ako *AdjustedFitness*.
- Funkcia *Spawn* vracia náhodného jedinca z druhu. Nevyberá však zo všetkých jedincov v druhu, ale počet z ktorého vyberá zníži prenásobením parametrom

dSurvivalRate, čo znamená, že keď sú jedinci zoradení zostupne podľa fitness, vráti táto funkcia náhodného jedinca z množiny jej najlepších jedincov.

5.1.8 CMinesweeper

- Udržuje informácie o tom, kde sa vybraný jedinec práve nachádza a kadiaľ už prešiel
- Stará sa o vykreslenie jedincov, správne natočenie
- Funkcia *Update* na vstupe dostane informácie zo senzorov, s týmito parametrami volá funkciu *Update* na sieti, ktorá vráti dvojicu výstupov. Tieto určujú rýchlosť otáčok ľavého a pravého kolesa minesweepera. Po správnom natočení jedinca a jeho pohybe, uloží jeho novú pozíciu ako miesto, ktoré už navštívil.
- Funkcia *TestSensors* kontroluje či senzory jedinca nenarazili na prekážku alebo už prejdenú cestu. Vracia vektor vzdialeností získaných zo senzorov podľa toho, ako ďaleko je prekážka, ak v dosahu senzora žiadne prekážky nie sú, vráti daný senzor hodnotu -1.

5.1.9 SNeuronGene

- Štruktúra, ktorá definuje neurón v genotype
- Udržiava *id* génu, hodnotu aktivačného prahu, typ génu (vstupný, skrytý, výstupný)

5.1.10 SLinkGene

- Štruktúra, ktorá definuje link v genotype
- Pamätá si, ktoré neuróny spája, váhu spojenia, či je rekurentný, a tiež svoje inovačné číslo

5.1.11 SLink

- Štruktúra reprezentujúca link vo fenotype
- Namiesto indexov na neuróny, ktoré spája má ich smerníky, ináč sa od *SLinkGene* nelíši

5.1.12 SNeuron

- Štruktúra reprezentujúca neurón vo fenotype
- Obsahuje vektor linkov, ktoré doň vstupujú a tiež vektor linkov, ktoré z neho vystupujú
- Pamätá si svoju aktuálnu pozíciu, svoj výstup a tiež sumu vynásobených váh a vstupov

5.2 Modifikácia NS na FSA

V rámci transformácie riešenia neurónovou sieťou na riešenie stavovým automatom, musíme zmeniť niektoré vlastnosti algoritmu. Základnou zmenou je topológia, NS má na začiatku všetky linky dopredné, v našej verzii však musia hneď na začiatku existovať aj v ponímaní neurónových sieti rekurentné linky, aby bolo možné sa dostať z jedného stavu do druhého a späť. Taktiež si budeme musieť pamätať, ktorý stav je aktívny, v NS prechádza výpočet všetkými neurónmi. Linky už nebudú potrebovať váhy, namiesto nich im pribudnú podmienky stavových prechodov. Taktiež potrebujeme hodnoty pre motory. V NS sú týmito hodnotami výstupy siete, v našom riešení bude mať každý stavový prechod aj hodnotu výstupu pre motory. Mutačné operátory zmeníme tak, aby mutovali podmienky stavových prechodov a tiež akcie linkov. Obdobne zmeníme operátor kríženia aby kombinoval navzájom prechodové podmienky.

V pôvodnom algoritme sme opravili chybu, ktorá nastávala vždy, keď zanikol v poradí prvý druh.

5.2.1 SLink, SLinkGene

Zmeny v týchto štruktúrach budú identické. Linkom pridáme vektorovú premennú *action*, ktorá bude udržiavať informáciu o rýchlostiach motorov jedinca. Pri inicializácii sa nastavia na náhodné hodnoty. Pridáme tiež vektorovú premennú *conditions*. Tá bude udržiavať vektor 10 reálnych čísel z intervalu <0, 1>. Pre regulárnosť môžeme túto štruktúru nazývať stavový prechod.

5.2.2 Cga

V tejto triede potrebujeme zmeniť len funkciu kríženia – *Crossover*. V tej už nejde o vybratie jedného z génov do nového jedinca, ale o podmienky, ktoré daný jedinec v géne dostane. Ak má niektorý gén iba jeden z rodičov, dostane ho potomok iba za predpokladu, že daný jedinec má vyššie fitness. Ak majú rovnaký gén obaja, potomok dostane gén od toho, ktorý je lepší a zároveň má nižšie inovačné číslo. Náhodne je však vybratá jedna hodnota z vektoru *conditions* nahradená hodnotou z adekvátneho pozície vo vektore *conditions* zavrhnutého rodiča. Ak majú obaja rodičia rovnaké inovačné číslo, náhodne sa vyberie bod rozdelenia vektorov *conditions* oboch rodičov a jedna časť je prijatá z jedného rodiča, druhá z druhého rodiča.

5.2.3 CNeuralNet

Tejto triede pridáme vektorovú premennú *action*, ktorá bude udržiavať rýchlosti motorov jedinca. Jej hodnota sa nastaví na hodnotu *action* linku ktorým sme sa prešli z predošlého stavu do nového stavu.

Do tejto triedy pridáme funkciu *fit_to_conditions(input, conditions).* Na vstupe príjme dáta zo senzorov(*input*) a *conditions* z niektorého linku. Hlavný algoritmus funkcie:

```
1.  double ret = 0;
2.  for (int i=0; i<conditions.size(); i++)
3.  {
4.      if ( abs(input.at(i)) >= conditions.at(i) )
5.      {
6.          ret += abs(input.at(i)) - conditions.at(i);
7.      }
8.      else
9.      {
10.         ret += 1000;
11.     }
12. }
13. return ret;
```

Kontrolujeme všetky hodnoty (2.). Ak je absolútna hodnota vstupu menšia ako *conditions*, pridá, tak to znamená, že vstup nevyhovel podmienke zakódovanej v premennej *conditions* a do výstupu sa pripočíta nejaká vysoká hodnota (10.). V opačnom prípade (6.) sa do výstupu pripočíta len ich rozdiel, čo bude nízke číslo, menšie ako 1. Po skontrolovaní všetkých hodnôt funkcia vráti výstup *ret* (13.). Hodnota *ret* menšia ako 1000 znamená, že vstup vyhovel podmienkam; čím nižšia je hodnota *ret*, tým viac vstup vyhovuje.

Funkciu *Update* zmeníme úplne. Neprechádzame všetkými neurónmi ako pri NS, ale vyberieme si stav (neurón), ktorý je aktívny a kontrolujeme všetky linky, ktoré z neho vychádzajú. *Conditions* z každého takéhoto linku overíme funkciou *fit_to_conditions*. Ak niektorý z linkov vyhovel, zmení sa aktuálny stav na koncový stav linku a jedincovi sa nastaví *action*, ktorá je uložená vo vlastnostiach linku. Táto *action* je potom aj výstupom funkcie.

5.2.4 CGenome

- Konštruktor – tu sme museli zrušiť pridávanie biasu do topológie a upraviť s tým súvisiace indexovanie stavov. Taktiež sme pridali naplnenie nových premenných genotypu *conditions* a *action* náhodnými hodnotami pri každom stave
- *GetCompatibilityScore* – túto funkciu, počítajúcu odlišnosť dvoch genotypov, sme upravili tak, aby namiesto porovnávania váh porovnávala rozdiely v *conditions*.

Počíta sa ako priemerná odchýlka medzi porovnávanými prvkami vektorov *condtions*. Zarátavanie počtu rovnakých stavov a spojov medzi nimi, sme ponechali nezmenené.

- *MutateWeights* – v cykle s každým linkom s pravdepodobnosťou *dMutationRate* vojde do mutácie a tam s pravdepodobnosťou *dProbabilityWeightReplaced* nahradí hodnoty celého vektoru *conditions* úplne náhodnými hodnotami z intervalu <0, 1>, v opačnom prípade s pravdepodobnosťou 0.5 zníži alebo zvýši o náhodné číslo z intervalu <-1, 1> prenásobené parametrom *dMaxWeightPerturbation* < 1 jednu náhodnú hodnotu z vektora *conditions* alebo to urobí so všetkými hodnotami vektora.
- *AddNeuron* – s pravdepodobnosťou dChanceAddNode pridá do genotypu nový stav medzi dva už spojené stavy, pričom pôvodný link medzi nimi teraz smeruje do pridaného stavu a vytvorí sa nový link, ktorý spája pridaný stav s pôvodným adresátnym stavom. Nový link má náhodný vektor *conditions* a tiež vektor *action*.
- *AddLink* – s pravdepodobnosťou *dChanceAddLink* pridá do genotypu nový link medzi stavy, ktoré ešte nie sú navzájom prepojené. Nový link má náhodný vektor *conditions* a tiež vektor *action*
- *CreatePhenotype* – túto funkciu bolo potrebné zmeniť len trochu, aby noví jedinci dostali z genotypu aj nové parametre *conditions* a *action*.

5.2.5 Konfiguračné nastavenia

Do konfiguračného súboru sme pridali štyri nové konštanty:

- *iMaxEpoque* – pomocná konštanta určujúca, koľko epoch sa má algoritmus opakovať kým skončí, pretože v pôvodnom algoritme nebolo žiadne kritérium zastavenia
- *iFile* – pomocná konštanta pri testovaní určujúca číslo testu, ktorý sa vykonáva
- *iNumInputs* – určuje počet vstupných neurónov, v našom prípade stavov
- *iNumOutputs* – určuje počet výstupných neurónov, v našom prípade stavov

5.3 Modifikácie pre experiment 2

V 2. experimente majú jedinci za cieľ naučiť sa navštevovať rohy v užívateľom zadanom poradí. Jedinci dostávajú na vstupe údaje z piatich senzorov o tom ako ďaleko je prekážka v okruhu piatich krokov jedinca (25 pixlov) a uhol o aký je jedinec otočený

oproti zvislému smeru zdola nahor. Pri príchode do správneho rohu dostane jedinec na vstup jednorázovo aj identifikačné číslo rohu, ktorý ma navštíviť ako nasledujúci.

Rohy sú indexované od ľavého horného rohu v smere hodinových ručičiek hodnotami 0,25; 0,5; 0,75; 1. Navštíviť roh znamená dostať sa do vzdialenosti menej ako 10 pixlov horizontálne aj vertikálne od rohu.

Pre tento experiment musíme upraviť počet vstupných neurónov v NS, pretože nebude už dostávať informácie o tom kde už jedinec bol. Do oboch verzií pridáme novú funkciu na rátanie fitness(detailnejšie v kapitole 5.3.3). Do FSA verzie algoritmu pridáme každému linku nový parameter: vektor *jumping*(detailnejšie v kapitole 5.3.2).

5.3.1 Konfiguračné nastavenia

Pridávame konštantu *iPath*, ktorá určuje správne poradie navštívenia rohov mapy. Hodnota je prirodzené nanajvýš 9 ciferné číslo zložené z číslic z množiny {1, 2, 3, 4}.

5.3.2 CGenome

Do genotypu sme pridali nový parameter *jumping*. Tento parameter sme pridali ako určujúci element prechodu do iného stavu ak sa jedinec dostane do nového rohu. Algoritmus prezrie všetky linky vychádzajúce z aktívneho stavu, a za nový aktívny vrchol sa považuje adresátny stav linku, ktorého hodnota *jumping* je najbližšia hodnote postupnosti už prejdenej cesty.

- *MutateWeights* – zmeny v tejto funkcii sa týkajú len FSA verzie algoritmu. Podobne ako *conditions* a *action*, s istou pravdepodobnosťou mutujeme jednu alebo všetky hodnoty vektora *jumping*. Táto mutácia spočíva v nahradení danej hodnoty náhodnou z hodnôt 0,25; 0,5; 0,75; 1.

5.3.3 CMinesweeper

Tejto triede sme pridali novú premennú *recentPath*(typu int), v ktorej si budeme pamätať poradie rohov, ktoré sme už prešli. Napríklad hodnota 141 znamená, že jedinec postupne navštívil ľavý horný roh, ľavý dolný roh a znovu ľavý horný roh.

- *Update* – do tejto funkcie pridávame volanie novej funkcie *UpdateFitness*, ktorá aktualizuje fitness po každom kroku jedinca.
- *UpdateFitness* – táto funkcia slúži na nový výpočet fitness jedinca. Počíta sa po každom jeho kroku, len ak jedinec ešte nenavštívil nesprávny roh. Ak sa jedinec nenachádza v žiadnom rohu, jeho fitness sa nemení. Ak je jedinec znovu v poslednom navštívenom rohu jeho fitness sa zníži o hodnotu 2, aby nemal

tendenciu v tom rohu ostať. Ak je jedinec v rohu, ktorý nie je ďalším v poradí, ani posledným navštíveným rohom, jeho fitness je penalizované o polovicu bonusu, a v ďalších krokoch sa už jeho fitness nemení. Bonus je hodnota rovná povolenému počtu krokov daného jedinca v rámci jednej epochy. Ak jedinec navštívi roh, ktorý má správne nasledovať, jeho fitness sa zvýši o hodnotu bonusu a zníži sa o polovicu počtu krokov, ktoré dosiaľ vykonal. To zabezpečí, že jedinci, ktorí sa do daného rohu dostanú skôr budú mať vyššie fitness než pomalšie jedince. Ak je navštívený vrchol zároveň posledným určeným vrcholom na ceste, fitness jedinca sa zvýši o trojnásobok počtu krokov, ktoré ešte môže vykonať.

- *jumpToOtherState* – táto funkcia sa využíva len v FSA verzii algoritmu. Volá sa vždy, keď sa jedinec dostane do nového rohu mapy. Na linkoch zo súčasne aktívneho stavu vyhľadá ten, ktorého hodnota parametra *jumping* je najbližšie k hodnote parametra *recentPath*. Za aktívny stav potom nastaví ten, do ktorého vchádza vybraný link a vráti vektor *action* daného linku ako výstup funkcie.

5.4 Algoritmus

Pri inicializácii sa načítajú parametre z konfiguračného súboru params.ini. Pomocou týchto parametrov sa potom nastavia vlastnosti prvej populácie náhodných genotypov, ktoré sa funkciou *CreatePhenotype* prekódujú do fenotypov. Títo jedinci(fenotypy) sa začnú pohybovať z rovnakého miesta v 2D ohraničenom prostredí s prekážkami. Plochu tvoria štvorcové dlaždičky veľkosti 5x5 pixlov.

Úlohou jedincov v 1. experimente je prejsť čo najväčší počet týchto dlaždičiek, pričom opätovné stúpenie na dlaždičku sa nepočíta. Tento počet potom určuje fitness jednotlivých jedincov. Fitness sa jedincom počíta len na konci epochy.

Úlohou jedincov v 2. experimente je prejsť po vopred určených rohoch mapy v správnom poradí v čo najkratšom čase. Poradie vrcholov pritom jedincom nie je vopred známe, musia sa ho počas vývoja naučiť. Fitness sa jedincom ráta priebežne počas pohybu.

Po ohraničenom počte krokov sa daná epocha končí, a na genotypy jedincov sú aplikované evolučné operátory. Celá populácia sa najprv rozdelí na druhy. Uplatňuje sa elitizmus, teda najlepší jedinec z každého druhu sa automaticky prenáša do novej generácie. Procesom selekcie sú vyradení najhorší jedinci z každého druhu. Rodičia vybratí selekciou vstupujú do kríženia a mutácie. Vzniknutí jedinci majú ešte šancu na pridanie linku alebo stavu do svojej topológie. Noví jedinci vstupujú do ďalšej generácie.

6 Experimenty

6.1 Experiment 1

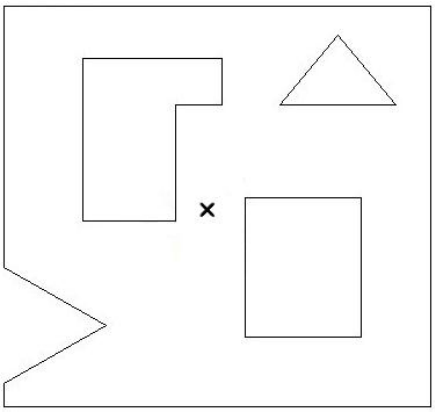
Nami vytvorenú implementáciu budeme porovnávať s pôvodnou verziou evolvujúcou NS. Väčšinu parametrov ponechávame rovnaké ako boli nastavené autormi danej implementácie, kvôli predpokladu, že vybrali pre sieť najlepšie vyhovujúce parametre.

V našej implementácii budeme meniť počet stavov v prvotnom jedincovi. Aplikáciu budeme testovať s 2, 4, 6 a 8 počiatočnými stavmi pre každého jedinca. V pôvodnej aplikácii ponecháme počet 11 vstupných a 2 výstupných neurónov. Každú konfiguráciu budeme testovať po dobu 300 epoch pričom parametre, ktoré budeme meniť sú: počet jedincov v populácii, počet krokov, ktoré môžu v každej epoche vykonať a parameter *maxWeightPerturbation*. Tento parameter ako sme už uviedli určuje, o akú maximálnu hodnotu sa môžu zmeniť hodnoty váh / stavových podmienok v priebehu mutácie. Každú z týchto konfigurácii budeme testovať zo štatistických dôvodov desaťkrát.

Našou hypotézou je, že FSA verzia aplikácie by si pri niektorých z týchto konfigurácii mala počínať rovnako, v niektorých prípadoch dokonca lepšie ako NS verzia aplikácie, a to aj napriek tomu, že priestor riešení je pri FSA mnohonásobne väčší než pri NS.

Taktiež očakávame, že vyššia hodnota parametra *maxWeightPerturbation*, ktorý budeme testovať s hodnotami 0,1 a 0,5, by mala priniesť vylepšenie pre NS aj pre FSA reprezentácie, pretože dovoľuje rýchlejšie prehľadať väčší priestor riešení.

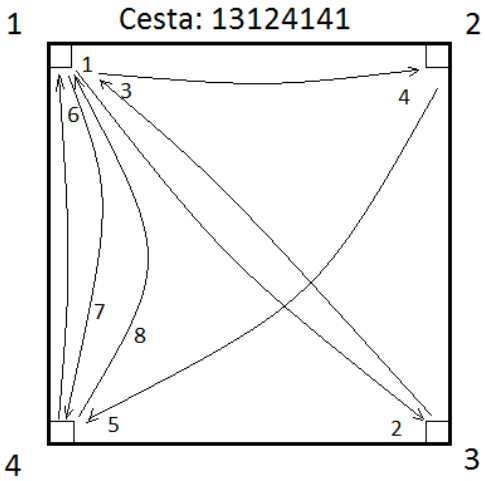
Všetky tieto testy budú uskutočnené na 2D mape veľkosti 345×368 pixlov, so štartom na pozícii (180, 200) a štyrmi prekážkami ako je znázornené na obrázku 11.



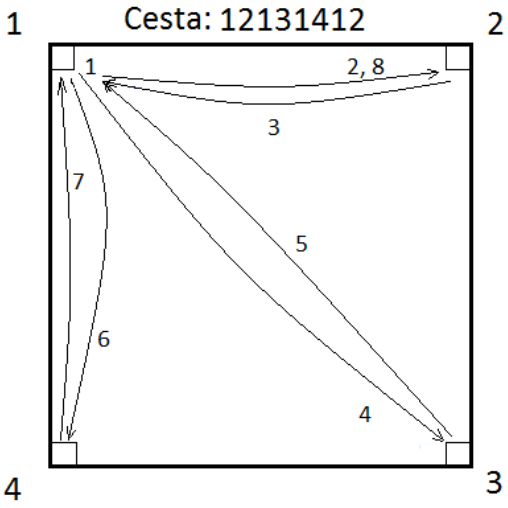
Obr. 11 2D mapa prostredia, po ktorej sa pohybujú jedinci, x určuje počiatočnú pozíciu jedincov.

6.2 Experiment 2

Pri ďalšom experimente použijeme štvorcovú 2D mapu rozmerov 160×160 pixlov. Každú konfiguráciu oboch implementácií (NS aj FSA) budeme testovať päťkrát s dvomi rôznymi cestami. Prvým parametrom, ktorý budeme v konfiguráciách meniť je počet počiatočných stavov. Aplikáciu budeme testovať s 8 a 12 počiatočnými stavmi. Ďalším parametrom, ktorý budeme v testoch meniť, je veľkosť populácie. Hodnoty tohto parametra budú: 80, 100, 150, 200. Tretím parametrom, ktorý budeme meniť je povolený počet krokov. Hodnoty tohto parametra budú 2000 a 4000. Posledným parametrom bude cesta po ktorej má jedinec prejsť. Hodnoty tohto parametra budú 13124141 a 12131412. Na obrázkoch 12 a 13 sú znázornené trajektórie s vyznačeným správnym poradím navštívených vrcholov.



Obr. 12 Znázornenie správneho poradia navštívenia rohov cesty 13124141.



Obr. 13 Znázornenie správneho poradia navštívenia rohov cesty 12131412.

7 Výsledky a diskusia

7.1 Experiment 1

Výsledky zo všetkých testovacích behov sú zahrnuté v tabuľkách 1 až 5. Z desiatich behov programu sú v tabuľkách uvedené vždy priemerné, minimálne a maximálne hodnoty dosiahnutých fitness. Priemerné fitness sa ráta ako aritmetický priemer všetkých dosiahnutých hodnôt fitness v danej konfigurácii. V tabuľkách s FSA verziou sú doplňujúce stĺpce, ktoré udávajú či bol dosiahnutý výsledok lepší ako výsledok NS verzie.

V tabuľke 1 sú údaje z testov NS verzie algoritmu. V tabuľke 2 sú údaje z testov FSA verzie algoritmu s 2 stavmi na počiatku. V tabuľke 3 sú údaje z testov FSA verzie algoritmu so 4 stavmi na počiatku. V tabuľke 4 sú údaje z testov FSA verzie algoritmu so 6 stavmi na počiatku. V tabuľke 5 sú údaje z testov FSA verzie algoritmu s 8 stavmi na počiatku.

Tabuľka 1 NS verzia aplikácie.

MWP	PopSize	NumTicks	Avg Fitness	Max Fitness	Min Fitness
0,1	50	300	147,5	151	143
0,1	50	600	271,7	275	266
0,1	50	1500	633,4	648	622
0,1	50	2500	986,5	1014	965
0,1	80	300	147,8	151	145
0,1	80	600	271	276	263
0,1	80	1500	635,6	647	627
0,1	80	2500	993,9	1021	954
0,1	100	300	148,6	154	145
0,1	100	600	276	281	270
0,1	100	1500	639,6	653	622
0,1	100	2500	994,8	1015	976
0,1	150	300	148,1	150	145
0,1	150	600	275,8	277	273
0,1	150	1500	640,1	651	630
0,1	150	2500	999,3	1014	970
0,5	50	300	147,7	152	145
0,5	50	600	278,6	285	271
0,5	50	1500	651,5	657	643
0,5	50	2500	1017,3	1048	994
0,5	80	300	148,8	152	145
0,5	80	600	279,2	282	276
0,5	80	1500	657,5	669	641
0,5	80	2500	1022,8	1032	1010

0,5	100	300	149,3	153	144
0,5	100	600	281,3	291	272
0,5	100	1500	658,9	664	651
0,5	100	2500	1020	1032	1008
0,5	150	300	149,7	157	148
0,5	150	600	284,7	299	277
0,5	150	1500	655,5	665	641
0,5	150	2500	1025,9	1042	1007

Tabuľka 2 FSA verzia aplikácie s 2 počiatočnými stavmi.

MWP	PopSize	NumTicks	Avg Fitness	NS	Max Fitness	NS	Min Fitness	NS
0,1	50	300	137,3	-	172	+	107	-
0,1	50	600	229,2	-	261	-	205	-
0,1	50	1500	496,9	-	554	-	484	-
0,1	50	2500	744,4	-	826	-	729	-
0,1	80	300	129,3	-	163	+	106	-
0,1	80	600	234,9	-	266	-	206	-
0,1	80	1500	522	-	587	-	488	-
0,1	80	2500	773,2	-	850	-	730	-
0,1	100	300	152,9	+	172	+	124	-
0,1	100	600	238,9	-	277	-	206	-
0,1	100	1500	509,3	-	613	-	481	-
0,1	100	2500	767,8	-	860	-	736	-
0,1	150	300	152,7	+	173	+	137	-
0,1	150	600	253,3	-	285	+	211	-
0,1	150	1500	524,5	-	628	-	484	-
0,1	150	2500	773,5	-	856	-	736	-
0,5	50	300	138	-	150	-	114	-
0,5	50	600	234,2	-	262	-	200	-
0,5	50	1500	528,5	-	614	-	477	-
0,5	50	2500	767,7	-	920	-	719	-
0,5	80	300	144	-	155	+	125	-
0,5	80	600	253,1	-	268	-	226	-
0,5	80	1500	550,1	-	628	-	487	-
0,5	80	2500	778,9	-	860	-	731	-
0,5	100	300	144,6	-	157	+	136	-
0,5	100	600	249,7	-	284	-	208	-
0,5	100	1500	504,33	-	573	-	474	-
0,5	100	2500	791,6	-	880	-	718	-
0,5	150	300	154,3	+	169	+	141	-
0,5	150	600	258,1	-	270	-	212	-

0,5	150	1500	585,78	-	651	-	515	-
0,5	150	2500	856	-	946	-	792	-

Tabuľka 3 FSA verzia aplikácie so 4 počiatočnými stavmi.

MWP	PopSize	NumTicks	Avg Fitness	NS	Max Fitness	NS	Min Fitness	NS
0,1	50	300	159,9	+	174	+	142	-
0,1	50	600	271,3	-	289	+	262	-
0,1	50	1500	592,7	-	627	-	502	-
0,1	50	2500	855,3	-	901	-	790	-
0,1	80	300	163,2	+	178	+	145	0
0,1	80	600	280,5	+	309	+	265	+
0,1	80	1500	609,6	-	645	-	577	-
0,1	80	2500	889,1	-	945	-	841	-
0,1	100	300	169,7	+	177	+	155	+
0,1	100	600	275,8	-	297	+	266	-
0,1	100	1500	614,78	-	641	-	564	-
0,1	100	2500	924,8	-	966	-	869	-
0,1	150	300	172,4	+	180	+	159	+
0,1	150	600	277,5	+	292	+	268	-
0,1	150	1500	638,5	-	668	+	616	-
0,1	150	2500	902,7	-	943	-	871	-
0,5	50	300	160,78	+	176	+	142	-
0,5	50	600	269,3	-	291	+	260	-
0,5	50	1500	598,2	-	627	-	547	-
0,5	50	2500	889	-	935	-	836	-
0,5	80	300	160	+	178	+	143	-
0,5	80	600	271,78	-	278	-	264	-
0,5	80	1500	609,11	-	657	-	576	-
0,5	80	2500	924,78	-	982	-	854	-
0,5	100	300	164,78	+	171	+	157	+
0,5	100	600	274,78	-	298	+	263	-
0,5	100	1500	624,56	-	672	+	594	-
0,5	100	2500	921	-	1000	-	857	-
0,5	150	300	161,56	+	172	+	148	0
0,5	150	600	275,1	-	288	-	266	-
0,5	150	1500	627,1	-	694	+	597	-
0,5	150	2500	923	-	949	-	883	-

Tabuľka 4 FSA verzia aplikácie so 6 počiatočnými stavmi.

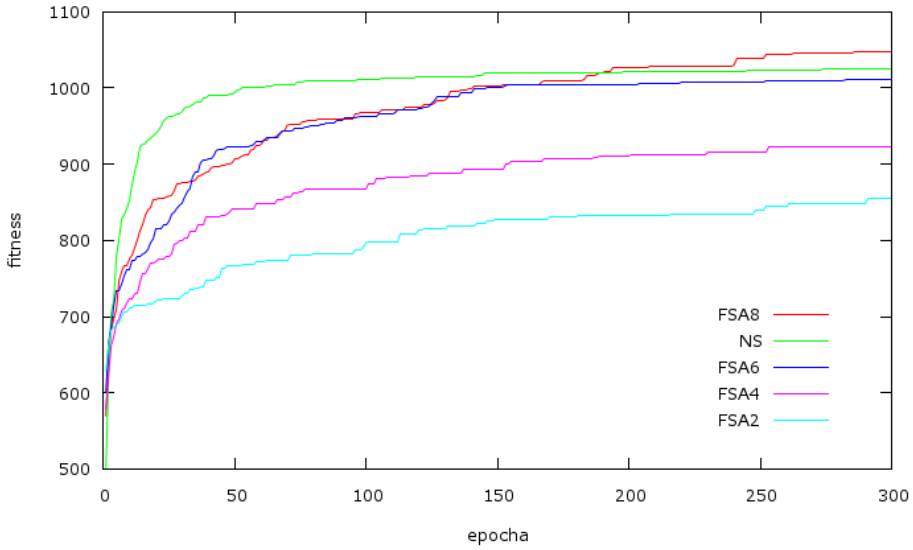
MWP	PopSize	NumTicks	Avg Fitness	NS	Max Fitness	NS	Min Fitness	NS
0,1	50	300	166	+	181	+	141	-
0,1	50	600	281,8	+	315	+	262	-
0,1	50	1500	624	-	675	+	594	-
0,1	50	2500	909,3	-	959	-	853	-
0,1	80	300	165,3	+	181	+	151	+
0,1	80	600	283,4	+	298	+	266	+
0,1	80	1500	627,8	-	707	+	502	-
0,1	80	2500	938,6	-	971	-	888	-
0,1	100	300	170,22	+	183	+	151	+
0,1	100	600	292,5	+	316	+	265	-
0,1	100	1500	636,4	-	668	+	603	-
0,1	100	2500	970,9	-	1126	+	888	-
0,1	150	300	174,2	+	181	+	151	+
0,1	150	600	302	+	331	+	276	+
0,1	150	1500	644,9	+	670	+	614	-
0,1	150	2500	964,1	-	1054	+	914	-
0,5	50	300	165,1	+	178	+	142	-
0,5	50	600	282,6	+	301	+	267	-
0,5	50	1500	620,5	-	668	+	583	-
0,5	50	2500	947,3	-	1063	+	767	-
0,5	80	300	162,7	+	181	+	143	-
0,5	80	600	279,6	+	300	+	269	+
0,5	80	1500	650,4	-	716	+	612	-
0,5	80	2500	979,6	-	1097	+	881	-
0,5	100	300	165,4	+	176	+	144	0
0,5	100	600	286,3	+	299	+	274	+
0,5	100	1500	648,5	-	720	+	605	-
0,5	100	2500	1010,5	-	1134	+	915	-
0,5	150	300	169,7	+	179	+	152	+
0,5	150	600	287,67	+	316	+	274	-
0,5	150	1500	647,6	-	676	+	625	-
0,5	150	2500	1011,8	-	1191	+	911	-

Tabuľka 5 FSA verzia aplikácie s 8 počiatočnými stavmi.

MWP	PopSize	NumTicks	Avg Fitness	NS	Max Fitness	NS	Min Fitness	NS
0,1	50	300	168,7	+	180	+	146	+
0,1	50	600	297,1	+	302	+	279	+
0,1	50	1500	663,4	+	760	+	604	-
0,1	50	2500	955,9	-	990	-	910	-
0,1	80	300	172,8	+	181	+	150	+
0,1	80	600	296,8	+	319	+	275	+
0,1	80	1500	652,8	+	697	+	607	-
0,1	80	2500	975,7	-	1167	+	924	-
0,1	100	300	175,3	+	182	+	152	+
0,1	100	600	294,4	+	314	+	282	+
0,1	100	1500	662,9	+	704	+	613	-
0,1	100	2500	986,8	-	1060	+	929	-
0,1	150	300	169,6	+	181	+	148	+
0,1	150	600	302,2	+	363	+	274	+
0,1	150	1500	679,2	+	764	+	624	-
0,1	150	2500	971,7	-	1094	+	914	-
0,5	50	300	163,6	+	173	+	143	-
0,5	50	600	283,9	+	298	+	266	-
0,5	50	1500	635,9	-	687	+	612	-
0,5	50	2500	994,3	-	1118	+	880	-
0,5	80	300	167,3	+	174	+	153	+
0,5	80	600	284,7	+	318	+	273	-
0,5	80	1500	680,5	+	749	+	647	+
0,5	80	2500	1065,3	+	1176	+	970	-
0,5	100	300	172,9	+	177	+	165	+
0,5	100	600	286,2	+	294	+	276	+
0,5	100	1500	665,9	+	735	+	627	-
0,5	100	2500	1051,2	+	1156	+	846	-
0,5	150	300	172,7	+	177	+	167	+
0,5	150	600	294	+	310	+	275	-
0,5	150	1500	682,9	+	708	+	660	+
0,5	150	2500	1048,6	+	1112	+	967	-

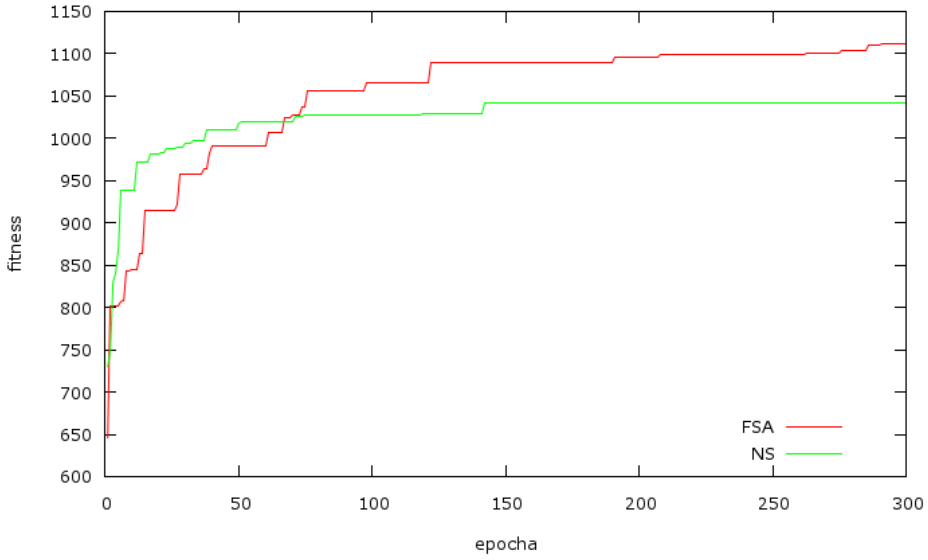
Z nameraných výsledkov je zrejmé, že najlepšie si počínali stavové automaty s 8 počiatočnými stavmi, i keď minimálne hodnoty nimi dosiahnuté boli o dosť horšie než hodnoty NS. FSA so 6 stavmi sa podarilo vo väčšine prípadov dosiahnuť lepšie maximálne fitness a tiež priemerné fitness pri testoch s menším počtom povolených krokov než NS.

Ako vidieť na obrázku 14 fitness jedincov reprezentovaných neurónovými sieťami sa zo začiatku vyvíja rýchlo, ale už okolo 40. epochy stagnuje a stúpa minimálne. Jedinci reprezentovaní stavovými automatmi, sa vyvíjajú pomalšie, ale rovnomerne počas celej doby trénovania. Napriek tomu v časovom horizonte 300 povolených krokov, len jedinci s ôsmimi počiatočnými stavmi dosahujú vyššiu fitness než NS.



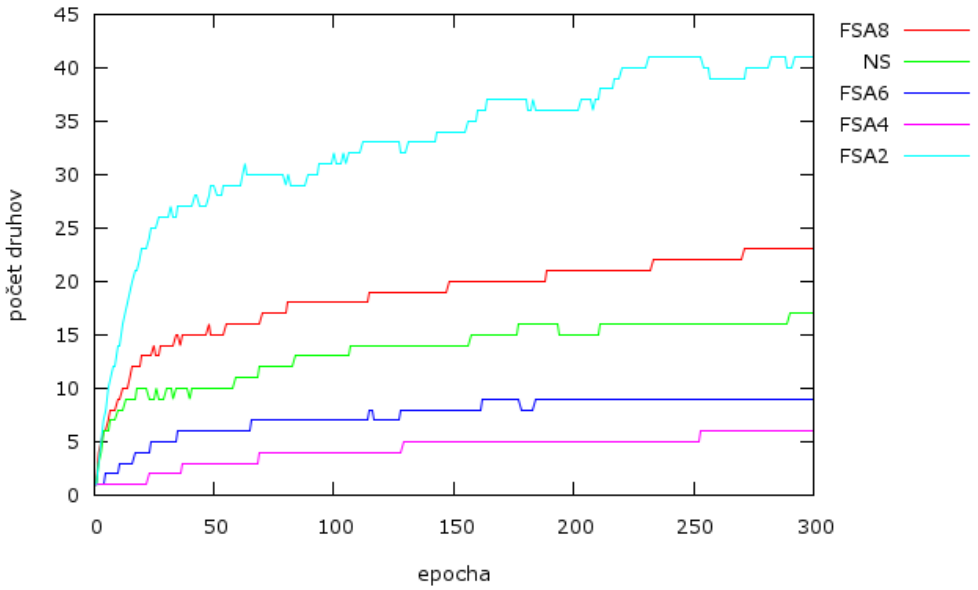
Obr. 14 Vývoj priemernej fitness FSA a NS verzií aplikácie (Konfigurácia: *PopSize*=150, *NumTicks*=2500, *MWP*=0,5).

Obrázok 15 znázorňuje vývoj najlepšej dosiahnutej fitness oboch verzií aplikácie. Ako je z grafu vidieť, aj vývoj maximálnych fitness má rovnaký priebeh, v NS verzii sa vyvíja len niekoľko epoch a potom stagnuje, zatiaľ čo FSA verzia fitness stúpa po celý čas.



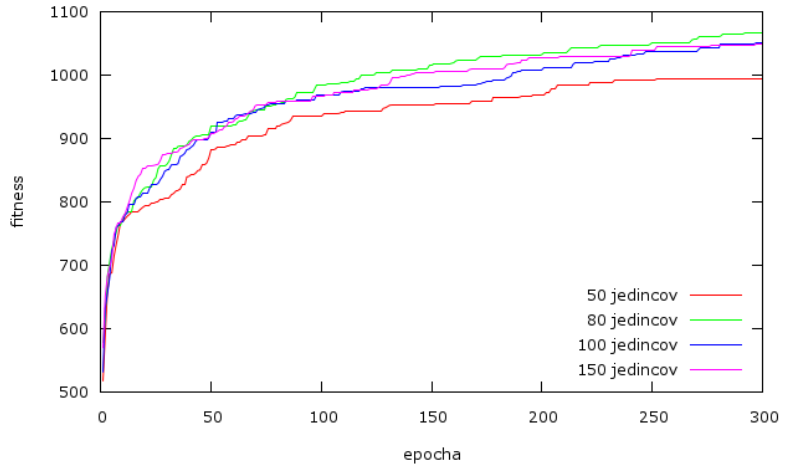
Obr. 15 Vývoj fitness najlepšieho jedinca FSA verzie s 8 počiatočnými stavmi a NS verzie (Konfigurácia: *PopSize*=150, *NumTicks*=2500, *MWP*=0,5).

Obrázok 16 znázorňuje vývoj počtu druhov v jednotlivých konfiguráciách FSA a NS verzií aplikácie. Vo všeobecnosti je vidieť, že počet druhov postupne narastá, populácia teda diverguje. Najväčší počet druhov sa vytvára pri stavových automatoch s 2 počiatočnými stavmi. Dá sa to logicky odôvodniť tým, že funkcia rátajúca rozdiely medzi jedincami a rozdeľujúca ich na druhy dáva veľký dôraz na počet stavov. Preto aj rozdiel o jeden stav znamená veľkú zmenu genotypu.

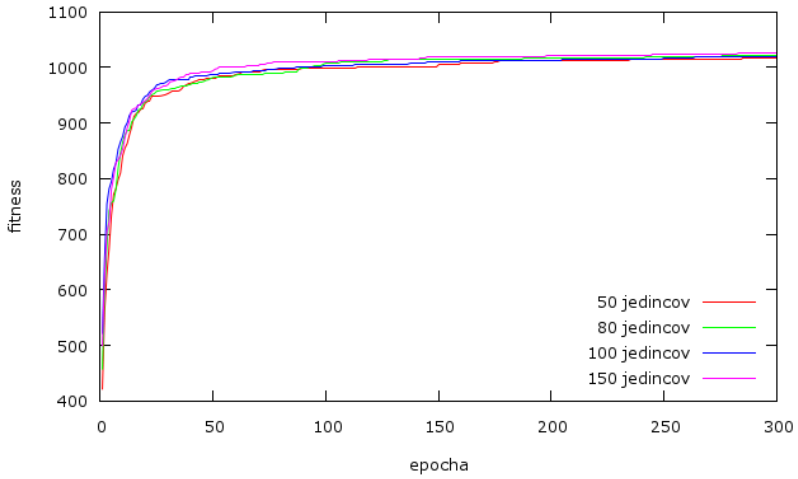


Obr. 16 Vývoj priemerného počtu druhov pre jednotlivé verzie aplikácie (Konfigurácia: *PopSize*=150, *NumTicks*=2500, *MWP*=0,5).

Obrázky 17 a 18 znázorňujú vývoj priemernej fitness populácie FSA s ôsmimi počiatočnými stavmi a populácie NS. Z grafov je zrejmé, že početnosť populácie jedincov NS nemá väčší vplyv na vývoj priemernej fitness. Naproti tomu rozdiely pri FSA sú viac viditeľné. Populácia 50 jedincov tu značne zaostáva za početnejšími populáciami (80, 100 a 150 jedincov). ktorých priemerné fitness je relatívne na rovnakej úrovni.

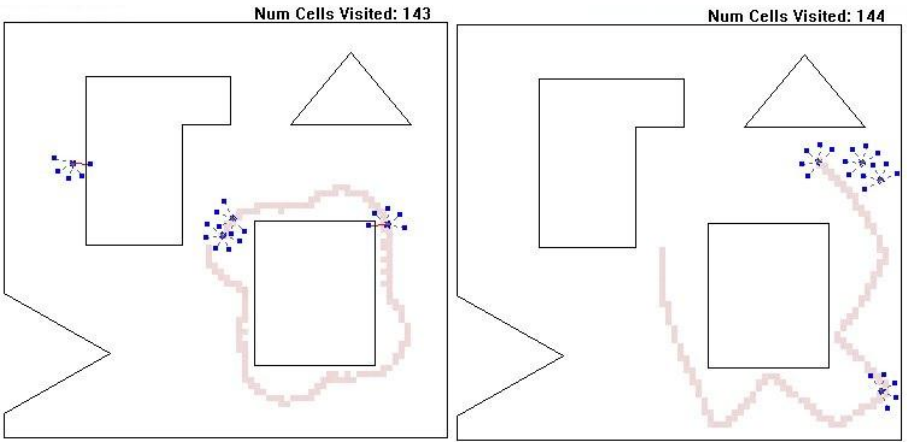


Obr. 17 Vývoj priemernej fitness podľa veľkosti populácie FSA verzie aplikácie (Konfigurácia: *NumTicks*=2500, *MWP*=0,5).

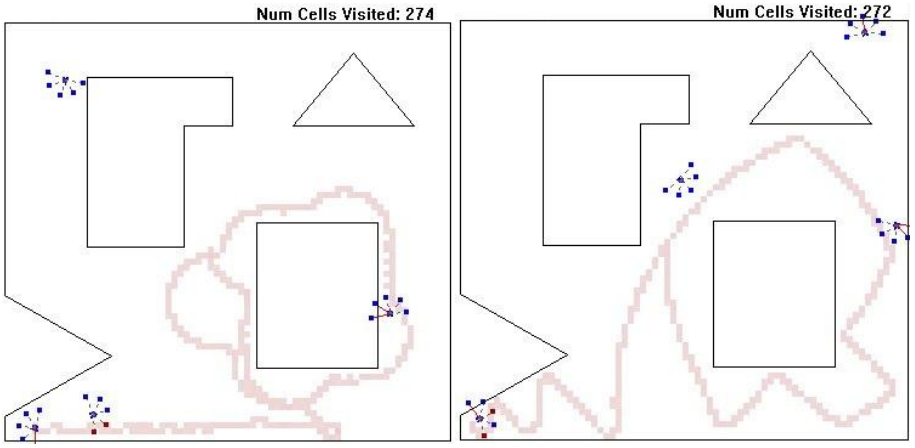


Obr. 18 Vývoj priemernej fitness podľa veľkosti populácie NS verzie aplikácie (Konfigurácia: *NumTicks*=2500, *MWP*=0,5).

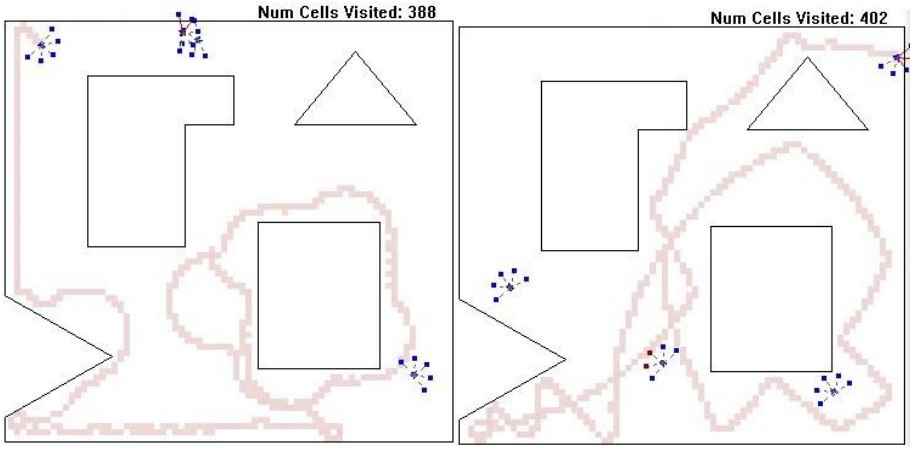
Na nasledujúcich obrázkoch(19-23) je vyznačená trajektória najlepšieho jedinca z oboch verzií aplikácii ukladaná postupne po každom tristom kroku. Konfigurácia týchto aplikácii je nasledovná: počet jedincov je 80, povolený počet krokov je 1500. Počiatočný počet stavov automatov bol nastavený na hodnotu 8. Obrázky sú zo 150. epochy oboch algoritmov. Spolu sú na každom obrázku znázornení 4 najlepší jedinci, ale zobrazená je len trajektória toho najlepšieho. Na obrázkoch je vidieť, že FSA verzia jedinca koná systematickejšie, teda nekrižuje svoju kategóriu, ale maximálne tesne kopíruje. Naproti tomu NS verzia svoju trajektóriu bez zaváhania niekoľkokrát križuje.



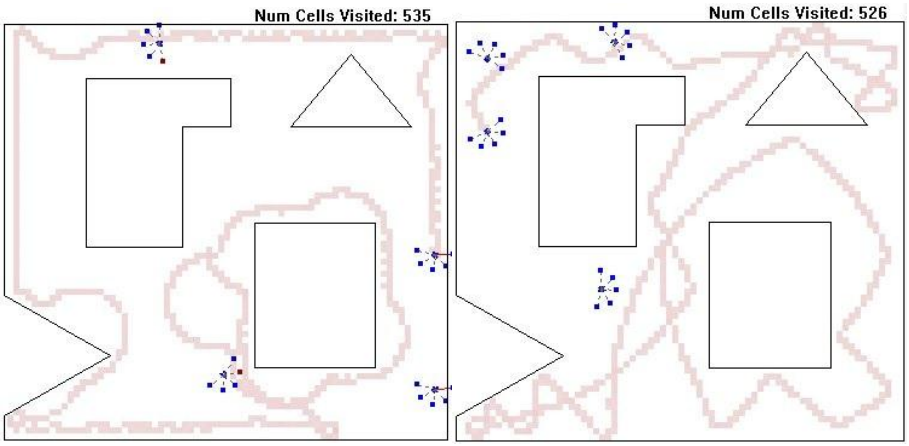
Obr. 19 Najlepší jedinci FSA(vľavo) a NS verzií aplikácie po 300 prejdených krokoch.



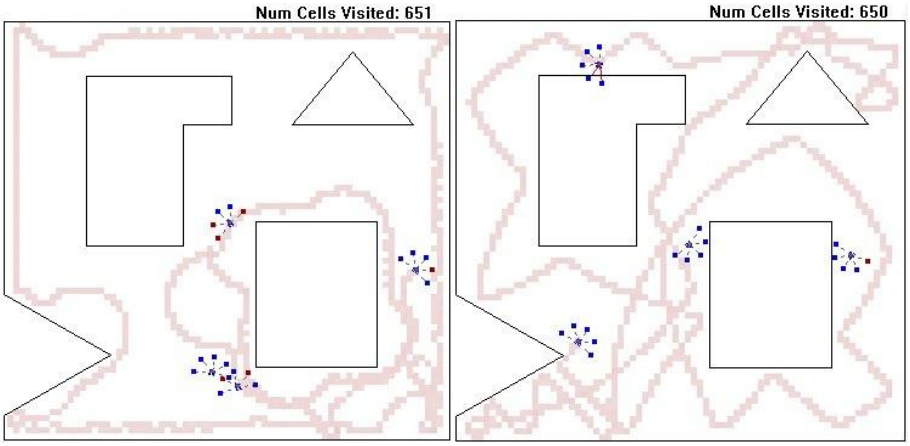
Obr. 20 Najlepší jedinci FSA(vľavo) a NS verzií aplikácie po 600 prejdených krokoch.



Obr. 21 Najlepší jedinci FSA(vľavo) a NS verzií aplikácie po 900 prejdených krokoch.



Obr. 22 Najlepší jedinci FSA(vľavo) a NS verzií aplikácie po 1200 prejdených krokoch.



Obr. 23 Najlepší jedinci FSA(vľavo) a NS verzií aplikácie po 1500 prejdených krokoch.

7.2 Experiment 2

Výsledky z 2. experimentu sú zapísané v tabuľkách 6 až 11. Tabuľky 6 a 9 reprezentujú výsledky NS verzie algoritmu a tabuľky 7, 8, 10 a 11 reprezentujú výsledky FSA verzie algoritmu. Podobne ako v 1. experimente sú aj tu pre každú konfiguráciu aplikácií vyhodnotené hodnoty priemerných, maximálnych a minimálnych dosiahnutých fitness. Priemerné hodnoty sú vypočítané ako aritmetický priemer zo všetkych behov v danej konfigurácii. Tabuľky pre FSA verzie algoritmov obsahujú navyše tri stĺpce, ktoré udávajú či je FSA verzia aplikácie v danej konfigurácii lepšia (+), rovnaká (0) alebo horšia (-) ako NS verzia aplikácie.

Tabuľka 6 NS verzia 2. experimentu s cestou 13124141.

PopSize	NumTicks	Avg Fitness	Max Fitness	Min Fitness
80	2000	7955,0	9362	6790

80	4000	18366,8	22709	15099
100	2000	8621,8	9669	7255
100	4000	26029,8	32269	20176
150	2000	9818,4	10604	9250
150	4000	30618,4	33470	24690
200	2000	9274,6	10651	6963
200	4000	27751,0	32956	14876

Tabuľka 7 FSA verzia 2. experimentu s 8 počiatočnými stavmi a cestou 13124141.

PopSize	NumTicks	Avg Fitness	NS	Max	NS	Min	NS
80	2000	9282,0	+	13036	+	6390	-
80	4000	19544,4	+	38343	+	10167	-
100	2000	7029,2	-	10065	+	5703	-
100	4000	15335,6	-	20861	-	11969	-
150	2000	8386,8	-	9372	-	7114	-
150	4000	26171,2	-	36652	+	14715	-
200	2000	7923,0	-	9641	-	6813	-
200	4000	27479,6	-	34387	+	21417	+

Tabuľka 8 FSA verzia 2 experimentu s 12 počiatočnými stavmi s cestou 13124141.

PopSize	NumTicks	Avg Fitness	NS	Max	NS	Min	NS
80	2000	7226,6	-	8610	+	5805	-
80	4000	17882,6	-	27201	+	13420	-
100	2000	8129,8	-	9349	-	7316	+
100	4000	23693,6	-	37158	+	17392	-
150	2000	8023,6	-	9981	-	7222	-
150	4000	22671,2	-	32606	-	14549	-
200	2000	10414,0	+	13753	+	6944	-
200	4000	28560,2	+	36708	+	18214	+

Tabuľka 9 NS verzia 2.experimentu s cestou 12131412.

PopSize	NumTicks	Avg Fitness	Max Fitness	Min Fitness
80	2000	9714,6	12524	8104
80	4000	18672,8	18818	18451
100	2000	7821,2	8586	5803
100	4000	24578,2	30847	17958
150	2000	8978,0	10091	8382
150	4000	26379,2	33797	20777
200	2000	10585,6	13613	8669
200	4000	23755,2	27034	18345

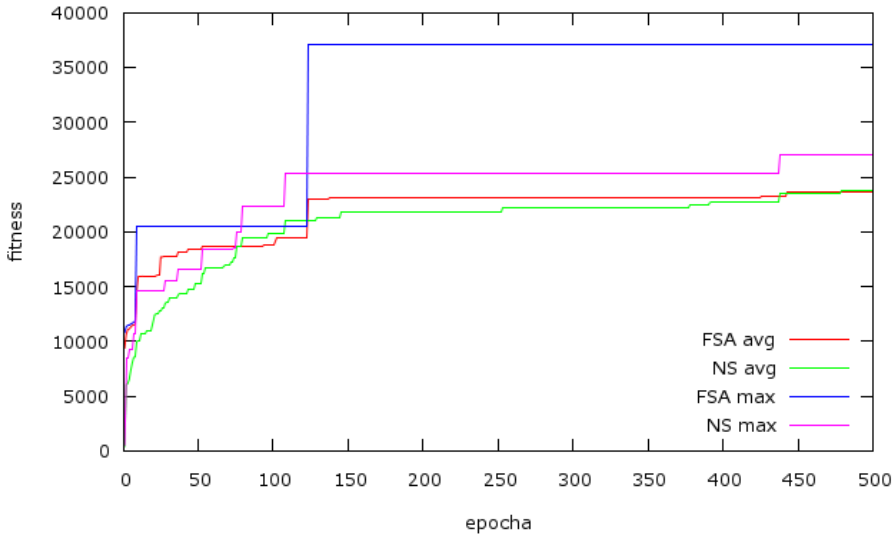
Tabuľka 10 FSA vezia 2. experimentu s 8 počiatočnými stavmi a cestou 12131412.

PopSize	NumTicks	Avg Fitness	NS	Max	NS	Min	NS
80	2000	9227,0	-	9535	-	8635	+
80	4000	16084,6	-	19312	+	11904	-
100	2000	10533,6	+	13526	+	7694	+
100	4000	23252,0	-	34882	+	16636	-
150	2000	9644,0	+	11565	+	8635	+
150	4000	20755,5	-	22654	-	18705	-
200	2000	9932,6	-	12855	+	8731	+
200	4000	24020,4	+	36328	+	17899	-

Tabuľka 11 FSA verzia 2. experimentu s 12 počiatočnými stavmi a cestou 12131412.

PopSize	NumTicks	Avg Fitness	NS	Max	NS	Min	NS
80	2000	9300,8	-	10093	-	9011	+
80	4000	21536,2	+	30589	+	19113	+
100	2000	9459,6	+	9520	+	9330	+
100	4000	18738,8	-	19734	-	16200	-
150	2000	10409,8	+	12508	+	8370	-
150	4000	25824,5	-	33729	-	19228	-
200	2000	9438,4	-	9545	-	9272	+
200	4000	23625,6	-	37029	+	19130	+

Ako vyplýva z tabuliek FSA verzia algoritmu si na tejto úlohe počínala v priemernom meradle horšie ako NS verzia. Napriek tomu vo viac než polovici úloh dosiahla lepšie maximálne fitness než NS verzia. Na obrázku 25 je znázornený vývoj fitness FSA a NS verzií aplikácie s populáciou veľkosti 200, počtom povolených krokov 4000 a predurčenou cestou 12131412.



Obr. 24 Vývoj priemerných a maximálnych fitness oboch verzií aplikácie.

Záver

V tejto práci sme sa oboznámili s rôznymi evolučnými algoritmami, predstavili sme si neurónové siete a bežné metódy, ktoré spravidla evolúciou upravovali ich váhy, ale aj metódy, ktoré zároveň evolvovali ich topológiu.

Špeciálne sme sa venovali metóde NEAT, ktorá evolvuje neurónové siete a rieši niekoľko problémov, s ktorými iné predchádzajúce metódy mali problémy.

Predstavili sme si model stavových automatov a porovnaním s rekurentnými neurónovými sieťami sme dospeli k názoru, že neurónové siete v metóde NEAT by mohli byť úspešne nahradené stavovými automatmi, ak by sme v algoritme NEAT vykonali niektoré potrebné zmeny.

Cieľom tejto práce bolo navrhnúť vlastnú metódu evolúcie stavových automatov. Tento cieľ sa nám úspešne podarilo splniť. Pri návrhu tejto metódy a jej implementácii sme sa inšpirovali metódou NEAT, ktorá evolvuje rekurentné neurónové siete. Využili sme pri tom fakt, že rekurentné neurónové siete a stavové automaty majú veľa podobných atribútov. Najvýraznejším takýmto atribútom je podobnosť topológie oboch reprezentácií.

Ďalším našim cieľom bolo vyskúšať našu metódu na vhodnej úlohe a porovnať jej výsledky s výsledkami pôvodného riešenia tejto úlohy metódou NEAT. Aj tento cieľ sme úspešne splnili. Testovali sme našu metódu s rôznymi počtami počiatočných stavov. Z našich výsledkov vyplynulo, že najlepšie si počínala naša metóda s automatmi, ktoré mali pri inicializácii osem stavov, teda najviac z testovaných konfigurácií. Porovnaním s pôvodným riešením tejto úlohy metódou NEAT sme zistili, že naša metóda s ôsmymi počiatočnými stavmi dosahovala lepšie priemerné aj maximálne výsledky.

Z našich pozorovaní tiež vyplynulo, že fitness stavových automatov evolvovaných upravenou verziou metódy NEAT rastie na danej úlohe plynulejšie než fitness pôvodnej reprezentácie neurónovými sieťami, ktoré sa dokázali na úlohu rýchlo adaptovať, avšak po istom počte epoch postupne strácali schopnosť zlepšovať sa, respektíve rýchlosť vývoja sa razantne znížila.

Zaujímavým zistením bolo to, že veľkosť populácie v pôvodnej reprezentácii nehrá v podstate žiadnu úlohu. Rôzne veľké populácie dosahovali porovnateľné výsledky. Naproti tomu populácia v našej metóde zohrávala väčšiu úlohu, pričom väčšie populácie dosahovali spravidla lepšie výsledky.

V druhom experimente, na ktorý sme aplikovali našu metódu, dosiahla pôvodná reprezentácia neurónovými sieťami väčšinou lepšie priemerné riešenie ako nami vybraná reprezentácia stavovými automatmi. Napriek tomu stavové automaty dosiahli v nadpolovičnej väčšine testov lepšie maximálne hodnoty fitness. To nás vedie k presvedčeniu, že aj pri tejto úlohe je reprezentácia stavovými automatmi vhodná, ale vyžaduje si viac testov s rôznymi konfiguráciami na dosiahnutie maximálnej kvality.

Stavové automaty ako reprezentácia jedincov evolvovaných metódou NEAT sa v týchto úlohách ukázali ako výborná konkurencia pre neurónové siete, ba v niektorých prípadoch ich aj porazili. Tento poznatok sa teraz ukazuje ako dôležitý míľnik metódy NEAT a stavových automatov, ktorý si zasluhuje viac pozornosti, než mu bolo dosiaľ venované. Vhodným problémom na testovanie tejto metódy by mohol byť napríklad problém vyrovnávania rovnováhy prevráteného kyvadla.

Zaujímavý efekt by mohlo mať aj prenastavenie funkcie počítania podobnosti jedincov na stavové automaty s menším počtom počiatočných stavov. Ak by sa zvýšila benevolentnosť tejto funkcie, je možné, že by aj tieto automaty dosahovali dobré výsledky.

Zoznam použitej literatúry

[1] **BABJAK, J. 2003.** *Neurónové siete sú čiernou skrinkou.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.root.cz/clanky/neuronove-siete-su-ciernou-skrinkou>.

[2] **BROOKS, R.A. 1986.** A Robust Layered Control System For A Mobile Robot. [online]. *In IEEE Journal of Robotics and Automation*. 1986, vol. 2, no. 1, s. 14–23. [cit. 2011-05-01]. Dostupné na internete: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1087032&tag=1>. ISSN 0882–4967.

[3] **BUNDZEL, M. 1999.** *Neurónové siete s adoptívnou topológiou*. [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.ai-cit.sk/source/publications/thesis/phd_thesis_prospectuses/1999/bundzel/html/node13.html>.

[4] **CHELLAPILLA, K – CZARNECKI, D. 1999.** A preliminary investigation into evolving modular finite state machines. [online]. In *Evolutionary Computation*. 1999, vol. 2, s 1349–1356. [cit. 2011-05-01]. Dostupné na internete: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=782607>. ISBN 0–7803–5536–9.

[5] **CLELLAND, CH.H. – NEWLANDS, D.A. 1994.** PFSA Modelling of Behaviour Sequences by Evolutionary Programming. [online]. In *Complex Systems: Mechanism for Adaptation*. IOS Press, s 165–172. [cit. 2011-05-01]. Dostupné na internete: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.2119>.

[6] **GAVALIER, M. 2000.** *Topológia sietí.* [online]. [cit. 2011-05-01]. Dostupné na internete:<http://www.ai-cit.sk/source/publications/thesis/master_thesis/2000/gavalier/html/node25.html>.

[7] **HOCKENMAIER, J. 2008.** *Finite state Transducers and Morphology.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.cs.uiuc.edu/class/fa08/cs498jh/Slides/Lecture2.pdf>.

[8] **HOLÍK, L. 2007.** *Abstraktní regulární model checking a separační logika jako vybrané metody verifikace pointerových programů.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.fit.vutbr.cz/study/courses/TJD/public/0607TJD-Holik.pdf>.

[9] **HOPCROFT, J.E. – ULLMAN, J.D. 1978.** *Formálne jazyky a automaty.* Bratislava : Alfa, 1978. 343 s.

[10] **HUDÁKOVÁ, I. 2007.** *Zložitostné aspekty konečných automatov* : bakalárska práca. [online]. Bratislava : Katedra informatiky, Fakulta matematiky, fyziky a informatiky UK Bratislava, 2007. 31 s. [cit. 2011-05-01]. Dostupné na internete: <http://www.dcs.fmph.uniba.sk/bakalarky/obhajene/getfile.php/Zlozitost_KA.pdf?id=9&fid=15&type=application%2Fpdf>.

[11] **KOCUR, P. 2000.** *Úvod do teorie konečných automatů a formálních jazyků*. [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.kvd.zcu.cz/cz/materialy/kafj/_kafj1.html>.

[12] **KOTRÍK, R. 2011.** *Reinforcement learning.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://ii.fmph.uniba.sk/~farkas/Courses/Neurocomp/Presentations/kotrik.rl.pdf>.

[13] **KOVÁČ, V. 2003.** *Genetické programovanie*. [online]. [cit. 2011-04-25]. Dostupné na internete: <http://neuron.tuke.sk/~kovacv/minimovka/html/node13.html>.

[14] **MACH, M. 2009.** *Evolučné algoritmy – prvky a princípy.* [online]. Košice : Elfa, s.r.o., 2009. 250 s. [cit. 2011-05-01]. Dostupné na internete: <http://neuron-ai.tuke.sk/~machm/publications/kniha-eapp.pdf>. ISBN 978–80–8086–123–0.

[15] **MAKULA, M. 2009.** *Sila a obmedzenia rekurentných neurónových sietí pri spracovaní postupností symbolov* : dizertačná práca. [online]. Bratislava : Ústav aplikovanej informatiky, Fakulta informatiky a informačných technológií, STU Bratislava, 2009. 133 s. Dostupné na internete: < http://ii.fmph.uniba.sk/~benus/theses/Matej_thesis.pdf>.

[16] **MRÁZ, F. 2007.** *Evolučná robotika*. [online]. 2007. [cit. 2011-04-25]. Dostupné na internete: <http://ksvi.mff.cuni.cz/~mraz/EvoRob/index.html>.

[17] **PETROVIČ, P. 2008.** Evolving Behaviour Coordination for Mobile Robots using Distributed Finite-State Automata. [online]. In *Frontiers in Evolutionary Robotics*. 2008, s 413 –438. [cit. 2011-05-01]. Dostupné na internete: <http://www.intechopen.com/source/pdfs/865/InTech-Evolving_behavior_coordination_for_mobile_robots_using_distributed_finite_state_automata.pdf>. ISBN 978–3–902613–19–6.

[18] **ROCHE, E. 1995.** Smaller representations for finite-state transducers and finite-state automata. [online]. In Computer Science. 1995, vol. 937, s 352–365. Dostupné na internete: <http://www.springerlink.com/content/pl836v20r7401024/>.

[19] **SCHWEFEL, H.P – MÄNNER, R. 1991.** *Parallel Problem Solving from Nature*. Berlin : Springer, 1991. 485 s. ISBN 3–540–54148–9.

[20] **SEJNOWSKI, T.J. 1999.** *Unsupervised learning.* [online]. US : Massachutsetts Institute of Technology, 1999. 398 s. [cit. 2011-05-01]. Dostupné na internete: <http://books.google.com/books?id=yj04Y0lje4cC&dq=unsupervised+learning&hl=sk&source=gbs_navlinks_s>.

[21] **SEKAJ, I. 2009.** Evolúcia v počítači alebo evolučné a genetické algoritmy. [online]. In *Posterus*. 2009, vol. 2, no. 10. [cit. 2011-05-01]. Dostupné na internete: <http://www.posterus.sk/?p=3364>. ISSN 1338–0087.

[22] **SIČÁK, P. – ANDREJKOVÁ, G. 2002.** *Neurónové siete – inžiniersky prístup.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://ics.upjs.sk/~novotnyr/home/skola/neuronove_siete/nn_sincak_andrejkova/neuronky2.pdf>.

[23] **SIEGELMANN, H.T. – SONTAG, E.D. 1992.** Some Recent Results on Computing With „Neural Nets“. [online]. In *Decision and Control*. 1992, no. 31, s 1476–1481. [cit. 2011-04-29]. Dostupné na internete: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=371169&tag=1>. ISBN 0–7803–0872–7.

[24] **STANLEY, K.O – MIIKKULAINEN, R. 2002.** Evolving Neural Networks Through Augmenting Topologies. [online]. In *Evolutionary Computation*. 2002, vol. 10, no. 2, s 99–127. [cit. 2011-04-25]. Dostupné na internete: <http://nn.cs.utexas.edu/?stanley:ec02>. ISSN 1063–6560.

[25] **ŠUBA, S. 2006.** *Systémy a ich formálne špecifikácie.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://webcache.googleusercontent.com/search?q=cache:94NIKh7HMwUJ:hornad.fei.tuke.sk/~suba/LPI/referat_TMII.doc+mealyho+vs+moorov&cd=9&hl=sk&ct=clnk&gl=sk&client=firefox-a&source=www.google.sk>.

[26] **TUHÁRSKY, J. – REIFF, T. – SINČÁK, P. 2009.** *Evolutionary Approach for Structural and Paramentric Adaptation of BP-like Multilayer Neural Networks.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://www.bmf.hu/conferences/cinti2009/2_cinti2009_submission.pdf>.

[27] **VALPOLA, H. 2000.** *Supervised vs. unsupervised learning.* [online]. [cit. 2011-05-01]. Dostupné na internete: <http://users.ics.tkk.fi/harri/thesis/valpola_thesis/node34.html>.

[28] **WERBOS, P.J. 1990.** Backpropagation Through Time: What It Does and How to Do It. [online]. In Proceesings of the IEEE, vol. 78, no. 10, s 1550–1560. Dostupné na internete: < http://www.sysc.pdx.edu/classes/Werbos-Backpropagation%20through%20time.pdf>.

[29] **ZELINKA, I. et al. 2009.** *Evoluční výpočetní techniky – principy a aplikace*. Praha : BEN – technická literatura, 2009. 536 s. ISBN 978–80–7300–218–3.

[30] **WIKIMEDIA COMMONS. 2006.** *Neuron sk.* [online]. [cit. 2011-05-02]. Dostupné na internete: <http://commons.wikimedia.org/wiki/File:Neuron_sk.jpg>.

[31] **WIKIMEDIA COMMONS. 2006.** DFAexample. [online]. [cit. 2011-05-02]. Dostupné na internete: <http://commons.wikimedia.org/wiki/File:DFAexample.svg>.

[32] **MIIKKULAINEN, R. 2010.** Neuroevolution. [online]. In *Encyclopedia of Machine Learning*, s 716–720. [cit. 2011-05-01]. Dostupné na internete:<http://www.springerlink.com/content/k8404476m7860060>. ISBN 978-0-387-30768-8.