Evolúcia stavových automatov inšpirovaná metódou NEAT

Pavel Petrovič

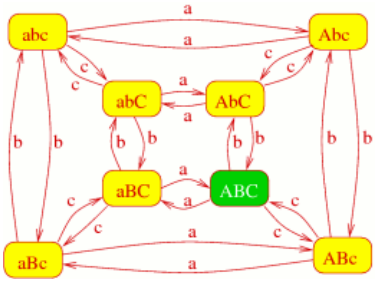
Katedra aplikovanej informatiky, Fakulta matematiky, fyziky a informatiky, Univerzita Komenského
Mlynská dolina, 842 48 Bratislava
ppetrovic@acm.org

Abstrakt

Konečno-stavový automat je formalizmus, ktorý je schopný do veľkej miery modelovať činnosť robotického systému. Ba, činnosť robotického systému možno pomocou stavových automatov priamo riadiť: riadiaci systém robota môže tvoriť jeden alebo viacero stavových automatov, presnejšie transducerov, kde prechody medzi stavmi generujú akcie. Bolo ukázané, že stavové automaty ako riadiaci systém mobilného robota možno generovať automaticky využitím evolučných algoritmov. Stavové automaty sa svojou topológiou podobajú na úplne prepojené rekurentné neurónové siete, pričom metóda NEAT je jedna z najúspešnejších evolučných metód na ich návrh vrátane topológie. Tento príspevok sa zaoberá adaptovaním myšlienok NEAT na evolúciu stavových automatov.

1 Úvod

Štandardný formalizmus konečno-stavových automatov (finite-state automata – FSA, alebo finite-state machines – FSM) je dobre známy. Formálna definícia [19], považuje automat za päticu (S,Σ,δ,q_0,T), kde S je konečná množina stavov, Σ je abeceda, t.j. konečná množina vstupných symbolov, δ je zobrazenie z $S\times\Sigma$ do S, ktoré dvojici aktuálny stav a vstupný symbol priradí nový stav, $q_0 \in S$ je počiatočný stav, a $T \subset S$ je množina koncových stavov. Automat akceptuje slovo $x=x_0x_1...x_N$, zložené zo symbolov $x_i \in \Sigma$, ak $\delta(...\delta(\delta(q_0,x_0),x_1),...,x_N)=p$ pre nejaký stav $p \in T$, čiže postupnosť stavových prechodov generovaná postupne symbolmi vstupného slova x vedie z počiatočného stavu do nejakého koncového stavu. Takto zadefinovaný konečný automat rozpoznáva regulárne jazyky (t.j. akceptuje slová z jazyka generovateľného regulárnou gramatikou), má rovnakú vyjadrovaciu silu ako regulárne gramatiky. Napríklad, konečno-stavový automat môže rozpoznať jazyk nad abecedou $\{a,b,c\}$, ktorý obsahuje nepárny počet symbolov a, nepárny počet symbolov b a nepárny počet symbolov c, príklad takého automatu je na obrázku 1. V prípade, že slovo patrí do tohto jazyka, výpočet automatu sa v konečnom čase zastaví. Tento automat je formálne definovaný ako $(\{abc,Abc,ABc,aBc,abC,AbC,ABC,aBC\},\{a,b,c\},\delta,abc,\{ABC\})$, kde $\delta(s_1,t)=s_2$, ak na obr. 1 je šípka zo stavu s_1 do stavu s_2 označená symbolom t. Naopak, neexistuje konečno-stavový automat, ktorý rozpoznáva jazyk pozostávajúci s rovnakého počtu symbolov a, b, c. V tejto podobe teda nie je ekvivalentný triede jazykov rozpoznávaných Turingovým strojom a nedokáže ani vypočítať ľubovoľnú vypočítateľnú funkciu. Ak automatu dovolíme produkovať výstup, t.j. prekladať vstupné slová na výstupné slová, dostávame konečno-stavový prekladač (finite-state transducer), ktorého každý stavový prechod je označený symbolom, ktorý ho vyvoláva a symbolom, ktorý je vyslaný na výstup pri využití stavového prechodeu. Konečno-stavové automaty boli študované v mnohých teoretických prácach, od klasickej publikácie [14] po súčasné práce teoretickej informatiky, napr. [5].



Obr. 1. Príklad konečno-stavového automatu.

Z hľadiska inteligentného agenta, ktorý vykonáva akcie v prostredí, vstupné symboly v tomto formalizme môžu zodpovedať vnemom agenta a výstupné symboly zodpovedajú akciám. Takýto systém môže produkovať reaktívne správanie, avšak obsahuje aj vnútorný stav, takže nejde nutne o výlučne reaktívneho agenta (pure reactive). Formálnu výpočtovú silu stavového formalizmu môžeme zväčšiť, ak zavedieme množinu registrov/premenných, ktoré môžu uchovávať dynamickú stavovú informáciu, a ktoré možno testovať a meniť v podmienkach/akciach stavových prechodov. V prípade,

že akcie reprezentujú volania ľubovoľných procedúr a každému stavu priradíme ďalšiu osobitnú procedúru, ktorá sa štartuje resp. vykonáva, keď sa systém dostane resp. zotrváva v príslušnom stave, získame silný a univerzálny formalizmus. V taketo podobe môže byť stavový formalizmus vnímaný ako užitočný nástroj na popísanie procesov a scenárov ako množiny stavov, kde prechody medzi stavmi zodpovedajú dynamike procesov. Stav zodpovedá kontextu, alebo referenčnému rámcu: každý stav definuje dôležité interakcie, ktoré v danom kontexte môžu nastať a ako na ne systém reaguje.

Každý robot, ktorý vykonáva nejakú činnosť zotrváva v nejakom stave, kým aktívne odpovedá na vnemy z prostredia okamžitými akciami, alebo zmenou stavu. Aktivitu robotického (alebo aj iného) agenta takto možno verne modelovať pomocou stavového diagramu. Tento model môže robotický agent priamo interpretovať a považovať za svoj program, ktorý tak získa prirodzenú štruktúru. Inými slovami, program pre mobilného robota, ktorý pôsobí v dynamickom, nedeterministickom a nepredvídateľnom prostredí je možné formulovať ako rozšírený konečno-stavový automat. Konečno-stavové automaty boli v minulosti navrhnuté ako formalizmus na implementáciu správaní robota aj priamo v Brooksovej Subsumpčnej architektúre [6], sú využité v Rosencheinovej práci o situovaných automatoch [21] a sú využívané v rôznych robotických modeloch od koordinácie pohybu nôh kráčajúcich robotov po zložité problémy riadenia.

Našou hypotézou je, že riadiace systémy mobilných robotov, ktoré sú založené na formalizme konečno-stavových automatov tiež môžu podliehať adaptácii a ich štruktúra, topológia a samotná stavová prechodová funkcia môže byť predmetom učenia. Medzi bežne používané metódy adaptácie patria i evolučné výpočty a v tejto práci sa zaoberáme problémom automatického návrhu konečno-stavových automatov pomocou evolučných výpočtov. V nasledujúcich statiach postupne priblížime predchádzajúce práce, kde stavové formalizmy boli použité ako reprezentácia genotypu v evolučných výpočtoch, spomenieme porovnanie dvoch reprezentácií: konečno-stavových automatov a stromových programových štruktúr evolvovaných pomocou Genetického Programovania, uvedieme príklad práce na ktorú nadväzujeme, kde konečno-stavové automaty boli využité na koordináciu správaní riadiacej architektúry mobilného robota, stručne zopakujeme princípy úspešnej metódy NEAT na evolúciu rekurentných neurónových sietí, navrhneme jej modifikáciu pre stavové automaty a na záver zrhnieme základný prínos práce a načrtneme ďalšie smerovanie.

2 Evolúcia stavových automatov v doterajších prácach

Konečno-stavové automaty boli využité ako reprezentácia genotypu v rozličných prácach, hoci táto reprezentácia nepatrí k hlavnému prúdu v rámci oblasti výskumu a aplikácií evolučných výpočtov.

Evolučné programovanie, [9 – 12] ako osobitný prístup v oblasti evolučných výpočtov pôvodne využívalo pre reprezentáciu genotypu konečno-stavové automaty, ale v ďalšom vývoji sa odklonilo k vektorom reálnych čísel. EP pôvodne nevyužívalo operátory kríženia a spoliehalo sa na operátory mutácie.

Chellapilla a Czarnecki v [7] skúmajú modulárne FSM, ktoré sú vyjadrovacou silou ekvivalentné nemodulárnym FSM, ale ich topológia je obmedzená. FSM je rozdelený na niekoľko vnorených podautomatov, ktoré sa aktivujú len prostredníctvom ich začiatočných stavov. Autori v práci využívajú modulárne FSM na reprezentáciu evolvovaných programov pre známy problém mravca, ktorý bol predtým úspešne vyriešený pomocou FSA reprezentovaných ako bitové reťazce [17]. Autori dokumentujú, že 1) modulárne FSM dosahujú v tejto úlohe lepšie výsledky ako nemodulárne FSM a 2) priama reprezentácia FSM so štrukturálnymi operáciami (mutácia) je výhodnejšia ako reprezentácia bitovými reťazcami použitými v práci [17]. Myšlienku modulárnych FSM prevzali Acras a Vergilio v práci [1], kde je navrhnutý všeobecný rámec pre modulárne evolučné programovanie. Rámec je prezentovaný na dvoch konkrétnych príkladoch.

Angeline a Pollack [2] experimentujú s emergenciou modulárnych FSA. Navrhujú počas evolúcie dočasne fixovať (zmraziť) časti FSA a tým zabrániť, aby evolučné operátory mohli tieto časti meniť. Zmrazenie častí prebieha náhodne, pričom autori predpokladajú, že evolučné procesy prirodzeného výberu uprednostnia tie jedince, v ktorých sú zafixované užitočné moduly. Opäť na probléme mravca autori dosahujú lepšie výsledky v behoch s fixáciou častí FSA, ako v bežných behoch. Uvažujú takto: ``Tieto výsledky si vysvetľujeme tak, že proces fixácie nachádza a ochraňuje tie časti, ktoré sú dôležité pre životaschopnosť potomka generovaného evolučnými operátormi. Nové mutácie môžu meniť iba tie časti, ktoré nie sú kľúčovo nevyhnutné.

Lucas v [18] evolvuje konečno-stavové transducery (FSTs), čiže automaty, ktoré produkujú výstupy, resp. mapujú reťazce vo vstupnej doméne s reťazcami vo výstupnej doméne. Autorovi úspešne naevolvoval FST, ktoré transformujú reťazové obrazové kódy (image chain codes), pričom porovnával využitie troch rôznych mier v

účelovej funkcii: presnú rovnosť, Hammingovu vzdialenosť a „editovaciu“ vzdialenosť (t.j. počet editovacích operácii nevyhnutných na dosiahnutie zhody). Hammingova vzdialenosť pochopiteľne dosahuje lepšie výsledky ako presná rovnosť. V nadväzujúcej práci [23] autori dokončujú porovnanie a ukazujú, že editovacia vzdialenosť dosahuje lepšie výsledky ako Hammingova vzdialenosť a hoci je zároveň výpočtovo najnáročnejšia, celkový prínos k efektivite algoritmu vďaka zníženiu počtu nevyhnutných vyhodnotení jedincov je kladný. Autori v práci tiež porovnávajú svoju evolučnú metódu (založenú na (1+1)-evolučnej stratégii s metódou OSTIA – deterministickým heuristickým algoritmom induktívneho učenia na trénovanie konečno-stavových transducerov (Onward subsequential transducer inference algorithm, [24]), založenom na zlučovaní stavov, ktorý v tejto triede algoritmov dosahuje najlepšie výsledky. V porovaní evolučnej metódy a OSTIA získali autori porovnateľné alebo lepšie výsledky pomocou evolučnej metódy a to zvlášť v prípadoch, keď trénovacie dáta sú zašumené – čo vedie k významnému narušeniu výkonnosti metódy OSTIA.

Zaujímavou je aj práca [13], kde autori evolvujú FSM (Mealy machine) pre riešenie niekoľkých testovacích (benchmark) funkcií, ktoré sú na rozdiel od problému mravca spojité. V tejto práci je stavová prechodová funkcia reprezentovaná ako strom (podobne ako v Genetickom programovaní) – vo vyšších uzloch stromu sú vetviace podmienky a v nižších jednoltivé stavové prechody a generované výstupy – t.j. smer prehľadávania definičného oboru optimalizovanej funkcie. Systém sa naučil rôzne úspešné stratégie prehľadávania, ktoré vykazovali prvky robustnosti.

Hsiao vo svojej dizertačnej práci [15] využíva evolúciu FSA na generovanie vstupných sekvencií pre digitálne obvody za účelom ich verifikácie a detekcie chýb. Na rôznych obvodoch dosahuje jeho metóda najlepšie výsledky, avšak nie na tých, kde si aktivácia chyby vyžaduje dlhú sekvenciu symbolov.

Horihan a Lu v [16] evolvujú FSM, ktoré akceptujú slová jazyka generovaného rôznymi regulárnymi gramatikami (tento proces nazývajú *genetická inferencia*). V práci využívajú inkrementálnu evolúciu, pričom automaty, ktoré naevolvovali pre jazyky zodpovedajúce jednoduchším gramatikám postupne trénujú na zložitejšie a zložitejšie jazyky.

Clelland a Newlands [8] využívajú Evolučné programovanie s pravdepodobnostnými FSA (PFSA) v úlohe nájdenia pravidelných závislostí vo vstupných dátach. PFSA je FSA, v ktorom sú stavové prechody ohodnotené pravdepodobnosťami výberu príslušného prechodu, zmeranými na vstupných slovách. Úlohou EP je návrh topológie FSA – t.j. stanoviť optimálny počet stavov a ich prepojenie, a nájsť stavové prechody. Takýto formalizmus môže byť využitý na rýchle pochopenie vnútornej štruktúry postupností. Cieľom je nájsť čo najjednoduchší automat popisujúci daný jazyk v zmysle miery MML (minimum message length). Na rozdiel od heuristických metód založených na inkrementálnom pridávaní reťazcov do jazyka, ktoré sú schopné nájsť len lokálne optimálne riešenie, genetická inferencia môže nájsť globálne optimálne riešenie: aby však evolúcia fungovala, bolo do abecedy potrebné pridať špeciálny symbol, ktorý sa používa v prípade, že pri spracovávaní vstupného slova automat nemôže pokračovať (can't consume symbol) – každý stav potom obsahuje slučkový stavový prechod označený týmto symbolom – účelová funkcia penalizuje automaty, ktoré tieto symboly využívajú a automaty, ktoré obsahujú nevyužité stavové prechody. Okrem toho, že PFSA sú minimálnym modelom jazyka, sú užitočné na jednoduchú predikciu nasledujúceho symbolu v spracovávanom slove.

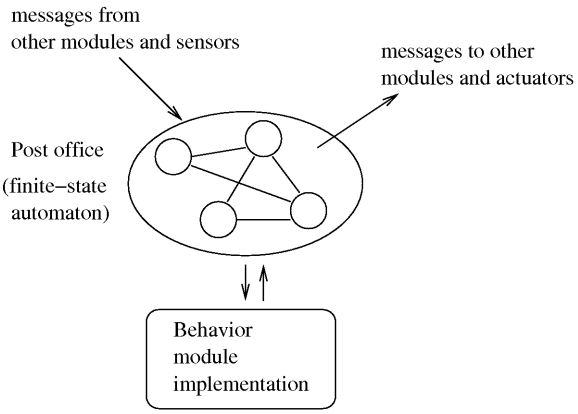
Ashlock et al. [3] evolvovali FSM na klasifikáciu reťazcov DNA primerov na správne a nesprávne v simulovanom procese replikácie DNA. Výsledkom po 600 generáciách sú automaty so 64 stavmi. Autori zistili, že ak evolvovali automaty, ktoré klasifikovali primery len svojím konečným stavom, výsledky boli slabé. Namiesto toho použili riešenie, kde automat klasifikuje v každom kroku svojho výpočtu. To zároveň mení odpoveď automatu z binárnej (áno-nie) na číselnú (kde výstup určuje ako dobre príslušný primer vyhovuje). Najlepšie naevolvované automaty mali úspešnosť 70%, tá sa však dala ďalej zlepšiť pomocou hybridizácie na 77% úspešnosť, kde sa 1/6 populácie inicializuje najlepšími automatmi z predchádzajúcich 100 evolučných behov. Na túto prácu však nadväzuje [4], kde prístup pomocou FSM je porovnaný s bežnejšou metódou IMM (Interpolated Markov Models), ktorá dosahuje lepšie výsledky ako FSM.

Inšpiratívna štúdia [22] analyzuje evolúciu navigačných stratégií v hre o súťaži o zdroje. Stratégie sú reprezentované ako FSM, agent sa pohybuje na 2D mriežke a obsadzuje voľné polia a súťaží s druhým agentom, ktorý polia obsadzuje pomocou pevnej, ale stochastickej stratégie. Hra končí, keď sú všetky polia obsadené. Agenti nemôžu opustiť vlastné teritórium. Autori zistili, že úloha je náchylná na cyklické správanie, ktoré je pre FSM typické a preto implementovali osobitné kontroly za účelom zamedzenia cyklov. Ďalej analyzujú otázky operátorov – nakoľko je výhodné stavy úplne mazať, alternatívami sú dočasná deaktivácia a neskoršia re-aktivácia, alebo zlučovanie, resp. rozdeľovanie existujúcich stavov. Podobne sa zaoberajú porovnaním

evolúcie automatov s pevným a premenlivým počtom stavov. Pri evolúcii s premenlivým počtom stavov sú stavy, ktoré pribudli neskôr počas evolučného behu väčšinou využívané menej. Zaujímavou je aj diskusia o nutnosti viacnásobných štartov ohodnocovaného automatu, vzhľadom na stochastickú podobu problému. Autori nesúhlasia s predchádzajúcimi prácami, že namiesto viacnásobných štartov stačí zväčšiť počet generácií, pretože dochádza k príliš veľkej chybe – odchýlke účelovej funkcie od skutočnej kvality automatu. Autori ponúkajú dvojvrstvové riešenie: najskôr ohodnotiť jedince na menšom počte štartov a v prípade, že jeho ohodnotenie presahuje najlepšie automaty z predchádzajúcej generácie, jedinec je otestovaný na ďalšej množine štartov.
Ďalšie práce využívajúce FSM ako reprezentáciu v evolučných algoritmoch môže čitateľ nájsť napr. v sekciách EP zborníkov konferencií GECCO a CEC.

3 Porovnanie stavových automatov a genetického programovania z hľadiska vhodností reprezentácie

Práca [20] porovnáva vlastnosti stavových automatov (transducerov) a programov vo forme stromov navrhnutých pomocou Genetického programovania na niekoľkých úlohách generovania symbolických postupností a jednoduchých úlohách 2D navigácie. Podľa výsledkov vynikajú obe reprezentácie jedna nad druhou – v závislosti na type úlohy. Vo svojich vlastnostiach sa dopĺňajú. Stavové automaty sú vhodné v úlohách, kde sa vyskytujú cyklické interakcie. Programové stromy sú vhodné v úlohách, kde treba generovať špecifickú dlhú a neopakujúcu sa postupnosť. V úlohách, kde interakcia s prostredím je nepredvídateľná a nedeterministická a kde systém prechádza cez veľa možných kombinácii stavov, počet opakovaní je nepredvídateľný a kde sa očakáva okamžitá odozva na druh podnetu lokálnou alebo podstatnou zmenou stavu sa ukázali vhodnejšie stavové automaty. Práca sa podrobnejšie zaoberá evolúciou stavových automatov z hľadiska evolvovateľnosti. Vzhľadom na to, že priestor riešení je pre stavové automaty značne nespojitý a hrbolatý (veľmi podobné automaty líšiace sa napr. len v jednom stave alebo stavovom prechode) môžu mať celkom odlišné správanie, je úloha ich automatického generovania pomocou evolučných algoritmov *ťažká*. Úlohu je preto nutné zjednodušiť, napr. poskytnutím nejakého podporného rámca. Jedným z takýchto rámcov je inkrementálna evolúcia, kde sa cieľová úloha postupne mení od ľahšej po zložitejšiu. Zjednodušenie riešenej úlohy sa môže diať pozdĺž rôznych osí: a) zjednodušenie prostredia, b) zjednodušenie požadovaného správania, c) zjednodušenie vyjadrovacej sily reprezentácie, d) zjednodušenie morfológie agenta (týka sa predovšetkým robotických úloh), e) zjednodušenie rozhrania agenta s prostredím (možnosti senzorov). Aj pri tomto zjednodušení ale môže dôjsť k tomu, že výsledná náročnosť evolučného algoritmu sa zväčší. Deje sa to vtedy, keď zjednodušením úlohy prinútime algoritmus, aby sledoval nami vopred zvolenú cestu prehľadávacím priestorom, pričom takto algoritmu zamedzíme, aby využil iné cesty postupných riešení, ktoré vedú k cieľu. Ak výhoda, ktorú zjednodušením docielime je slabšia, ako nevýhoda plynúca z vylúčenia časti prehľadávacieho priestoru (túto nevýhodu nazývame *inkrementálny bias*), celkový ikrementálny algoritmus bude výpočtovo náročnejší.

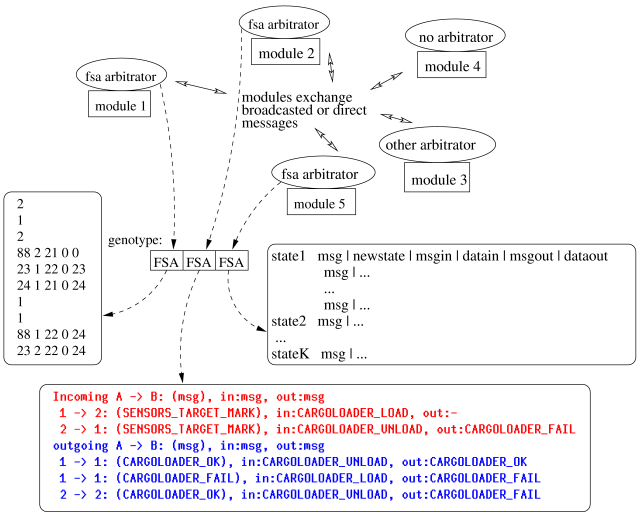


Obr. 2. Riadiaca architektúra mobilného robota – množina paralelne vykonávaných modulov (správaní) je koordinovaná množinou konečnostavových automatov (každý modul má svoj automat).

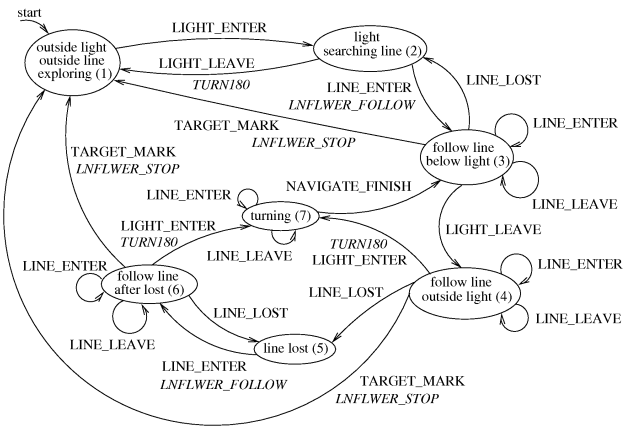
4 Príklad: evolúcia koordinácie správaní pre mobilného robota s behavior-based architektúrou

V nadväzujúcej časti práce [20] je distribuovaná sieť stavových automatov koordinujúcim mechanizmom riadiaceho programu mobilného robota s behaviorálnou architektúrou, obrázok 2. Jednotlivé správania, hotové predprogramované moduly, sú vykonávané paralelne a plnia dielčie úlohy. Celková úloha, ktorú má mobilný robot vykonávať, nie je vopred známa a podľa toho, o akú úlohu ide, sa má systém naučiť koordinovať akcie a komunikáciu medzi jednotlivými modulmi. Komunikácia medzi modulmi je založená na posielaní správ (message passing) a tie sú filtrované, resp. prekladané pomocou množiny stavových automatov (transducerov). Aj tu sa ukazuje, že úloha (v tomto prípade ide o opakované prevážanie nákladu medzi určenými miestami v prostredí s prekážkami a statickými i dynamickými prvkami) je

priťažká, pričom ikrementálny prístup rozdelenia úlohy na dielčie podúlohy algoritmu umožní nájsť dobré riešenie. Na evolúciu stavových automatov sme využili evolučný algoritmus s priamou reprezentáciou, kde jednotlivé polia genotypu s dynamickou dĺžkou zodpovedajú popisu stavov a prechodov stavovej prechodovej funkcie, obr. 3.



Obr. 3. Reprezentácia automatov v genotype: vľavo skutočná číselná reprezentácia, vpravo všeobecný význam, dolu konkrétny príklad automatu (automat rozlišuje správy prichádzajúce pre príslušný riadiaci modul (incoming) a správy, ktoré sú vysielané samotným modulom (outgoing).



Obr. 4. Príklad automatu koordinujúceho správanie jedného z modulov. Pre názornosť je vybratý ručne navrhnutý automat, automaticky evolované automaty mali podobnú štruktúru a funkčnosť, ale o niečo viac stavov a prechodov.

Algoritmus využíva mnohoraké štrukturálne operátory mutácie: vyvorenie alebo zrušenie náhodného nového prechodu, vytvorenie nového stavu s minimálnym počtom vstupných a výstupných prechodov, zrušenie náhodného stavu a všetkých incidujúcich prechodov, náhodná zmena niektorého prechodu, vytvorenie náhodného jedinca (celého automatu), vytvorenie nového stavu uprostred existujúceho prechodu, zmena počiatočného stavu. Operátor kríženia zvolí náhodné podgrafy dvoch krížených automatov a ich kombináciou vytvorí nový automat, pričom čo najviac zachová korešpodenciu prechodov z pôvodných automatov, čiže prechody vedúce do rovnakého stavu v pôvodnom automate budú viesť do rovnakého stavu aj v novom automate. Príklad automatu, ktorý riadi správanie jedného z modulov vo výslednej riadiacej architektúre je na obrázku 4.

5 Neuroevolution of Augmenting Topologies (NEAT)

Neurónové siete s dopredným šírením sú štandardným nástrojom na regresiu funkcií reálnych premenných a ich základnou vlastnosťou je, že neobsahujú vnútorný stav. Systémy, ktoré nie sú čisto reaktívne, ale vnútorný stav vyžadujú. Tento nedostatok riešia rekurentné neurónové siete, pre ktoré už ale neexistujú jednoduché metódy na trénovanie váh, čo platí hlavne pre triedy sietí s ľubovoľnou topológiou (modifikácie štandardných metód, ako napr. BPTT majú svoje ťažkosti). Preto je výhodné využiť všeobecné prehľadávacie metódy akými sú napr. evolučné algoritmy. NEAT (Neuroevolution of augmenting topologies [25]) je metóda na trénovanie rekurentných neurónových sietí s ľubovoľnou topológiou, ktorá na viacerých úlohách dosahuje najlepšie výsledky spomedzi známych metód. Základnými vlastnosťami algoritmu sú: využitie systematického operátora kríženia, ktorý zachováva topologickú štruktúru, inicializácia jedincov len minimálnymi sieťami, ktoré postupne rastú až počas behu evolúcie, rozlíšenie jedincov (sietí) na druhy za účelom zachovania funkcie naevolvovaných štruktúr. Na rozdiel od iných metód, ktoré hľadajú iba vhodné váhy pre vopred zvolenú topológiu siete, NEAT hľadá aj samotnú topológiu. Hoci značná časť výpočtovej sily musí byť použitá na nájdenie správnej topológie, užívateľ je odbremenený od často neprehľadnej analýzy problému, ktorú si vyžaduje nanuálny odhad vhodnej topológie. Naviac, NEAT, keďže začína s malými sieťami, je vhodnejší na hľadanie riešení problémov, kde sa žiada využiť postupný, inkrementálny prístup. V nasledujúcom krátko opíšeme ako NEAT dosahuje vyššie uvedené vlastnosti. Jedna z hlavných ťažkostí využitia operátora kríženia v evolučných algoritmoch nastáva v prípade, že jedno riešenie je v príslušnej reprezentácii

genotypu možné vyjadriť viacerými rôznymi, ale ekvivalentnými spôsobmi. Jav sa v literatúre označuje ako Competing Conventions Problem, Permutations Problem, alebo Synonymic Representations. Jav je typický pre neurónové siete (aj pre stavové automaty). Prejavuje sa tak, že potomok kríženia dvoch významovo podobných jedincov je s veľkou pravdepodobnosťou nefunkčný. Toto môže dramaticky sťažovať prehladávanie priestoru riešení, populácia môže ľahko konvergovať k nevhodnému variantu zápisu toho istého riešenia – napr. takému, ktorý nemá vo svojom okolí vhodných jedincov na ceste k optimálnemu riešeniu – algoritmus to nevie dopredu 'uhádnuť". Metóda NEAT sa inšpiruje genetickými procesmi v prírode, kde sa bežne môžu križiť len gény na zodpovedajúcich pozíciach v genotype (sú homologické). Preto NEAT o každom géne – čiže spojení uzlov siete eviduje jeho genetický pôvod (históriu), tzv. innovation number. V prípade kríženia tak môže identifikovať, ktoré gény majú rovnaký pôvod a nachádzajú sa v oboch rodičoch (tie vyberá náhodne z jedného alebo druhého rodiča) a ktoré sú špecifické len pre jedného z rodičov (tie vyberá len z rodiča s lepšou fitness). Gény s rovnakou genetickou históriou s veľkou pravdepodobnosťou plnia rovnakú alebo podobnú funkciu vo výpočte siete. Ďalším typickým problémom je, že prínos štrukturálnej mutácie (pridanie nejakého nového uzla alebo spojenia) sa neprejaví hneď v nasledujúcej generácii, ale na optimalizáciu spojení a váh potrebuje nová štruktúra siete niekoľko generácií, počas ktorých dosahuje nižšiu fitness ako rodičovská sieť. NEAT preto ochraňuje takéto inovácie pomocou zdieľania fitness. Podobne ako v prírode, rozdelenie organizmov na druhy umožňuje populácii preskúmať genetický priestor príslušného druhu, pričom jednotlivci príslušného druhu nesúťažia s globálnou populáciou všetkých jedincov, ale vrámci svojho druhu a globálna populácia je skôr súťaž jednotlivých druhov ako súťaž jedincov. Takto sú jedince chránené vrámci svojho druhu pred dominanciou jedného mimoriadne kvalitného jedinca a evolúcia vrámci druhu experimentuje s inovatívnymi mutáciami. NEAT rozdeľuje jedince na druhy podľa počtu zdieľaných a odlišných génov a rozdielov vo váhach spojení medzi uzlami: ak miera odlišnosti dvoch genotypov prekročí prah δ_t, sú považované za rozličné druhy:

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

kde E je počet génov jedinca s inovačným číslom väčším, ako je rozsah inovačných čísel druhého jedinca a D je počet génov jedinca s inovačným číslom, ktoré je v rozmedzí inovačných čísel jedinca, ale príslušný gén sa v druhom jedincovi nenachádza, W' je priemerný rozdiel veľkosti váh spoločných génov, c_i sú váhovacie koeficienty – parametre algoritmu. V algoritme NEAT je ochrana druhov dosiahnutá tak, že jedince vrámci toho istého druhu explicitne zdieľajú fitness. Znamená to, že ich skutočná fitness je vydelená počtom jedincov v ich druhu. Následne žiaden zvlášť dobrý jedinec nemôže obsadiť svojimi kópiami celú populáciu a spôsobiť prudký úhyn ostatných druhov. Upravená fitness jedinca je určená vzťahom:

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

kde funkcia *sh(x)* vráti 1, ak odlišnosť x je menšia ako prahová hodnota δ_t, inak vráti 0. Napokon, dôležitou vlastnosťou NEAT je, že populácia je inicializovaná na siete bez vnútorných uzlov, t.j. siete obsahujúce len vstupné a výstupné uzly. Je to dôležité pre správne zabezpečenie funkcie inovačného čísla a pre zamedzenie uvedenia náhodného kódu do populácie, ktorého prítomnosť nie je možné z hľadiska histórie odôvodniť (napr. ak by sa sieť inicializovala s niekoľkými náhodnými uzlami s rôznymi prepojeniami, ktoré nevznikli štruktúrovanou mutáciou, ktorá viedla k zlepšeniu fitness, alebo bola v rámci druhu s relatívne dobrou fitness).

6 Adoptovanie NEAT pre evolúciu stavových automatov

Myšlienky metódy NEAT považujeme za zásadné nielen pre evolúciu rekurentných neurónových sietí, ale pre evolúciu akýchkoľvek grafových štruktúr. Naším cieľom bude overiť ich prínos pre evolúciu stavových automatov a transducerov.

6.1 Topologická podobnosť neurónových sietí a stavových automatov

Topológiu ľubovoľne prepojenej neurónovej siete tvorí obyčajný orientovaný graf s hranami ohodnotenými váhami. Počas výpočtu je každý vrchol grafu ohodnotený svojim stavom – reálnym číslom, ktoré vyjadruje aktiváciu uzla. Stavové automaty majú topológiu zhodnú s neurónovými sieťami – líšia sa len druhom ohodnotenia hrán: namiesto váh pri spojeniach medzi uzlami neurónovej siete obsahuje stavový automat podmienky pri stavových prechodoch a transducer aj zodpovedajúce akcie. Líši sa aj samotný výpočet (beh) jedinca – v prípade stavového automatu je naraz aktívny iba jeden uzol, priebeh výpočtu však nemá podstatný vplyv na proces návrhu automatu.

6.2 NEAT evolúcia stavových automatov

Keďže rozdielne ohodnotenie hrán grafu nijako nediskvalifikuje ani jednu uvedenú myšlienku algoritmu NEAT, adoptujeme ho na evolúciu stavových automatov priamočiaro: globálne počítadlo hrán siete ostane zachované, uzly budú zodpovedať stavom, evolúcia bude incializovaná automatom s dvomi stavmi, každý s jedným prechodom vedúcim do druhého stavu, štrukturálna mutácia ostáva zachovaná: pridáva nový stav alebo stavový prechod, podobnosť jedincov sa určí podľa rovnakého vzorca, pričom za priemerný rozdiel ohodnotenia hrán budeme považovať škálovaný počet odlišných podmienok a akcií priradených stavovým prechodom.

7 Experiment

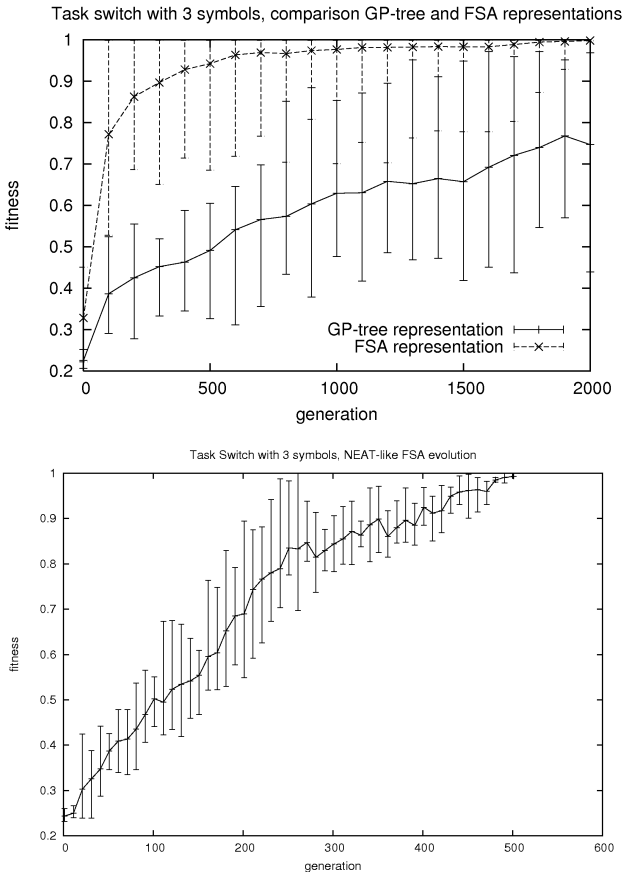
V prvom experimente na overenie efektívnosti metódy vychádzame z voľne prístupnej implementácie pôvodného algoritmu NEAT v C++ [27], ktorý sme upravili pre evolúciu stavových automatov: nahradili sme operátory inicializácie, mutácie a kríženia. Snažili sme sa zachovať čo najviac pôvodného kódu a preto naša reprezentácia bola zhodná s pôvodnou reprezentáciou pre neurónovú sieť. Stavové prechody (podmienka + akcia) boli zakódované do jedného čísla – ohodnotenia hrany medzi uzlami, ktoré zodpovedalo váhe spojenia v sieti. Mutácia potom s pravdepodobnosťami *prob_mut_action* a *prob_mut_condition* (v našom prípade 0.035) náhodne zmenila ľubovoľný stavový prechod – na rozdiel od evolúcie siete sme nepoužili gaussovskú mutáciu ale rovnomerné rozdelenie nad všetkými akciami a podmienkami. Podobne, operátor kríženia nepočítal priemer váh príslušných génov oboch rodičovských jedincov, ale len použil kópiu stavového prenosu z jedného z rodičov.

V úvodnom experimente sme evolvovali automaty (transducery), ktoré riadia stroj prepisujúci postupnosť symbolov na vstupnej páske. Táto úloha bola úspešne vyriešená v minulosti [20], pričom automaty boli evolvované klasickým evolučným algoritmom bez čŕt NEAT. Podľa našej hypotézy by evolúcia inšpirovaná NEAT mala dosahovať podobné, alebo lepšie výsledky. V tejto úlohe stroj nahrádza symboly 0 za symboly 1-3 podľa najbližšieho nenulového symbolu na páske smerom vľavo. Napríklad, automat má prepísať nasledujúce slovo na páske takto (hore vstupné, dole výstupné slovo):

2000030000000000000302001000000020000200010003002222233333333333333332221111111122222222211113333

Graf na obrázku 5 hore zobrazuje priebeh evolúcie pôvodnou metódou (v porovnaní s Genetickým programovaním, ktoré sa ukázalo byť pre túto konkrétnu úlohu menej vhodnou reprezentáciou ako stavové automaty). Krivka v grafe je priemer z 10 behov, pričom zvislé úsečky v grafe zobrazujú celkový rozsah hodnôt fitness v danej generácii. Populácia obsahovala 300 jedincov a počet ohodnotení je v našom prípade priamo úmerný číslu generácie. Veľkosť populácie bola nastavená na získanie optimálneho výsledku (pri menších populáciach sa neprehľadával dostatočne široký priestor riešení, pri väčších populáciách bola zotrvačnosť evolúcie priveľká a nové nádejné jedince mali len malú šancu presadiť sa vo veľkej populácii). V prípade metódy NEAT je populácia rozdelená na viacero druhov a preto optimálna veľkosť populácie je väčšia – v našich experimentoch sme použili populáciu s 1000 jedincami. Hoci niektoré behy našli riešenie pomerne skoro, niektoré potrebovali takmer 2000 generácii. Graf na obrázku 5 dole zobrazuje priemer z 10 behov evolúcie NEAT pre ten istý problém, avšak s ohraničenou páskou. Po 500 generáciách bolo riešenie nájdené vo všetkých 10 behoch. Príklad evolvovaného automatu je na obrázku 7. V prípade experimentov s ohraničenou páskou ide však o mierne zjednodušenú verziu problému – slovo je zacyklené a automat pri vyjdení zo slova smerom vľavo pokračuje na konci slova (a naopak). V prípade nekonečnej pásky môže automat presúvať hlavu v oboch smeroch donekonečna. Tento čiastočný výsledok nám však jednoznačne nasvedčuje, že NEAT evolúcia stavových automatov je možná a je pomerne efektívna. Naevolvovať automat pre problém s nekonečnou páskou sa nám zatiaľ nepodarilo (obrázok 6 dole), ale dôvodom je pravdepodobne to, že sme doposiaľ nenašli vhodné nastavenie parametrov pre evolúciu stavových automatov. Napríklad, nameraný počet druhov sa pohyboval na priveľmi vysokej úrovni (cez 50), automaty neboli normalizované (obsahovali viacero prechodov s rovnakou podmienkou, čo môže mať negatívny vplyv na zloženie druhov a postup evolúcie). Vo všetkých uvedených behoch sme každého jedinca testovali na 10 rôznych, náhodne vygenerovaných testovacích páskach (10 až 60 symbolov, pravdepodobnosť nulového symbolu 0.82). Keďže jedince dosahovali na rôznych páskach rôznu úspešnosť, účelová funkcia bola veľmi zašumená, čo vidieť aj na grafoch v obrázku 6. V ďalšom experimente sme skúmali, či evolúcia bude priamočiara, ak budeme počas všetkých generácií používať rovnakú sadu 10 testovacích pások. Tieto behy dosahovali výslednú fitness okolo 0.74, ale špecializovali sa na zadanú sadu a neboli rovnako univerzálne. Vyskúšali sme preto po každých 70 generáciach vygenerovať novú sadu. Tento postup zodpovedá prirodzeným zmenám prostredia po určitom čase – vždy po nejakom čase trvania evolučnej epochy, keď je prostredie stabilné a organizmy sa mu môžu dokonalo prispôsobiť, nastáva zmena podmienok. V tomto experimente sme dosahovali najlepšie výsledky, obr. 8, avšak výsledné riešenie sa ani

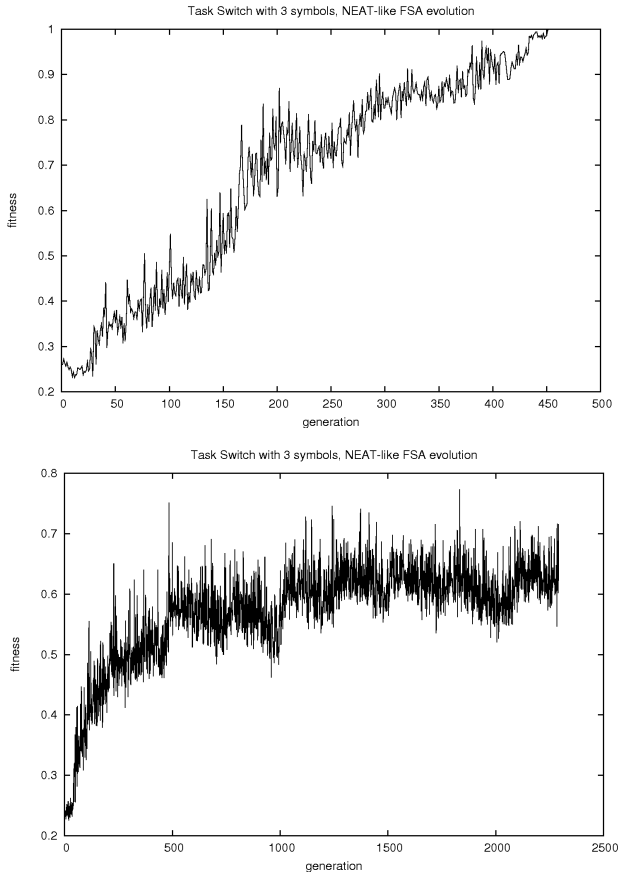
tak nepodarilo nájsť, táto úloha ostáva do ďalšieho výskumu, v ktorom tiež plánujeme reimplementáciu algoritmu tak, aby bol od základov prispôsobený pre problém evolúcie stavových automatov. Zdrojové kódy a výsledky experimentov sú k dispozícii na adrese [28].



Obr. 5. Hore: priebeh evolúcie pomocou klasického evolučného algoritmu, v grafe je zobrazený aj vývoj najlepšej fitness pri riešení rovnakého problému pomocou Genetického programovania, čo dokumentuje, že táto úloha je vhodná pre riešenie pomocou stavových automatov. Dole: priebeh evolúcie pomocou NEAT evolúcie stavových automatov.

8 Zhrnutie a námety do budúcnosti

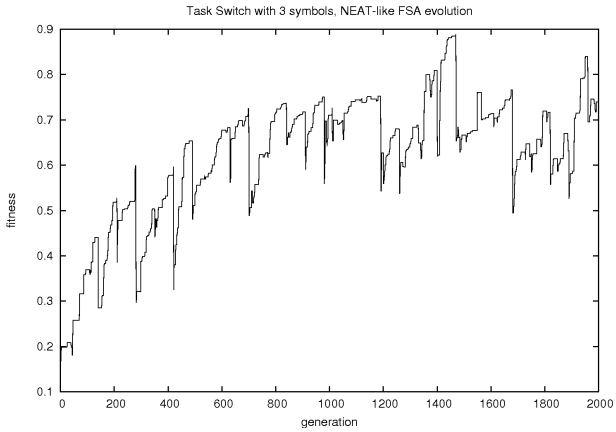
Konečno-stavové automaty a prekladače (transducery) sú užitočný formalizmus, vhodný na modelovanie i riadenie správania sa zložitých systémov. Na rozdiel od iných formalizmov sú pomerne zrozumiteľné človeku a na rozdiel od iných druhov reprezentácii (napr. neurónových sietí, kde je informácia distribuovaná v stovkách váh spojení) je správanie systému zrejmé, transparentné a vysvetliteľné. Stavové automaty sú predmetom súčasného výskumu vrátane možností ich učenia sa – či už heuristickými deterministickými metódami alebo stochastickými optimalizačnými algoritmami, akými sú evolučné algoritmy. V práci sme uviedli príklady, v ktorých evolučné algoritmy generujú optimálnejšie výsledné automaty, alebo sú tieto algoritmy efektívnejšie, jednoduchšie, alebo robustnejšie. Topologická štruktúra konečnostavových automatov a prekladačov je takmer identická so štruktúrou ľubovoľne prepojených rekurentných neurónových sietí. Medzi najlepšie známe metódy na evolúciu topológie rekurentných neurónových sieti patrí metóda NEAT. V tejto práci sme ukázali, ako je túto metódu možné adoptovať na evolúciu stavových automatov a prekladačov a na úvodnom príkade sme demonštrovali efektívnosť tejto metódy v porovnaní s klasickým evolučným algoritmom použitým na tom istom probléme v minulosti. V ďalšej práci sa zameriame na podrobnejšie experimentálne porovnanie metódy na rôznych druhoch úloh, prípadne adoptovanie neskorších rozšírení metódy NEAT [26] na problém evolúcie stavových automatov.



Obr. 6. Hore: priebeh jedného behu evolúcie pre ohraničenú pásku, dole: pre nekonečnú pásku (pozri text).

```
Node 1:
 --[6.000000]---2---Left---> 2
 --[7.000000]---3---Left---> 12
 --[5.000000]---1---Left---> 114
 --[4.000000]---0---Left---> 1675
Node 2:
 --[9.000000]---1---Write0---> 2
 --[12.000000]---0---Write1---> 12
 --[3.000000]---3---Right---> 366
 --[2.000000]---2---Right---> 451
Node 12:
 --[23.000000]---3---Write3---> 2
 --[10.000000]---2---Write0---> 12
 --[16.000000]---0---Write2---> 114
 --[1.000000]---1---Right---> 451
Node 114:
 --[2.000000]---2---Right---> 12
 --[9.000000]---1---Write0---> 2
 --[0.000000]---0---Right---> 114
 --[7.000000]---3---Left---> 366
Node 366:
 --[20.000000]---0---Write3---> 2
 --[2.000000]---2---Right---> 12
 --[3.000000]---3---Right---> 366
 --[13.000000]---1---Write1---> 114
Node 451:
 --[23.000000]---3---Write3---> 2
 --[12.000000]---0---Write1---> 12
 --[10.000000]---2---Write0---> 552
 --[1.000000]---1---Right---> 1675
Node 552:
 --[11.000000]---3---Write0---> 366
 --[2.000000]---2---Right---> 2
 --[13.000000]---1---Write1---> 12
 --[16.000000]---0---Write2---> 1675
Node 1675:
 --[12.000000]---0---Write1---> 2
 --[10.000000]---2---Write0---> 12
 --[23.000000]---3---Write3---> 1675
 --[25.000000]---1---Done---> 366
Node 3185:
 --[20.000000]---0---Write3---> 366
 --[18.000000]---2---Write2---> 451
Node 3303:
 --[22.000000]---2---Write3---> 3185
 --[27.000000]---3---Done---> 366
Node 3339:
 --[24.000000]---0---Done---> 366
 --[21.000000]---1---Write3---> 552
 --[15.000000]---3---Write1---> 12
Node 3384:
 --[2.000000]---2---Right---> 451
 --[12.000000]---0---Write1---> 114
 --[1.000000]---1---Right---> 12
```

Obr. 7. Príklad evolvovaného automatu (ohraničená, resp. zacyklená páska). V hranatých zátvorkách je uvedené ohodnotenie hrany v sieti.



Obr. 8. Priebeh jedného behu evolúcie pre nekonečnú pásku, pričom testovacia množina pások sa mení len v každej 70. generácii.

Literatúra

[1] Acras R N, Vergilio S R (2004) Splinter: A Generic Framework for Evolving Modular Finite-State Machines, in *SBIA'2004*, 356-365, Springer-Verlag.

[2] Angeline P J, Pollack J (1993) Evolutionary Module Acquisition, in *Proceedings of the Second Annual Conference on Evolutionary Programming*.

[3] Ashlock D, Wittrock A, Wen T-J (2002) Training Finite State Machines to Improve PCR Primer Design, in *Proceedings of the Congress on Evolutionary Computation CEC'02*, 13-18.

[4] Ashlock D et al (2004) A Comparison of Evolved Finite State Classifiers and Interpolated Markov Models for Improving PCR Primer Design, in *Proceedings of the Congress on Evolutionary Computation CEC'02*, 13-18.

[5] Badr A, Geffert V, Shipman I (2008) Hyper Minimizing Minimized Deterministic Finite State Automata in *RAIRO Theoretical Informatics and Applications*.

[6] Brooks R A (1987) A Hardware Retargetable Distributed Layered Architecture for Mobile Robot Control, in *Proceedings of the IEEE International Conference on Robotics and Automation*, 106-110.

[7] Chellapilla K, Czarnecki D (1999) A Preliminary Investigation into Evolving Modular Finite State Machines, in *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99, vol. 2.*

[8] Clelland C H, Newlands D A (1994) PFSA Modelling of Behavioural Sequences by Evolutionary Programming, in *Complex'94 - Second Australian Conference on Complex Systems*, IOS Press, 165-72.

[9] Fogel L J (1962) Autonomous Automata, *Industrial Research*, vol. 4, number 2, 14-19.

[10] Fogel L J (1964) *On the Organization of Intellect*, PhD thesis, UCLA.
[11] Fogel L J, Owens A J, Walsh M J (1966) *Artificial Intelligence through Simulated Evolution*, John Wiley.
[12] Fogel L J, Angeline P J, Fogel D B (1995) An Evolutionary Progamming Approach to Self-Adaptation on Finite-State Machines, in *Proceedings of the 4th Annual Conference on Evolutionary Programming*, MIT Press.
[13] Frey C, Laugering G (2001) Evolving Strategies for Global Optimization - A Finite State Machine Approach, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 27-33.
[14] Hopcroft J E, Ullman J D (1969) *Formal languages and their relation to automata*, Addison-Wesley.
[15] Hsiao M S (1997) *Sequential Circuit Test Generation using Genetic Techniques*, PhD thesis, University of Illionis at Urbana-Champaign.
[16] Horihan J, Lu Y-H (2004) Improving FSM Evolution with Progressive Fitness Functions. In *GLSVLSI04*.
[17] Jefferson D at al. (1992) Evolution as a Theme in Artificial Life: The Genesis/Tracker System, *Artificial Life II*, 10:549-578.
[18] Lucas S M (2003) Evolving Finite State Transducers: Some Initial Explorations. In *European Conference on Genetic Programming, EuroGP'2003,* 130-141.
[23] Lucas S M, Reynolds T J (2007) Learning Finite-State Transducers: Evolution Versus Heuristic State Merging, *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 3, 308-325.
[19] Moore E F (1956) Gedanken experiments on sequential machines, in: *Automata studies*, Princeton, New Jersey, Princeton University Press, 129-153.
[24] Oncina J, Garcia P, Vidal E (1993) Learning Subsequential Transducers for Pattern Recognition Interpretation Tasks, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 5.
[20] Petrovic P (2008) Evolving Behavior Coordination for Mobile Robots using Distributed Finite-State Automata, in *Frontiers in Evolutionary Robotics*, ITECH, 413-438.
[21] Rosenschein S J (1985) Formal theories of knowledge in AI and robotics, *New Generation Computing*, vol. 3, num. 4, 345-357.
[22] Spears W M, Gordon D F (2000) Evolving Finite-State Machine Strategies for Protecting Resources, in *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems*, Springer-Verlag, 166-175.
[25] Stanley K O, Miikkulainen R (2002) Evolving Neural Networks through Augmenting Topologies, *Evolutionary Computation,* vol 10, no. 2, 99-127.
[26] D'Ambrosio D B, Stanley K O (2007) A Novel Generative Encoding for Exploiting Neural Network Sensor and Output Geometry, *Proceedings of GECCO 2007*.
[27] Stanley K O (2001) NEAT Software DOC File, University of Texas at Austin, *part of the NEAT software package.*
[28] NEAT modified for FSA, results from experiments. `http://capek.ii.fmph.uniba.sk/fsa-neat/`