BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2016
http://excel.fit.vutbr.cz

Evoluční přístup k vytváření topologií neuronových sítí

Tomáš Černík*, Štěpán Karásek**

Abstrakt

Neuroevoluce – vyvíjení rekurentních sítí pomocí evolučních metod – je aktivní oblast výzkumu. Pro pouze učící metody je nutné, aby byla předem vytvořena topologie sítě. Ta však musí být navrhnuta ručně a nemusí být pro daný problém vhodná. U rekurentních sítí je tento problém daleko výraznější, neboť propojení mezi neurony může být libovolné. Algoritmus, který umožňující kromě učení také tvorbu topologie, tak může značně zefektivnit vývoj rekurentní sítě. Použití evolučních metod v kombinaci s neuronovými sítěmi však sebou přináší řadu problémů, které je potřeba řešit. Mezi základní problémy patří kódování, které musí být schopné unikátně popsat síť a zároveň umožnit efektivní křížení a mutace. Dále je nutné řešit nestejně dlouhé genomy různorodých sítí a postavit se strašáku dimenzionality řešení. Na obtížnost problému ukazují velmi rozdílné přístupy některých metod. V tomto článku jsou představeny dvě metody, které se k neuroevoluci postavily rozdílným způsobem. Přestože je v tomto oboru dosahováno stále nových pokroků, je vytváření rekurentních neuronových sítí včetně topologie stále velmi náročným problémem. Představené algoritmy však mohou nejenom ukázat možné přístupy k problému, ale také lépe osvětlit problém neuroevluce jako takové. Cílem článku je představit dvě rozdílné metody – Celulární kódování a NEAT – a na základě vlastní implementace ověřit výsledky metody NEAT.

Klíčová slova: Neuronové sítě — Evoluční algoritmy — NEAT — Cellular Encoding

Přiložené materiály: *N/A*

* xcerni08@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*
* xkaras24@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Úvod

V článku budou ukázány dvě metody neuroevoluce. První z nich – NEAT – je zástupcem klasických genetických algoritmů s přímým kódováním a lineárním genomem. Navíc jako jeden z mála algoritmů dokázal upotřebit operátor křížení. S touto metodou jsou navíc provedeny experimenty na vlastní implementaci algoritmu. Pomocí NEATu je vyřešen typický benchmarkový problém balancování tyčí – (*pole balancing*) – a to i náročnější verze bez informace o rychlostech. Uvedeny jsou výsledky experimentů i zobrazení některých získaných sítí.

Druhým zástupcem je celulární kódování v kombinaci s genetickým programováním. Tato metoda naopak využívá nepřímého kódování založeného na stromové struktuře. Nepřímé kódovaní je sice abstraktnější, ale také umožňuje vznik opakujících se motivů.

2. Neuroevoluce rozšiřujících se topologií - NEAT

Jednou z metod evoluce neuronových sítí je algoritmus NEAT (NeuroEvolution of Augmenting Topologies) [1]. K tomuto algoritmu existuje celá řada různých úprav a rozšíření, algoritmus NEAT však představuje základ ze kterého všechny vycházejí a ukazuje způsob, jak lze efektivně využít genetické algoritmy k vývoji rekurentních neuronových sítí.

2.1 Kódování

NEAT používá lineární genomy s přímým kódováním (každý gen odpovídá jednomu spojení / neuronu v

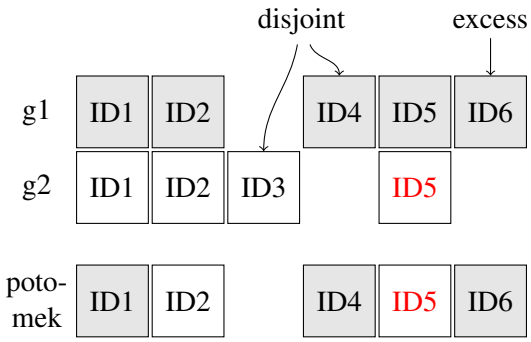
síti). Aby bylo umožněno použití operátoru křížení, musí být zvolené kódování schopno určit, který gen odpovídá kterému. Z tohoto důvodu NEAT používá ID genů (resp. ID vyjadřující pořadí jejich vzniku). Vznikajícímu novému genu je přiřazeno nové ID z globálního čítače. Při křížení jedinců se ID genů již nemění. Pomocí hodnoty ID je tedy možné zjistit původ genu.

Tato vlastnost umožňuje nejenom efektivní zarovnání genů při křížení, ale také porovnání podobnosti topologií. Toho je využíváno v dalších částech algoritmu.

Kromě ID obsahují geny i tzv. enable bit. Jedná se o hodnotu udávající, jestli je gen aktivní. Během mutací může dojít k vypnutí některého z genů. Ten se však neodstraní z genomu, ale pouze se zneaktivní.

2.2 Křížení

Díky zvolenému kódování je možné vytvořit operátor křížení, který je schopen pracovat nad nestejně dlouhými genomy a díky párování genů se stejným ID nevytváří poškozené potomky.



Obrázek 1. Křížení v algoritmu NEAT. Genom g1 má lepší hodnotu fitness. Gen ID5 je v g2 vypnutý a má proto větší šanci být zděděn.

Celá operace začne seřazením genů podle jejich ID. Poté je z každého páru genů se shodným ID vložen do potomka gen z prvního či druhého rodiče, a to se stejnou pravděpodobností. Pokud je gen v některém z rodičů vypnutý, existuje 75% šance, že bude vypnutý tento gen i v potomkovi. Geny, které nemají v druhém rodiči odpovídající pár, jsou převzaty pouze z rodiče s lepším fitness.

2.3 Mutace

Stejně jako u ostatních neuroevolučních algoritmů lze mutace v NEAT rozdělit na **strukturální** a **parametrické**. Strukturální mutace jsou přidání spojení („*add connection*“) a vložení neuronu („*add node*“). Mezi parametrické mutace patří mutace vah („*weight mutation*“) a znovuzapnutí genu („*toggle link*“).

Při strukturálních mutacích dochází k rozšiřování genomu. NEAT registruje různé strukturální mutace během jedné generace a pokud se vyskytne stejná mutace ve více genomech, nově vzniklé geny sdílí stejné ID. Cílem je omezit množství shodných genů s různými hodnotami ID.

2.3.1 Přídání spojení

Při mutaci „přidání spojení“ jsou nalezeny dva neurony, mezi kterými ještě neexistuje spojení a to je mezi ně přidáno. Spojení nemůže vést zpět do vstupních neuronů, ale může také vytvářet rekurentní vazby (to lze v případě potřeby omezit). Váha spojení je volena náhodně v rozmezí daném uživatelem. Jelikož se jedná o strukturální mutaci při které vznikne nový gen, je mu přiřazeno jeho ID podle „*innovation*“ čítače a ten je inkrementován.

2.3.2 Vložení neuronu

Mutace *vložení neuronu* nalezne již existující spojení a do něj „vloží“ nový neuron. Toho dosáhne tím, že původní spojení vypne (gen nadále zůstane v genomu) a na jeho místo vloží nový neuron. Ten je napojen na neurony z původního spojení. Aby byl vliv mutace co nejméně destruktivní, je váha spojení vedoucí do nového neuronu nastavena na 1 a váha výchozího spojení na váhu shodnou s váhou původního spojení. Nově vzniklé geny samozřejmě získají nové ID.

2.3.3 Mutace vah

Způsob provedení mutace vah se liší podle různých publikací i konkrétních implementací. K. Stanley v původním článku pouze poznamenává, že každá váha je s určitou pravděpodobností pozměněna („*perturbed*“) [2]. Ve své disertační práci [1] pak upřesňuje, že je k aktuální váze přidána reálná hodnota ze zadaného rozsahu s uniformním rozložením.

Další verzí je pak samotná implementace algoritmu NEAT K. Stanleym. V té je použita sofistikovanější verze založená na myšlence, že starší spojení jsou již z větší části optimálizována a není je tedy potřeba příliš měnit. Velikost mutací se pak volí podle stáří genu (respektive jeho ID, které je větší pro mladší geny), kdy určitá část nejmladších genů má zvýšenou šanci na výraznější změnu váhy.

2.3.4 Znovuzapnutí genu

Jedná se o velice přímočarou metodu mutace, kdy je v genomu vyhledán dříve vypnutý gen a je opět zapnut. Vypnuté geny vznikají při mutaci „přidání spojení“. V některých variantách může tato mutace vypínat i zapínat gen.

2.4 Speciation – rozdělení do druhů

Při strukturální mutaci vede změna v genomu často k momentálnímu zhoršení odezvy celé neuronové sítě a snížení fitness jedince. Zároveň jedince, jejichž genom je menší, je snazší optimalizovat a dosahují dříve vyšší fitness (méně hodnot k optimalizaci). Aby nebyly změny ztraceny při procesu selekce, jsou v NEATu jedinci rozděleni do několika druhů („*species*“) na základě jejich podobnosti. Evoluce probíhá v každém z druhů zvlášť, což umožňuje souběžnou evoluci genomů s rozdílnými topologiemi. Pokud je některý z jedinců mutací a křížením příliš pozměněn a příliš se liší od ostatních existujících druhů, je pro něj vytvořen nový druh. V kontextu evolučních algoritmů se jedná o *multimodální optimalizaci*.

Metriku pro měření rozdílnosti genomů poskytuje samo kódování NEATu. Díky faktu, že geny obsahují své ID, lze určit, které geny jsou shodné a které rozdílné. NEAT využívá tři faktory pro zjištění rozdílnosti genomů. Prvními dvěmi faktory je počet rozdílných genů. Ty se rozdělují na tzv. „*disjoint*“ a „*excess*“ geny. „*Excess*“ geny jsou všechny geny jednoho z jedinců, které mají větší ID, než je největší ID v druhém genomu. Všechny ostatní geny bez páru v druhém jedinci jsou označovány jako „*disjoint*“ geny. Třetím faktorem je rozdílnost vah mezi jedinci.

NEAT měří příslušnost genomu do druhu pomocí hranice kompatibility δ_t („*compatibility treshold*“) a vzdálenosti dvou jedinců δ. Jedná se o lineární kombinaci výše zmíněných faktorů, kterou lze zapsat jako:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W} \tag{1}$$

Kde N je délka většího z genomů použitá k normalizaci, E je počet „*excess*“ genů, D je počet „*disjoint*“ genů a $\overline{W}$ je průměrný rozdíl mezi váhami shodných genů spojení, které se vyskytují v obou genomech. Dále jsou použity tři koeficienty, c_1, c_2 a c_3, kterými je možné měnit vliv jednotlivých faktorů. Existují i implementace algoritmu, které používají N na hodnotě 1.

Využití koeficientů lze demonstrovat na příkladě řešení úloh citlivých na změnu výstupu (např. „double pole balance“). Může být výhodnější zvýšit koeficient c_3, čímž se začnou druhy více odlišovat na základě rozdílných vah. Při křížení se pak setkají jedinci s podobnými váhami, což umožní preciznější evoluci vah.

Pro určení, jestli je jedinec kompatibilní s druhem lze využít výpočetně náročnější přístup, kdy je použita průměrná vzdálenost jedince od všech ostatních jedinců v druhu. Druhou možností je jednodušší přístup, kdy je měřena vzdálenost pouze od jednoho jedince – reprezentanta druhu.

2.4.1 Multimodalita a fitness sharing

Pro dosažení vyrovnaných subpopulací – druhů, je v NEATu využit přístup z *multimodální optimalizace* – nichingu. Ten používá upravenou fitness funkci pro multimodální úlohy „*explicit fitness sharing*“. Původní fitness jedince je značena f_i a upravená fitness f_i'.

$$f_i' = \frac{f_i}{\sum_{j=1}^{n} sh(\delta(i,j))} \tag{2}$$

Funkce sdílení $sh()$ je rovna 0 pokud platí $\delta(i,j) > \delta_t$, jinak je 1. V důsledku to znamená, že fitness jedince je podělena množstvím jedinců v jeho druhu. Každému z druhů je pak vypočítána jeho celková fitness na základě sumy fitness všech jedinců v daném druhu. Na jejím základě je určeno, kolik potomků z kterého druhu vznikne. Pomocí sdílení fitness jedinců v rámci druhu je zaručeno, že celková fitness druhu, a tím pádem i množství potomků, je závislá na kvalitě jedinců a nikoliv na jejich množství.

2.5 Nová generace

Na začátku každé nové generace dojde nejdříve k **ohodnocení nových potomků** a jejich **přidělení do druhů** („*speciation*“). K udržení druhů slouží jejich seřazený seznam (některé implementace řadí druhy podle fitness, jiné pouze udržují pořadí vzniku). Jednotliví potomci jsou postupně testováni na příslušnost do druhů. Jedinec je přiřazen do prvního kompatibilního druhu (viz kap. 2.4).

Před vytvořením nových potomků dojde k **selekci**. Ta probíhá v rámci každého druhu zvlášť. Použita je elitistická selekce. **Počet potomků** z rodičů jednoho druhu je vypočítán podle poměru fitness druhu vůči celkové fitness všech druhů dohromady:

$$N_{off_i} = \frac{f_{spcs_i}}{\sum_{j=1}^{n} f_{spcs_j}} \cdot PopSize \tag{3}$$

Kde f_{spcs_i} je fitness druhu a $PopSize$ je velikost celé populace. Nová generace je tvořena pouze potomky. Před rozřazením potomků do druhů jsou z nich nejdříve odstraněni rodiče.

2.6 Vlastnosti

NEAT je komplexní systém pro neuroevoluci. Za použití přímého kódování a identifikace genů byl, narozdíl od jiných algoritmů, schopný použít operátor křížení. Taktéž byl úspěšně použit pro řešení „double pole balance“ problému a to s i bez informace o rychlostech.

Na druhou stranu NEAT obsahuje množství různých parametrů, které potřebují být optimálně nastaveny.

Taktéž použití přímého kódování limituje algoritmus pouze na menší sítě, neboť musí optimalizovat každé spojení zvlášť. Řešení omezení přímého kódování dalo vzniknout nové metodě využívající nepřímého kódování v kombinaci s algoritmem NEAT – HyperNEAT.

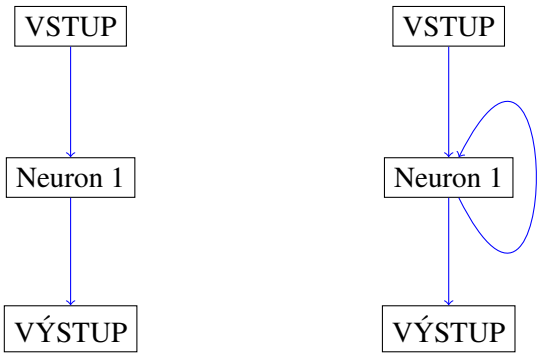
3. Celulární kódování

Celulární kódování je metoda, která se využívá pro kódování neuronových sítí. Tato technika byla vyvinuta v roce 1994 [3] a využívá nepřímé kódování. Algoritmus představují buňky, které interpretují tzv. „celulární kód“ a ve chvíli kdy dosáhnou konce kódu, tak se přeměňují v neurony. Celulární kód lze kódovat jako strom, jak lze vidět na levé straně obrázku 3. Každá buňka interpretuje instrukce dle její pozice v kódu, a ve chvíli, kdy buňka dosáhne konce programu „zanikne“ a přemění se na neuron.

Každá buňka obsahuje několik registrů. Některé registry jsou využívány v průběhu vyvíjení a některé pro vytváření spojení a nastavení aktivačních funkcí neuronů.

Konkrétní celulární kódování je definováno počátečním grafem buněk a množinou instrukcí.

Existují dva důležité grafy - cyklický a acyklický. Rozdíl mezi nimi můžeme vidět na obrázku 2.



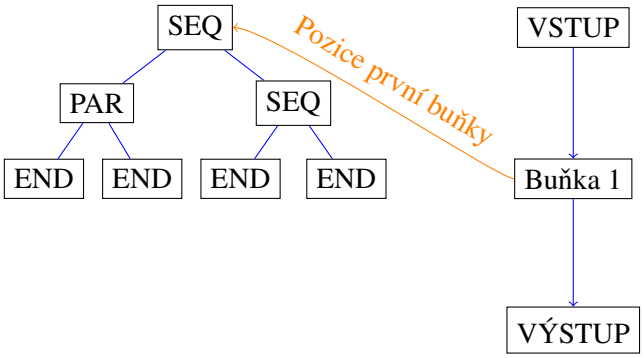
Obrázek 2. Počáteční grafy buněk. Na levé straně je znázorněn acyklický, na pravé cyklický graf.

Druhou vlastností celulárního kódování je množina instrukcí, ze kterých lze skládat celulární kód. Z článku vyplývá, že dostatečná i úplná instrukční sada je následující: { *CYC* , *END* , *PAR* , *SEQ* , *CUT* , *INCLR* , *ON* , *OFF* , *MRG* , *WAIT* , *VAL* }. Pomocí těchto instrukcí lze vytvořit libovolnou neuronovou síť. První instrukcí je definován počáteční strom - cyklický (obrázek 2).

- **END** - značí konec programu
- **PAR** - jedná se o paralelní rozdělení buňky
- **SEQ** - jedná se o sekvenční rozdělení buňky
- **CUT** - odstraní spojení dvou neuronů
- **INCLR** - inkrementuje registr
- **ON** - zapne spojení určené registrem
- **OFF** - vypne spojení určené registrem
- **MRG** - spojení dvou buněk do jedné
- **WAIT** - buňka přeskočí krok
- **VAL** - zjistí hodnotu spojení a uloží ji do registru

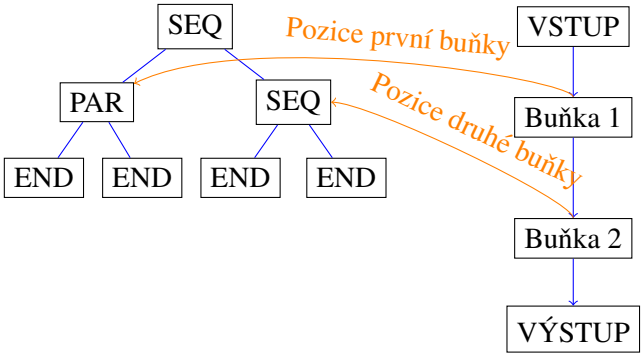
3.1 Ukázka algoritmu

V následující části je ukázán běh algoritmu na jednoduchém **celulárním kódu**. Algoritmus začíná s jednou buňkou na začátku kódu, jak ukazuje obrázek 3.



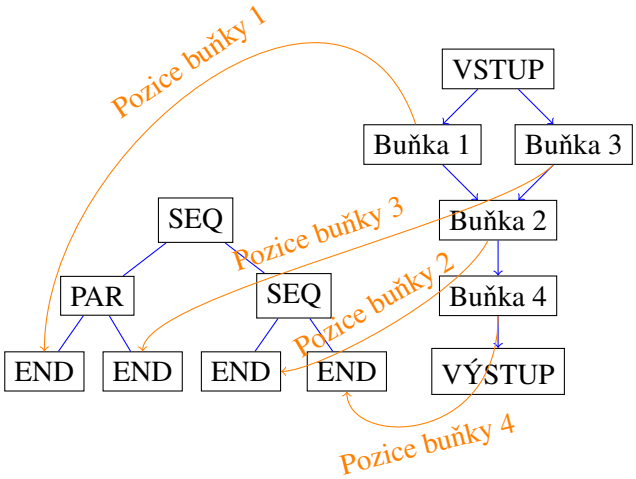
Obrázek 3. Začátek algoritmu

Buňka provede instrukci, v tomto případě **SEQ**. Tato instrukce zapříčiní rozdělení buňky na dvě, kdy se nová buňka připojí na výstup buňky první. Buňky potom nastaví ukazatele na další instrukce obsažené ve stromě, jak je ukázáno na obrázku 4.



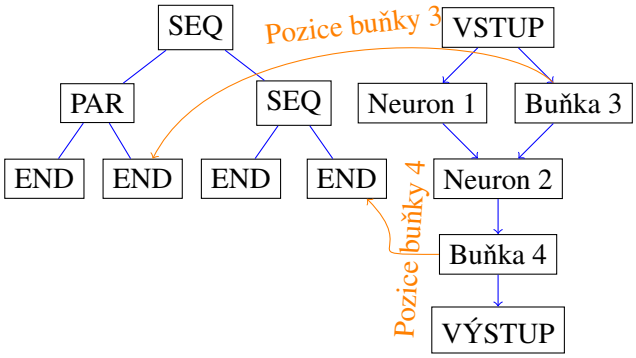
Obrázek 4. Stav algoritmu po prvním kroku

V dalším kroku algoritmu všechny buňky opět provedou instrukci. Nyní buňka 1 ukazuje na instrukci **PAR** a nově vytvořená buňka 2 ukazuje na instrukci **SEQ**. Instrukce **SEQ** již byla popsána v předešlém kroku, nyní dojde ke stejné akci. Instrukce **PAR** je také dělící instrukcí, ale tvoří novou buňku, která má stejné vstupy i výstupy - jedná se tedy o paralelní dělení. Výsledek tohoto dělení můžeme vidět na obrázku 5



Obrázek 5. Druhý krok algoritmu

V posledním kroku algoritmu vidíme, jak se buňky mění na neurony, což je způsobeno instrukcí **END**. Tuto přeměnu je možné vidět na obrázku 6, který ukazuje výslednou neuronovou sít pro tento celulární kód a počáteční graf.



Obrázek 6. „Umírající“ buňky a vznikající neurony v posledním kroku algoritmu

3.2 Genetické programování vhodné pro celulární kód

V článku jsme se zabývali vytvářením celulárního kódu pomocí genetického programování. Toto programování patří mezi evoluční algoritmy a je podobné biologické evoluci pro vytváření počítačových programů.

V článku je taktéž zmíněno, že je vhodné dynamicky měnit velikost populace, aby nenarůstal čas potřebný ke zhodnocení jejich fitness. Jak se populace průběžně vyvíjí, genetická informace jedinců se zvětšuje a z toho plyne, že narůstá i doba pro vytvoření neuronové sítě.

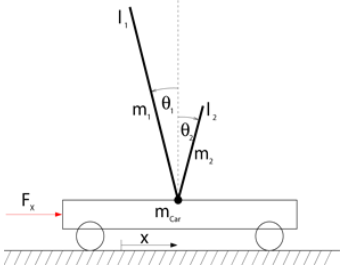
3.3 Vlastnosti

Celulární kódování je metoda, která přináší spoustu možností. Jedním z nastavení algoritmu je množina instrukcí. Přidáním nových, komplikovanějších instrukcí, lze docílit rychlejší konvergence. Stejně tak můžeme některé odebrat a nenastavovat váhy, které můžeme vyvíjet pomocí učících algoritmů. Další z možností je změna počátečního grafu.

Pokud mezi instrukce přidáme instrukci, která umožňuje vytváření cyklů, dáme tak celulárnímu kódování možnost vytvářet jednoduše symetrické sítě.

4. Experimenty – balancování tyčí

Klasickým benchmarkovým testem pro rekurentní neuronové sítě vytvořené pomocí neuroevolučních algoritmů je takzvané balancování tyčí („*pole balancing*“) [2]. Tato úloha má za cíl balancovat dvě tyče s různou délkou (existuje i jednodušší varianta pouze s jednou tyčí), které jsou připevněny k pohyblivému vozíku. Vozík se smí hýbat pouze v jedné ose a navíc nesmí překročit předem dané hranice (musí zvládnout balancování na omezeném prostoru). Neuronová síť rozhoduje o síle, která má být na vozík aplikována. Tyč je považována za vybalancovanou, pokud je její úhel náklonu v rozmezí [-36°,36°].



Obrázek 7. Ukázka problému balancování dvou tyčí.

Vstupy neuronové sítě jsou:

1. pozice vozíku - normalizovaná na rozsah [-1,1] mezi krajními pozicemi
2. rychlost vozíku v m/s
3. úhel θ_1 první tyče - normalizovaný na rozsah [-1,1] mezi krajnímy úhly
4. rychlost změny úhlu θ_1 v rad/s
5. úhel θ_2 druhé tyče - normalizovaný na rozsah [-1,1] mezi krajnímy úhly
6. rychlost změny úhlu θ_2 v rad/s

Existují dvě verze problému. První je jednodušší a obsahuje všechny výše uvedené vstupy. Druhá, složitější, neobsahuje informaci o rychlostech (body 2,4 a 6). Aby bylo možné tyče vyrovnat, musí být síť schopná rychlosti sama odvozovat.

Výjimkou může být situace, kdy se síť naučí vyrovnávat tyče pomocí rychlého kývavého pohybu ze strany na stranu. Aby byla síť donucena používat k vyvažování i rychlosti, byla vytvořena speciální fitness funkce penalizující oscilace pohybu (rovnice 6).

$$f_1 = t/1000 \tag{4}$$

$$f_2 = \begin{cases} 0 & pro \quad t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}(|x^i|+|\dot{x}^i|+|\theta_1^i|+|\dot{\theta}_1^i|)} & jinak \end{cases} \tag{5}$$

$$f = 0.1 \cdot f_1 + 0.9 \cdot f_2 \tag{6}$$

Kde t odpovídá kroku simulace, x^i vzdálenosti vozíku od středu, $\dot{x}^i$ rychlosti vozíku, θ_1^i velikosti úhlu velké tyče a $\dot{\theta}_1^i$ rychlosti náklonu velké tyče. Tato rovnice je definována vždy pro 1000 kroků. Do výsledné hodnoty se započítávají i odchylky od středu a rychlosti. Jejich minimalizací lze zlepšit hodnotu fitness a tím pádem penalizovat vyrovnávání kmitavým pohybem.

Počáteční podmínky simulace jsou použity shodně s [1]. Délka trasy, na které lze balancovat je 4.8m. Kratší tyč měří 0.1m a míří kolmo vzhůru. Delší z tyčí měří 1.0m a je nakloněná pod úhlem 1°. Délka kroku simulace je 0.01s a cílová doba simulace je 100 000 kroků. Síť je použita při každém kroku simulace a je provedena vždy jedna aktivace. Výstup sítě je upraven do rozsahu [-10,10] a odpovídá síle aplikované na vozík v Newtonech. K výpočtu simulace je použit algoritmus Runge-Kutta 4. řádu.

4.1 Parametry algoritmu NEAT

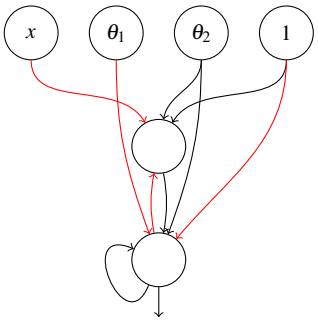
Parametry algoritmu jsou – obvzláště u algoritmu NEAT – zásadní pro výsledné chování a výsledky. Parametry použité pro experiment jsou shodné s parametry použitými v článku [2]. Oproti článku však bylo zjištěno zlepšení výsledků při snížení síly mutací. Velikostí mutací bylo testováno více. Další parametry byly voleny následovně: Velikost populace N: 1000, Max. počet generací: 300, Max. váha spojení: [-8,8], Aktivační funkce: $f(x) = \frac{1}{1+e^{-4.9 \cdot x}}$, Počáteční síť: plně propojená bez skrytých neuronů, Počáteční rozsah vah: [-0,01;0,01], Pst. (pravděpodobnost) křížení: 75%, Pst. křížení bez mutací: 80%, Pst. přidání neuronu: 3%, Pst. přidání spojení: 30%, Pst. křížení průměrem: 40%, Síla mutací: 0,75 / 1,0 / 1,3 / 1,8, Rozsah nově generovaných vah: [-2,2], Mezidruhová hranice δ: 4,0, $c_1 = 1, \quad c_2 = 1, \quad c_3 = 3$, Procento rodičů: 20%.

Experiment byl pro každou rozdílnou sílu mutací spuštěn padesátkrát. Výjimečně nastala situace, kdy nebylo nalezeno řešení (z 200 běhů nebylo nalezeno řešení ve 3 případech). V případech, kdy řešení nebylo nalezeno, došlo během prvních generací ke slibnému nárustu fitness. Ten se ovšem zastavil v lokálním extrému, ze kterého se algoritmus nedokázal dostat. Výsledky ze všech experimentů lze vidět níže:

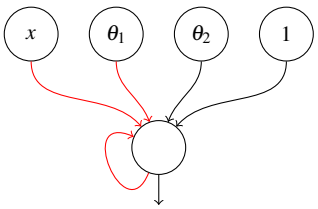
Síla mutací	Generace		Evaluace	
	Průměr	S. odchylka	Průměr	S. odchylka
0,75	24,69	22,85	25321,55	22424,62
1,00	22,00	16,97	22392,46	15710,51
1,30	26,24	19,18	25729,75	17573,92
1,80	30,55	37,43	28397,00	31735,94

Tabulka 1. Výsledky experimentů s DPNV pro různé hodnoty síly mutací. Hodnoty vychází z 50 pokusů pro každou z hodnot mutací.

Zajímavým faktem vycházejícím z tabulky 1 je, že průměrný počet generací (a tedy i evaluací fitness funkce) je výhodnější pro slabší sílu mutací, než jaká je udávána v článku [2]. To může být způsobeno implementačními rozdíly v nezdokumentovaných úpravách algoritmu.



Obrázek 8. Vygenerovaná síť řešící DPNV se skrytým neuronem.



Obrázek 9. Vygenerovaná síť řešící DPNV s minimální topologií.

Nejčastěji vyvýjenou topologií byla minimální možná topologie pro vyřešení DPNV obsahující jedno rekurentní spojení výstupního neuronu se sebou samým (viz obr. 9). Toto řešení ukazuje snahu algoritmu NEAT hledat minimální řešení. Na obrázku 4 pak lze vidět komplikovanější síť využívající skrytý neuron a zpětnou vazbu do něj vedoucí z výstupního neuronu.

4.2 Celulárního kódování a výsledky

Celulární kódování na tomto problému nebylo příliš úspěšné. Algoritmus se velice často dostával do lokálních extrémů.

Expreimentovali jsme s různým nastavením algoritmu - sadou instrukcí se kterou algoritmus pracuje. Nejlépe dopadl algoritmus s instrukcemi, které nemění

topologii, ale pouze váhy. Naopak pokud měl algoritmus k dispozici i instrukce, které mění samotnou topologii, tak končil velice často neúspěšně.

Výsledky algoritmu je možné vidět v tabulce 2. V můžeme vidět, v jaké genraci průběžně došlo k vyřešení problému (nevyřešené běhy algoritmu byly ignorovány) a kolik kroků algoritmus zvládl. V posledním sloupci je možné vidět v kolika procentech byl agoritmus úspěšný.

Instrukce	Prům. generace	Počet kroků	Úspěch alg.
Plná sada	430	300	20%
Úpravy vah	99	20153	80%

Tabulka 2. Výsledky experimentů s DPNV pro různé sady instrukcí. Hodnoty vycházejí z 30 pokůsů pro každou sadu instrukcí.

5. Závěr

V článku jsme představili dvě metody spadající do kategorie neuroevoluce. Tyto algoritmy je možné úspěšně použít pro konstrukci neuronových sítí, dopředných i rekurentních.

V kapitole o algoritmu NEAT jsme si ukázali možnost použití identifikátoru s informací o vzniku genu. Toho lze využít pro křížení genomů nestejné délky i porovnávání podobnosti dvou topologií. Tyto vlastnosti umožňují použití klasických genetických algoritmů.

Algoritmus NEAT byl také ověřen na vlastní implementaci. Použit byl typický benchmarkový problém pro tvorbu rekurentních neuronových sítí – balancování tyčí (*pole balancing*). Byla ukázána schopnost algoritmu vyvíjet sítě a snaha o vytváření kompaktních řešení.

V kapitole o celulárním kódování jsme si ukázali dostatečné instrukce pro tvorbu jakékoliv neuronové sítě a představili si dva možné základní grafy - cyklický a acyklický.

Literatura

[1] Kenneth O. Stanley. Efficient evolution of neural networks through complexification, 2004.

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

[3] Frédéric Gruau, L'universite Claude Bernard lyon I, Of A Diplome De Doctorat, M. Jacques Demongeot, Examinators M. Michel Cosnard, M. Jacques Mazoyer, M. Pierre Peretto, and M. Darell Whitley. Neural network synthesis using cellular encoding and the genetic algorithm., 1994.