Crawler - chodící robot pomocí HyperNEAT

Jan Bouček, FIT ČVUT

Upozornění: tato dokumentace je z velké části pouze zkrácená, aktualizovaná a pro BI-ZUM upravená verze původní dokumentace (zde) odevzdané jako součást maturitní práce, která detailněji popisuje jednotlivé využité techniky.

Cíl projektu

Cílem projektu je vyvinout ovladač simulovaného čtyřnohého 2D robota, který dosahuje co nejrychlejší plynulé chůze v jednom směru, pomocí HyperNEAT.

Chůzi lze definovat jako sérii stavů kráčivého tělesa, kde každý stav lze zjednodušit jako soubor informací popisujících jednotlivé pohyblivé prvky robota. V rámci tohoto projektu budeme pracovat s dvojrozměrnou simulací čtyřnohého robota, jehož každá noha je ovládána dvěma klouby. Stav robota tak lze vystihnout jako osmirozměrný vektor $\vec{s}$. Hledáme pak ovladač robota, který pro každý stav malezne osmirozměrný vektor $\vec{\omega}$, ten každému kloubu určuje úhlovou rychlost, která ovlivní pohyb v dalším snímku.

Využitá teorie

NEAT

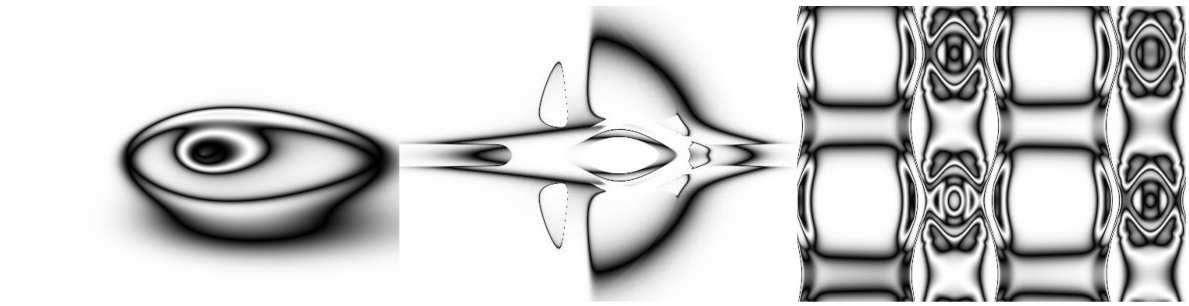
Neuroevolution of augmenting topologies (ref) - NEAT - je metodou, která definuje jeden ze způsobů, kterými lze vyvíjet neuronové sítě pomocí evolučního algoritmu. Ideou tohoto systému je způsob, kterým zapisuje neuronové sítě, jakožto *živočichy* v genetickém algoritmu. Tento systém každou neuronovou síť popisuje jako ohodnocený graf pomocí dvou seznamů *genů* - seznamu vrcholů a seznamu hran. Každému vrcholu a hraně přisuzuje číselné identifikátory, pomocí kterých dokáže udržovat přehled o tom, kteří jedinci jsou si *geneticky* podobní. Stanley a Miikkulainen díky tomu zavádí i proces *speciace*, který ještě před rozmnožováním roztřídí jedince na různé *druhy* podle příbuznosti tak, aby se spolu křížily jen sítě s menšími rozdíly. Každý druh je pak ohodnocen svojí průměrnou hodnotou fitness, pomocí které se určí počet potomků v další generaci daného druhu. Při procesu rozmnožování je z každého druhu vybrána *silnější* část, ze které se vytvoří požadovaný počet potomků, z nichž každý může vzniknout křížením - většinu genů zdědí po silnějším rodiči, ale část od slabšího, nebo bez křížení, kdy se jedinec pouze zkopíruje do další generace. Pak na všech potomcích proběhnou mutace, přičemž je náhodně rozhodnuto, které z druhů mutací na nich proběhnou, Stanley a Miikkulainen jich popisují několik:

- přidání nové hrany do sítě
- rozdělení hrany na dvě hrany s novým neuronem uprostřed
- změna všech vah o malou hodnotu, nebo na náhodnou hodnotu

Po zmutování jsou všichni potomci prohlášeni za současnou generaci a algoritmus pokračuje znovu hodnocením jedinců.

CPPN-NEAT

Tím, že v NEAT algoritmu umožníme každému neuronu využívat jinou aktivační funkci, můžeme dosáhnout tvorby sítí, které jsou dobře uzpůsobené ke generování fyzické geometrie živočichů. Metoda CPPN-NEAT (ref) využívá různých vlastností funkcí - symetrie, repetice apod. tak, že průchodem celou sítí jsou výsledná data nakombinována pomocí komplexní složené funkce, která si uchováva všechny tyto vlastnosti. Pokud například vytvoříme CPPN síť s dvěma vstupy - souřadnicemi x a y s jedním výstupem, získáme dvojrozměrné útvary (viz obr. níže), které mají velmi blízko k fyzickému rozložení v reálných živočiších, např. dokáže nakombinovat repetici s variací a generovat útvary podobné prstům ruky nebo pomocí symetrie a variace útvar podobný lidskému oku (viz obr. níže). V samotném algoritmu stačí jen při tvoření nových neuronů určit náhodnou aktivační funkci a přidat mutační operátor, který změní funkci u náhodného neuronu.



HyperNEAT

Technika HyperNEAT (ref) využívá CPPN sítě ke generování čtyřrozměrného prostoru, který ve výsledku slouží jako definice vah v další neuronové síti. Tzn. "organická struktura", kterou generujeme pomocí CPPN-NEAT je "mozek". To znamená, že generujeme CPPN síť, která má čtyři vstupy - x_1, y_1, x_2, y_2. Výstup nám pak určuje váhu spoje z neuronu na *fyzických* souřadnicích $[x_1; y_1]$ do neuronu na souřadnicích $[x_2; y_2]$. Stačí nám pak navrhnout *fyzickou* strukturu sítě k využití této techniky. V tomto projektu je použito rozložení zvané *state-space sandwich* (ref) o třech vrstvách podobně jako v referovaném článku, kde výsledná síť je rozvrstvená do trojrozměrného prostoru a z každého neuronu vede spoj do každého neuronu v další vrstvě. CPPN síť má pak dva výstupy, kde první určuje váhu mezi souřadnicemi $[x_1; y_1]$ z první vrstvy do $[x_2; y_2]$ v druhé a druhý výstup určuje stejným způsobem váhy mezi druhou a třetí vrstvou.

Implementace

Projekt je implementován v jazyce Kotlin (původní verze byla v Javě), byl vybrán zejména pro dobrou objektovou strukturu a protože se mi v něm programuje nejpohodlněji. V následním textu je popsáno, jak jednotlivé komponenty projektu fungují a jaké všechny vylepšení jsem vystavěl na základním algoritmu.

Program zkouší evoluci 20 krát s různým seedem generátoru náhodných čísel.

NEAT

NEAT je impementován velmi blízko tomu, jak byl výše popsán. Kromě několika základních jsem ještě přidal několik dalších mutačních operátorů:

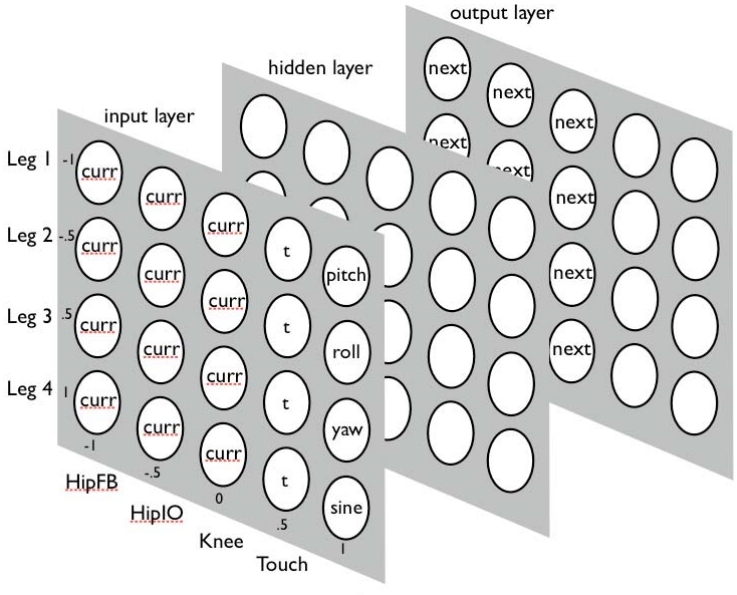
- aktivace/deaktivace jedné hrany
- změna jedné váhy na náhodnou hodnotu
- změna jedné váhy o maximálně $\pm 5\%$

Díky této malé úpravě dokáže algoritmus ladit neuronovou síť v mírně detailnějším měřítku.

HyperNEAT

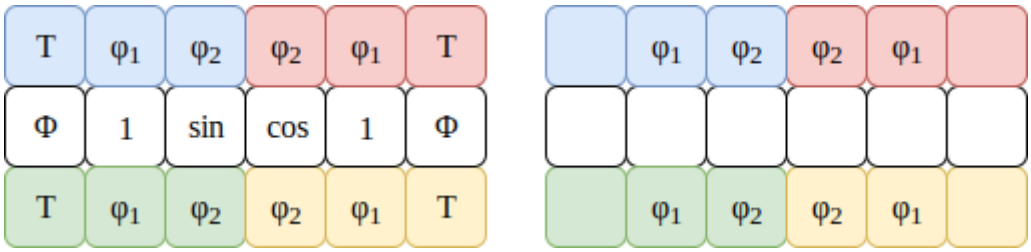
HyperNEAT je také implementován docela standardním způsobem. Největší otázkou bylo, jak ve *state-space sandwichi* rozmístit neurony.

Na rozdíl od klasických neuronových sítí jsou si sítě generované HyperNEATem vědomé geometrických souvislostí (ref). Clune et al. (ref) umisťuje do každého řádku v *substrátu* jednu nohu a do posledního sloupce přidává neuspořádaně zbývající informace:



Tento postup sice dává geometricky najevo související buňky - podobné informace jsou ve stejných sloupcích, ale opomíjí rozložení nohou v prostoru, proto se v tomto projektu používá struktura, která za pomoci symetrie dává najevo souvislosti typu *přední/zadní pár nohou* a *levá/pravá noha*. Pro nejlepší výsledky je v rozložení nohou středová souměrnost a zbývající informace jsou v prostřední řadě mezi nohama (viz obr níže). Každá noha je popsána pomocí dvou úhlů kloubů (označeno ϕ) a dotykového *senzoru* (vstup označený T) na spodním článku nohy, jehož hodnota je nastavena na -1 bez dotyku a 1 s dotykem, pro plynulost pohybu mají tyto hodnoty lineární přechod o délce 25 snímků. Jako přídavné proměnné jsou v tomto rozložení $sin(t/\tau)$, $cos(t/\tau)$, kde τ je experimentálně určená konstanta 40, dále úhel těla robota se zemí Φ a 1 jakožto *bias*. Z výstupní vrstvy jsou využívány pouze neurony v pozicích úhlů nohou, které jsou převedeny na úhlovou rychlost podobně jako v (ref): $\omega = 4,5 \cdot (\phi_{new} - \phi_{cur})$.

Pro ilustraci této struktury poslouží následující obrázek:

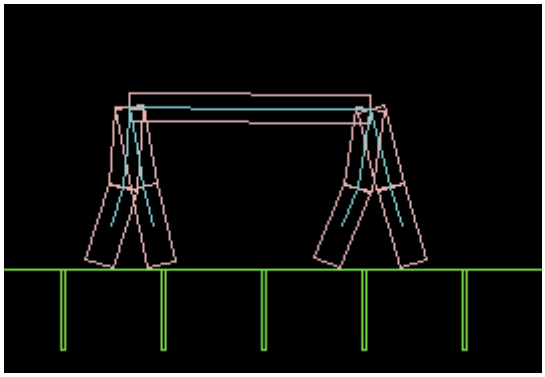


symetrické rozložení sítě, barvy označují jednotlivé nohy, vlevo vstupní vrstva, vpravo výstupní vrstva, prostřední skrytá vrstva nevyobrazena

Simulace

K simulaci je použita knihovna JBox2D (ref), která poskytuje v celku věrohodný model dvojrozměrného světa, ale bohužel nesimuluje úplně deterministicky, takže každý běh programu se ve výsledcích liší.

Robot se skládá z hlavního těla, na které jsou napojeny nohy (viz obr. níže). Každá z nich je napojena pomocí rotujícího *RevoluteJointu* a stejným způsobem je napojen i spodní segment v koleni.



Měření vzdálenosti probíhá 3000 snímků při 60 snímcích za sekundu a dalších 1500 snímků se čeká, jestli robot po konci měření upadne na zem. Tím se zabraňuje strategiím, kdy místo snahy o chůzi robot pouze skočí směrem kupředu a robota je nucen chodit vzpřímeně místo plazení.

Výsledná vzdálenost se přímo využívá jako fitness v evolučním algoritmu, pokud robot upadl, počítá se jako 0.

Pro zrychlení se simulace pouští paralelně a výsledky pro konkrétní genotyp se ukládají do cache, aby se simulace pro stejný genotyp nemusela počítat znovu (což je poměrně častý jev, protože některé genotypy se rozmnoží bez mutace).

Jedna z nových úprav je také cachování poolů objektů pro JBox2D v rámci každého vlákna, což spolu s několika dalšími optimalizacemi znamenalo desetinásobné zrychlení evolučního algoritmu oproti původní verzi. Ve výsledku všech 20 běhů programu trvá na průměrném laptopu (Intel i5-8250U) přibližně 10 hodin.

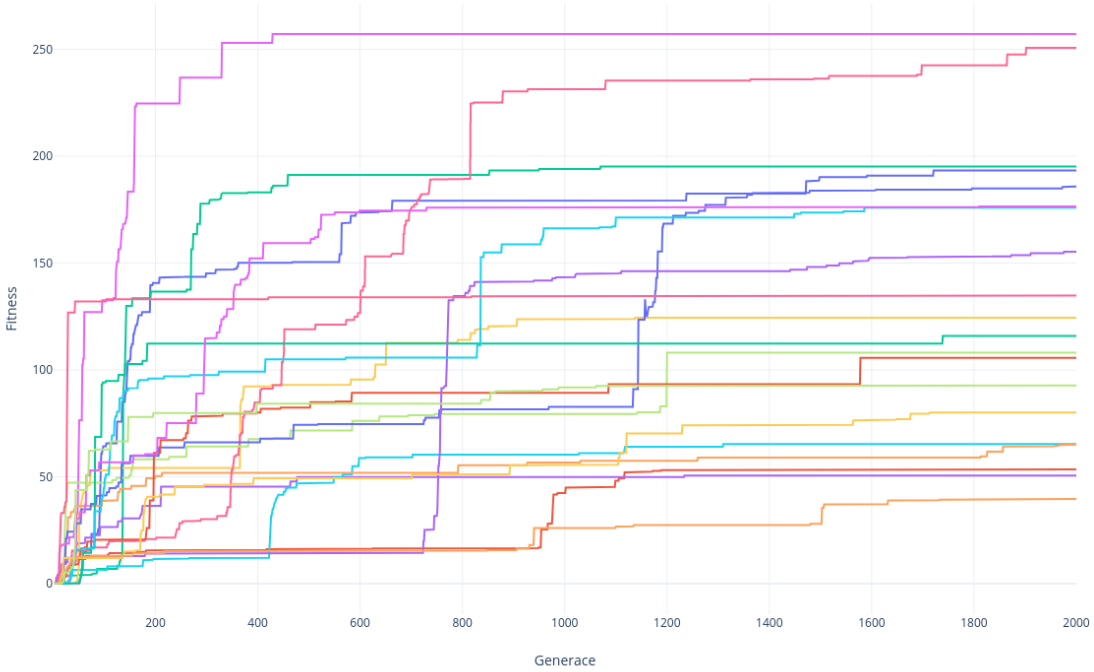
Výsledky

Nejlepším výsledkem programu je ovladač robota, který je schopen během 3000 snímků ujít 257 délkových jednotek. Výsledá "chůze" je ale ve skutečnosti spíš skákaní, které sice nebylo původním cílem, ale v rámci daných pravidel je zřejmě efektvnější než opravdová chůze.

U několika ovladačů se ale opravdu vyvinulo něco, co se za chůzi považovat dá, nejlepší takový ovladač dosahuje vzdálenosti 195. K lidské, či zvířecí chůzi ale chybí jedna zásadní vlastnost - při chůzi moc nezvedá nohy. K tomu dochází zejména z toho důvodu, že při chůzi na úplně hladkém povrchu k tomu ani není důvod a stálý kontakt se zemí mu dodává další stabilitu.

nejlepší výsledek: video

výsledek s chůzí: video



průběh vývoje fitness podle generace napčíč všemi 20 běhy

Uživatelská příručka

Výstupem projektu jsou dva programy - evoluční algoritmus a prohlížeč výsledků. Kompilace i běh programů vyžaduje Javu ve verzi 8 a novější, projekt byl otestován na Linux Mint 19.3 s Oracle JDK, kompatibilita s jinými systémy je velmi pravděpodobná. Kód lze zkompilovat pomocí Mavenu příkazem `mvn package`. Bez kompilace lze využít dodávaný předkompilovaný archiv.

Po kompilaci nebo extrakci archivu lze programy spustit **v hlavní složce projektu** v terminálu následovně.

Prohlížeč výsledků:

```
java -jar target/crawler-gui-jar-with-dependencies.jar
```

Evoluční algoritmus:

```
java -jar target/crawler-learn-jar-with-dependencies.jar
```

Prohlížeč výsledků

Grafické prostředí je ovládané poměrně intuitivně, v levé části je na výběr postupně:

- běh programu (podle názvu složky)
- generace programu
- jedinec z generace

Na pravé straně se zobrazuje simulace společně s užitečnými daty umístěnými v levé části simulačního panelu:

- souřadnice x těla robota
- maximální dosažená souřadnice x těla robota v současné simulaci

- maximální dosažená souřadnice x těla robota po 3000 snímků
- počet uběhlých snímků
- souřadnice y těla robota
- současné hodnoty vstupní vrstvy neuronové sítě
- současné hodnoty skryté vrstvy neuronové sítě
- současné hodnoty výstupní vrstvy neuronové sítě

Napravo od těchto dat je vyobrazena CPPN síť. Vstupní neurony jsou na jejím levém okraji, výstupní neurony na pravém okraji, mezi nimi jsou náhodně rozloženy skryté neurony, jejichž barva značí použitou aktivační funkci. Spoje v síti mají svůj cílový neuron označený kratším obarveným koncem. Neaktivní spoje jsou vyznačeny šedou barvou.

Evoluční algoritmus

Evoluční program vyžaduje soubor `config.cfg`, který obsahuje nastavení několika klíčových konstant, které lze měnit. Program ukládá výsledky do složky `results`, kam také ukládá záznam průběhu programu, který rovněž vypisuje do konzole. Vypisuje několik důležitých informací, jako nejdelší dosaženou vzdálenost, počet potomků, popis jednotlivých druhů apod.