UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PROGRAM PRE SÚŤAŽ SIMULATED CAR RACING

BAKALÁRSKA PRÁCA

2013

Ondrej Balún

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

PROGRAM PRE SÚŤAŽ SIMULATED CAR RACING

BAKALÁRSKA PRÁCA

Kód práce:	0b752c44-8b39-4f30-b345-845b910ce785
Študijný program:	Aplikovaná informatika
Študijný odbor:	2511. Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	Mgr. Pavel Petrovič, PhD.

Bratislava, 2013

Ondrej Balún

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

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

Názov: Program pre súťaž Simulated Car Racing

Cieľ: Cieľom práce je preštudovať rozličné metódy evolučných algoritmov, analyzovať ich z hľadiska konkrétnej aplikácie, vybrať vhodnú metódu a niekoľko alternatív, navrhnúť vlastné reprezentacie, operátory, porovnať ich vzhľadom na ich úspešnosť, prezentovať a prediskutovať výsledky porovnania. Výsledný softvér sa zapojí do súťaže organizovanej pri konferenciách o evolučných výpočtoch (GECCO, EVOSTAR, CIG).

Literatúra:
A.E.Eiben, J.E.Smith: Introduction to Evolutionary Computing, Springer, 2008.
W.Banzhaf et al.: Genetic Programming, Morgan Kaufmann, 1998.
I.Zelinka et al.: Evoluční výpočetní techniky, Principy a aplikace, BEN, 2009.

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

Dátum zadania: 04.10.2012

Dátum schválenia: 04.10.2012

doc. RNDr. Mária Markošová, PhD.
garant študijného programu

Balún
.....
študent

Pavel Petrovič
.....
vedúci práce

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

.....

podpis autora práce

Poďakovanie

Chcem úprimne poďakovať svojmu školiteľovi Mgr. Pavelovi Petrovičovi, PhD. za jeho pomoc a rady, organizátorom súťaže Simulated Car Racing za ústretovú komunikáciu a spolužiakovi Dominikovi Kertysovi za pomoc pri formálnych úpravách textu.

Abstrakt

Práca je zameraná na tému evolučných algoritmov a ich využití pri riešení reálnych úloh. Prináša základný prehľad dôležitých smerov v strojovom učení, s ťažiskom na evolučné výpočty a neurónové siete. Cieľom je využiť tieto poznatky na vytvorenie autopilota do prostredia TORCS (The Open Racing Car Simulator). Po analýze niektorých zaujímavých existujúcich riešení uvádzam svoj návrh, využívajúci neuroevolúciu (NEAT). Dôležitá úloha je nastavenie parametrov neurónovej siete, ako aj operátorov pre evolučný algoritmus. Až potom je spustená evolúcia. Kandidáti na riešenie súťažia medzi sebou prostredníctvom hodnoty fitness, ich vyhodnotenie prebieha simuláciou na konkrétnej trati. V závere je natrénovaný vodič porovnaný s ručne naprogramovaným vodičom a tiež najrýchlejším pilotom v TORCS na viacerých tratiach. Výsledky poukazujú na silu neuroevolúcie, ktorá je schopná konkurovať ostatné úspešné riešenia.

Kľúčové slová

evolučné algoritmy, strojové učenie, závod áut, GECCO

Abstract

This thesis is focused on the evolutionary algorithms and their use in practice. In the beginning, I introduce the most important approaches in the field of machine learning with evolutionary computing and neural networks in the centre. The goal is to use this knowledge to propose racing bot for TORCS (The Open Racing Car Simulator). After analysing some interesting existing solutions I come with my own proposal using neuro-evolution (NEAT). The important task is to set the parameters of the neural network, as well as the operators for the evolutionary algorithm. Then the evolution is started. The fitness of the candidate solution is computed through a simulation performed with TORCS. In the end, I compare the performance of my approach to the hand-programmed bot and to the performance of the fastest controller available for TORCS. The results show, that neuro-evolution is able to reach good performance comparable to the other successful solutions.

Keywords

evolutionary algorithms, machine learning, car racing, GECCO

Predhovor

Témou mojej bakalárskej práce sú evolučné algoritmy, ich aplikácie v praxi, konkrétne v mojom prípade - využitie pri riadení závodného auta. Osobne považujem toto zadanie za zaujímavé, pretože nachádza miesto v súčasnej modernej dobe, kde dochádza k automatizácií princípov takmer vo všetkých odvetviach.

Ukázalo sa, že dobre nastavený stroj dokáže presnejšie a spoľahlivejšie vykonávať úlohy, v ktorých ľudský faktor zohráva negatívnu rolu. Napriek mnohým úspechom existujú odvetvia, pre ktoré dnešná úroveň umelej inteligencie nie je postačujúca. Tu patrí téma riadenia automobilu robotom. Problematika je veľmi komplexná a zlyhanie automaticky znamená veľmi negatívny dopad.

Vo svojej práci sa pokúsim priblížiť niekoľko existujúcich riešení a navrhnúť svoj návrh s využitím evolučných výpočtov. Výsledky porovnám, prinesiem zoznam negatív a pozitív, tiež návrhy pre budúce riešenia.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Človek, okrem toho, že sa nazýva tvor rozumný, je predovšetkým tvor lenivý a pohodlný. Bez ohľadu na to, ako to znie, je to pravda, ktorá je dokázaná každou sekundou ľudskej existencie na tejto planéte. Pre mnohých vlastnosť považovaná za neduh, je jednou z hnacích motorov dejín a dôvodom, prečo píšem túto bakalársku prácu.

Lenivosť a pohodlnosť viedla ľudskú rasu k hľadaniu riešení, ktoré by uľahčili život. Tak, ako bolo vynájdené koleso, aby sa nemusel nosiť ťažký náklad na ľudských pleciach, tak isto počas priemyselnej revolúcie prišli vynálezy strojov, nahrádzajúcich stovky ľudí a tak isto sa v dnešnej dobe hľadajú riešenia inteligentného riadenia rôznych prvkov všedného života, od kuchynského robota, po inteligentné systémy, riadiace proces výroby, medicínske a vojenské operácie a v neposlednom rade sa v záujme bezpečnosti na cestách a optimalizácie logistiky hľadajú sofistikované riešenia riadenia automobilov, lietadiel atď. Práve riadenie automobilu, konkrétne závodného vozidla je v predmete záujmu mojej práce.

V praxi existuje mnoho návrhov, experimentov a testov, ale stále panuje strach v spoločnosti, ako sa tieto autá budú správať najmä v rizikových situáciách, keďže dnešná úroveň umelej inteligencie stále postráda uspokojivú úroveň spolupráce s človekom. Určite stojí za zmienku projekt spoločnosti Google, ktorá svoje bezpilotné autá testovala na tisíckach kilometroch a dokonca niektoré americké štáty prijali zákon, ktorý povoľuje tieto vozidlá. Problém spočíva v tom, že riadenie takého automobilu, musí brať do úvahy obrovské množstvo faktorov, ktoré si ani neuvedomujeme, musí ich vedieť správne vyhodnotiť a keďže rôznymi kombináciami týchto faktorov vznikajú mnohé nepredvídateľné situácie a nové problémy, systém riadenia sa musí byť schopný učiť a vyvíjať v takom prostredí, kde nie sú ospravedlnené chyby, kde neexistuje možnosť „odskúšať si“, kde aj malý omyl môže spôsobiť ťažké následky. Na súčasnej úrovni vedeckého pokroku, nie je možné nájsť optimálne riešenie pre taká obrovské množstvo vstupov a jediná schodná cesta je použiť rôzne heuristiky.

Keďže osobne preferujem aktivity, kde môžem byť kreatívny, kde nie je vyšliapaná len jedna cesta, kde sa riešenie stále hľadá, rozhodol som sa zapojiť do problematiky bezpilotného riadenia automobilu a venovať tomu svoju bakalársku prácu. Verím, že sa mi podarí vytvoriť riešenie, ktoré nebude zapĺňať len priestor fakultnej knižnice, ale bude ďalší krok na ceste k cieľu inteligentného riadenia.

Kapitola 1

Ciele a štruktúra práce

1.1 Ciele práce

Cieľom mojej bakalárskej práce je preštudovať rôzne metódy pre evolučné algoritmy, analyzovať ich a využiť na návrh riešenia pre autopilota súťaže Simulated Car Racing. Navrhnúť vlastné aplikácie evolučných algoritmov a získané výsledky porovnať, najmä s výsledkami dosiahnutými ručným programovaním. Vlastné zistenia budú uplatnené pri optimalizácii môjho riešenia s cieľom navrhnúť čo najúspešnejšieho autopilota v zmysle pravidiel spomínanej súťaže organizovanej pri konferenciách o evolučných výpočtoch (GECCO, EVOSTAR, CIG).

1.2 Štruktúra práce

Bakalárka práca bude pozostávať z nasledujúcich hlavných častí:

- **Predstavenie problému, východiská, existujúce riešenia** – v tejto časti by som rád pomenoval problematiku, ktorej sa venuje moja bakalárska práca, ako aj predstavil východiská pre jej vypracovanie, budú tu objasnené základné pojmy a pravidlá súťaže Simulated Car Racing. Tiež priblížim existujúce riešenia a postupy.
- **Návrh riešenia** – návrh vlastného riešenia problematiky
- **Realizácia** – samotná implementácia návrhu a rôzne experimenty v testovacom prostredí za cieľom optimalizácie riešenie
- **Záver** – zhrnutie výsledkov, prínosov, návrhy a vízie do budúcnosti

1.3 Použité pojmy a skratky

Zoznam použitých pojmov a skratiek v práci

- TORCS – The Open Racing Car Simulator
- Autopilot, jazdec, vodič, bot – inteligentný agent riadiaci vozidlo v TORCS
- Tick – jednotka času hry (približne 20ms simulovaného času)
- Naivný autopilot – implemetácia pilota dodávaného autormi softvéru pre Simulated Car Racing

Kapitola 2

Opis problému, východiská a existujúce riešenia

V tejto kapitole je popísaný problém, na ktorý sa bakalárska práca zameriava - súťaž Simulated Car Racing, jej zmysel, prostredie a pravidlá. Sú tu vysvetlené dôležité pojmy najmä k evolučným algoritmom. Kapitola ponúka prehľad východísk k bakalárskej práci. Stručne sú vymenované a opísané existujúce riešenia.

2.1 Predstavenie problému

Každoročne sa koná niekoľko konferencií k evolučným algoritmom a ich využití, kde sa prezentujú aktuálne výsledky, pokroky, odborná verejnosť môže prispievať do týchto konferencií svojimi článkami a tiež sa zúčastniť osobne prednášok pozvaných hostí.

2.1.1 Konferencie o evolučných výpočtoch

V krátkosti zhrnuté najznámejšie konferencie o evolučných výpočtoch.

CIG - Computational Intelligence in Games

Konferencia, ktorá sa zameriava predovšetkým na umelú inteligenciu a hlavne evolučné výpočty v hrách. Medzi hlavné témy patrí napr.: ko-evolúcia, neurónové siete, fuzzy logika v hrách, multi-agent a multi-strategy učenie, aplikácie teórie hier.

EvoStar – The Leading European Event on Bio-Inspired Computation

V rámci tejto konferencie existuje niekoľko sekcií:

- **EuroGP** - International Conference on Genetic Programming

- **EvoCOP** - International Conference on Evolutionary Computation in Combinatorial Optimization
- **EvoBIO** - International Conference on Evolutionary Computation, Machine Learning and Data Mining in Computational Biology
- **EvoMUSART** - International Conference on Evolutionary and Biologically Inspired Music, Sound, Art and Design

Gecco – Genetic and Evolutionary Computation Conference

Najväčšia konferencia na poli genetických a evolučných výpočtov. Popri tejto konferecii sa organizuje aj niekoľko súťaží ako napr. The Evolution Art, kde sa účastníci pomocou gnetických výpočtov snažia vytvoriť niečo umelecké (maľba, hudba, modelovanie), „Humies“, čo je ocenenie pre významný výsledok nejakého problému získaný genetickými výpočtami.

2.1.2 Simulated Car Racing SCR

Súťaž[1] organizovaná popri medzinárodných konferenciách o evolučných výpočtoch. Výstup mojej práce by mal byť schopný zúčastniť sa tejto súťaže, čiže musí byť v súlade s platnými pravidlami.

Ciele

Vyvinúť ovládanie pre pretekárske auto, ktoré je schopné zvládnuť niekoľko vopred neznámych tratí, najskôr na čas, potom proti súperom. Automobil je vybavený sadou senzorov, snímajúcich všetky potrebné faktory okolia ako napr. rýchlosť, pozícia na trati, vzdialenosť od prekážky, množstvo paliva. Odpoveďou je reakcia pilota auta ako napr. natočenie volantu, plyn, spojka, brzda.

Pravidlá

Každý závod pozostáva s troch úrovní:

- **Warm-up (tréning)** – auto má určitý čas, ktorý môže stráviť na dráhe a cieľom je získavať dôležité informácie o trati, ktoré si zapamätá
- **Qualifying (kvalifikácia)** – každý jazdec jazdí sám na trati proti času, 8 jazdcov, ktorí prejdú najväčšiu vzdialenosť, postúpia do finále

[1] Simulated Car Racing. `http://scr.geccocompetitions.com/`

- **Final (finále)** – 8 súperov jazdí na trati proti sebe, bodovanie je systémom F1, pričom štartov je toľko, aby každý jazdec štartoval z každej pozície na štarte. Dodatočný bonus dosiahne pilot, ktorý prešiel trať najrýchlejšie a tiež hráč s najnižším poškodením auta

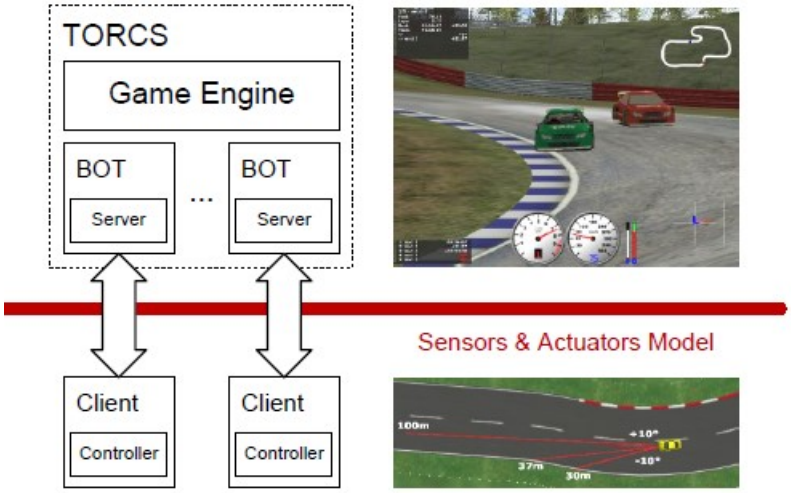
2.1.3 Prostredie TORCS

The Open Racing Car Simulator (TORCS)[1] je open-source simulátor automobilových závodov, poskytujúci 3D vizualizáciu, sofistikovaný fyzikálny engine a dynamiku auta, zachytávajúcu poškodenie, aerodynamiku, spotrebu paliva atď. Prostredie obsahuje množstvo rôznych tratí, áut, niekoľko hracích módov (tréning, rýchly závod, turnaj). Väčšinu nastavení môže užívateľ priamo zmeniť prostredníctvom grafického rozhrania alebo prepísaním konfiguračných XML súborov. Hra obsahuje niekoľko naprogramovaných jazdcov a umožňuje pridávať vlastných pilotov, vlastné trate. TORCS poskytuje vodičovi informácie o pozícii na trati, rýchlosti, vzdialenosti od prekážky prostredníctvom diaľkomerov, stave oponentov a pod. Pilot môže ovládať plynový, spojkový a brzdový pedál, radiť prevodové stupne. Hoci prostredie TORCS nebolo primárne vyvinuté na strojové učenie, svojou modulárnou architektúrou a kvalitným modelom senzorov umožňuje použiť tento softvér na poli strojového učenia. Napriek tomu, že prostredie TORCS je naprogramované v jazyku C++ s použitím OpenGL, v rámci Simulated Car Racing sú dostupné implementácie klienta v jazyku C++, Java a od roku 2013 aj v Python. Implementácia servera je dostupná pre platformu Windows ako aj Linux.

API

V rámci súťaže Simulated Car Racing bol softvér upravený na podobu klient/server architektúry (obrázok 2.1), čo umožňuje separáciu herného enginu od jazdcov a tiež vytvára fair-play prostredie, v ktorom nikto neblokuje iného ani nie je inak zvýhodnený. Klient (pilot) sa cez externý proces pripojí k serveru prostredníctvom UDP spojenia. Každý herný tick (zhruba 20ms simulovaného času) odošle server informácie získané so senzorov (tabuľka 2.1) každému klientovi, ktorý musí do 10ms reálneho času vyslať svoju reakciu v podobe efektorov (tabuľka 2.2). Niektoré hodnoty senzorov môžu byt zašumené, čím sa vnášajú do modelu nepresné hodnoty napodobňujúce reálne prostredie. Ak server nezachytí odpoveď v stanovenom čase, pokračuje v poslednej akcii vodiča. Špeciálnym signálom vyslaným k serveru si klient môže vyžiadať reštart.

[1] The open racing car simulator. `http://torcs.sourceforge.net/`



Obr. 2.1: Architektúra TORCS pre Simulated Car Racing

Tabuľka 2.1: Zoznam dostupných senzorov

Názov	**Rozsah**	**Popis**
angle	[-π; π] (rad)	Uhol zvierajúci os auta s osou trate.
curLapTime	[0; ∞] (s)	Čas odjazdený počas aktuálneho kola.
damage	[0; ∞] (body)	Aktuálne poškodenie auta (čím vyššia hodnota, tým väčšie poškodenie).
distFromStart	[0; ∞] (m)	Vzdialenosť auta od štartovej čiary.
distRaced	[0; ∞] (m)	Vzdialenosť prejdená od začiatku preteku.
focus	[0; 200] (m)	Vektor 5 diaľkomerov; každý vracia vzdialenosť medzi okrajom trate a autom v rozsahu 200 metrov. Senzory vzorkujú 5 stupňový priestor pozdĺž smeru poskytnutého od klienta (akciou focus). Senzor môže byť použitý len raz za sekundu simulovaného času, v opačnom prípade poskytne hodnotu mimo rozsahu (platí aj v prípade, že auto je mimo trate).
fuel	[0; ∞] (l)	Aktuálne množstvo paliva.
gear	-1,0,1,...,7	Aktuálny prevod (-1 je spiatočka, 0 neutrál).
lastLapTime	[0; ∞] (s)	Čas prejdenia posledného kola.

Pokračovanie na ďalšej strane

Tabuľka 2.1 – *Pokračovanie*

Názov	**Rozsah**	**Popis**
opponents	[0; 200] (m)	Vektor 36 senzorov zameraných na súperov: každý pokrýva úsek 10 stupňov v rozsahu 200 metrov a vracia vzdialenosť najbližšieho oponenta v pokrytej oblasti. Senzory pokrývajú celý priestor okolo auta, v smere hodinových ručičiek od π po -π vzhľadom k osi auta.
racePos	1,2,...,N	Pozícia v preteku vzhľadom na protivníkov.
rpm	[2000; 7000] (ot/min)	Počet otáčok motora za minútu.
speedX	[-∞; ∞] (km/h)	Rýchlosť auta pozdĺž pozdĺžnej osi auta.
speedY	[-∞; ∞] (km/h)	Rýchlosť auta pozdĺž priečnej osi auta.
speedZ	[-∞; ∞] (km/h)	Rýchlosť auta pozdĺž osi z auta.
track	[0; 200] (m)	Vektor 19 diaľkomerov; každý vracia vzdialenosť medzi okrajom trate a autom v rozsahu 200 metrov. Senzory pokrývajú priestor pred autom, v smere hodinověch ručičiek od -π/2 po π/2 vzhľadom k osi auta. Ak sa auto nachádza mimo trate, návratov hodnota je mimo rozsahu.
trackPos	(-∞; ∞)	Vzdialenosť medzi autom a osou trate. Hodnota je normalizovaná vzhľadom na šírku trate: 0 predstavuje auto na osi trate, -1 v prípade, že auto je na pravom okraji a 1 v prípade, že auto je na ľavom okraji trate.
wheelSpinVel	[0; ∞] (rad/h)	Vektor 4 senzorov reprezentujúci otáčky kolies.
z	[-∞; ∞] (m)	Vzdialenosť ťažiska auta od povrchu trate pozdĺž osi z.

Tabuľka 2.2: Zoznam dostupných efektorov

Názov	**Rozsah**	**Popis**
accel	[0; 1]	Virtuálny plynový pedál
brake	[0; 1]	Virtuálny brzdový pedál
clutch	[0; 1]	Virtuálny spojkový pedál
gear	-1,0,1,...,7	Rýchlostný stupeň
steering	[-1; 1]	Hodnota zatočenia: -1 a 1 znamenajú naplno doprava resp. doľava, čo zodpovedá uhlu 0,785398 rad.
focus	[-90; 90]	Smer focus diaľkomerov v stupňoch.
meta	0; 1	Príkaz meta-riadenia: 0 predstavuje žiadnu zmenu; 1 žiadosť pre súťažný server o reštart preteku.

2.2 Východiská

Pre kvalitné vypracovanie bakalárskej práce bolo dôležité preštudovať a pochopiť viaceré prístupy, ktoré sú nevyhnutné pre návrh vlastného riešenia problému.

2.2.1 Strojové učenie

Podľa [11] strojové učenie patrí medzi moderné pojmy v informatike, konkrétne v oblasti umelej inteligencie a kognitívnej psychológie, ktorým sa dnes venuje veľká pozornosť a výskum. Je inšpirované kognitívnymi procesmi u človeka. Strojové učenie sa zameriava na výpočtové procesy, ktoré tvoria základ učenia nielen u ľudí, ale aj u strojov. Snaží sa vytvoriť a sformulovať popis pojmu na základe jeho charakteristických vlastností, atribútov.
Rozdelenie algoritmov:

- **Učenie s učiteľom** – pre vstupné dáta je už vopred známy správny výstup a hľadá sa funkcia, ktorá mapuje vstupy na požadovaný výstup
- **Učenie bez učiteľa** – nepoznáme správny výstup
- **Kombinácia s učiteľom a bez učiteľa** – pre určitú množinu vstupov poznáme správny výstup, ale pre zvyšok (väčšinou väčšia časť vstupov) nepoznáme správny výstup, snaha predpovedať nové výstupy z odpozorovaných špeciálnych prvkov vstupnej množiny (trénovacia množina)
- **Učenie posilňovaním** – skúma sa vplyv akcie na prostredie, pričom prostredie cez spätnú reakciu vplýva na učiaci algoritmus

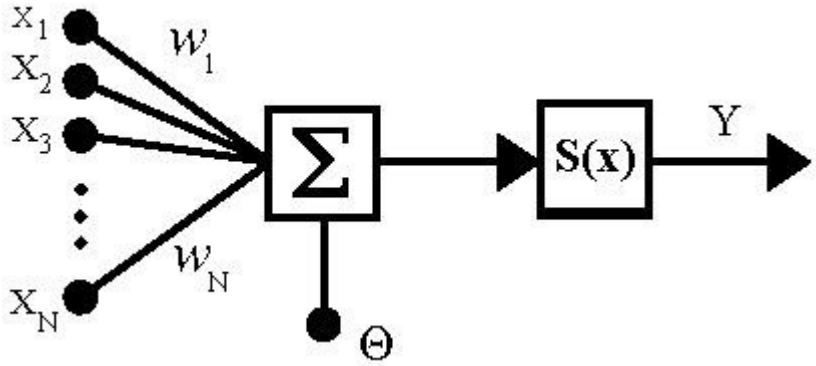
- **„Samoučenie“** – učenie sa základe vlastnej skúsenosti

Vstupom učiaceho algoritmu je množina typických príkladov (trénovacia množina) a na výstupe definícia naučeného pojmu. Typický príklad je reprezentovaný množinou n atribútov, ktoré predstavujú vlastnosti príkladu. Na overenie stupňa presnosti klasifikácie sa používa testovacia množina.

Téma strojového učenia je pre mňa základným východiskom, konkrétne učenie z vlastnej skúsenosti, nakoľko nie je vopred známa správna postupnosť krokov k dosiahnutiu najlepšieho výsledku.

2.2.2 Neurónové siete

Neurónová sieť je jeden z výpočtových modelov súčasnej modernej umelej inteligencie v informatike. Model je inšpirovaný *konekcionizmom*, ktorý je založený na biologických poznatkoch fungovania mozgu. Sieť je zložená z umelých neurónov, ktoré sú pospájané do jedného celku, nazývaného neurónová sieť (ďalej NN). Každý neurón môže mať ľubovoľný počet vstupov, ale vždy len jeden výstup. V rámci tejto siete si neuróny odovzdávajú signály. Na nasledujúcom obrázku (obrázok 2.2) vidíme formálny popis umelého neurónu.



Obr. 2.2: Model neurónu, x_i – vstupy neurónu, w_i – synaptické váhy, Θ - prah, S(x) – prenosová funkcia neurónu (aktivačná funkcia), Y – výstup neurónu

Výhody NN:

- Dokáže pracovať nelineárne, dá sa paralelizovať
- Robustnosť
- Tolerantná voči chybe – zlyhanie niektorého neurónu neohrozí celú sieť

- Dokáže uchovávať kontextové informácie (v synaptických váhach)

Využitie NN je veľmi široké napr. pri spracovaní obrazu, vo finančníctve, fyzike, chémii a medicíne. Pre moju bakalársku prácu je NN jednou z alternatív, predovšetkým modely NN, ktoré využívajú genetické algoritmy. Ako vstupy môžeme chápať signály z prostredia, ktoré tento autopilot má k dispozícii a NN vráti reakciu autopilota.

2.2.3 Pravidlové systémy

Pravidlové systémy sú z bežného života ľuďom veľmi blízke svojou povahou typu, IF podmienka THEN činnosť. Taktiež pri hľadaní odpovedí na otázky uvažujeme napr. Ak meno domáceho zvieraťa má tri písmená a prvá je písmeno „p“, tak to môže byť napríklad pes. Využíva sa dedukcia.

Pravidlové systémy využívajú expertné znalosti na riešenie problémov reálneho sveta. Používajú sa v systémoch riadenia (napr. fuzzy logika), pri spracovaní jazyka (napr. regulárne výrazy).

Od klasických logických systémov sa odlišujú nemonotónnym uvažovaním a možnosťou spracovania neurčitosti. Neurčitosť sa môže vyskytnúť jednak v predpokladoch pravidla, alebo sa môže týkať pravidla ako celku. Na rozdiel od neurónových sietí, pravidlové systémy sú schopné riešiť širokú škálu aj komplexných problémov, dokážu sa prispôsobovať a riešenie vedia vyjadriť jazykom blízkym človeku.
Pravidlový systém sa skladá z týchto základných častí:

- **množina pravidiel (znalostná báza)** – zoznam pravidiel typy IF *podmienka* THEN *dôsledok/akcia*

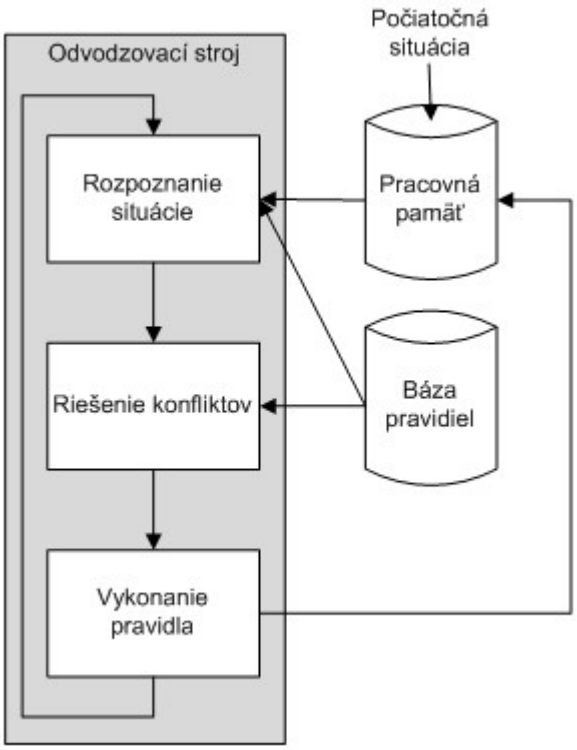
- **pracovná pamäť** – *„uchováva údaje o špecifickom prípade práve riešeného problému a o stave riešenia. Na začiatku odvodzovania (inferencie) je naplnená faktami o riešenom probléme a špecifikáciou počiatočného stavu. V priebehu odvodzovania odvodzovací stroj pridáva, modifikuje alebo odstraňuje fakty z tejto pamäte“* [15]

- **odvodzovanie (inferencia)** – *„hľadajú sa požadované informácie v znalostnej báze a používajú sa pri riešení problémov, v procese odvodzovania musí odvodzovací stroj nájsť správne fakty, interpretácie a pravidlá a správne ich pospájať. Celý proces je cyklický.“* [15]
  Existujú dva hlavné spôsoby odvodzovania:

  – dopredné zreťazenie

  - spätné zreťazenie

- **používateľské rozhranie** - pre komunikáciu používateľa so systémom



Obr. 2.3: Proces odvodzovania v pravidlovom systéme

Pravidlový systém považujem za efektívny rozhodovací systém. Jedno existujúce riešenie autopilota používa práve fuzzy logiku na riadenie. Nevýhodou pravidlového systému sú však zložité a komplikované pravidlá, ktoré sa potom dostávajú do konfliktu a je nevyhnutné investovať mnoho úsilia do správneho riešenia konfliktov.

2.2.4 Evolučné algoritmy

Téma evolučných algoritmov je kľúčová pre bakalársku prácu. *„Význam evolučných algoritmov prichádza s modernou dobou, kedy sa hľadajú riešenia komplexných problémov v priestore s obmedzenou kapacitou, čo si vyžaduje robustné algoritmy dávajúce uspokojivé výsledky v relatívne krátkom čase.“* [7]

Základná charakteristika

Evolučné algoritmy patria do skupiny prehľadávacích algoritmov, čo znamená, že riešenia vyberajú (hľadajú) z množiny vygenerovaných potencionálnych riešení. *„Evolučné*

algoritmy nie sú zamerané na riešenie jednej úlohy, ale snažia sa problém rozbiť na zložky (atribúty) a tie ohodnotiť, čiže dokážu problém riešiť bez toho, aby poznali o ňom veľa informácií, čo ich zaraďuje medzi tzv. slabé algoritmy.“ [10] Sú to populačné algoritmy, ich princíp je biologicky motivovaný. Evolučné algoritmy sa dajú použiť takmer v každej oblasti, napríklad v hrách, v armáde na evolvovanie možných bojových stratégií, všade tam, kde sa hľadá správne nastavenie atribútov a problém je natoľko zložitý, že neexistuje nájdenie ideálneho riešenia v reálnom čase pri súčasnom výpočtovom výkone.

Používané termíny

Podľa čoho považovať niektoré riešenie za lepšie ako iné? Potrebujeme zaviesť ohodnotenie - funkciu, ktorá ohodnotí každé riešenie. Nazýva sa *fitness funkcia*. Vhodnosť riešenia je priamo úmerná výške fitness funkcie. Každú úlohu, problém treba zakódovať pomocou *reprezentácie*. Na reprezentácii veľmi záleží - zle zvolená reprezentácia môže výrazne znížiť výkonnosť algoritmu, dokonca ho natoľko oslabiť, že nájdené riešenia budú slabé. Používa sa binárna reprezentácia, permutačná, reprezentácia reálnymi číslami a stromová reprezentácia. Problém je zadaný v pojmoch, čo nazývame *fenotyp* a tiež riešenie je vo fenotype, ktorý by sa mal dať rozložiť na množinu dvojíc atribút-hodnota. Úloha sa však rieši v *genotype*, čo je jej zložkový tvar (atribúty). Trvanie jednej množiny riešení sa nazýva *generácia*. Pod pojmom *populácia* rozumieme množinu jedincov (potenciálnych riešení) v rámci generácie.

Priebeh algoritmu

Priebeh genetického algoritmu je veľmi podobný evolúcii, ktorú môžeme sledovať v prírode.

Inicializácia

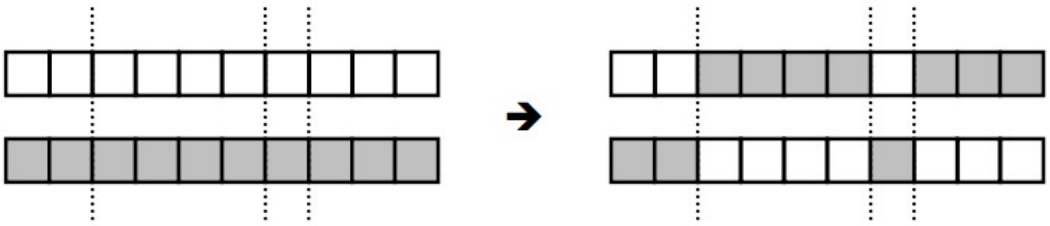
Vygeneruje sa nultá generácia riešení, väčšinou sú to náhodné riešenia a tieto sa vyhodnotia fitness funkciou. Ak sa podarí splniť ukončovaciu podmienku, algoritmus končí, inak sa pokúša nájsť nové riešenia cez reprodukciu.

Reprodukcia

- **Selekcia** - z aktuálnych riešení sa vyberá množina rodičov, čo sa nazýva selekcia. Tu je veľmi dôležité, ktoré riešenia sa majú vybrať, pretože tie pokračujú ďalej v zlepšovaní riešenia, zatiaľ čo ostatné zanikajú. Medzi najčastejšie prístupy výberu patrí *elitizmus* – výber riešení s najvyššou fitness hodnotou, avšak tu sa postrádajú

slabšie riešenia, a algoritmus môže uviaznuť v lokálnom extréme. Ďalej sa používa *ruletový výber*, kde riešenie s vyšším fitness má väčšiu šancu na výber, ale šancu majú aj ostatní, priamo úmerne k hodnotám fitness. Najoptimálnejšia selekcia sa zatiaľ javí *turnaj*, kde sa vyberie s rovnakou pravdepodobnosťou niekoľko riešení, tie potom navzájom súperia a víťazi sa stávajú rodičmi. *„Počet účastníkov turnaja zvyšuje tlak selekcie. Malý počet účastníkov turnaja spôsobí nízky tlak a veľký turnaj vytvorí veľký tlak selekcie.“* [1]

- **Generovanie potomkov** – z vybratých rodičov je potrebné vyrobiť potomkov pomocou genetických operátorov, kde zaraďujeme *kríženie* (crossing-over) a *mutáciu*. Kríženie znamená, že rodičia si navzájom vymenia časť genotypu (obrázok 2.4). Pri mutácii sa časť rodičovského genotypu nahradí inou hodnotou. Cieľom kríženia aj mutácie je vytvoriť novú populáciu riešení s lepším fitness



Obr. 2.4: Kríženie dvoch genotypov

- **Vyhodnotenie potomkov** – vygenerovaní potomkovia sú vyhodnotení pomocou fitness funkcie a pôvodná generácia sa nahradí novou a pokračuje opäť proces selekcie, kým sa nesplní ukončovacia podmienka

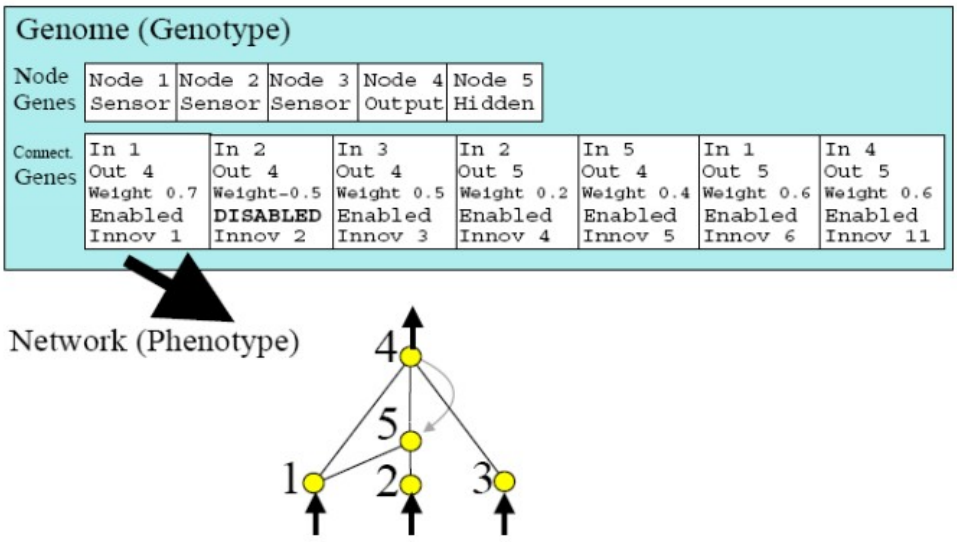
2.2.5 NEAT

NEAT (NeuroEvolution of Augmenting Topologies) je neurónová sieť, v ktorej sa používa neuroevolúcia pomocou evolučného algoritmu. Patrí do rodiny TWEANN (Topology and Weight Evolving Artificial Neural Network) sietí, čo znamená, že sa evolvujú nielen synaptické váhy, ale aj topológia siete.

O. Stanley a Risto Miikkulainen, autori tejto myšlienky, vo svojej štúdii [19] predstavujú svoj model a poukazujú na niektoré výhody, ktoré má táto sieť oproti iným modelom, a ktoré robia túto sieť podstatne efektívnejšiu.

Historical Markings

NEAT používa priame kódovanie, čo znamená, že genóm explicitne špecifikuje fenotyp. Priame kódovanie má tiež výhodu viacerých kódovacích alternatív ako napr. binárne alebo pomocou grafov.



Obr. 2.5: Kódovanie NEAT siete

Gény genotypu rozdeľujeme na uzly a spojenia. Uzly si v sebe uchovávajú informáciu, v ktorej vrstve siete sa nachádzajú a gény kódujúce spojenie obsahujú informáciu, ktoré uzly spájajú, akú majú váhu, či sú aktívne alebo vypnuté a tiež si uchovávajú inovačné číslo, ktoré charakterizuje históriu génu. Táto myšlienka rieši *Competing Conventions Problem* [12], ktorý poukazuje na otázku: Ako krížiť dve siete rozumným spôsobom, aby sa nekrížili gény, ktoré majú rôzny pôvod a teda aj funkciu. Inovačné číslo zachytáva históriu génu a ak dva gény majú rovnakú históriu, považujú sa za homologické, čo má zmysel pri identifikácii génov počas kríženia. Kríženie v NEAT znamená pridávanie nového uzlu alebo vznik nového spojenia.

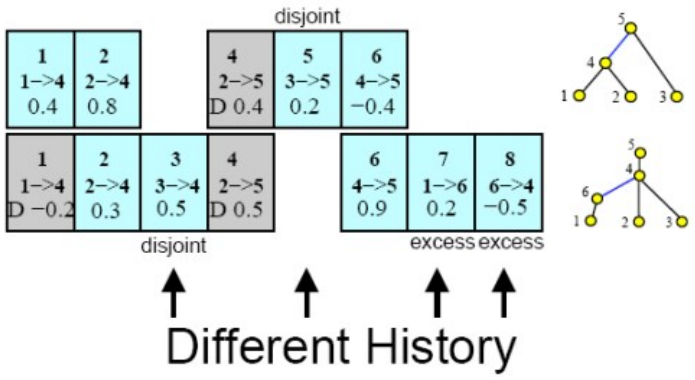
Speciation

Za účelom ochrániť užitočné inovácie v sieti, napriek tomu, že spočiatku sa zdajú byť nevýhodné, zavádza NEAT ochranu inovácií. Pridaním nového uzla resp. mutácia niektorého génu je nová sieť spočiatku neefektívna a trvá dlhší čas (vzhľadom na topológiu), než sa takáto sieť optimalizuje.

NEAT používa clusterizáciu jedincov na základe vzájomnej kompatibility (2.1), ktorá sa počíta ako lineárna kombinácia génov s vyšším inovačným číslom, ako maximálne inovačné číslo porovnávaného jedinca E (excess), génov, ktoré sú v niektorom s jedincov nad-

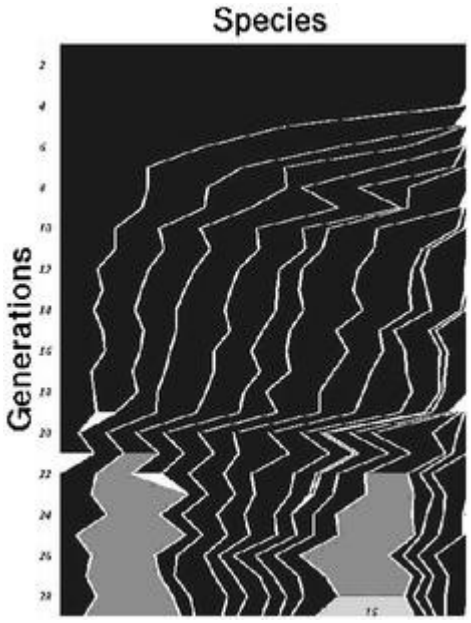
bytočné D (disjoint) a priemerného rozdielu váh odpovedajúcich si génov $\overline{W}$. Konštanty c_1, c_2 a c_3 určujú dôležitosť jednotlivých faktorov a N je počet génov väčšieho genotypu.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \overline{W} \tag{2.1}$$



Obr. 2.6: Kompatibilita dvoch jedincov v NEAT

Na základe stanoveného *compatibility thresholdu* sa potom siete rozdelia do druhov *species*. Jedinci sa potom množia a súťažia v rámci svojho druhu, podobne ako je to v prírode. Zdieľajú spoločnú fitness hodnotu, čo sa nazýva *explicit fitness sharing*. [9]



Obr. 2.7: NEAT speciation

Fitness sharing

- Chráni aby jeden druh neovládol celú populáciu, udržuje diverzitu druhov
- Fitness jedného druhu sa počíta ako priemer skutočnej fitness jedincov

- Podľa skutočnej fitness sa určuje pomer jedincov v rámci druhu, čo zabraňuje nadmerné klonovanie niektorého úspešného jedinca v druhu
- Počíta sa aj rozšírená fitness, ktorá závisí od kompatibility jedinca s každým ďalším jedincom v populácii

Complexification

Pri neurónový sieťach bolo preukázané, že na mnohých príkladoch sú siete so zložitou topológiou menej efektívne ako menšie siete, pretože na optimalizáciu veľkej siete je potrebné množstvo času. Väčšie siete sa preto často penalizujú vo fitness funkcii. Väčšina TWEANN modelov inicializuje prvé siete s náhodnou topológiou, čo síce zabezpečí okamžitú diverzitu, avšak tento prístup sa, vzhľadom na pomalú optimalizáciu, ukázal ako nevhodný.
NEAT inicializuje vždy minimálnu topológiu – všetky vstupné uzly sú priamo napojené na všetky výstupy. Zložitejšie topológie sú uchované, len ak sa ukážu ako užitočné. Toto nesie dve základné výhody:

- Riešenie sa hľadá už na triviálnej topológii, čo zabraňuje plytvaniu času na rozsiahlych sieťach.
- Limitovanie dimenzionality prehľadávaného priestoru počas evolúcie

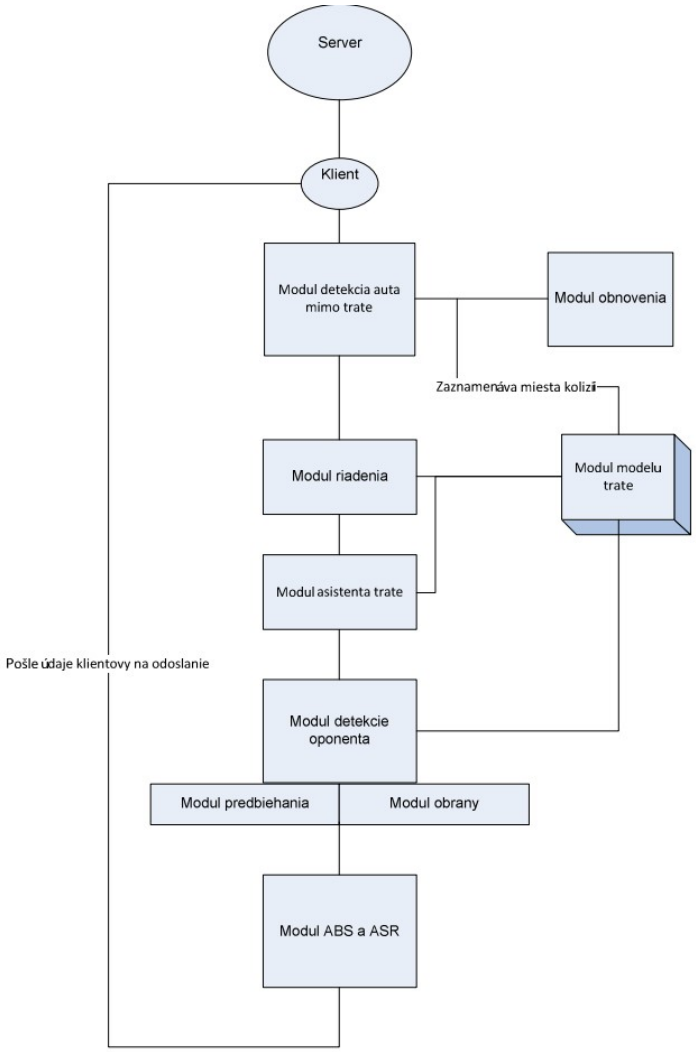
Autori aplikujú svoj model NEAT na riešenie XOR experimentu a tiež na balansovanie opačných kyvadiel (Double pole balancing with/without velocities [19]), kde dosiahli výborné výsledky. Ďalšie smerovanie NEAT prinieslo niekoľko modifikácií ako napríklad rtNEAT (real-time NEAT), ktorá sa používa najmä v hrách.

2.3 Existujúce riešenia

V tejto kapitole predstavujem niekoľko existujúcich návrhov pilotov, ktoré som študoval, porovnával. Hodnotím to ako veľmi užitočné, vyhol som sa mnohým komplikáciám, na ktoré poukazujú tieto riešenia, našiel som množstvo nových informácií a inšpiráciu pre moju prácu.

2.3.1 READY2WIN

Implementácia autopilota z roku 2011. Ide o prácu siedmich bakalárov z FIIT STU v Bratislave. S týmto riešením autori obsadili konečné 3. miesto. V [3] tvorcovia navrhli pre svojho pilota modulárnu architektúru zobrazenú na obrázku 2.8.



Obr. 2.8: Ready2win modulárna architektúra

- **Modul riadenia** – má na starosti prispôsobovať rýchlosť a smer auta. Rýchlosť je počítaná ako maximálna rýchlosť v, pri ktorej je auto ešte schopné ubrzdiť na brzdnej dráhe b pri tiažovom zrýchlení g a frickcii μ[1].

$$v = \sqrt{2g\mu b} \tag{2.2}$$

  Modul zabezpečuje prerad'ovanie prevodových stupňov, detekciu pilota mimo trate – v tomto prípade aktivuje modul obnovy. Úlohou je tiež detekcia oponentov na trati a voľba modulu predbiehania oponenta alebo obrany.

- **Modul obnovenia** – Ak sa automobil dostane mimo trať, aktivuje sa tento modul, ktorý na základe informácií o natočení auta voči trati a informácií, v ktorej časti mimo trati sa nachádza, má za úlohu čo najrýchlejšie vrátiť auto na trať.

[1] Koeficient trenia medzi kolesami vozidla a vozovkou

- **Modul modelu trate** - modul zaznamenáva prejdenú trasu. Každým kolom sa model aktualizuje, zaznamenávajú sa miesta kolízií. Boli navrhnuté viaceré modely na reprezentáciu trate ako napr. 2D mapa, model zákrut pomocou diaľkomerov, pomocou osi trate, ale tieto modely zlyhali na nepresnosti resp. relatívnej povahe dát zo senzorov. Nakoniec boli zákruty rozpoznávané podľa natočenia volantu. Vrchol zákruty bol určený ako postupnosť málo meniacich sa hodnôt točenia volantom pri čo najprudšom vytočení. Časom bol doplnený modul podpory riadenia pred vjazdom do zákrut, ktorého úlohou je upraviť smer auta pred vjazdom do zákruty.
- **Modul asistenta trate** – určovanie ideálnej stopy na trati
- **Modul detekcie oponenta** – reakcia na dobiehajúceho alebo približujúceho oponenta, volí sa buď prebiehanie alebo obrana
- **Modul predbiehania** – aktivuje predbiehanie oponenta
- **Modul obrany** – obrana pred predbehnutím
- **Modul ABS a ASR** – zabránenie súvislému šmyku a preklzávaniu kolies pri zrýchľovaní

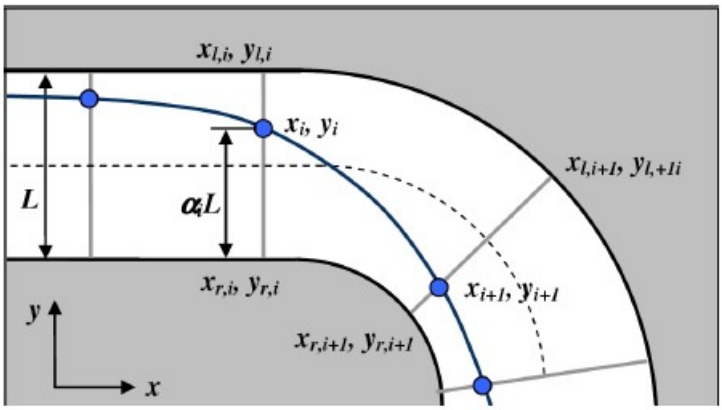
Zhodnotenie

Toto riešenie autopilota mi poskytlo mnoho dôležitých rád, avšak absentuje tu umelá inteligencia, ktorá je tu len na úrovni rozhodovania medzi modulmi. Tiež je veľmi málo uplatnené učenie. Dôraz sa kládol na experimentálne výsledky ručného programovania. Tím vytvoril niekoľko verzií pilota, vrátane prototypu, z ktorého som čerpal pri tvorbe ručne naprogramovaného pilota.

2.3.2 Automatic Racing Lines Generation For High-End Car Games

Práca Alessandra Pietra Bardelliho [2] zameraná na teóriu hľadania optimálnej trajektórie. Ide o rozsiahlu štúdiu vychádzajúcu predovšetkým z Braghinovej práce. [5] Nájdenie optimálnej trajektórie je jedna z kľúčových tém pri tvorbe akejkoľvek závodnej hry. Je dôležité si uvedomiť, že optimálna trasa nezávisí len od geometrického modelu trate, ale aj od parametrov a fyzikálnych vlastností automobilu, ktorý ju má absolvovať. Braghin vo svojej štúdii rozdelil problém na dve časti, hľadanie optimálnej stopy na čisto geometrickom základe a v druhom prístupe berie do úvahy aj parametre auta. Predpokladá, že optimálna trajektória je kombinácia najkratšej trasy a trasy s najmenšou zákrutovosťou, čiže takej, kde sú oblúky čo najhladšie.

Ako prvé, je potrebné urobiť diskretizáciu trate na uniformné segmenty. Trajektória je potom postupnosť bodov P_1 až P_n, ktoré sú pospájané. Každý bod je reprezentovaný dvojicou $\langle \delta_i, \alpha_i \rangle$, kde δ_i predstavuje vzdialenosť bodu od štartu pozdĺž osi trate a α_i, predstavuje laterálnu vzdialenosť od jedného okraja trate, normalizovanú k šírke trate.



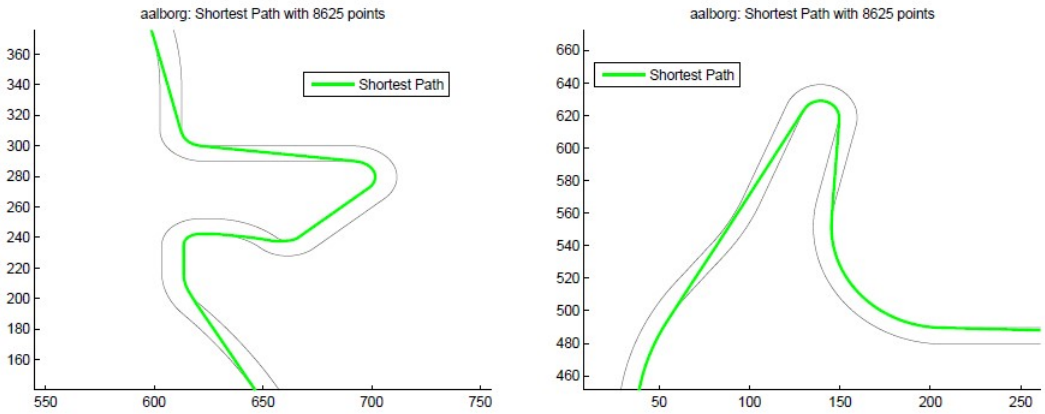
Obr. 2.9: Diskretizácia trajektórie

Najkratšia trajektória

Predstavuje postupnosť bodov, ktoré umožňujú prejsť jedno kolo na trati s minimálnou vzdialenosťou. Na základe práce bol formulovaný vzťah na výpočet najkratšej cesty. Problém sa dá vyriešiť pomocou kvadratického programovania nasledovne:

$$min_\alpha \quad \alpha^T H_S \alpha + B_S \alpha \quad 0 \leq \alpha_i \leq 1 \ \ \forall i \in \{1, ..., n\}, \tag{2.3}$$

kde H_S je $n \times n$ matica, B_S je $1 \times n$ vektor, závisiaci priamo od súradníc okraja trate a α je vektor $\langle \alpha_1, \alpha_2, ..., \alpha_n \rangle^T$. Autor riešil túto minimalizáciu pomocou MatLab-u. Táto optimalizácia je kvadraticky úmerná segmentom trate, matica H_S je však triangonálna, čiže sa dá zadefinovať ako *sparse*.



Obr. 2.10: Nájdená najkratšia trajektória

Trajektória s minimálnou zákrutovosťou

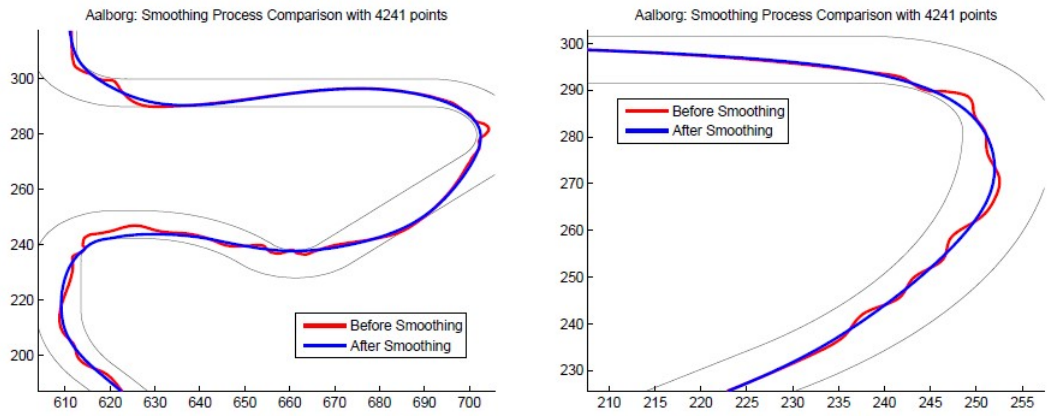
Tu rozumieme trajektóriu, ktorej zákruty sú čo najoblejšie, čiže automobil v nich môže ísť vyššou rýchlosťou, ako keby boli ostré. Plynulé zákruty sú hlavný faktor hľadania optimálnej trajektórie. Zákrutovosť vieme vypočítať ako inverznú funkciu polomeru zákruty. Najmenej zákrutovú trajektóriu vypočítame ako sekvenciu bodov, ktoré minimalizujú $\tilde{\Gamma}^2$:

$$\tilde{\Gamma}^2 = \sum_{i=1}^{n} \tilde{\Gamma}_i^{\,2}, \tag{2.4}$$

kde $\tilde{\Gamma}_i$ predstavuje zakrivenie v bode P_i kubickej krivky (*cubic spline*), ktorá interpoluje trajktóriu medzi bodmi P_i a P_{i+1}. Po úpravách dostávame kvadratický problém, ktorý podobne ako pri výpočte najkratšej trajektórie vyriešime pomocou kvadratického programovania:

$$min_\alpha \quad \alpha^T H_\Gamma \alpha + B_\Gamma \alpha \quad 0 \leq \alpha_i \leq 1 \;\; \forall i \in \{1, ..., n\}, \tag{2.5}$$

kde H_Γ je $n \times n$ matica, B_Γ je $1 \times n$ vektor, ktorý vypočítame priamo zo súradníc niektorého z okrajov trate a α je vektor $\langle \alpha_1, \alpha_2, ..., \alpha_n \rangle^T$. Znovu bolo na túto optimalizáciu použité prostredie MatLab, avšak surové výsledky ukázali, že nájdená trajektória obsahuje množstvo oscilácií a nežiadúcich zakrivení, ktoré spôsobujú veľmi negatívny efekt, čo sa týka času na prejdenie tejto trajektórie. Preto autor používa ďalšie spracovanie výsledku. Bola použitá vyhladzovacia metóda, ktorá pri zvolení vhodného vyhladzovacieho parametra upraví nájdenú trasu (obrázok 2.11).



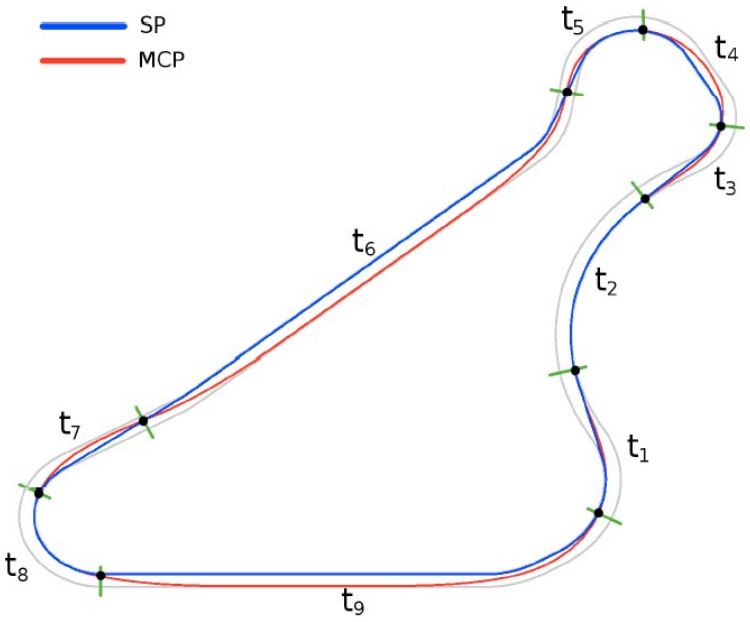
Obr. 2.11: Nájdená najplynulejšia trajektória

Optimimálna trajektória

Po nájdení najkratšej a najmenej zákrutovej trajektórie ostáva posledný krok: nájsť optimálnu trajektóriu, čo podľa Braghina je pomer medzi týmito dvoma trajektóriami. Čiže pre každý bod P_i určíme laterálnu vzdialenosť od okraja α_i ako kombináciu najkratšej a najmenej zákrutovej trasy:

$$\alpha = (1-\varepsilon)\cdot\alpha_{MCP} + \varepsilon\cdot\alpha_{SP} \quad 0 \leq \varepsilon \leq 1, \tag{2.6}$$

kde α_{MCP} predstavuje najmenej zákrutovú trasu a predstavuje α_{SP} najkratšiu trajektóriu. ε predstavuje váhu daného riešenia.
Braghin vo svojej práci rovnomerne vybral niekoľko hodnôt pre ε porovnal výsledky získané simuláciou a vybral najlepší koeficient. Tento prístup však postráda možnosť určiť koeficient ε pre každý úsek trate rôzny, pretože v niektorých častiach trate je vhodnejšia najkratšia trajektória a niekde opačne. Preto Bardelli rozšíril tento prístup o aplikáciu evolučného algoritmu. Trať rozdelil na na niekoľko sekcí $t_1, ..., t_m$ pričom hraničné body týchto úsekov sú priesečníky najkratšej trajektórie a trajektórie s najmenšou zákrutovosťou (obrázok 2.12).



Obr. 2.12: Rozdelenie trajektórie na sekcie

Pre každý úsek t_j sa počíta osobitná hodnota koeficientu ε. Na hľadanie riešenia (hodnôt koeficientov) bol použitý *C++ Genetic Algorithm Toolbox* [16]. Veľkosť populácie bola nastavená na 30, na selekciu bol použitý turnaj a proces evolúcie bežal 100 generácií. Evaluácia riešenia prebehla v dvoch fázach: 1) genóm bol dekódovaný na výpočet trajektórie 2) prebehla simulácia v podobe dvoch kôl na trati za použitia *Follower bota*, čo je jemná modifikácia *Simplix bota* [4].

Záver Bardelliho práce prináša porovnanie Braghinovho prístupu, riešenia s evolučným algoritmom, Simplix bota a Follower bota na trajektórii s minimálnou zákrutovosťou, na 11 tratiach. Výsledná tabuľka ukázala, že v ôsmich prípadoch najlepší čas na trati malo riešenie s evolučným algoritmom a v zvyšných prípadoch dominoval Simplix bot.

Zhodnotenie

Osobne sa mi páčila táto práca a rozhodol som sa využiť túto teóriu pre nájdenie optimálnej trajektórie.

2.3.3 A Genetic Algorithm in the Game Racetrack

Ide o bakalársku prácu [13] dvoch švédskych študentov Roberta Olssona a Andreasa Tarandiho, v ktorej sa snažia vytvoriť agenta, ktorý prejde trať bez akýchkoľvek znalostí o

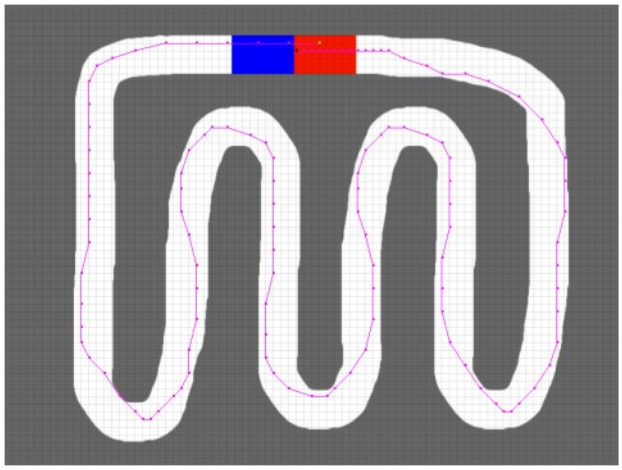
trati vopred. Racetrack[1] je klasická hra s perom na štvorčekovom papieri, avšak existujú aj rôzne počítačové implementácie. Hráč sa snaží prejsť trať v najkratšom možnom čase, pričom nesmie naraziť do steny. V každom ťahu môže hráč pokračovať v smere v predchádzajúceho ťahu a posunúť sa o rovnakú vzdialenosť ako v predošlom ťahu, alebo môže zrýchliť resp. spomaliť o jeden štvorček v každom smere.

Autori si vytvorili testovacie prostredie (okrem grafiky). Agent má k dispozícii 3 diaľkové senzory, ktoré vracajú vzdialenosť agenta od najbližšej steny. Veľkosť genómu bola stanovená fixne a závisí od počtu možných pohybov, počtu permutácií hodnôt získaných zo senzorov a tiež od počtu možností zrýchlení/spomalení.

Genetický algoritmus bol spustený na populácii 100 jedincov, pričom v každej generácii sa vybrali najlepší jedinci a tí boli krížení s ostatnými jedincami. Pravdepodobnosť kríženia dvoch rodičov bola v pomere ich fitness. Tiež bol aplikovaný operátor mutácie na náhodný gén genómu. Fitness funkcia bola po niekoľkých optimalizáciách počítaná nasledovne:

$$F(pos, n) = w \cdot goalDistance(pos) + n, \tag{2.7}$$

kde pos je pozícia agenta na trati, n je počet zákrut, w je koeficient, ktorý pri niektorých leveloch pridáva bonus, ak sa hráč dostane bližšie k cieľu a $goalDistance()$ vráti vzdialenosť agenta od cieľa.



Obr. 2.13: Nájdená trasa v racetrack

Experimenty ukázali, že pre niektoré zložitejšie trate je vhodné použiť väčší genóm, tiež sa ukázalo ako užitočné klonovať rodičov do ďalšej generácie a tým zabrániť stagnácii riešení. Oplatilo sa spúšťať algoritmus s menšou hodnotou pre mutáciu.

[1] Racetrack hra: `http://www.papg.com/show?1TPE`

Ukázalo sa, že optimálna hodnota je v rozmedzí 0.01 – 0.05.

Zhodnotenie

Páčilo sa mi na tejto práci, že základné princípy genetických algoritmov tu boli v réžii autorov a nebol len použitý predpripravený balík na genetické výpočty. Pochopil som dôležitosť kódovania genómu a keďže tu komunikácia prebiehala tiež prostredníctvom senzorov a efektorov, poskytlo mi to návod pre implementáciu v rozhraní určeným mojej bakalárskej práce.

2.3.4 Neuroevolúcia

V tejto časti by som rád prestavil dve existujúce riešenia využívajúce neuroevolúciu – konkrétne model NEAT. Ide o prácu Matt Simmersona [18], ktorý je aj autorom implementácie NEAT pre Javu s názvom Neat4J, a prácu tímu Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi [6], ktorí sa venujú vývoju autopilota už dlhší čas a navrhli niekoľko zaujímavých a úspešných návrhov.

Ako vstupy neurónovej siete boli v oboch prípadoch použité niektoré senzory dostupné v TORCS API. Taliani navrhli sieť s ôsmimi vstupmi, kde bolo zastúpených 7 diaľkomerov vzorkujúcich priestor pred autom a aktuálna rýchlosť. Simmerson použil až 29 vstupov, kde okrem viacerých diaľkomerov, použil aj laterálnu rýchlosť naprieč traťou, aktuálne otáčky motora a kolies, prevodový stupeň a pozíciu auta na trati. Vstupy boli normalizovaná na intervaly $[0,1]$ resp. $[-1,1]$ podľa povahy hodnôt zo senzorov. Toto bolo nevyhnutné, aby vysoké hodnoty napr. pre rýchlosť neodstavili ostatné vstupné signály s nižšími hodnotami. Na výstupných uzloch bola v oboch prípadoch použitá sigmoidná aktivačná funkcia. Taliani merali dva kontinuálne výstupy v intervale $[-1,1]$, pre ovládanie volantu vpravo/vľavo podľa znamienka výstupu a signál pre akceleráciu resp. brzdenie – tiež podľa polarity výstupu. Ak senzory nezachytili prekážku v dosahu 100 metrov, plynový pedál bol automaticky nastavený na 1 a brzdový na 0. Zmenu prevodového stupňa tiež riadila naprogramovaná tabuľka, kde sa podľa aktuálnych otáčok motora zmenil stupeň nahor alebo nadol. Simmerson použil o jeden výstup naviac pre indikáciu zmeny prevodového stupňa.

Piloti boli evolvovaní na trati, ktorá obsahuje komplexnú škálu zákrut a rovín. Na trati strávil každý pilot určitý počet tickov (Taliani 10000, Simmerson 4000) a v rámci hodnotiaceho kola bola počítaná hodnota fitness:

Taliani:

$$F = C1 - T_{out} + C2 \cdot \overline{s} + d, \tag{2.8}$$

kde T_{out} je čas strávený mimo trate, $\overline{s}$ je priemerná rýchlosť a d je vzdialenosť, ktorú pilot prešiel na trati. $C1$ a $C2$ boli nastavené na 1000.
Simmerson:

$$F = 2d - r - T_{out} + s_{max} + C, \tag{2.9}$$

kde d predstavuje prejdenú vzdialenosť, r zničenie auta, T_{out} čas strávený mimo trate, s_{max} maximálnu rýchlosť a C konštanta, nastavená na 10000.
Dôležitú úlohu zohrala aj politika návratu auta na trať, nastavenie veľkosti populácie a počtu generácií.

Zhodnotenie

Práce dokázali porovnateľnosť výsledkov získaných neuroevolúciou s výsledkami naprogramovaných pilotov. Poskytli mi množstvo námetov na experimenty, avšak nevýhoda je časovo náročná evolúcia, pretože každú sieť je nutné otestovať simuláciou priamo na trati.

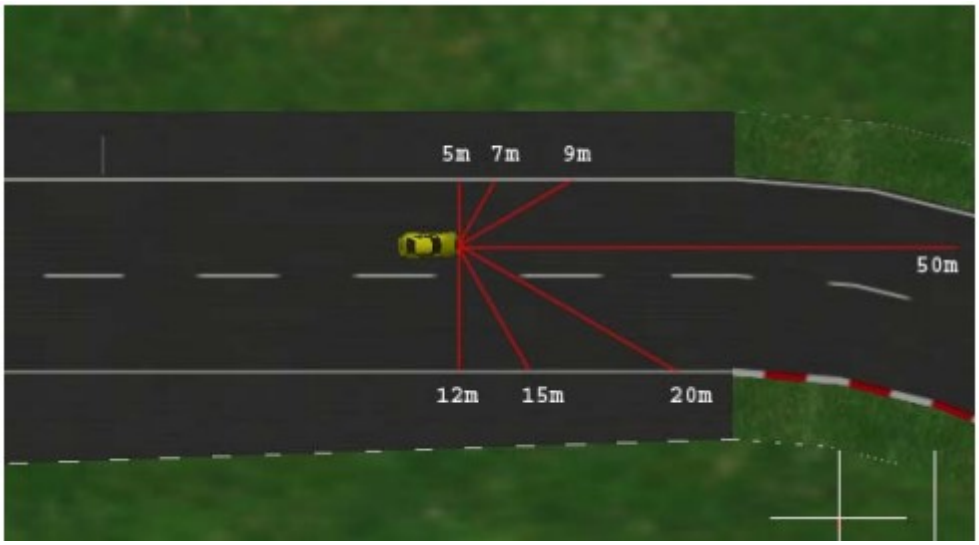
Kapitola 3

Návrh riešenia

V tejto kapitole opisujem svoje návrhy riešenia problematiky autopilota. Vytvoril som niekoľko návrhov, väčšinou inšpirovaných existujúcimi riešeniami, pretože som považoval za rozumné stavať na skúsenostiach a práci iných autorov, ktorí sa téme venujú už niekoľko rokov.

3.1 Prototyp

Bakalárska práca je zameraná predovšetkým na aplikáciu evolučných algoritmov. V záujme porovnania evolvovaného vodiča s ručne naprogramovaným jazdcom, bolo potrebné vytvoriť návrh jednoduchého, ale úspešného pilota, ktorý sa riadi striktne naprogramovanou stratégiou. Pilot ovláda volant, spojkový, brzdový a plynový pedál. Prototyp bol do veľkej miery inšpirovaný riešením [3].



Obr. 3.1: Diaľkomery v TORCS

Točenie volantom

Viaceré tímy počítajú natočenie volantu podľa senzora, ktorý vráti najvyššiu vzdialenosť od okraja (obrázok 3.1). Túto myšlienku som uplatnil v návrhu prototypu, nakoľko je to jednoduché naprogramovať a výsledky sú prekvapivo optimistické. Prototyp kontroluje všetkých 19 diaľkomerov, vyberie jeden resp. množinu tých, ktoré zaznamenajú najvyššiu hodnotu a ich uhol k osi vozidla je potom spriemerovaný a použitý na výpočet uhla, o ktorý je treba otočiť volant. Tu je veľmi dôležité si uvedomiť skutočnosť, že nesprávne alebo neprimerané mykanie volantom výrazne spomaľuje auto a ľahko ho môže dostať do šmyku, kedy pilot stráca kontrolu nad vozidlom. Aby sa vyhlo zbytočnému prudkému vytáčaniu volantu, pilot netočí volantom naraz plný uhol, ale postupne robí menšie otočenia za sebou, čo môže spôsobiť mierne vybiehanie vozidla mimo trať. Ak je potrebné otočiť volant o príliš veľký uhol, použije sa maximálne otočenie vpravo/vľavo.

Pridávanie/brzdenie

Na určenie pridávania alebo brzdenia sa používa porovnanie aktuálnej rýchlosti auta a maximálnej možnej rýchlosti (výpočet 2.2) vzhľadom na polohu vozidla. Ak je aktuálna rýchlosť nižšia ako maximálne prípustná, vodič odošle reakciu vo forme zrýchlenia. Hodnota akcelerácie je ešte filtrovaná cez ASR aby nedošlo k prekĺzavaniu kolies. Podobne ak má dôjsť k brzdeniu, použije sa filter ABS aby sa auto nedostalo do šmyku.

Prevodovka

Základný softvér klienta pre SCR obsahuje implementáciu naivného jazdca, ktorý si uchováva tabuľku hraničných otáčok motora, pri ktorých dochádza k zmene prevodového stupňa. Tím Ready2Win prostredníctvom experimentov mierne upravil túto tabuľku a takto upravenú ju používa aj môj prototyp. Používa sa optimalizácia v podobe pamätania histórie posledných otáčok motora a k zmene prevodového stupňa dôjde len v prípade, že otáčky majú stúpajúci alebo klesajúci trend. Toto zabraňuje častému prerad'ovaniu, ktoré spomaľuje automobil. Spojkový pedál je použitý len pri štarte, kedy sa postupne uvoľňuje, čo spôsobí plynulejší a agresívnejší štart.

Návrat na trať

Ak sa auto nachádza mimo vozovky resp. vo veľkom uhle voči osi trate a tento stav trvá niekoľko tickov, použije sa návrat na trať implementované v naivnom vodičovi dodávaného autormi klientskeho softvéru.

3.2 Follower jazdec

V tomto návrhu nehrá rozhodujúcu úlohu vodič, ktorý v podstate len nasleduje nájdenú trajektóriu, z čoho je odvodený aj názov pre tento návrh. Ide o napodobnenie riešenia spomínaného v kapitole 2.3.2. Je však jeden zásadný rozdiel. Celý matematický model tohto návrhu pracuje s údajmi o trati, najmä súradnicami, vzdialenosťami a pod. Autori spomínaného riešenia dospeli k záveru, že je výhodnejšie čítať tieto údaje z XML súborov, ktoré popisujú jednotlivé trate. Tento prístup neumožňuje použiť riešenie v súťaži, kde sú trate vopred neznáme a účastníci nemajú prístup k zdrojovým súborom. Preto sa moje riešenie opiera o údaje namerané v prostredí TORCS.

Najskôr je potrebné, aby nejaký jednoduchý jazdec prešiel pomaly trať a snažil sa ísť čo najviac po osi trate. Má za úlohu namerať vzdialenosti od okrajov trate, z čoho sa dá potom vypočítať šírka vozovky a vytvoriť 2D mapa. Aby bol model presnejší, vodič prejde trať niekoľkokrát a vypočíta sa priemer odpovedajúcich hodnôt. V ďalšom kroku musíme trať rovnomerne rozdeliť na úseky. Čím bude vzorkovacia frekvencia vyššia, model bude presnejší, ale výpočty budú časovo náročnejšie. Následne zostrojíme potrebné matice a pomocou kvadratického programovania dostaneme údaje najskôr pre najkratšiu trajektóriu a potom pre najmenej zákrutovú trasu. Identifikujú sa priesečníky týchto dráh a podľa spoločných bodov sa trať rozdelí na niekoľko úsekov. Výsledná stopa, ktorú má vodič nasledovať, sa vypočíta evolučným algoritmom tak, ako v kapitole 2.3.2, ktorý pre každý úsek nájde kombináciu najkratšej a najplynulejšej trajektórie.

Následne sa už zostrojí jednoduchý jazdec, ktorý dostane údaje o stope, a v čo najvyššej presnosti sa pokúsi túto stopu nasledovať. Porovnáva sa aktuálna pozícia auta na priečne na trati s požadovanou a podľa toho sa otáča volantom. Akcelerácia, brzdenie a zmena prevodového stupňa je počítaná ako v návrhu prototypu.

3.3 NEAT jazdec

Návrh finálneho pilota. Ide o riešenie postavené prevažne na sile evolučných výpočtoch. Informácie o trati sú nepodstatné, keďže ide o adaptáciu vodiča.

Využíva sa tu idea NEAT na naučenie pilota jazdiť na trati. Ako vstup siete sa vezmú údaje z diaľkomerov, ktoré snímajú priestor pred autom v zornom poli 180°. Do úvahy sa berie aj aktuálna rýchlosť. Zmena rýchlostného stupňa sa riadi ako v predošlých návrhoch – podľa tabuľky s otáčkami motora. Obnova vodiča na trať využíva default postup

naivného vodiča. Na začiatku sa vytvorí populácia neurónových sietí s čo najjednoduchšou topológiou, každá sieť je vyhodnotená prostredníctvom simulácie v TORCS, z ktorej potom môžeme vypočítať fitness hodnotu pre každého jedinca. Nasleduje zaradenie do druhu, aktualizácia údajov a generovanie potomkov. Cez tento proces sa pilot zdokonaľuje.

Tento návrh je pomerne jednoduchý, avšak ponúka široké pole optimalizácií. Zmenou počtu vstupov siete sa mení náročnosť učenia takejto siete, avšak je tu predpoklad lepších výsledkov. Rôzne nastavenia kompatibility jedincov znamenajú zmeny v počte a početnosti druhov, viac druhov umožňuje väčšiu variabilitu, ale aj slabé riešenia. Pravdepodobnosť mutácií, veľkosť populácie, množstvo generácií, toto všetko dokáže výrazne ovplyvniť výkon modelu NEAT. Tieto nastavenia sú ťažiskom tohto návrhu.

Kapitola 4

Realizácia

Kapitola zachytáva priebeh implementácie návrhov, vyberá zaujímavé implementačné problémy, ktoré som riešil a prináša zoznam optimalizácií pre každý návrh. Testovanie prebehlo na počítači *Intel Core i7-2670QM (quad-core), 8 GB RAM, 500 GB HDD*.

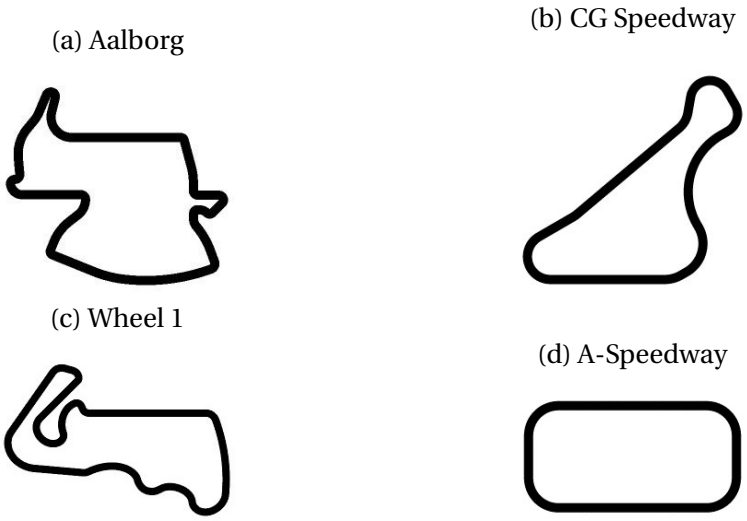
4.1 Prototyp

Implementácia

Implementácia prototypu prebehla hladko a rýchlo. Po dôkladnej štúdii zdrojového kódu tímu Ready2Win a implementácie naivného autopilota, som bol schopný skombinovať tieto riešenia.

Testovanie

Prototyp bol odskúšaný na niektorých tratiach (obrázok 4.1), výsledky zobrazuje tabuľka 4.1. Problém nastával, ak sa vodič dostal mimo trať. Stratégia obnovy vodiča bola jednoduchá a občas zlyhávala, čo sa prejavilo na pomerne veľkej diverzite výsledkov v rámci jednej trate.



Obr. 4.1: Trate distribuované TORCS

Tabuľka 4.1: Výsledky prototypu - aritmetický priemer z 5 kôl

Trať	**Čas (s)**
Aalborg	93.848 ± 5.578
CG Speedway	45.97 ± 0.79
Wheel 1	99.262 ± 0.522
A-Speedway	62.73 ± 5.8

Optimalizácie

- Zmena hodnoty frikcie – zvýšenie prinieslo rýchlejší prejazd, ale vyššie riziko šmyku
- Spojka pri štarte – postupné uvoľňovanie spojkového pedálu zrýchlilo štart
- História otáčok motora – algoritmus vyhodnocuje otáčky za posledných 30 tickov, čo zabraňuje príliš častému prerad'ovaniu

4.2 Follower jazdec

Implementácia

Najskôr bol naprogramovaný jednoduchý vodič, ktorého úlohou bolo prejsť pomaly trať a uložiť údaje zo senzorov do súboru. Tu bolo treba vyriešiť niekoľko problémov:

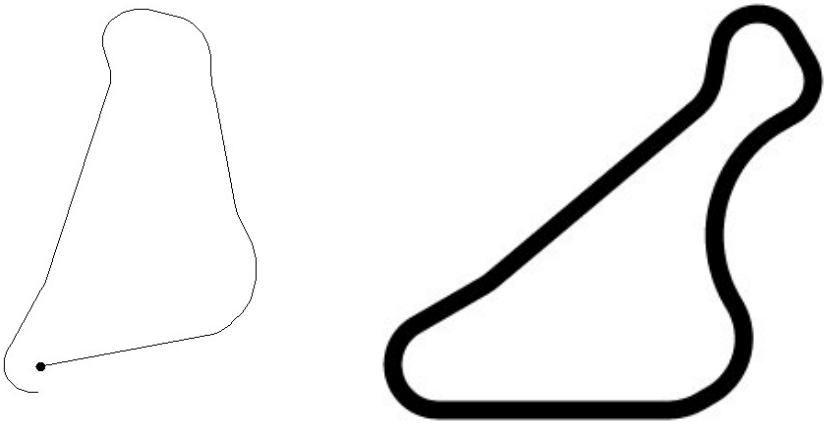
- Údaje sú vždy relatívne vzhľadom na aktuálnu pozíciu auta, poznáme uhol k osi trate a vzdialenosť pozdĺž osi trate, ale natočenie volantu nie vždy korešponduje so

zakrivením trate – môj vodič meral hodnoty, len keď bol mierne odchýlený od osi a mal vyrovnaný volant. Táto metóda je niektorých tratiach nepoužiteľná.

- Dáta zo senzorov sú aj pri vypnutom šume málo presné na vytvorenie presného 2D modelu, každá odchýlka spôsobí chybu, ktorá sa ďalej zväčšuje, čím je výsledok veľmi nepresný (obrázok 4.2a). O to viac, čím je trať členitejšia. Riešil som to neuniformným vzorkovaním trate, čiže v zákrutách som meral hodnoty častejšie a na rovine menej, čím sa výsledok výrazne zlepšil. Porovnanie prináša obrázok 4.2.

(a) CG Speedway - generovaná

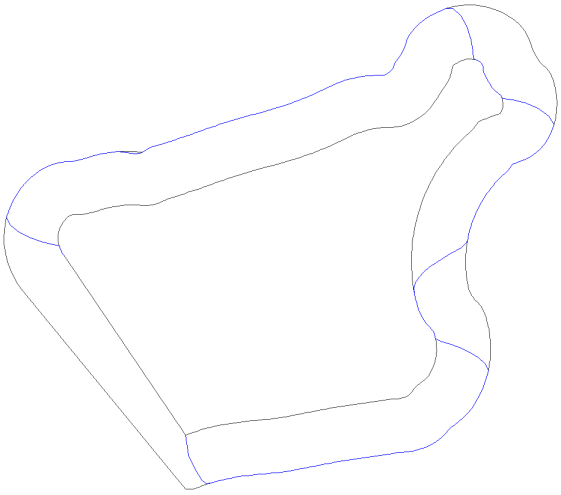
(b) CG Speedway - pôvodná



Obr. 4.2: Porovnanie generovanej a pôvodnej trate CG Speedway

Napriek nepresnostiam v zozbieraných informáciách som pokračoval v implementácii hľadania najkratšej cesty. Vytvoril som potrebné matice podľa postupu v [2] som vytvoril matice H_S, B_S. Na prácu s maticami som použil Java knižnicu Colt [8], určenú pre vedecké a technické výpočty. Podobne ako autori v MatLab-e, aj ja som využil možnosť zadefinovať maticu ako *sparse*, čiže „riedku“. To urýchli výpočty nad maticou. Nasledoval proces minimalizácie α (výpočet 2.3).

Naštudoval som si teóriu minimalizácie kvadratického problému a pomocou knižnice JMSL [14] som minimalizoval svoj problém. Následne som vytvoril trajektóriu (obrázok 4.3 modrá čiara).



Obr. 4.3: Nájdená najkratšia trasa

Kvalita výsledku odpovedala presnosti údajov, ktoré boli namerané. Miesta, kde stopa prechádza z jednej strany dráhu na druhú nie sú započítavané do dĺžky trasy z dôvodu nerovnomerného rozdelenia trate na úseky.

V implementácii som už nepokračoval ďalej. Hľadanie najplynulejšej stopy by sa riešilo obdobne, avšak tam aj pri presných údajoch je potrebné výsledné hodnoty ešte upravovať pomocou filtrov. Na základe výsledkov najkratšej cesty bolo evidentné, že tento návrh moc rozsiahly a časovo by som to nestíhal. Taktiež je tu využitie evolučného algoritmu až v závere, čo som považoval za mierny nesúlad so zadaním práce.

Testovanie ani ďalšie optimalizácie neprebehli. Vyskúšal som si tento prístup a pochopil som, že z hľadiska bakalárskej práce nemá zmysel pri ňom strácať ďalší čas.

4.3 NEAT jazdec

Implementácia

Použil som NEAT framework pre jazyk Java NEAT4J [17]. Je to implementácia Matta Simmersona, inšpirovaná pôvodným návrhom Kenntha O Stanleyho. Medzi základné výhody tohto frameworku patria

- Jednoduchá konfigurácia NEAT parametrov pomocou konfiguračných súborov
- Výborná modularita, flexibilita na zadaný problém
- Distribuované učenie cez remote prístup

Konfigurácia siete

Stanovil som počet vstupov na 12 – 7 hodnôt z diaľkomerov vzorkujúcich priestor pred autom po 30°, pozícia vzhľadom na šírku trate, uhol k trati, rýchlosť po všetkých osiach x, y, z. Výstupy boli dva – pridávanie/brzdenie a točenie volantom. Následne som nakonfiguroval konfiguračné súbory frameworku. Veľkosť populácie bola stanovená na 100 a počet generácií na 300. Počet druhov bol 3. Sieť začína s minimálnou topológiou, kde je každý vstup priamo prepojený na každý výstup. Vstupy boli normalizované na hodnoty 0..1 resp. −1..1 a vstupné uzly sú lineárne. Skrytá vrstva používa $Tanh$ aktivačnú funkciu a výstupné uzly $Sigmoid$ funkciu, upravenú na kontinuálne výstupy −1..1. Rekurentné spojenia boli povolené. Fitness funkcia som podobne ako autori v [6] počítal nalsedovne:

$$F = C1 - (C2 - T_{out} + C2 \cdot \overline{v} + d), \tag{4.1}$$

kde T_{out} je čas strávený mimo trate, $\overline{v}$ priemerná rýchlosť, d prejdená vzdialenosť. Konštanty $C2$ a $C3$ boli zavedené, aby výsledná hodnota bola kladná. Nastavil som ich na počet tickov strávených pilota na trati. $C1$ zabezpečuje aby úspešnejší jedinec mal nižšiu hodnotu fitness ako slabší jedinec. Táto inverzia je potrebná kvôli predčasnej terminácii učeniu, ak najlepší jedinec dosiahne fitness napr. 0,5.
Mutácia, kríženie a selekcia boli zdedené z frameworku. Selekcia prebieha pomocou turnajov, pričom najlepší jedinec sa vždy uchováva.

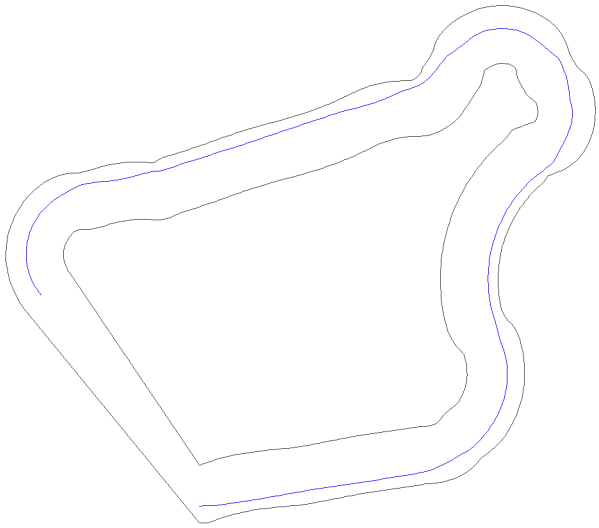
Trénovacie prostredie

Hodnota fitness funkcie závisí od hodnôt, ktoré si nevyhnutne vyžadujú simuláciu jedinca na trati. Softvér pre TORCS obsahuje možnosť spustiť program v textovom móde, kedy simulácia prebehne rýchlejšie (cca 6 sekúnd reálneho času pre 10000 tickov). Pri vyhodnotení jedinca sa spustí nový proces a pomocou *mutex-u* sa čaká až na ukončenie simulácie, vypočíta sa fitness hodnota, ktorá sa uloží do globálnej triedy viditeľnej aj pre proces, v ktorom beží trénovanie. Semafor sa otvorí, reštartuje sa TORCS server a pokračuje vyhodnotenie ďalšieho jedinca. Tento proces zachytáva nasledujúca ukážka zdrojového kódu:

```
public double evaluate(Chromosome genoType) {
        this.createNetFromChromo(genoType); // vytvorí sa sieť na základe genotypu
        this.save("genotype.ser", genoType); // uloženie objektu
        Mutex.close(); // proces si berie mutex
        // spustenie simulácie
        Client.main(new String[]{"scr.NEATDriver", "maxSteps:10000"});
        while(!Mutex.isOpen()) // čaká sa, kým simulácia skončí
        return ActualFitness.fitness; // prečíta sa statická hodnota fitness
}
```

Testovanie

Spočiatku som sa oboznamoval s frameworkom a vytvoril si niekoľko jednoduchších úloh, ktoré som sa snažil riešiť. Medzi inými aj napr. hľadanie optimálnej trajektórie, kde bola použitá jednoduchá fitness funkcia, ktorá počítala vážený priemer zakrivenia stopy a jej dĺžku. Jeden z výsledkov vidno na obrázku 4.4.



Obr. 4.4: Trajektória nájdená pomocou NEAT

Akonáhle som použil viac výstupov siete, NEAT4J knižnice hlásili chybu a po dlhom hľadaní som v distribuovanej verzii našiel chybu, ktorú som opravil. Problém sa vyskytol aj v samotnom softvéri TORCS. Ak beží program v textovom režime, po každej simulácii sa server reštartuje, aby sa mohol vyhodnotiť ďalší jedinec. Tieto reštarty však nie sú úplné a spôsobujú narastajúcu spotrebu operačnej pamäti, čo pri 200 generáciách predstavuje

asi 2GB pamäti, čo spôsobí pád programu a zastavenie učenia. Toto je veľmi nepríjemné, najmä ak trénovanie beží niekoľko hodín. Finálny vodič bol trénovaný na trati Wheel 1 (obrázok 4.1c), pretože som ju považoval za komplexnú vďaka takmer všetkým typom zákrut a aj dlhších rovných úsekov. Avšak niektoré trate (obrázok 4.1a) tento vodič nezvládol. Ideálne by bolo počas trénovania použiť viac tratí, to ale softvér TORCS nedovoľuje.

Optimalizácie

NEAT poskytuje veľký priestor na zmenu parametrov. Konkrétne implementácia NEAT4J používa konfiguračné súbory obsahujúce parametre uvedené v prílohe (tabuľky 4.3, 4.4, 4.5). Skúšal som trénovať na populácii 150 jedincov, no neprinieslo to výrazné zlepšenie, akurát dlhšie trénovanie. Podstatné zlepšenie prinieslo zníženie počtu vstupných uzlov. Uvažoval som len 7 diaľkomerov a rýchlosť po osi trati. Takáto sieť má jednoduchšiu topológiu, čo umožňuje rýchlejšie natrénovanie aj vzhľadom na problém zastavenia programu spomínaného vyššie. Zmena pravdepodobnosti mutácie, kompatibility jedincov priniesla niekedy lepšie výsledky, ale v mnohých prípadoch horšie, nakoniec zostali hodnoty v default podobe. Použil som aj modul recovery od tímu Ready2Win, ktorý rýchlejšie vráti vodiča na trať. Neprinieslo to zásadnú zmenu, lebo pri trénovaní majú všetky jedince rovnaké podmienky aj po vylepšení obnovy vodiča.

4.4 Výsledky

Navrhnutí piloti (prototyp, NEAT jazdec) boli porovnaní s existujúcimi riešeniami (Follower a Simplix bot) na niekoľkých tratiach. Každý pilot jazdil osamote 5 kôl a následne bol vypočítaný priemerný čas v sekundách a tiež odchýlka. Trate som vybral tak, aby boli zastúpené náročné kľukaté dráhy a tiež plynulé. Výsledky možno vidieť v nasledujúcej tabuľke 4.2. V troch prípadoch dominuje Follower bot používajúci presný matematický model trate a zvyšnú trať vyhral Simplix. Mne sa však podarilo natrénovaným pilotom (NEAT) prekonať ručne naprogramovaného (Prototyp), čo poukazuje na fakt, že natrénovaný pilot pomocou neuroevolúcie dokáže konkurovať riešenia, postavené na odborných teóriách z oblasti riadenia automobilu.

Tabuľka 4.2: Porovnanie pilotov. Výsledky sú v sekundách.

Trať	**Prototyp**	**NEAT**	**Follower**	**Simplix**
Aalborg	93.848 ± 5.578	250.384 ± 30.204	69.928 ± 0.023	70.762
CG Speedway	45.97 ± 0.79	42.268 ± 0.012	39.372 ± 0.003	40.120
Wheel 1	99.262 ± 0.522	90.75 ± 0.33	74.887 ± 0.109	73.924
A-Speedway	62.73 ± 5.8	31.518 ± 0.022	24.701 ± 0.004	27.558

Záver

V tejto kapitole je zhrnutý priebeh práce, uvádzam tu svoj osobný pohľad na tvorbu a dosiahnuté výsledky. Naznačené sú aj niektoré ďalšie možnosti pokračovania v téme za účelom získania lepších výsledkov.

Zhrnutie

V práci som sa zaoberal témou evolučných algoritmov, konkrétne v spojení s neurónovými sieťami. Najprv som vytvoril prototyp - ručne naprogramovaného pilota - riadiaceho sa podľa diaľkomeru, vracajúceho najväčšiu vzdialenosť od prekážky. Potom pomocou frameworku NEAT4J som vytvoril prostredie na trénovanie vodiča. Nastavil som vstupy a výstupy neurónovej siete a jednotlivé operátory pre genetický algoritmus, ktorý túto sieť evolvuje. Veľmi dôležitú rolu hrá dobre zvolená fitness funkcia, ktorá na základe simulácie kandidáta na jednej z tratí v TORCS, vyhodnotí jeho výkonnosť. Po niekoľkých optimalizáciách a dostatočnom počte evolúcií som získal vodiča, ktorého výsledky prekonali výsledky prototypu a poukázali na schopnosť neuroevolúcie, konkurovať riešenia využívajúce teórie riadenia športových vozidiel.

Pohľad autora

Vybraná bakalárska téma bola pre mňa výzva. Doposiaľ som nemal žiadne informácie o strojovom učení, neurónových sieťach a evolučných algoritmoch. Strávil som veľa času štúdiom týchto tém a často som mával pocit, že to nezvládnem, ale vtedy ma podržal školiteľ. Následne som prečítal viaceré články z konferencií o evolučných výpočtoch, dokumentácie existujúcich riešení a vytváral som si obraz o problematike. Pomaly sa črtal priestor pre moje návrhy a pokusy. Pri návrhu Follower jazdca som zažíval sklamania, pretože matematický model trate bol veľmi nepresný a čas strávený štúdiom tohto prístupu sa z hľadiska výsledkov ukazoval ako premárnený. Na týchto nedostatkoch som pochopil, že procesy v reálnom prostredí ťažko napasovať do striktného matematického modelu.

Spojenie neurónových sietí a evolúcie ma oslovilo, čo viedlo k dôkladnému študovaniu idey NEAT. Po niekoľkých pokusoch na jednoduchých príkladoch sa mi podarilo vytvoriť prostredie, v ktorom sa bude autopilot schopný učiť. Výsledky ma príjemne potešili. Napriek tomu, že program po určitom počte generácií padol, podarilo sa mi natrénovať vodiča s porovnateľnými výsledkami ako mali účastníci súťaže Simulated Car Racing.

Naučil som sa hľadať riešenie komplexného problému, kde zatiaľ nepoznáme optimálny prístup. Získal som mnoho informácií a návrhov v oblasti umelej inteligencie, učenia agentov najmä v hernom prostredí, ktoré je obmedzené časom.

Ďalšie smerovania

Priestor na zlepšenie riešenia vidím v spojení výhod ručného programovania a evolúcie, čo by šetrilo čas elimináciou veľmi slabých jedincov. Taktiež vidím možné riešenie pomocou real-time neuroevolúcie rtNEAT od tvorcov pôvodnej idey NEAT. Zaujímavý je aj model multiobjektovej evolúcie, kde sa problém rozloží na opakujúce sa situácie a pre každú situáciu sa natrénuje správanie – tým dôjde k užšej špecifikácii na konkrétny problém a nebude musieť jeden model zvládať všetky situácie.

Prílohy

Prehľad konfiguračných súborov NEAT4J

Zoznam konfiguračných parametrov pre jednotlivé súbory NEAT4J. Pole „Hodnota“ predstavuje použitú hodnotu pri implementácii.

Tabuľka 4.3: Prehľad trénovacích parametrov (súbor .ga)

Názov parametra	**Popis**	**Rozsah**	**Hodnota**
PROBABILITY.MUTATION	Hodnota určuje pravdepodobnosť mutácie váhy spojení ako aj sigmoid faktor pre výstupné uzly.	0 - 1	0.25
PROBABILITY.CROSSOVER	Pravdepodobnosť kríženia jedincov v rámci jedného druhu.	0 - 1	0.2
PROBABILITY.ADDLINK	Pravdepodobnosť pridania nového spojenia.	0 - 1	0.1
PROBABILITY.ADDNODE	Pravdepodobnosť pridania nového uzla.	0 - 1	0.03
PROBABILITY.MUTATEBIAS	Pravdepodobnosť mutácie bias hodnoty neurńu.	0 - 1	0.3
PROBABILITY.TOGGLELINK	Pravdepodobnosť s akou sa môže spojenie dvoch neurónov prepínať z enabled na disabled.	0 - 1	0.0
PROBABILITY.WEIGHT. REPLACED	Pravdepodobnosť resetu váhy spojenia na určitú hodnotu bez ohľadu na aktuálnu hodnotu.	0 - 1	0.0
EXCESS.COEFFICIENT	Koeficient dôležitosti počtu génov s vyšším inovačným číslom pri počítaní kompatibility dvoch jedincov.	>=0	1

Pokračovanie na ďalšej strane

Tabuľka 4.3 – *Pokračovanie*

Názov parametra	**Popis**	**Rozsah**	**Hodnota**
DISJOINT.COEFFICIENT	Koeficient dôležitosti počtu prebytočných génov pri počítaní kompatibility dvoch jedincov.	>=0	1
WEIGHT.COEFFICIENT	Koeficient dôležitosti rozdielu váh génov pri počítaní kompatibility dvoch jedincov.	>=0	0.4
COMPATABILITY.THRESHOLD	Určuje hranicu kompatibility dvoch jedincov.	>=0	0.5
COMPATABILITY.CHANGE	Ak je 0, potom sa COMPATABILITY.THRESHOLD nezmení, čím sa neudržuje počet druhov. Ak je > 0, potom sa COMPATABILITY.THRESHOLD dynamicky mení (hore/dole) o túto hodnotu a udržuje počet druhov na SPECIE.COUNT.	>=0	0.1
SPECIE.COUNT	Definuje počet druhov.	>=1	3
SURVIVAL.THRESHOLD	Aká časť jedincov v druhu sa môže páriť. Napr. ak je hodnota 0.2, potom len najlepších 20% v druhu sa smie páriť.	>=0	0.2
SPECIE.AGE.THRESHOLD	Hranica, kedy je vek druhu penalizovaný SPECIE.OLD.PENALTY.	>=1	80
SPECIE.YOUTH.THRESHOLD	Ak je vek druhu nižší ako táto hodnota, fitness jedincov sa násobí SPECIE.YOUTH.BOOST.	>=1	10
SPECIE.OLD.PENALTY	Penalizácia fitness hodnoty. Ak je NATURAL.ORDER.STRATEGY true, hodnota musí byť >= 1, inak <= 1.	>=0	1.2
SPECIE.YOUTH.BOOST	Zvýšenie fitness hodnoty. Ak je NATURAL.ORDER.STRATEGY true, hodnota musí byť <= 1, inak >= 1.	>=0	0.7

Pokračovanie na ďalšej strane

Tabuľka 4.3 – *Pokračovanie*

Názov parametra	**Popis**	**Rozsah**	**Hodnota**
SPECIE.FITNESS.MAX	Koľko epoch môže fitness hodnota druhu zostať bez zmeny.	>=1	15
OPERATOR.XOVER	Trieda definujúca operátor kríženia.		NEAT-Crossover
OPERATOR.FUNCTION	Trieda definujúca fitness funkciu.		Torcs-Final-Fitness
OPERATOR.PSELECTOR	Trieda definujúca selekciu jedincov.		Tournament-Selector
OPERATOR.MUTATOR	Trieda definujúca operátor mutácie.		NEAT-Mutator
MAX.PERTURB	Hodnota o koľko sa môže zvýšiť/znížiť váha spojenia.	>=0	0.5
MAX.BIAS.PERTURB	Hodnota o koľko sa môže zvýšiť/znížiť hodnota bias neurónu.	>=0	0.1
FEATURE.SELECTION	Definuje počiatočnú topológiu. Ak je true, náhodný vstup sa napojí na nejaký výstup, pokiaľ všetky výstupy nie sú spojené. Ak je false, všetky vstupy sú napojené na všetky výstupy.	true/-false	FALSE
RECURRENCY.ALLOWED	Povoľuje rekurentné spojenia.	true/-false	TRUE
INPUT.NODES	Počet vstupov siete.	>=1	8
OUTPUT.NODES	Počet výstupov siete	>=1	2
NN.CONFIG	Cesta ku konfiguračnému súboru neurónovej siete	String	*.net

Pokračovanie na ďalšej strane

Tabuľka 4.3 – *Pokračovanie*

Názov parametra	**Popis**	**Rozsah**	**Hodnota**
ELE.EVENTS	Určuje, či sú ELEs (Extinct Life Events) povolené. Vynásobí sa compatiblity_threshold 5 a udrží sa n% nažive podľa ELE.SURVIVAL.COUNT.	true/-false	FALSE
ELE.SURVIVAL.COUNT	Aká časť výsledného druhu prežije.	>=0	0.1
ELE.EVENT.TIME	Počet epoch medzi ELE.	>=1	1000
KEEP.BEST.EVER	Parameter určuje, či druh s najlepším jedincom bude existovať aj po prekročení SPECIE.FITNESS.MAX.	true/-false	TRUE
EXTRA.FEATURE.COUNT	Počet génov, ktoré nemajú vplyv na topológiu ale na definovanie vstupných hodnôt pre evolúciu.	>=0	0
POP.SIZE	Veľkosť populácie.	>=1	100
NUMBER.EPOCHS	Počet epoch na trénovanie.	>=1	600
TERMINATION.VALUE	Hodnota fitness, ktorá ak je prekročená, trénovanie končí.	>=0	0.001
NATURAL.ORDER.STRATEGY	Definuje, čí sú fitness hodnoty lepšie keď sa blížia k 0 (true) alebo keď narastajú (false).	true/-false	TRUE
SAVE.LOCATION	Miesto, kam sa uloží najlepší jedinec populácie. Java serializable objekt.	String	*.ser

Tabuľka 4.4: Parametre pre aplikáciu využívajúcu natrénovanú NEAT (súbor .app)

Názov parametra	**Popis**	**Rozsah**	**Príklad**
INPUT.NODES	Počet vstupných uzlov pre NEAT	>=1	8
OUTPUT.NODES	Počet výstupných uzlov pre NEAT	>=1	2
AI.SOURCE	Cesta k uloženému chromozómu	String	*.ser
NN.CONFIG	Cesta ku konfiguračnému súboru NEAT	String	*.net
INPUT.DATA	Cesta k testovacím dátam	String	*.csv
LEARNABLE	Trieda poskytujúca učiace prostredie	String	GALearnable

Tabuľka 4.5: Nastavenie trénovacích dát (súbor .net)

Názov parametra	**Popis**	**Rozsah**	**Príklad**
LEARNABLE	Trieda poskytujúca učiace prostredie	String	GALearnable
TRAINING.SET	Cesta k trénovacím dátam	String	*.csv

Obsah CD:

↦ zdrojové súbory k programu

↦ text ku práci - bakalarskaPracaBalun.pdf

Literatúra

[1] Banzhaf. *Genetic Programming: An Introduction on the Automatic Evolution of Computer Programs and Its Applications /Wolfgang Banzhaf...[et Al.]*. The Morgan Kaufmann Series in Artificial Intelligence Series. Elsevier Science & Technology Books, 1998.

[2] Alessandro Pietro Bardelli. Automatic racing lines generation for high-end car games. Master's thesis, Politecnico di Milano, Facoltà di Ingegneria dei Sistemi, 2010.

[3] Maroš Bednár, Adam Brček, Marek Briš, Marian Florek, Vojtech Juhász, Juraj Kosmeľ, and Ivan Valenčík. *Simulated Car Racing Competition 2011*, 2011. Použité: November 2012.

[4] Wolf-Dieter Beelitz. The simply mixed best practice torcs robot. `http://www.simplix.wdbee-aorp.de/SIMPLIX/SimplixDefault.aspx`. Použité: marec 2013.

[5] F. Braghin, F. Cheli, S. Melzi, and E. Sabbioni. Race driver model. *Computers & Structures*, 86:1503–1516, 2008.

[6] Luigi Cardamone, Daniele Loiacono, and Pier Luca Lanzi. Evolving competitive car controllers for racing games with neuroevolution. In *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1179–1186, New York, NY, USA, 2009. ACM.

[7] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, 2010.

[8] CERN European Organization for Nuclear Research. Colt project, 2010. Verzia 1.2.0.

[9] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 41–49, Hillsdale, NJ, USA, 1987. L. Erlbaum Associates Inc.

[10] Marian Mach. *Evolutionary algorithms: elements and principles*. Elfa, 2009.

[11] Kristína Machová. Strojové učenie, princípy a algoritmy, 2002. `http://people.tuke.sk/kristina.machova/pdf/SU4.pdf`. Použité: január 2013.

[12] David J Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the eleventh international joint conference on artificial Intelligence*, volume 1, pages 762–767. San Mateo, CA, 1989.

[13] Robert Olsson and Andreas Tarandi. *A Genetic Algorithm in the Game Racetrack*, Stockholm 2011.

[14] Inc. Rogue Wave Software. Jmsl™ numerical library for java™ applications, 2007. Verzia 5.0.

[15] Ivan Ruttkay-Nedecký. Odvodzovanie v pravidlových systémoch, 2009. `http://www2.fiit.stuba.sk/~kapustik/ZS/Clanky0910/ruttkay-nedecky/index.html`. Použité: december 2012.

[16] Kumara Sastry. Single and multiobjective genetic algorithm toolbox for matlab in c++. *IlliGAL Report*, (2007017), 2007.

[17] Matt Simmerson. Naet4j - neuroevolution for augmenting topologies for java, 2006. Verzia 1.0.

[18] Matt Simmerson. Neat4j, 2008. Car Racing Competition at WCCI2008.

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