Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Autonómne jazdiaci agent pre hru Trackmania

Diplomová práca

2025
Timotej Melkovič

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Autonómne jazdiaci agent pre hru Trackmania

Diplomová práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	Ing. Alexander Šimko, PhD.

Bratislava, 2025
Timotej Melkovič

13699118

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Timotej Melkovič
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Autonómne jazdiaci agent pre hru Trackmania
Autonomous artificial driver for Trackmania

Anotácia: Trackmania je počítačová hra, v ktorej hráči súťažia, kto dokáže zadanú dráhu prejsť v najkratšom čase. Hra ponúka API, ktoré umožňuje vytvárať umelých autonómne jazdiacich agentov. Tvorbe takéhoto agenta bola venovaná práca [1].

Cieľ: Cieľom tejto práce je nadviazať na dosiahnuté výsledky a navrhnúť a implementovať lepšiu verziu autonómne jazdiaceho agenta. K zlepšeniu má dôjsť vo viacerých aspektoch. Samotná jazda agenta sa má zlepšiť a agent sa má časom jazdy priblížiť ľudským hráčom. Proces trénovania agenta má byť plne automatizovaný a nemá vyžadovať zásahy človeka. Agent má byť schopný jazdiť niele na dráhach, ktoré ležia v jednej rovine, ale aj na takých, ktoré obsahujú vertikálne zmeny, ako sú kopce a svahy. Agent má vedieť jazdiť na rôznych povrchoch ako asfalt, hlina a tráva. Súčasťou práce bude empirické vyhodnotenia kvality jazdy agenta na vhodne zvolenej metrike a jeho porovnanie s pôvodnou verziou agenta.

Literatúra:
1. Timotej Melkovič. Trénovanie autonómneho vozidla v hre Trackmania pomocou strojového učenia. Bakalárska práca. FMFI UK. 2024.
2. Geoff Skinner and Toby Walmsley. Artificial Intelligence and Deep Learning in Video Games A Brief Review. In 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS). Pages 404–408. IEEE, 2019.
3. Tuomas Haarnoja, et al. Soft Actor-Critic Algorithms and Applications. arXiv preprint. arXiv:1812.05905, 2018.

Vedúci: Ing. Alexander Šimko, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 28.11.2024

Dátum schválenia: 04.12.2024 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

Abstrakt

Táto diplomová práca sa zaoberá návrhom a implementáciou autonómne jazdiaceho agenta pre počítačovú hru Trackmania, ktorý dokáže efektívne prejsť rôzne typy tratí bez zásahu človeka. Nadväzuje na bakalársku prácu, v ktorej bol vytvorený základný agent trénovaný pomocou metódy učenia posilňovaním. V tejto práci sú identifikované nedostatky predchádzajúceho riešenia – najmä absencia automatického trénovacieho procesu, slabá interpretácia výstupov a nerešpektovanie obmedzení reálneho času.

Cieľom práce je vytvoriť robustnejší framework, ktorý umožní tréning autonómneho agenta v realistických podmienkach so zohľadnením oneskorení a paralelného spracovania. Riešenie zahŕňa asynchrónne riadenie interakcie medzi prostredím a modelom, zlepšenú reprezentáciu trate a návrh odmenovej funkcie, ktorá motivuje k plynulej a efektívnej jazde. Experimentálne časti práce sa zameriavajú na porovnanie rôznych architektúr a tréningových nastavení s cieľom nájsť prístup, ktorý najlepšie vyhovuje hre Trackmania.

Kľúčové slová: autonómne vozidlo, učenie posilňovaním, Trackmania

Abstract

This thesis focuses on the design and implementation of an autonomously driving agent for the computer game Trackmania, capable of completing various types of tracks without human intervention. It builds upon a bachelor's thesis in which a basic agent was developed using reinforcement learning. This work identifies several shortcomings of the previous solution — particularly the lack of an automated training process, limited interpretation of results, and disregard for real-time constraints.

The aim of the thesis is to develop a more robust framework that enables the training of an autonomous agent under realistic conditions, taking into account delays and parallel processing. The proposed solution includes asynchronous coordination between the environment and the model, an improved representation of the track, and a reward function that encourages smooth and efficient driving. The experimental part of the thesis focuses on comparing various architectures and training settings in order to find an approach that best fits the requirements of the Trackmania environment.

Keywords: autonomous vehicle, reinforcement learning, Trackmania

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Tu bude text úvodu

Kapitola 1

Teoretické východiská

1.1 Autonómny agent a riadenie v herných prostrediach

V tejto práci sa zameriavame na autonómneho agenta, ktorý sa učí riadiť vozidlo v závodnej hre. Skôr než prejdeme k samotným algoritmom strojového učenia, je užitočné ujasniť si základné pojmy, s ktorými budeme ďalej pracovať, a zasadiť problém do širšieho kontextu hernej umelej inteligencie.

Základné pojmy

V rámci učenia posilňovaním a všeobecne pri modelovaní agentov sa spravidla pracuje s nasledovnou terminológiou.[12]

- **Agent** je entita, ktorá v danom prostredí vykonáva akcie. V našom prípade ide o virtuálneho vodiča, ktorý ovláda auto v závodnej hre.
- **Prostredie** (angl. *environment*) je všetko, čo nie je agent. Zahŕňa hernú fyziku, geometriu trate, prekážky a pravidlá hry. Prostredie reaguje na akcie agenta a vracia mu spätnú väzbu vo forme nového stavu a (prípadne) odmeny.
- **Stav** (angl. *state*) opisuje aktuálnu situáciu v prostredí z pohľadu agenta. Ide napríklad o polohu a rýchlosť auta, informácie o zakrivení trate, vzdialenosť od stien či checkpointov a podobne. Ideálny stav by mal obsahovať dostatok informácií na to, aby agent vedel robiť informované rozhodnutia.
- **Akcia** (angl. *action*) je rozhodnutie, ktoré agent v danom stave urobí. V závodnej hre to môže byť nastavenie volantu, pridanie plynu alebo brzdenie. Formálne ide o prvok z množiny akcií $\mathcal{A}$.

- **Politika** (angl. *policy*) opisuje správanie agenta. Politika priraďuje každému stavu pravdepodobnostné rozdelenie na akciách, prípadne priamo konkrétnu akciu. V tejto práci budeme neskôr uvažovať politiky reprezentované neurónovou sieťou.

- **Epizóda** je jedno ucelené behanie agenta v prostredí, typicky od počiatočného stavu až po koniec hry alebo splnenie cieľa. V závodnej hre môže epizóda zodpovedať jednému pokusu o prejdenie trate.

Takto definovaný agent v simulovanom prostredí nám umožňuje formulovať úlohu riadenia auta ako optimalizačný problém: hľadáme takú politiku, ktorá vedie k čo najlepšiemu správaniu podľa zvolených metrík (čas na kolo, prejdená vzdialenosť, počet kolízií a pod.).

Klasická herná AI a učiaci sa agenti

Tradične bola herná umelá inteligencia vo videohrách vytváraná prevažne ručne navrhnutými pravidlami. Typickým príkladom sú jednoduché skripty, konečné automaty (*finite-state machines*) alebo stromové štruktúry správania (*behavior trees*), v ktorých dizajnér hry presne špecifikuje reakciu postavy na rôzne situácie. Takýto prístup má výhodu v predvídateľnosti a plnej kontrole nad správaním, ale zároveň je časovo náročný na návrh a zle sa škáluje na zložitejšie prostredia a scenáre.[13]

V posledných rokoch sa čoraz častejšie používajú tzv. učiaci sa agenti, ktorí svoje správanie nezískavajú ručne písanými pravidlami, ale učením z interakcie s prostredím. Patria sem najmä metódy učenia posilňovaním a neuroevolúcie, ktoré sa využívajú pri riadení nehráčskych postáv (NPC) a v rôznych simuláciách. Schrum a Miikkulainen napríklad ukazujú, ako je možné evolúciou neurónových sietí s viacerými cieľmi konštruovať komplexné správanie NPC v hernom prostredí.[10]

Hlavnou výhodou učiacich sa agentov je ich schopnosť prispôsobiť sa prostrediu na základe skúsenosti. Namiesto toho, aby sme pre každú možnú situáciu na trati ručne špecifikovali reakciu, definujeme iba metriku, ktorá hovorí, čo znamená „dobrá jazda“, a agent sa učí túto metriku maximalizovať. Tento prístup je obzvlášť zaujímavý v prostrediach, kde je priestor stavov a akcií veľký a dopredu nevieme ľahko navrhnúť optimálne správanie.

Závodné hry ako testovacie prostredie

Závodné hry predstavujú vhodné testovacie prostredie pre výskum autonómneho riadenia z viacerých dôvodov. Po prvé, ide o relatívne izolované a dobre definované prostredie: auto sa pohybuje po trati, ktorá má presne danú geometriu a pravidlá. Po druhé,

simulácia je lacná a opakovateľná – agent môže prejsť tú istú trať veľakrát bez rizika poškodenia reálneho vozidla. Po tretie, kvalitu jazdy vieme prirodzene merať pomocou metrík ako čas dokončenia, prejdená vzdialenosť, počet kolízií či miera opustenia trate.[13]

Tieto vlastnosti viedli k tomu, že viacero prác využíva závodné hry ako platformu pre testovanie metód učenia posilňovaním alebo evolučných algoritmov. Fuchs a kol. demonštrujú, že hlboké učenie posilňovaním môže dosiahnuť až nadľudský výkon v hre Gran Turismo Sport.[6] Podobné prístupy sa objavili aj v iných prostrediach, napríklad v rámci frameworku Unity ML-Agents,[9] kde je možné trénovať agentov v rôznych simulovaných svetoch.

Konkrétne hra Trackmania sa ukázala ako vhodný testbed pre výskum autonómnych agentov: má jednoduché, ale rýchle závodné mechaniky, presné meranie času a možnosť vytvárať vlastné trate. Viacerí autori skúmali použitie učenia posilňovaním pre riadenie auta v Trackmanii,[2, 8] pričom sa zameriavali najmä na návrh vhodného stavového priestoru, odmeny a tréningovej procedúry. Táto diplomová práca na tieto myšlienky nadväzuje, avšak skúma alternatívny prístup založený na evolučných algoritmoch a neuroevolúcii politiky agenta.

Zhrnutím, autonómneho vodiča v závodnej hre môžeme vnímať ako agenta, ktorý v každom časovom kroku pozoruje stav prostredia a rozhoduje o ďalšej akcii. Prostredie (hra) reaguje na tieto akcie a poskytuje spätnú väzbu, na základe ktorej sa agent postupne zlepšuje. V ďalších sekciách preto najprv stručne predstavíme základy strojového učenia a učenia posilňovaním, ktoré tvoria teoretický rámec pre zvyšok práce.

1.2 Strojové učenie

Strojové učenie (angl. *Machine Learning*) je oblasť informatiky, ktorá sa zaoberá návrhom algoritmov schopných učiť sa z dát a zlepšovať svoje správanie na základe skúseností. Namiesto explicitného programovania postupov pre každú situáciu sa model učí hľadať vzťahy v dátach a následne ich využívať pri predikcii alebo rozhodovaní. Cieľom je typicky minimalizovať chybu na trénovacích príkladoch a zároveň dosiahnuť dobré generalizačné vlastnosti na nových, neznámych dátach.

Typický proces strojového učenia je možné stručne popísať nasledovne. Na začiatku máme dátovú množinu, ktorá obsahuje jednotlivé príklady (vzorky). Každý príklad je reprezentovaný vektorom príznakov (angl. *features*), ktoré popisujú daný objekt alebo situáciu. Následne si zvolíme triedu modelov (napr. lineárne modely, rozhodovacie stromy, neurónové siete) a pomocou optimalizačného algoritmu hľadáme také parametre modelu, ktoré na tréningových dátach dosahujú čo najlepší výsledok podľa zvolenej stratovej funkcie. Časť dát si odložíme na validáciu alebo testovanie, aby sme

vedeli overiť, ako sa model správa na dátach, ktoré počas učenia nevidel.

Podľa toho, aké informácie máme pri učení k dispozícii, rozlišujeme niekoľko základných typov strojového učenia. Najčastejšie sa uvádzajú tri skupiny: učenie pod dohľadom, učenie bez dohľadu a učenie posilňovaním.[12, 1]

Učenie pod dohľadom

Pri učení pod dohľadom (angl. *supervised learning*) má každý trénovací príklad okrem vstupného vektora príznakov aj požadovaný výstup (tzv. *label*). Úlohou modelu je naučiť sa mapovanie zo vstupov na výstupy tak, aby vedel čo najpresnejšie predpovedať label pre nové, neznáme príklady. Typickými úlohami sú klasifikácia (napr. rozoznávanie písmen, kategorizácia obrázkov) a regresia (predpovedanie reálnych hodnôt, napr. cena nehnuteľnosti).

Kvalitu modelu hodnotíme pomocou stratovej funkcie, napr. strednej kvadratickej chyby alebo krížovej entropie. Optimalizácia parametrov modelu prebieha najčastejšie pomocou gradientných metód (napr. stochastický gradientný zostup), prípadne rôznych variantov evolučných alebo heuristických algoritmov.

Učenie bez dohľadu

V učení bez dohľadu (angl. *unsupervised learning*) k dispozícii nemáme požadované výstupy, ale iba samotné vstupné dáta. Cieľom je odhaliť štruktúru a vzťahy v dátach, napríklad zoskupenie podobných príkladov do klastrov, hľadanie hlavných smerov variability v dátach alebo odhaľovanie anomálií.

Typickými príkladmi úloh bez dohľadu sú klastrovanie (napr. algoritmus k-means), dimenzionálna redukcia (napr. PCA) alebo učenie vektorových reprezentácií (embeddingov), ktoré sa následne používajú v ďalších modeloch.

Učenie posilňovaním v kontexte strojového učenia

Učenie posilňovaním (angl. *Reinforcement Learning*, RL) tvorí tretiu základnú kategóriu strojového učenia. Na rozdiel od učenia pod dohľadom nemá agent pre každý vstup k dispozícii správny výstup, ale iba spätú väzbu vo forme odmeny, ktorá hodnotí kvalitu jeho rozhodnutí v čase.[12] RL sa typicky používa v úlohách sekvenčného rozhodovania, kde jedna akcia ovplyvňuje nielen okamžitý zisk, ale aj budúce situácie, do ktorých sa agent dostane.

V nasledujúcej sekcii sa budeme učením posilňovaním zaoberať podrobnejšie, keďže ide o základný teoretický rámec, z ktorého vychádzajú aj práce zamerané na autonómne riadenie vozidiel a agentov v simulovaných prostrediach.[1]

1.3 Učenie posilňovaním

Učenie posilňovaním (angl. *Reinforcement Learning*, RL) je oblasť strojového učenia, v ktorej sa *agent* učí rozhodovať na základe spätnej väzby z prostredia. Na rozdiel od učenia pod dohľadom nemá k dispozícii správne akcie alebo požadované výstupy pre jednotlivé situácie, ale iba skalárnu odmenu, ktorá hodnotí kvalitu jeho správania. Cieľom agenta je zvoliť takú stratégiu správania, ktorá maximalizuje dlhodobú kumulatívnu odmenu.[12]

Formálny model prostredia

Štandardným formálnym rámcom pre učenie posilňovaním je *Markovovský rozhodovací proces* (MRP, resp. MDP – *Markov Decision Process*). Základné pojmy v tejto časti vychádzajú z knihy Suttona a Barta.[12] MDP môžeme zapísať ako päticu

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \gamma),$$

kde

- $\mathcal{S}$ je množina stavov prostredia,
- $\mathcal{A}$ je množina akcií, ktoré môže agent vykonať,
- $P(s' \mid s, a)$ je prechodová pravdepodobnosť, že po vykonaní akcie $a \in \mathcal{A}$ v stave $s \in \mathcal{S}$ sa prostredie presunie do stavu $s' \in \mathcal{S}$,
- $R(s, a, s')$ je odmena, ktorú agent získa za prechod zo stavu s do s' pomocou akcie a,
- $\gamma \in [0, 1)$ je diskontný faktor, ktorý určuje, do akej miery agent uprednostňuje okamžité odmeny pred budúcimi.

Interakcia agenta s prostredím prebieha v diskrétnych časových krokoch $t = 0, 1, 2, \ldots$. V čase t sa prostredie nachádza v stave $S_t \in \mathcal{S}$, agent zvolí akciu $A_t \in \mathcal{A}$ a prostredie odpovie prechodom do nového stavu S_{t+1} a poskytnutím odmeny R_{t+1}. Agent teda pozoruje postupnosť

$$S_0, A_0, R_1, S_1, A_1, R_2, \ldots,$$

na základe ktorej sa učí meniť svoje rozhodovanie.[12]

Kľúčovým predpokladom MDP je *Markovovská vlastnosť*: pravdepodobnosť ďalšieho stavu a odmeny závisí len od aktuálneho stavu a zvolenej akcie, nie od celej histórie. Formálne:

$$P(S_{t+1} = s', R_{t+1} = r \mid S_0, A_0, R_1, \ldots, S_t, A_t) = P(S_{t+1} = s', R_{t+1} = r \mid S_t, A_t).$$

V praxi je táto vlastnosť často splnená len približne – stav je typicky odvodený z dostupných senzorických údajov (napr. snímky obrazovky, lidarové vzdialenosti, informácie o rýchlosti a polohe), ktoré nemusia obsahovať všetky informácie o skutočnom stave prostredia.

Politika a cieľ učenia

Správanie agenta je popísané *politikou* π, ktorá každému stavu priraďuje pravdepodobnostné rozdelenie na akciách:

$$\pi(a \mid s) = \Pr(A_t = a \mid S_t = s).$$

V deterministickom prípade má agent v každom stave jednoznačne určenú akciu $a = \pi(s)$.

Kvalitu politiky meriame pomocou očakávaného *návratu* (*return*). Návrat G_t v čase t je definovaný ako diskontovaný súčet budúcich odmien:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{1.1}$$

Cieľom učenia posilňovaním je nájsť politiku $\pi^\star$, ktorá maximalizuje očakávaný návrat z každého počiatočného stavu:

$$\pi^\star = \arg\max_{\pi} \; E_\pi\big[G_t \mid S_t = s\big] \quad \text{pre všetky } s \in \mathcal{S}.$$

Pre hodnotenie politiky sa zavádzajú *hodnotové funkcie*.[12] *Stavová hodnotová funkcia* V^π vyjadruje očakávaný návrat pri nasledovaní politiky π z daného stavu:

$$V^\pi(s) = E_\pi\big[G_t \mid S_t = s\big].$$

Analogicky *akčno–stavová hodnotová funkcia* Q^π priraďuje očakávaný návrat páru stav–akcia:

$$Q^\pi(s, a) = E_\pi\big[G_t \mid S_t = s, A_t = a\big].$$

Hodnotové funkcie spĺňajú tzv. Bellmanove rovnice, ktoré spájajú očakávaný návrat s jedným krokom interakcie a hodnotou nasledujúceho stavu. Pre stavovú funkciu V^π má tvar

$$V^\pi(s) = \sum_{a \in \mathcal{A}} \pi(a \mid s) \sum_{s' \in \mathcal{S}} P(s' \mid s, a)\big[R(s, a, s') + \gamma V^\pi(s')\big]. \tag{1.2}$$

Tieto rovnice tvoria základ pre množstvo známych RL algoritmov.[12]

Typy algoritmov učenia posilňovaním

Algoritmy učenia posilňovaním môžeme rozdeliť podľa viacerých kritérií. Prehľad modernej literatúry uvádzajú napríklad Arulkumaran a kol.[1]

Jedným z rozšírených delení je rozlíšenie medzi *modelovými* a *bezmodelovými* prístupmi. Modelové algoritmy predpokladajú alebo sa snažia naučiť prechodové pravdepodobnosti P a funkciu odmeny R a následne nad takýmto modelom riešia optimalizačnú úlohu (napr. dynamické programovanie). Bezmodelové algoritmy naproti tomu pracujú priamo s pozorovanými prechodmi $(S_t, A_t, R_{t+1}, S_{t+1})$ bez explicitného odhadu modelu prostredia.[12, 1]

Ďalšie dôležité rozdelenie je na *hodnotovo orientované* (*value-based*), *politicky orientované* (*policy-based*) a *aktor–kritik* algoritmy. Hodnotovo orientované metódy (napr. Q-learning, Deep Q-Networks) sa snažia naučiť optimálnu akčno–stavovú hodnotovú funkciu $Q^\star(s, a)$ a z nej odvodiť politiku, napríklad voľbou akcie s najvyššou hodnotou v danom stave. Politicky orientované metódy (napr. REINFORCE, Proximal Policy Optimization) parametrizujú priamo politiku π_θ a optimalizujú ju pomocou gradientných metód voči očakávanému návratu. Algoritmy typu aktor–kritik kombinujú obidva prístupy: majú samostatnú reprezentáciu politiky (aktor) aj hodnotovej funkcie (kritik).[1]

Osobitnú úlohu v RL zohráva *trade-off medzi prieskumom a využívaním* (angl. *exploration–exploitation trade-off*). Agent musí na jednej strane využívať už naučenú politiku, aby získaval vysokú odmenu, na druhej strane však musí občas skúšať aj menej perspektívne akcie, aby objavil lepšie stratégie. Jednoduchým mechanizmom je napríklad ε-greedy politika, ktorá s pravdepodobnosťou $1-\varepsilon$ vyberie momentálne najlepšiu známu akciu a s pravdepodobnosťou ε náhodnú akciu.[12] V hlbokom učení posilňovaním sa často používa aj regularizácia entropie politiky, ktorá podporuje dostatočnú mieru náhodnosti akcií počas učenia.[1]

Aplikácie v riadení a simulovaných prostrediach

Učenie posilňovaním sa prirodzene hodí na problémy *sekvenčného rozhodovania*, kde agent musí vykonať sériu akcií a kvalita ich kombinácie sa ukáže až na konci epizódy. Typickými príkladmi sú hry (šach, Go, rôzne video hry), robotické manipulácie, riadenie autonómnych vozidiel alebo úlohy riadenia v priemysle.[12]

V herných simuláciách má RL niekoľko výhod: prostredie je plne pod kontrolou, dá sa opakovane resetovať, odmeny či metriky výkonu (napr. čas dokončenia trate, počet kolízií, stabilita jazdy) sú jasne definované a relatívne ľahko merateľné. Zároveň ide často o vysokodimenzionálne a nelineárne problémy, kde je explicitné programovanie inteligentného správania mimoriadne náročné alebo neudržateľné. Preto vzniklo viacero prác, ktoré RL využívajú práve v prostrediach pre riadenie vozidiel alebo závodných

hier, napríklad v hre Gran Turismo Sport,[6] v rámci ML-Agents v Unity,[9] alebo priamo v hre Trackmania.[2, 8] RL tak poskytuje všeobecný rámec, v ktorom sa agent učí jazdiť alebo hrať na základe interakcie so simulovaným svetom a postupne zlepšuje svoje schopnosti.

1.4 Evolučné algoritmy

Evolučné algoritmy predstavujú triedu optimalizačných metód, ktoré sú inšpirované prírodnou evolúciou. Namiesto jedného kandidátneho riešenia pracujú s celou populáciou riešení, ktoré sa v priebehu generácií menia pomocou operátorov pripomínajúcich prirodzený výber, kríženie a mutáciu.[7, 5] Vďaka tomu dokážu tieto algoritmy hľadať riešenia aj v zložitých a nelineárnych priestoroch, kde tradičné gradientné metódy zlyhávajú alebo sa ťažko aplikujú.

V tejto časti stručne zhrnieme základný optimalizačný rámec, na ktorom sú evolučné algoritmy postavené, popíšeme základné komponenty genetického algoritmu a naznačíme, ako sa evolučné metódy ďalej rozvetvili do viacerých príbuzných smerov.

Optimalizačný problém a fitness funkcia

Mnohé praktické úlohy môžeme formulovať ako problém optimalizácie nejakej objektívnej funkcie. V najjednoduchšom prípade ide o jednokriteriálnu optimalizáciu, kde hľadáme prvok x z množiny možných riešení X, ktorý maximalizuje (alebo minimalizuje) reálnu funkciu f:

$$\max_{x \in X} f(x). \tag{1.3}$$

Vektor x môže predstavovať napríklad parametre riadiaceho algoritmu, váhy neurónovej siete alebo rôzne voľby konfigurácie. Množina X sa nazýva priestor riešení alebo priestor vyhľadávania.

V prípade, že sledujeme viacero cieľov naraz, môžeme definovať tzv. multiobjektívnu optimalizáciu, kde každému riešeniu priraďujeme vektor cieľových funkcií

$$F(x) = \big(f_1(x), f_2(x), \ldots, f_k(x)\big) \tag{1.4}$$

a snažíme sa nájsť riešenia, ktoré sú z hľadiska všetkých týchto funkcií čo „najlepšie“.[3] Formálne sa potom uvažuje o Pareto optimálnych riešeniach, ktoré nemožno zlepšiť v jednom cieli bez zhoršenia v inom. Podrobnejšie sa multiobjektívnej optimalizácii budeme venovať v samostatnej sekcii.

V evolučných algoritmoch sa na hodnotenie kvality riešenia často používa pojem *fitness* (fitness funkcia). Fitness funkcia f priraďuje každému riešeniu číslo vyjadrujúce, ako dobre daný jedinec spĺňa zadanú úlohu. V prípade riadenia autonómneho agenta to môže byť napríklad kombinácia prejdenej vzdialenosti, času a počtu kolízií. Dôležité je,

že evolučné algoritmy nepotrebujú explicitné informácie o odvodeninách tejto funkcie ani jej analytický tvar. Stačí vedieť každému riešeniu priradiť fitness na základe jeho správania v simulácii. Fitness funkcia tak môže byť v podstate „čiernou skrinkou“, ktorá na vstupe dostane kandidátne riešenie a na výstupe vráti jeho ohodnotenie.[5]

Základné komponenty genetického algoritmu

Genetický algoritmus (GA) je najznámejším predstaviteľom evolučných algoritmov. Bol pôvodne navrhnutý pre binárne reprezentácie riešení, ale neskôr bol rozšírený aj na reálne vektory alebo zložitejšie štruktúry.[7, 5]

Základné stavebné prvky genetického algoritmu sú:

- **Populácia** je konečná množina kandidátnych riešení (jedincov). V každej generácii GA pracuje s populáciou veľkosti N.
- **Chromozóm** je vnútorná reprezentácia jedinca, typicky v podobe vektora (napr. bitový reťazec, reálny vektor). Jednotlivé prvky chromozómu nazývame gény.
- **Fenotyp** je „skutočné“ riešenie v úlohe, ktoré vznikne interpretáciou chromozómu (napr. váhy neurónovej siete, parametre riadiaceho regulátora).
- **Fitness funkcia** priraďuje každému jedincovi číselné ohodnotenie, ktoré vyjadruje kvalitu jeho riešenia.
- **Selektívny mechanizmus** (selekcia) určuje, ktorí jedinci sa stanú rodičmi ďalšej generácie. Často používané sú napríklad ruletová selekcia (pravdepodobnosť výberu úmerná fitness), turnajová selekcia (náhodne vyberieme malú skupinu jedincov a z nej najlepšieho) alebo rangová selekcia.[5]
- **Kríženie** (crossover) kombinuje časti chromozómov dvoch (alebo viacerých) rodičov a vytvára nového potomka. Pri binárnej reprezentácii sa často používa jednobodové alebo dvojbodové kríženie, pri reálnych vektoroch napríklad aritmetické kríženie.
- **Mutácia** náhodne mení niektoré gény chromozómu, aby sa do populácie dostali nové genetické variácie. Vďaka mutácii algoritmus neuviazne v statickom stave a dokáže lepšie preskúmať priestor riešení.
- **Elitizmus** je mechanizmus, ktorý zabezpečí, že najlepší jedinci z aktuálnej generácie sa prenesú nezmenení do generácie nasledujúcej. Tým sa znižuje riziko, že dobré riešenia zaniknú náhodnými mutáciami alebo smolnou selekciou.

Typický priebeh genetického algoritmu môžeme zhrnúť do nasledujúcich krokov (inšpirované napr. [7, 5]):

1. Inicializujeme počiatočnú populáciu náhodnými jedincami.

2. Vyhodnotíme fitness všetkých jedincov v populácii.

3. Pokiaľ nie je splnená podmienka ukončenia (napr. maximálny počet generácií alebo dosiahnutie požadovanej kvality), opakujeme:

   (a) pomocou selekčného mechanizmu vyberieme rodičov,

   (b) nad vybranými rodičmi aplikujeme kríženie a vytvoríme potomkov,

   (c) na potomkoch aplikujeme mutáciu,

   (d) z rodičov a potomkov vytvoríme novú populáciu (prípadne s elitizmom),

   (e) vyhodnotíme fitness všetkých jedincov v novej populácii.

4. Ako výsledok algoritmu zoberieme najlepšieho jedinca, ktorý sa v populácii vyskytol.

Konkrétne voľby operátorov, parametre (napr. veľkosť populácie, pravdepodobnosť mutácie) a spôsob tvorby novej populácie môžu byť rôzne. Rôzne kombinácie týchto prvkov vedú k mnohým variáciám evolučných algoritmov.

Varianty evolučných metód

Genetické algoritmy sú len jednou z vetiev širšej rodiny evolučných metód. V literatúre sa často stretávame aj s pojmami ako evolučné stratégie, evolučné programovanie či genetické programovanie.[5]

Evolučné stratégie. Evolučné stratégie (ES) vznikli pôvodne v Nemecku a sú úzko spojené s optimalizáciou kontinuálnych (reálnych) parametrov. Jedinec je typicky reprezentovaný ako vektor reálnych čísel, ktorý predstavuje hľadané parametre, a súčasťou chromozómu môžu byť aj tzv. strategické parametre, ktoré určujú veľkosť mutačných krokov.[5] Základné schémy ES sa označujú ako (μ, λ) a $(\mu + \lambda)$:

- v schéme (μ, λ) vzniká v každej generácii λ potomkov z μ rodičov a nová generácia sa tvorí výlučne z týchto potomkov,

- v schéme $(\mu + \lambda)$ sa nová generácia vyberá zo zjednotenia rodičov a potomkov, čo umožňuje silnejší elitizmus.

Evolučné programovanie a genetické programovanie. Evolučné programovanie sa pôvodne zameriavalo na evolúciu konečných automatov a ich prechodových funkcií. Genetické programovanie (GP) ide ešte o krok ďalej: jedinci sú celé programy, ktoré sú reprezentované napríklad vo forme stromov (výrazových stromov) a evolučné operátory menia podstromy alebo časti programu.[5] GP sa osvedčilo v úlohách, kde chceme automaticky objavovať algoritmy alebo symbolické vzťahy, ale je výpočtovo náročné a výsledné programy bývajú ťažko interpretovateľné.

Evolúcia neurónových sietí. Osobitnú skupinu tvorí evolúcia umelých neurónových sietí, označovaná aj ako *neuroevolúcia*. V tomto prístupe jedinci reprezentujú buď parametre (váhy a biasy) neurónovej siete, alebo dokonca aj jej štruktúru (topológiu). Yao uvádza prehľad rôznych prístupov k evolúcii neurónových sietí a ich použitie pri riadení a aproximácii funkcií.[14] Neuroevolúcia je obzvlášť zaujímavá pri riadení agentov v prostrediach, kde nemáme jednoduchý spôsob, ako definovať gradienty a použiť klasický tréning pomocou spätného šírenia chyby.

V tejto práci sa budeme sústrediť práve na prístup, kde chromozóm jednotlivca predstavuje parametre neurónovej siete, ktorá slúži ako politika autonómneho agenta. Takýto prístup môžeme chápať ako špeciálny prípad evolučného algoritmu, konkrétne neuroevolúcie.

Výhody a obmedzenia evolučných algoritmov

Evolučné algoritmy majú niekoľko vlastností, ktoré ich robia atraktívnymi pre riešenie zložitých optimalizačných úloh:

- **Nezávislosť od derivácií.** Evolučné metódy nevyžadujú výpočet gradientov ani znalosti o vnútornej štruktúre fitness funkcie. Stačí vedieť pre dané riešenie vypočítať jeho ohodnotenie. To je výhodné pri úlohách, kde je fitness funkcia zadaná simuláciou alebo experimentom a nemá jednoduchý analytický tvar.

- **Flexibilná reprezentácia.** Chromozómy môžu mať rôzne podoby: binárne reťazce, reálne vektory, stromy, grafy. Evolučné algoritmy tak vedia pracovať s rozličnými typmi rozhodovacích štruktúr.

- **Práca s viacerými cieľmi.** Mnohé evolučné algoritmy sú prirodzene rozšíriteľné na multiobjektívne optimalizačné problémy, kde hľadáme kompromis medzi viacerými cieľmi. Príkladom je známy algoritmus NSGA-II.[4]

- **Paralelizácia.** Vyhodnocovanie jednotlivcov je často nezávislé, takže sa dá jednoducho paralelizovať na viacerých procesoroch alebo strojoch. To je dôležité najmä v prípadoch, keď je jedna simulácia drahá.

Na druhej strane majú evolučné algoritmy aj svoje obmedzenia:

- **Počet hodnotení.** Keďže pracujú s populáciou a postupne generujú nové riešenia, môžu vyžadovať veľké množstvo hodnotení fitness funkcie. Pri náročných simuláciách (napr. fyzikálne korektná simulácia vozidla) môže byť tento výpočtový náklad problém.
- **Nastavenie parametrov.** Výkon evolučného algoritmu závisí od voľby parametrov (veľkosť populácie, pravdepodobnosť mutácie a kríženia, výber selekčného mechanizmu). Neexistuje univerzálne nastavenie vhodné pre všetky úlohy a často je potrebné parametre empiricky ladiť.[5]
- **Absencia záruk globálneho optima.** Evolučné algoritmy sú heuristické metódy. Vo všeobecnosti neexistuje záruka, že nájdu globálne optimum, hoci v praxi často objavia dostatočne dobré riešenia.
- **Stochastický charakter.** Výsledky evolučného algoritmu sa môžu líšiť pri opakovaných spusteniach s rôznymi náhodnými seedmi. To treba mať na pamäti pri vyhodnocovaní a porovnávaní rôznych nastavení.

Napriek týmto obmedzeniam sa evolučné algoritmy osvedčili v mnohých oblastiach, vrátane návrhu regulátorov, riadenia robotov a evolúcie stratégií v hrách. V kontexte tejto práce nám umožnia hľadať takú politiku autonómneho vodiča, ktorá dosahuje dobré výsledky v simulovanej závodnej hre, bez potreby explicitného odvodenia gradientov alebo ručného návrhu pravidiel.

1.5 Neuroevolúcia – evolučné učenie neurónových sietí

V predchádzajúcej časti sme opísali základné princípy evolučných algoritmov. Jednou z oblastí, kde sa tieto metódy často uplatňujú, je tzv. neuroevolúcia, teda evolučné učenie umelých neurónových sietí.[14] V tomto prístupe jednotlivci v populácii nereprezentujú napríklad priamo parametre regulátora, ale neurónové siete, ktoré slúžia ako politika alebohodnotová funkcia pre agenta. Evolučný algoritmus potom hľadá také siete, ktoré vedú k čo najlepšiemu správaniu v danom prostredí.

V tejto sekcii stručne zhrnieme základnú architektúru neurónových sietí používaných pri riadení agentov, spôsoby ich kódovania do chromozómu a niekoľko príkladov použitia neuroevolúcie v herných prostrediach.

Umelé neurónové siete pre riadenie agentov

Umelá neurónová sieť (ANN) je parametrický model zložený z jednoduchých výpočtových jednotiek (neurónov), ktoré sú usporiadané do vrstiev a vzájomne prepojené

váhovanými spojmi.[14] Najčastejšie sa pri riadení agentov používa viacvrstvová perceptrónová sieť (MLP – *Multi-Layer Perceptron*), ktorá má niekoľko plne prepojených vrstiev a informácia ňou preteká smerom dopredu (feed-forward architektúra).

Základnú MLP tvoria:

- **Vstupná vrstva**, ktorá prijíma stavový vektor z prostredia (napr. vzdialenosti z lidarových senzorov, rýchlosť vozidla, informácie o polohe na trati),
- jedna alebo viac **skrytých vrstiev**, v ktorých sa vstupy transformujú pomocou lineárnych kombinácií a nelineárnych aktivačných funkcií,
- **výstupná vrstva**, ktorá produkuje akciu alebo parametrizuje rozdelenie na akciách (napr. hodnota plynu, brzdy a natočenie volantu).

Každý neurón počíta vážený súčet svojich vstupov a pripočíta bias:

$$z = \sum_i w_i x_i + b,$$

kde x_i sú vstupy, w_i váhy a b bias. Výstup neurónu vznikne aplikáciou aktivačnej funkcie ϕ, napríklad ReLU, sigmoidy alebo tangens hyperbolický:

$$y = \phi(z).$$

Nelineárne aktivačné funkcie sú dôležité preto, aby sieť dokázala modelovať aj nelineárne vzťahy medzi vstupmi a výstupmi.

Pre riadenie agentov v sekvenčných prostrediach sa okrem MLP používajú aj iné typy architektúr. Rekurentné neurónové siete (RNN) a ich moderné varianty (LSTM, GRU) umožňujú pracovať s vnútornou pamäťou a lepšie zachytiť závislosti v čase. Konvolučné neurónové siete (CNN) sa zase používajú pri spracovaní obrazových vstupov, napríklad keď agent vníma prostredie cez kamery.[1] Voľba konkrétnej architektúry závisí od typu vstupných dát a požiadaviek na správanie agenta.

Z pohľadu riadenia môžeme neurónovú sieť chápať ako funkciu

$$\pi_\theta : \mathcal{S} \to \mathcal{A},$$

ktorá každému stavu $s \in \mathcal{S}$ priradí akciu $a \in \mathcal{A}$. Parametre siete (váhy a biasy) sú označené súhrnne ako θ. Úlohou učenia je nájsť také θ, aby výsledná politika π_θ viedla k želanému správaniu agenta (napr. rýchla a bezpečná jazda po závodnej trati).

Spôsoby kódovania neurónovej siete do chromozómu

Aby sme mohli na neurónové siete aplikovať evolučné algoritmy, musíme zvoliť spôsob, ako ich reprezentovať v podobe chromozómu. V literatúre sa používa viacero rôznych prístupov.[14, 5]

Priame kódovanie parametrov. Najjednoduchší a veľmi často používaný prístup je *priame kódovanie*, kde chromozóm obsahuje všetky parametre neurónovej siete v jednom vektore. Všetky váhy a biasy v sieti sa „vyrovnajú“ do jedného dlhého reťazca reálnych čísel:

$$\mathbf{c} = (w_1, w_2, \ldots, w_n, b_1, b_2, \ldots, b_m).$$

Evolučný algoritmus potom pracuje s týmto vektorom ako s klasickým riešením v kontinuálnom priestore. Výhodou je jednoduchá implementácia a možnosť použiť štandardné operátory kríženia a mutácie nad reálnymi vektormi. Nevýhodou môže byť veľká dimenzia chromozómu pri rozsiahlejších sieťach.

Evolúcia topológie siete. Pri priamom kódovaní zvyčajne predpokladáme, že architektúra siete (počty vrstiev a neurónov) je pevne daná a evolúcia mení iba jej parametre. Alternatívou sú prístupy, ktoré evolvujú aj samotnú štruktúru neurónovej siete. Príkladom je algoritmus NEAT (*NeuroEvolution of Augmenting Topologies*), kde chromozómy reprezentujú grafy neurónových sietí a evolučné operátory umožňujú pridávať alebo odstraňovať neuróny a spojenia.[11] Sieť tak môže počas evolúcie postupne rásť od jednoduchých štruktúr k zložitejším, pričom sa zachováva kompatibilita medzi rôznymi topológiami.

Na NEAT nadväzujú aj ďalšie metódy ako HyperNEAT, ktoré sa snažia využiť geometrickú štruktúru problému a generovať rozsiahle, priestorovo usporiadané siete pomocou nepriamych kódovaní.

Nepriame kódovanie a generatívne prístupy. Okrem priameho kódovania parametrov existujú aj tzv. *nepriame* alebo generatívne kódovania, kde chromozóm neobsahuje priamo všetky váhy siete, ale skôr „recept“ na ich generovanie. Príkladom sú kompaktné neuronálne grafy (CPPN) alebo rôzne gramatiky, ktoré dokážu z krátkeho genotypu vytvoriť veľké a štruktúrované siete. Takéto prístupy môžu byť výhodné, ak má výsledná sieť vykazovať určitú symetriu alebo pravidelnosť.[14]

Výber konkrétneho spôsobu kódovania neurónovej siete do chromozómu závisí od charakteru úlohy a od praktických obmedzení (veľkosť siete, dostupný výpočtový výkon). V ďalších kapitolách sa preto budeme venovať návrhu konkrétnej reprezentácie a evolučných operátorov pre riadenie agenta v závodnej hre.

Neuroevolúcia v hrách

Neuroevolúcia sa v hrách a simulovaných prostrediach používa už dlhší čas. Yao vo svojom prehľade uvádza viacero príkladov, kde evolúcia neurónových sietí slúži na riadenie robotov, navigáciu alebo koordináciu jednoduchej kolektívnej inteligencie.[14]

Herné prostredia sú pre neuroevolúciu atraktívne najmä preto, že poskytujú kontrolované, opakovateľné a dobre merateľné podmienky.

V hernej AI sa neuroevolúcia používa napríklad na:

- návrh správania nehráčskych postáv (NPC), ktoré sa dokážu adaptovať na hráča a vytvárať rozmanité stratégie,
- evolúciu riadiacich politík pre závodné a akčné hry,
- hľadanie nových herných stratégií alebo taktických postupov, ktoré by dizajnér manuálne len ťažko vymyslel.

Schrum a Miikkulainen vo svojej práci demonštrujú použitie *multiobjektívnej* neuroevolúcie na konštrukciu komplexného správania NPC v hernom prostredí.[10] Ich prístup vychádza z toho, že požadované správanie nie je vhodné popísať jednou skalárnou fitness funkciou, ale viacerými cieľmi naraz. V ich experimentoch s bojovými agentmi sa napríklad uvažujú ciele ako:

- maximalizácia spôsobeného poškodenia súperovi,
- maximalizácia času prežitia agenta,
- minimalizácia prijatého poškodenia.

Tieto ciele sú prirodzene konfliktné – agresívny agent môže spôsobovať veľa poškodenia, ale zároveň rýchlo zomrieť, zatiaľ čo opatrnejší agent prežije dlhšie, ale spôsobí menej škody. Autori preto používajú multiobjektívny evolučný algoritmus založený na Pareto dominancii (inšpirovaný napr. NSGA-II[4]), ktorý udržiava v populácii množinu rôznych kompromisných riešení namiesto jedného skalárneho optima.[10]

Takýto prístup má viacero výhod. Umožňuje preskúmať rôzne štýly správania (agent môže byť viac agresívny, defenzívny, vyvažovať riziko a zisk) a až v následnom kroku vybrať konkrétny kompromis podľa preferencií dizajnéra. Zároveň sa tým prirodzene modeluje situácia, kde neexistuje jednoznačne „najlepšia“ stratégia, ale skôr množina riešení, ktoré sú optimálne z rôznych pohľadov.

V kontexte závodných hier môžeme analogicky uvažovať o viacerých cieľoch, ako sú minimalizácia času na kolo, minimalizácia počtu kolízií, maximalizácia plynulosti jazdy alebo minimalizácia odchýlky od ideálnej pretekárskej stopy. Neuroevolúcia umožňuje tieto ciele zohľadniť priamo vo fitness funkcii (alebo vo viacerých fitness funkciách) a hľadať politiky, ktoré s nimi pracujú v rámci jedného evolučného procesu.

Zhrnutím, neuroevolúcia spája flexibilitu umelých neurónových sietí s robustnosťou evolučných algoritmov. Umožňuje hľadať politiky pre agentov aj v prostrediach, kde je ťažké alebo nemožné použiť klasické gradientné učenie (napr. kvôli nederivovateľnej

simulácii, šumu alebo zložitým spätným väzbám). V ďalších kapitolách sa preto zameriame na návrh a experimentálne overenie konkrétneho neuroevolučného prístupu pre riadenie agenta v závodnej hre.

1.6 Multiobjektívna optimalizácia a hodnotenie jazdy

Pri návrhu autonómneho agenta pre závodnú hru zvyčajne nechceme optimalizovať iba jeden cieľ. Okrem čo najkratšieho času na kolo nás zaujíma aj počet kolízií, plynulosť jazdy, opustenie trate a ďalšie vlastnosti. Ide teda prirodzene o *multiobjektívny* problém, v ktorom je potrebné nájsť kompromis medzi viacerými často konfliktnými cieľmi.[3] V tejto sekcii stručne zhrnieme základný formálny aparát multiobjektívnej optimalizácie, spomenieme Pareto-based evolučné algoritmy a načrtneme typické metriky, ktoré sa používajú pri hodnotení kvality jazdy autonómneho agenta.

Multiobjektívny optimalizačný problém

Kým pri jednokriteriálnej optimalizácii sa snažíme maximalizovať (alebo minimalizovať) jednu objektívnu funkciu $f(x)$, v multiobjektívnom prípade máme sústavu viacerých cieľov. Každému riešeniu $x \in X$ priraďujeme vektor cieľových funkcií

$$F(x) = \big(f_1(x), f_2(x), \ldots, f_k(x)\big), \tag{1.5}$$

kde $f_i(x)$ reprezentuje i-ty cieľ (napríklad čas, počet kolízií, prejdenú vzdialenosť). Formálne chceme nájsť riešenia, ktoré sú z hľadiska všetkých týchto cieľov „čo najlepšie“.[3]

Na rozdiel od jednokriteriálneho prípadu však zvyčajne neexistuje jedno riešenie, ktoré by bolo optimálne vo všetkých cieľoch naraz. Preto sa používa pojem *Pareto dominancia*. Nech $x, y \in X$ sú dve riešenia. Povieme, že x *dominuje* riešeniu y (značíme $x \prec y$), ak sú splnené nasledujúce podmienky:

- x nie je horšie ako y v žiadnom cieli:

$$f_i(x) \geq f_i(y) \quad \text{pre všetky } i = 1, \ldots, k,$$

- x je striktne lepšie ako y aspoň v jednom cieli:

$$f_j(x) > f_j(y) \quad \text{pre niektoré } j.$$

Riešenie $x^\star$ sa nazýva *Pareto optimálne*, ak neexistuje žiadne iné riešenie y, ktoré by ho dominovalo. Množina všetkých Pareto optimálnych riešení sa nazýva *Pareto front*.[3]

V kontexte autonómnej jazdy sú ciele často konfliktné. Napríklad:

- **minimálny čas na kolo** vs. **minimálny počet kolízií** – agresívny štýl jazdy môže byť rýchly, ale zároveň rizikový,

- **maximálny progress po trati** vs. **minimálna prejdená vzdialenosť** – agent, ktorý jazdí cik–cak medzi stenami, môže prejsť rovnakú časť trate ako agent, ktorý ide po „ideálnej stope“, ale s podstatne väčšou celkovou vzdialenosťou,

- **maximálna priemerná rýchlosť** vs. **plynulosť jazdy** – časté prudké zmeny riadenia môžu viesť k nekomfortnej alebo nestabilnej jazde.

V takýchto situáciách nie je cieľom nájsť jedno „najlepšie“ riešenie, ale skôr množinu kompromisov (Pareto front), z ktorej si následne vieme vybrať riešenie podľa preferencií (napr. uprednostniť bezpečnejšieho agenta pred mierne rýchlejším, ale veľmi rizikovým).

Pareto-based evolučné algoritmy

Evolučné algoritmy sú prirodzeným nástrojom pre multiobjektívnu optimalizáciu, pretože pracujú s populáciou riešení, a teda dokážu naraz približovať celú Pareto frontu.[3, 5] Namiesto toho, aby všetky ciele zlúčili do jednej scalarnej fitness funkcie (napr. váženou sumou), udržiavajú v populácii množinu nedominovaných riešení a snažia sa ich rozumne rozložiť po celej fronte.

Jedným z najznámejších multiobjektívnych evolučných algoritmov je NSGA-II (*Non-dominated Sorting Genetic Algorithm II*).[4] Jeho základná myšlienka spočíva v dvoch kľúčových mechanizmoch:

- **Nedominované triedenie** (non-dominated sorting): populácia sa rozdelí do tzv. frontov podľa toho, ako sú riešenia medzi sebou dominované. Prvý front tvorí množina riešení, ktoré nie sú dominované žiadnym iným riešením v populácii. Druhý front tvorí riešenia, ktoré sú dominované iba riešeniami z prvého frontu, a tak ďalej. Pri selekcii majú prednosť riešenia z lepších (nižších) frontov.

- **Crowding distance**: aby sa zachovala rozmanitosť riešení na Pareto fronte, NSGA-II zavádza mieru „zahustenosti“ okolia každého riešenia. Uprednostňujú sa riešenia, ktoré sa nachádzajú v riedkejších oblastiach fronty, čím sa zabraňuje zhlukovaniu jedincov iba v jednej časti priestoru cieľov.

Kombinácia nedominovaného triedenia a crowding distance umožňuje NSGA-II efektívne približovať Pareto frontu a zároveň udržiavať rozumné pokrytie naprieč rôznymi kompromismi medzi cieľmi.[4]

Schrum a Miikkulainen používajú prístup inšpirovaný takýmito Pareto-based metódami pri multiobjektívnej neuroevolúcii správania NPC.[10] Namiesto jedného skalárneho fitness hodnotia agentov podľa viacerých cieľov (napr. spôsobené poškodenie, čas prežitia, prijaté poškodenie) a evolučný algoritmus udržiava populáciu rôznych

typov správania (agresívne, defenzívne, vyvážené). Takéto riešenie je atraktívne aj v závodných hrách, kde môžu existovať rôzne štýly jazdy (veľmi rýchla, ale agresívna, opatrnejšia a podobne) a nie je vopred jasné, ktorý z nich je „najlepší“.

Popri Pareto-based prístupoch sa v praxi stále používajú aj jednoduchšie scalarizačné metódy, kde sa viaceré ciele spoja do jednej fitness funkcie, napríklad váženou sumou

$$f(x) = \alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_k f_k(x),$$

alebo lexikografickým porovnávaním (najprv porovnávame hlavný cieľ, pri rovnosti sekundárny cieľ atď.).[3] Tieto prístupy sú jednoduchšie na implementáciu, ale vyžadujú dopredu zvoliť váhy alebo poradie dôležitosti cieľov.

Metriky kvality jazdy autonómneho agenta

Pri návrhu multiobjektívnej fitness funkcie pre autonómneho vodiča v závodnej hre je kľúčové rozhodnúť, ktoré aspekty jazdy chceme hodnotiť a akým spôsobom. Metriky môžeme rozdeliť do niekoľkých kategórií.

Výkonnosť. Táto skupina metrík vyjadruje, ako „dobrý“ je agent z hľadiska dosiahnutia cieľa závodu. Typické príklady sú:

- **čas na kolo alebo celkový čas jazdy**,
- **prejdená vzdialenosť** po trati,
- **percentuálny progress** po mape (napr. podiel prejdených checkpointov).

Vo väčšine závodných hier je primárnym cieľom minimalizovať čas na prejdenie trate, prípadne maximalizovať percento mapy, ktoré agent stihne prejsť v danom časovom limite.

Bezpečnosť a stabilita. Druhá skupina metrík sa týka bezpečnosti a stability jazdy:

- **počet kolízií** so stenou, prekážkami alebo inými vozidlami,
- **počet opustení trate** alebo nájazdov mimo ideálnej dráhy,
- **priemerná a maximálna bočná akcelerácia** (náznak rizikových manévrov).

Tieto metriky sú dôležité najmä vtedy, keď nechceme, aby agent dosahoval dobré časy za cenu extrémne rizikového správania. V multiobjektívnom nastavení môžeme napríklad minimalizovať čas a zároveň minimalizovať počet kolízií.

Plynulosť a komfort jazdy. Ďalšiu skupinu tvoria metriky plynulosti a komfortu:

- **variancia akcií** – napr. variancia natočenia volantu alebo pridávania plynu,
- **počet prudkých zmien riadenia**, ktoré môžu indikovať cik–cak jazdu,
- **hladkosť trajektórie** – napr. integrovaná absolútna hodnota zakrivenia dráhy.

Tieto metriky sú užitočné v prípadoch, kde nechceme len „funkčného“, ale aj esteticky alebo komfortne pôsobiaceho agenta (napr. pri simulácii pasažierskych vozidiel alebo v hrách zameraných na realistickú jazdu).

Formulácia cieľov a agregácia. Jednotlivé metriky môžeme použiť ako samostatné cieľové funkcie

$$f_1(x) = -\text{čas}(x), \quad f_2(x) = \text{progress}(x), \quad f_3(x) = -\text{počet kolízií}(x), \quad \ldots$$

a riešiť multiobjektívny problém pomocou Pareto-based evolučného algoritmu, ktorý hľadá množinu rôznych kompromisných politík.[3, 4, 10] Alternatívou je zvoliť jednu agregovanú fitness funkciu, v ktorej metriky vhodne skombinujeme (napr. váženou sumou alebo lexikografickým porovnávaním). V praxi sa často začína jednoduchšou scalarizáciou a v prípade potreby sa prechádza k plnohodnotnej multiobjektívnej formulácii.

Konkrétna voľba metrík a spôsob ich kombinovania závisí od cieľov, ktoré na autonómneho agenta kladieme. Iné požiadavky budeme mať pri čisto časovo orientovanom „time trial“ režime a iné pri realistickej simulácii premávky alebo pri učení viacerých štýlov jazdy. Detailný návrh metrík a ich využitie v evolučnom algoritme pre závodnú hru bude predmetom nasledujúcich kapitol.

1.7 Reprezentácia prostredia a vstupov

Správanie autonómneho agenta vždy závisí od toho, ako je preň prostredie reprezentované a aké informácie má k dispozícii pri rozhodovaní. V prípade závodných hier ide najmä o reprezentáciu trate a o voľbu vhodných senzorických vstupov, z ktorých agent odhaduje svoj stav. V tejto sekcii stručne načrtneme niekoľko bežných prístupov k reprezentácii závodnej trate a senzorických dát v simulovaných prostrediach.

Reprezentácia závodnej trate

Závodná trať môže byť reprezentovaná rôznymi spôsobmi v závislosti od potrieb hry a riadiaceho algoritmu.

Jednou z možností je **graf checkpointov**, kde trať rozdelíme na postupnosť bodov (checkpointov) alebo úsekov a tieto body prepojíme hranami.[13] Každý úsek môže

niesť dodatočné informácie, napríklad polomer zakrivenia, šírku vozovky alebo odporúčanú rýchlosť. Takáto reprezentácia je vhodná najmä vtedy, ak chceme pracovať s vyššou úrovňou abstrakcie, napríklad pri plánovaní ideálnej stopy alebo pri hodnotení progressu agenta po trati.

Ďalším bežným prístupom je reprezentovať trať ako **diskretizovanú dráhu v priestore**. V trojrozmerných hrách ide typicky o polygónovú sieť (mesh), ktorá popisuje povrch vozovky a jej okolie. Pre potreby riadenia sa táto geometria môže zjednodušiť na stredovú líniu trate (centerline) a funkciu, ktorá udáva šírku trate v jednotlivých miestach. S využitím spline kriviek alebo iných interpolácií je možné získať hladkú reprezentáciu dráhy a odvodiť z nej napríklad lokálne zakrivenie alebo smer trate v okolí vozidla.[13]

V niektorých prácach sa trať reprezentuje pomocou **blokov** alebo dlaždíc (angl. *tiles*). Každý blok predstavuje malý úsek trate s definovaným tvarom (rovinka, ľavá/pravá zákruta, skok a pod.) a trať ako celok vzniká ich pospájaním. Takýto modulárny prístup je typický najmä pre editory tratí v závodných hrách a umožňuje jednoduchšie generovanie rôznych konfigurácií trate.

Pre účely autonómneho riadenia je dôležité, aby zvolené reprezentácie umožňovali jednoducho vypočítať základné geometrické informácie, ako sú:

- vzdialenosť vozidla od okrajov trate,
- smer a zakrivenie trate v smere jazdy,
- poradie a vzdialenosť k nasledujúcim checkpointom,
- prípadne informácie o špeciálnych objektoch (skoky, prekážky).

Konkrétna voľba reprezentácie závisí od toho, či chceme pracovať skôr s geometrickým modelom trate, s grafom checkpointov alebo s blokovým opisom.

Senzorické vstupy agenta

Aby agent vedel v prostredí rozumne jazdiť, musí mať prístup k informáciám, ktoré popisujú jeho aktuálnu situáciu. V simulovaných prostrediach sa používajú rôzne typy „senzorov“, ktoré sú odvodené z interného stavu hry.

Raycast a lidarové senzory. Veľmi častým prístupom je použitie **raycast** alebo „lidarových“ senzorov. Z pozície vozidla sa v rôznych smeroch (napr. v určitom uhle doľava a doprava) vysielajú lúče a meria sa vzdialenosť k najbližšej prekážke alebo okraju trate. Takto vznikne vektor vzdialeností, ktorý poskytuje lokálny obraz o tvare trate a prekážkach v okolí vozidla. Podobný princíp sa používa aj v robotike pri navigácii mobilných robotov.[14, 1]

Raycast senzory majú výhodu v tom, že sú relatívne jednoduché na implementáciu a poskytujú informácie, ktoré sú priamo relevantné pre riadenie (napr. ako ďaleko je stena v danom smere). Zároveň umožňujú agentovi fungovať aj v prostredí, kde nemá k dispozícii globálne súradnice alebo kompletnú mapu.

Globálne stavové informácie. Okrem lokálnych vzdialeností sa často používajú aj **globálne** informácie o stave vozidla:

- pozícia na mape (napr. projekcia na centerline alebo index najbližšieho checkpointu),
- rýchlosť a zrýchlenie vozidla,
- uhol natočenia vozidla vzhľadom na smer trate,
- bočná rýchlosť alebo odchýlka od ideálnej stopy.

V niektorých prácach zameraných na autonómne riadenie v hrách (napr. Gran Turismo Sport[6] alebo prostredia založené na Unity ML-Agents[9]) sa tieto informácie používajú ako súčasť stavového vektora, ktorý vstupuje do riadiacej neurónovej siete.

Obrazové vstupy z kamery. Ďalšou možnosťou je reprezentovať vstupy agenta pomocou **obrazových dát**, napríklad snímok z virtuálnej kamery umiestnenej v kokpite alebo nad vozidlom. Takýto prístup sa spája s použitím konvolučných neurónových sietí (CNN), ktoré sú schopné extrahovať z obrázkov vizuálne príznaky relevantné pre riadenie (trajektória trate, okraje vozovky, prekážky).[1]

Obrazové vstupy sú najbližšie tomu, ako scenár vníma ľudský hráč, ale sú aj výpočtovo náročnejšie a vyžadujú väčšie modely. V prostrediach, kde ide skôr o rýchly výskum alebo experimentovanie, sa preto často uprednostňuje kombinácia jednoduchších senzorov (raycast, rýchlosť, pozícia) pred plne vizuálnym vstupom.

Stavový vektor a parciálna observabilita

Zvolená kombinácia senzorických vstupov tvorí **stavový vektor**, ktorý agent v každom časovom kroku pozoruje. V ideálnom prípade by tento vektor mal spĺňať Markovovskú vlastnosť, teda obsahovať všetky informácie potrebné na predikciu budúceho vývoja pri danej akcii.[12] V praxi sa však často pracuje len s *čiastočne* pozorovateľným stavom (agent nevidí celú mapu ani interný stav fyziky), čo kladie vyššie nároky na návrh politiky a môže motivovať použitie rekurentných sietí alebo pamäťových mechanizmov.

Pri návrhu stavového vektora je potrebné nájsť kompromis medzi:

- **informatívnosťou** (čím viac relevantných informácií, tým lepšie môže agent rozhodovať),

- **jednoduchosťou** (príliš vysoká dimenzia môže sťažovať učenie a zvyšovať nároky na výpočtový výkon),
- **generalizáciou** (stav by mal byť definovaný tak, aby agent dokázal preniesť naučené správanie na nové trate alebo konfigurácie).

Rôzne práce zamerané na autonómne riadenie v hrách používajú odlišné kombinácie vstupov: od čisto obrazových vstupov,[6] cez kombináciu raycast senzorov a globálnych parametrov,[2, 8] až po vysoko abstraktné reprezentácie založené na checkpointoch alebo ideálnej stope.[13]

Konkrétnu voľbu stavového vektora a senzorických vstupov, ktoré budú použité v tejto diplomovej práci, špecifikujeme v kapitole 3.

Kapitola 2

Súvisiace práce

2.1 Deep Reinforcement Learning in Real-Time Environments

[2]

2.2 Simulated Autonomous Driving Using Reinforcement Learning: A Comparative Study on Unity's ML-Agents Framework

[9]

2.3 Representing and Driving a Race Track for AI Controlled Vehicles

[13]

2.4 Improving Trackmania Reinforcement Learning Performance: A Comparison of Sophy and Trackmania AI

[8]

2.5 Super-Human Performance in Gran Turismo Sport Using Deep Reinforcement Learning

[6]

Kapitola 3

Analýza problému a výzvy

Kapitola 4

Návrh riešenia

Kapitola 5

Implementácia

Kapitola 6

Výskum a experimentálne výsledky

V tejto kapitole stručne prezentujeme prvé výsledky evolučného tréningu autonómneho agenta v hre Trackmania. Agent je reprezentovaný doprednou neurónovou sieťou s jednou skrytou vrstvou, pričom všetky váhy a biasy siete tvoria chromozóm jedinca v genetickom algoritme.

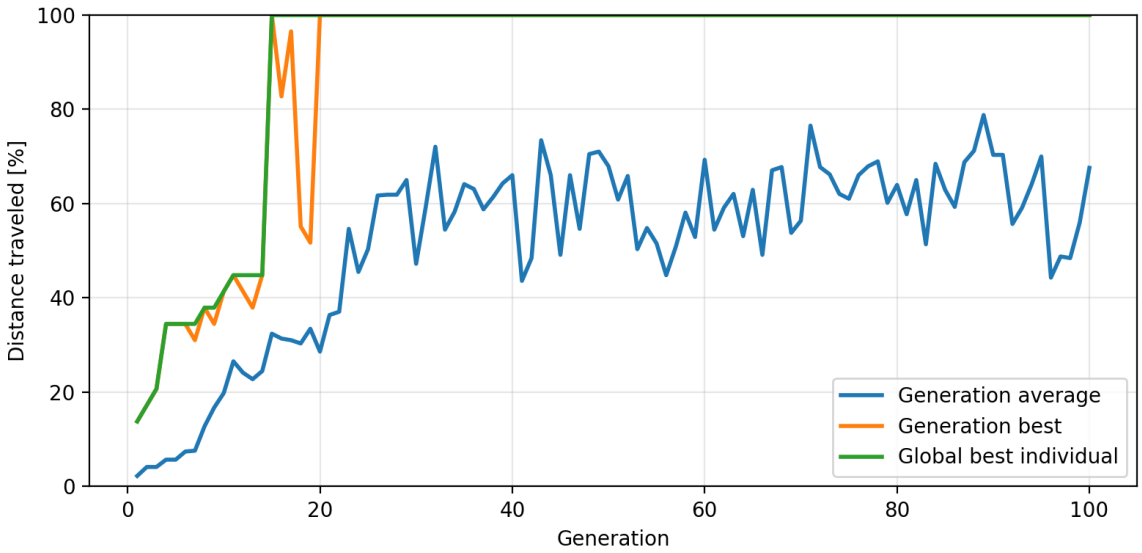
V experimente sme trénovali populáciu neuroevolučných agentov na jednej fixnej trati. Každý jedinec odjazdil jednu epizódu (jazdu) s pevne daným časovým limitom. Ak agent nestihol dôjsť do cieľa, epizóda sa považovala za nedokončenú (*DNF*) a jej čas bol pre účely vyhodnotenia nastavený na hodnotu časového limitu (v našom prípade 60 s). Tréning prebiehal niekoľko desiatok generácií, pričom v každej generácii bola populácia nových jedincov vytvorená selekciou, krížením a mutáciou z predchádzajúcej generácie. Celý beh je časovo náročný, keďže máme k dispozícii iba jednu inštanciu hry a jedincov musíme vyhodnocovať sekvenčne; jeden takýto experiment trval rádovo niekoľko hodín.

6.1 Progress po trati

Na obrázku 6.1 je znázornený vývoj percenta prejdenej trate v závislosti od generácie. Zobrazujeme tri krivky:

- priemerný jedinec v generácii (*Generation average*),
- najlepší jedinec v danej generácii (*Generation best*),
- globálne najlepší jedinec naprieč všetkými generáciami (*Global best individual*).

Na začiatku tréningu vidíme, že priemerný jedinec prejde len malú časť trate (typicky jednotky až desiatky percent). Postupne však priemerný progress rastie a po niekoľkých desiatkach generácií sa ustáli približne v intervale 50–70 %. To naznačuje,



Obr. 6.1: Vývoj percenta prejdenej trate počas evolučného tréningu.

že väčšina populácie už dokáže prejsť výraznú časť okruhu, ale nie všetci jedinci dôjdu do cieľa.

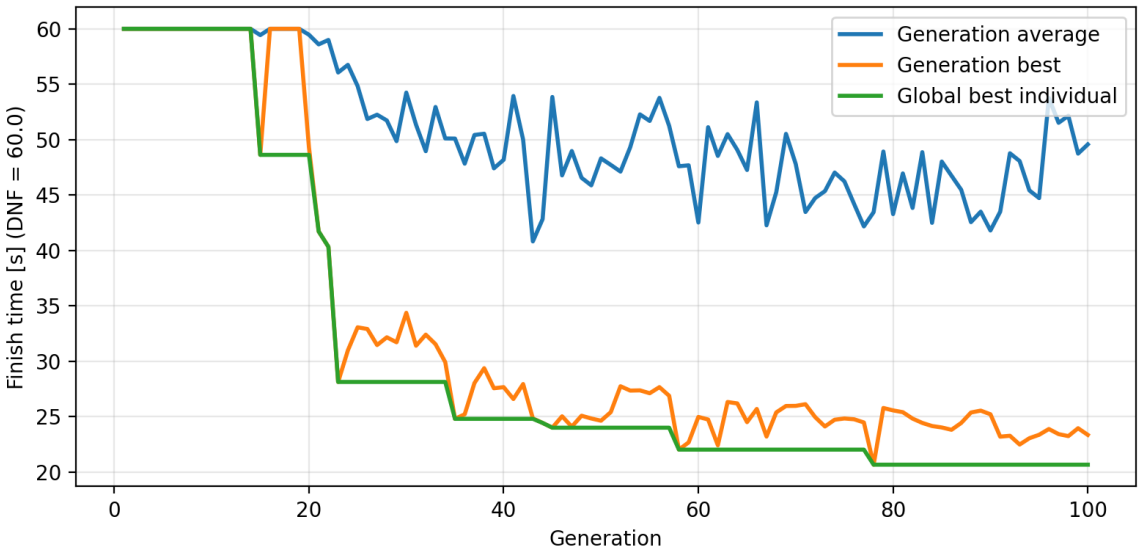
Zelená krivka (globálny najlepší jedinec) rastie po schodoch a pomerne rýchlo dosiahne hodnotu 100 %, t. j. evolúcia nájde jedinca, ktorý trať dokáže dokončiť. Od tej chvíle zostáva globálny najlepší progress na úrovni 100 %, keďže lepší než úplne dokončená jazda už z hľadiska tejto metriky neexistuje. Oranžová krivka (najlepší jedinec v generácii) sa po dosiahnutí 100 % často prekrýva so zelenou, pretože v ďalších generáciách sa podarí vyprodukovať jedincov, ktorí trať takisto dokončia.

6.2 Čas jazdy

Druhý graf (obrázok 6.2) zobrazuje vývoj času jazdy v sekundách. Pre nedokončené jazdy (DNF) je čas nastavený na hodnotu časového limitu 60 s, čo je v grafe explicitne uvedené.

Na začiatku tréningu sú priemerné časy vysoké, často na úrovni časového limitu, pretože veľká časť agentov trať nedokončí. Globálne najlepší jedinec (zelene) má spočiatku tiež čas 60 s, čo znamená, že ani najlepší agent v prvých generáciách nedokáže prísť do cieľa.

V priebehu evolúcie však vidíme výrazné zlepšenie: zelená krivka postupne klesá z 60 s na hodnoty okolo 20–25 s. To znamená, že genetický algoritmus postupne nachádza nielen agentov, ktorí trať dokončia, ale zároveň ju prejdú výrazne rýchlejšie. Oranžová krivka (najlepší jedinec v generácii) kopíruje tento trend a naznačuje, že v neskorších generáciách sa v populácii pravidelne objavujú rýchli a úspešní jazdci. Priemerný čas jedinca v generácii (modrá krivka) zostáva vyšší, čo je očakávané – populácia si stále



Obr. 6.2: Vývoj času prejazdu počas evolučného tréningu (nedokončené jazdy sú penalizované časom 60 s).

udržiava určitú mieru diverzity a obsahuje aj slabších jedincov.

Subjektívne pozorovanie jazdy ukazuje, že aj najlepší agenti majú stále tendenciu jazdiť pomerne agresívne a miestami cik–cak medzi okrajmi trate. Je to dôsledok nastavenia fitness kritérií: evolúcia výrazne preferuje rýchlosť a progress po trati, zatiaľ čo plynulosť jazdy je v tomto experimente penalizovaná iba nepriamo (napr. cez celkovú prejdenú vzdialenosť). Táto vlastnosť poskytuje motiváciu pre ďalšie experimenty s multiobjektívnym hodnotením, ktoré by lepšie vyvažovalo rýchlosť, bezpečnosť a plynulosť jazdy.

Záver

Tu bude záver

Literatúra

[1] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[2] Yann Bouteiller. *Deep Reinforcement Learning in Real-Time Environments*. Ecole Polytechnique, Montreal (Canada), 2021.

[3] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Chichester, 2001.

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[5] A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computing*. Springer, Berlin, 2003.

[6] Florian Fuchs, Yunlong Song, Elia Kaufmann, Davide Scaramuzza, and Peter Durr. Super-human performance in gran turismo sport using deep reinforcement learning. *IEEE Robotics and Automation Letters*, 6(3):4257–4264, July 2021.

[7] Vladimír Kvasnička, Jiří Pospíchal, and Peter Tiňo. *Evolučné algoritmy*. Slovenská technická univerzita, Bratislava, 2000.

[8] LJ Neinders. Improving trackmania reinforcement learning performance: A comparison of sophy and trackmania ai. B.S. thesis, University of Twente, 2023.

[9] Yusef Savid, Reza Mahmoudi, Rytis Maskeliūnas, and Robertas Damaševičius. Simulated autonomous driving using reinforcement learning: A comparative study on unity's ml-agents framework. *Information*, 14(5):290, 2023.

[10] Jacob Schrum and Risto Miikkulainen. Constructing complex NPC behavior via multi-objective neuroevolution. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008)*, pages 108–113, Stanford, California, 2008.

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

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2 edition, 2018.

[13] Simon Tomlinson and Nic Melder. Representing and driving a race track for ai controlled vehicles. *Game AI Pro*, 2014.

[14] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Príloha A: stránka agenta

V elektronickej prílohe priloženej k práci sa nachádza zdrojový kód programu a súbory s výsledkami experimentov. Zdrojový kód je zverejnený aj na stránke `https://github.com/Metcoler/Trackmania-BC`.