VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

PRINCIPY A APLIKACE NEUROEVOLUCE

NEUROEVOLUTION: PRINCIPLES AND APPLICATIONS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. JAN HEREC**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAL BIDLO, Ph.D.**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačových systémů Akademický rok 2017/2018

Zadání diplomové práce

Řešitel: **Herec Jan, Bc.**
Obor: Bioinformatika a biocomputing
Téma: **Principy a aplikace neuroevoluce**
Neuroevolution Principles and Applications
Kategorie: Umělá inteligence

Pokyny:

1. Nastudujte problematiku evolučních algoritmů a neuronových sítí.
2. Seznamte se možnými oblastmi aplikace neuronových sítí a konceptem neuroevoluce.
3. Navrhněte a implementujte systém využívající neuroevoluci ve zvolené oblasti.
4. Proveďte sadu experimentů demonstrujících schopnosti navrženého řešení.
5. Zhodnoťte dosažené výsledky a diskutujte možnosti pokračování projektu.

Literatura:

- Podle pokynů vedoucího projektu.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání, demonstrace prototypu návrhu dle bodu 3 zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bidlo Michal, Ing., Ph.D.**, UPSY FIT VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 23. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačových systémů a sítí
612 66 Brno, Božetěchova 2

L.S.

prof. Ing. Lukáš Sekanina, Ph.D.
vedoucí ústavu

Abstrakt

Práce se na teoretické úrovni zabývá evolučními algoritmy (EA), neuronovými sítěmi (NN) a jejich syntézou v podobě neuroevoluce. Z praktického hlediska je cílem práce ukázat uplatnění neuroevoluce na dvou odlišných úlohách. První úloha spočívá v evolučním návrhu architektury konvoluční neuronové sítě (CNN), která by dokázala klasifikovat s vysokou přesností ručně psané číslice (z datasetu MNIST). Druhá úloha spočívá v evoluční optimalizaci vah neurokontroléru, který řídí přistání 1. stupně rakety Falcon 9 ve 2D simulaci. Obě úlohy jsou výpočetně velmi náročné a proto byly řešeny na superpočítači. V rámci první úlohy se podařilo navrhnout takové architektury, které při správném natrénování dosahují přesnosti klasifikace 99,49%. Ukázalo se tak, že je možné návrh kvalitních architektur zautomatizovat s využitím neuroevoluce. V rámci druhé úlohy se podařilo optimalizovat váhy neurokontroléru tak, že pro definované počáteční podmínky dovede neurokontrolér model rakety k úspěšnému přistání. V obou úlohách tedy neuroevoluce uspěla.

Abstract

The theoretical part of this work deals with evolutionary algorithms (EA), neural networks (NN) and their synthesis in the form of neuroevolution. From a practical point of view, the aim of the work is to show the application of neuroevolution on two different tasks. The first task is the evolutionary design of the convolutional neural network (CNN) architecture that would be able to classify handwritten digits (from the MNIST dataset) with a high accurancy. The second task is the evolutionary optimization of neurocontroller for a simulated Falcon 9 rocket landing. Both tasks are computationally demanding and therefore have been solved on a supercomputer. As a part of the first task, it was possible to design such architectures which, when properly trained, achieve an accuracy of 99.49%. It turned out that it is possible to automate the design of high-quality architectures with the use of neuroevolution. Within the second task, the neuro-controller weights have been optimized so that, for defined initial conditions, the model of the Falcon booster can successfully land. Neuroevolution succeeded in both tasks.

Klíčová slova

Umělá inteligence, Biocomputing, Neuronová síť, Konvoluční neuronová síť, Evoluční algoritmus, Genetický algoritmus, Evoluční strategie, Diferenciální evoluce, Neuroevoluce, Neurokontrolér, Řídicí systém, Falcon 9, Superpočítání, MNIST, Klasifikace

Keywords

Artificial intelligence, Biocomputing, Neural network, Convolution neural network, Evolution algorithm, Genetic algorithm, Evolution Strategy, Differential Evolution, Neuroevolution, Neurocontroller, Control system, Falcon 9, Supercomputing, MNIST, Classification

Citace

HEREC, Jan. *Principy a aplikace neuroevoluce*. Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michal Bidlo, Ph.D.

Principy a aplikace neuroevoluce

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Michala Bidla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Herec
22. května 2018

Poděkování

Děkuji svému vedoucímu práce Ing. Michalu Bidlovi, Ph.D. za poskytnutí mnohé odborné i praktické pomoci. Také jsem mu vděčný za poskytnutí výpočetního času na superpočítači Salomon, kde jsem v rámci experimentů využil přes 170 tisíc jádrohodin. Bez této podpory by tato práce nemohla vzniknout.

Tato práce byla podpořena Ministerstvem školství, mládeže a tělovýchovy z podpory Velkých infrastruktur pro výzkum, experimentální vývoj a inovace v rámci projektu „IT4Innovations národní superpočítačové centrum - LM2015070“.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Mnoho netriviálních výpočetních problémů je dnes řešeno s využitím algoritmů inspirovaných přírodou. Pro řešení určitých problémů jsou totiž tyto algoritmy nejvhodnější nebo i jediné možné. Tyto algoritmy spadají do oblasti umělé inteligence, dnes jich existuje velmi pestrá paleta a neustále vznikají další. Jedná se tedy o velmi živou a neustále se rozvíjející oblast počítačových věd, resp. umělé inteligence. Kombinací některých takových přírodou inspirovaných algoritmů, konkrétně neuronových sítí a evolučních algoritmů, se zabývá tato diplomová práce.

Jedna z přírodou inspirovaných technik jsou již zmíněné neuronové sítě. Jsou tady s námi sice již několik desítek let, ale neustále v této oblasti probíhají výzkumy. Neuronové sítě se vyvíjejí a to je také důvod, proč je dnes o nich hodně slyšet a těší se značné popularitě. Neuronové sítě představují paralelní výpočetní model, který mapuje vstup na výstup, a jenž je inspirován fungováním sítí biologických neuronů v centrální nervové soustavě. Neurony mají určité vstupy, k těmto vstupům dané váhy a takto váhovaný vstup nelineárně transformují na výstup. Neurony mohou být v rámci sítě různě propojeny, takže výstup jednoho neuronu je pak vstupem dalších neuronů. Podle toho, jak jsou tedy nastaveny váhy a propojeny neurony, je kvalitní i odezva sítě. Procesem učení neuronové sítě se nastavují váhy tak, aby byla odezva co nejkvalitnější. Neuronové sítě své uplatnění nacházejí v řadě oblastí – zejména při řešení úloh strojového učení jako je např. klasifikace, predikce a shlukování. [24][13]

V poslední době je možné pozorovat značný zájem o oblast klasifikace obrazových dat, kde se ukázaly být velmi efektivní tzv. konvoluční neuronové sítě (CNN). Tyto se inspirují způsobem jakým je v mozku zpracovávána obrazová informace buňkami ve vizuálním kortexu. Obsahují konvoluční a další speciální vrstvy, které se starají o extrakci příznaků vstupních obrazových dat. Tyto příznaky jsou poté vstupem pro dopředné plně propojené vrstvy neuronů (několik vrstev neuronů je za sebou a výstupy neuronů z jedné vrstvy jsou propojeny pouze se vstupy neuronů následující vrstvy). Zde je obrovská výhoda v tom, že zde dochází k automatickému výběru příznaků (nemusíme navrhovat speciální extraktor příznaků, který navíc je typicky vhodný jen pro konkrétním typ obrazových dat) a že tento automatický výběr příznaků funguje dobře, protože se zlepšuje učením sítě. CNN se bude tato práce dále věnovat více než jiným typů neuronových sítí. [35][27][48]

Další již zmíněnou technikou, která je inspirována přírodními procesy jsou evoluční algoritmy. Podobně jako neuronové sítě jsou zde s námi několik desítek let a neustále se vyvíjí a nacházejí nová uplatnění. Evoluční algoritmy patří z obecného hlediska mezi paralelní adaptivní metody pro prohledávání stavového prostoru možných řešení daného problému. Umožňují řešit některé složité optimalizační problémy, které by byly tradičními metodami

(analytickými, deterministickými, apod.) výpočetně nespočítatelné, a to velmi efektivním způsobem. Inspirují se především biologickou evolucí, ale v širším slova smyslu se zde řadí i algoritmy, které se inspirují i jinými biologickými jevy – např. sociálním chováním. Evoluční biologický proces simulují tak, že pracují s populací tzv. jedinců, kdy se tato populace během generací působením selekčního tlaku zlepšuje, a získáváme jedince s požadovanými vlastnostmi. Jedinci představují možné řešení úlohy (toto řešení mají zakódované ve svých genech – soubor všech genů tvoří chromozom), přičemž každý jedinec je ohodnocen podle kvality řešení, které představuje. Lepší jedinci mají větší šanci předat své geny do další generace. Aby docházelo skutečně k prohledávání stavového prostoru a objevování nových potenciálně dobrých řešení, tak se vybraní jedinci neumístí přímo do další generace, ale dostanou se tam jejich potomci. Ti vzniknou křížením, popř. mutací chromozomů rodičů. [46][38][13][52]

Evoluční algoritmy je možné použít pro návrh a učení neuronových sítí a tuto jejich aplikaci poté nazýváme neuroevoluce [37][49]. Cílem této práce je v tomto ohledu ukázat uplatnění neuroevoluce na dvou odlišných úlohách. První úloha spočívá v evolučním návrhu architektury konvoluční neuronové sítě (CNN), která by byla schopna (po řádném natrénování) s vysokou přesností klasifikovat ručně psané číslice z datasetu MNIST. Druhá úloha spočívá v evoluční optimalizaci vah neuronové sítě, která řídí přistání 1. stupně rakety Falcon 9 v rámci 2D simulace [45].

Cílem první úlohy je tak automatické nalezení kvalitních architektur pro problém klasifikace ručně psaných číslic. Přičemž v rámci experimentu je prohledávací prostor architektur omezený na nepříliš komplexní architektury. Architektura CNN hraje totiž podstatnou roli v tom, jak dobře bude schopna daná CNN klasifikovat vstupní data [19][32]. CNN s kvalitními architekturami by se v přesnosti klasifikace měly blížit state-of-the-art výsledkům pro dataset MNIST, které odpovídají přesnosti klasifikace 99,79%.

V době kdy jsem začal pracovat na této práci neexistovalo příliš mnoho publikovaných přístupů, které by pomocí evolučních algoritmů navrhovaly architekturu CNN, nebo vůbec přístupů, které by aplikovaly neuroevoluci na CNN. Což může být kvůli tomu, že CNN jsou poměrně novou záležitostí (a taky jak jsem se později přesvědčil sám, to může být z důvodu výpočetní náročnosti). Přesto pár publikovaných článků, které se zabývají spojením neuroevoluce a CNN existovalo. Jeden přístup např. vyvíjí váhy CNN pomocí genetického algoritmu (GA) [62], což ovšem není přesně to, na co se zaměřuje tato práce, která se spíše zaměřuje na návrh obecné architektury CNN. Další přístup, který se již zabývá vývojem architektury je zmíněn v článku [18], kde se používá algoritmus EXACT, založený na neuroevolučním algoritmu NEAT, který hledá velikosti konvolučních filtrů a jejich propojení. Ani toto však není vývoj architektury na obecné úrovni, na což se zaměřuje tato práce. Proto můj zvolený přístup nestaví na existujících přístupech, ale jde svou vlastní cestou.

Cílem druhé úlohy je poté úspěšné přistání 1. stupně rakety Falcon 9 (dále označován také jako raketa), a to v rámci experimentů s jinými parametry simulace a jinými počátečními podmínkami. Neuronová síť, která řídí toto přistání, je označována jako neurokontrolér; a má v rámci experimentů fixní architekturu, pro niž optimalizujeme váhy s ohledem na co nejkvalitnější přistání rakety.

Pro úlohu přistání rakety v rámci dané 2D simulace zatím neexistuje podobné řešení v podobě neuroevoluce. Nicméně pro příbuzné úlohy jako je přistání lunárního modulu ve 2D simulaci [8] existuje řešení pomocí neuroevolučního algoritmu NEAT [17]. Podobná úloha, která řešená pomocí neuroevoluce je přistávání helikoptéry [33]. Neuroevoluce je také použita pro opačnou úlohu, kterou je řízení letu (nikoliv přistání) rakety ve 3D simulaci [23]. Pro řešení této úlohy je aplikován neuroevoluční algoritmus ESP. V neposlední řadě je

úloha podobná i klasické úloze inverzního kyvadla, kde je třeba provádět jeho vyvažování, aby se nepřevrátilo. A i zde existuje řada řešení pomocí neuroevolučních algoritmů jako je NEAT nebo CMA-ES [47]. Zde je tedy částečně na co navázat, a to i když se jedná o úlohu, na kterou je neuroevoluce aplikována s největší pravděpodobností poprvé.

V této práci se kapitola 2 zabývá evolučními algoritmy z hlediska jejich historie, problematiky jejich vymezení, společných principů a vlastností. Dále jsou probrány vybrané evoluční algoritmy, většina jen přehledově. V podobném duchu se nese i kapitola 3, která nás uvádí do problematiky umělých neuronových sítí. V této kapitole je důraz kladen zejména na CNN se kterými bude souviset i praktická část práce. Na znalosti získané v předchozích kapitolách navazuje kapitola 4, ve které je popsán koncept neuroevoluce, kdy jsou evoluční algoritmy aplikovány při návrhu a učení neuronových sítí. Také jsou v této kapitole popsány možné aplikace neuroevoluce. V Kapitole 5 je představena technika evolučního návrhu architektury CNN, a její aplikace a vyhodnocení na problému klasifikace ručně psaných písmen. Tato kapitola představuje jeden z praktických přínosů práce. Kapitola 6 pak představuje aplikaci neuroevoluce v úloze řízeného přistání 1. stupně rakety Falcon 9 v rámci 2D simulace. Je zde provedena řada experimentů pro různé počáteční podmínky rakety a parametry simulace. Tato kapitola tvoří další praktický přínos práce. A nakonec kapitola 7 shrnuje dosažené výsledky, zejména z pohledu vlastního autorova přínosu a nastiňuje možné pokračování v dané práci.

Kapitola 2

Evoluční algoritmy

Evoluční algoritmy, jak již bylo v úvodu nastíněno, řadíme obecně mezi prohledávací algoritmy. Jsou charakteristické tím, že se inspirují přírodními procesy, a díky této inspiraci prohledávají stavový prostor možných řešení daného problému paralelním adaptivním způsobem. Tradičně se zde řadí algoritmy, které jsou inspirovány biologickou evolucí, resp. neodarwinismem. Méně tradičně se zde ale řadí i algoritmy inspirované jinými přírodními procesy. Problematika přesného vymezení třídy evolučních algoritmů bude probrána později v tomto úvodu do evolučních algoritmů. [40][13][52][56]

Evoluční algoritmy se uplatňují při řešení optimalizačních problémů, se kterými se potkáváme v nejrůznějších oblastech od ekonomie, přes strojové učení, kybernetiku, bioinformatiku až po letecký průmysl. Optimalizační problém spočívá v nalezení takových hodnot parametrů účelové funkce, aby hodnota dané funkce dosáhla globálního extrému (minima v případě minimalizace účelové funkce, nebo maxima v případě maximalizace účelové funkce). Účelová funkce tedy slouží k vyhodnocení kvality potenciálního řešení a způsob jakým je konstruována má citelný dopad na výsledky optimalizace. Z hlediska optimalizace řadíme evoluční algoritmy mezi tzv. metaheuristiky. Jedná se o obecné algoritmy, které poskytují obecný rámec pro řešení optimalizačního problému, nejsou tedy zaměřeny pro řešení určitého problému, ale pro širokou škálu různých optimalizačních problémů. Pro tyto problémy je možné si tyto algoritmy přizpůsobit např. pomocí nastavení jejich parametrů, implementace některých jejich komponent, apod. Evoluční algoritmy jsou specificky zvláštní případ metaheuristik, jedná se o tzv. přírodou inspirované populační stochastické metaheuristiky. Populační znamená to, že pracují s populací tzv. jedinců, tito jedinci představují kandidátní řešení daného optimalizačního problému. Jsme tak schopni získat více možných řešení, obvykle nás však zajímá řešení nejlepší. Naproti tomu existují metody založené na jednom řešení, které iterativně zlepšují. Stochastické znamená, že využívají prvek náhody, takže při jejich opakovaném spuštění dostaneme různé výsledky i když jsou jinak počáteční podmínky (mimo inicializace generátoru pseudoháhodných čísel) stejné. Toto může být v některých případech nevýhoda, v jiných případech výhoda. Naproti tomu existují deterministické metody, které vždy pro stejné počáteční podmínky vrátí stejný výsledek. Evoluční algoritmy (a metaheuristiky obecně) můžeme hodnotit z hlediska jejich schopnosti prozkoumávat nové oblasti prohledávacího prostoru, které se mohou ukázat jako potenciálně slibné (diverzifikace, explorace) a schopnosti detailně prozkoumávat již objevené slibné oblasti prohledávacího prostoru za účelem nalezení ještě lepších řešení (intensifikace, exploitace). Tyto schopnosti tvoří základ každého evolučního algoritmu, ale směřují proti sobě. Je tak potřeba jejich optimálního vyvážení, což pak vede k lepší konvergenci algoritmu při hledání globálního optima. [46][58][61]

Pro určitou třídu optimalizačních úloh, které se vyznačují zejména svou komplexností (mnoho omezujících podmínek, rozsáhlý prohledávací prostor, složitý tvar účelové funkce, změna účelová funkce v čase, apod.) nejsou klasické optimalizační (exaktní) metody vhodné (kvůli paměťové nebo spíše časové náročnosti), nebo vůbec použitelné. Při řešení těchto složitých optimalizačních úloh jsou mnohem úspěšnější právě metaheuristiky, resp. evoluční algoritmy. Tyto metody totiž umožňují prohledat efektivním způsobem prohledávací prostor možných řešení problému a umožňují nalézt přijatelné řešení v přijatelném čase. Obecně však negarantují nalezení optimálního řešení ani negarantují mez pro nejhorší řešení – nejsou totiž na rozdíl od exaktních optimalizačních metod matematicky dokázány a vycházejí pouze z intuice a pozitivní zkušenosti, která ukazuje, že tyto algoritmy obvykle nacházejí dobrá řešení. Pokud by nás zajímalo, jak si vedou metaheuristiky, resp. evoluční algoritmy obecně (tedy nejen při řešení určité třídy komplexních optimalizačních problémů) oproti exaktním algoritmům, případně který evoluční algoritmus je nejlepší, tak v této souvislsoti je vhodné zmínit existenci tzv. No Free Luch Teorému. Tento v podstatě říká, že průměrná výkonnost všech optimalizačních algoritmů na všech možných problémech je stejná. Což znamená, že pokud je optimalizační algoritmus A schopný řešit určitou třídu problémů lépe než jiný algoritmus B, tak musí naopak existovat třída problémů kterou dokáže lépe řešit algoritmus B. Tedy evoluční algoritmy a obecně metaheuristiky ačkoliv jsou schopny řešit poměrně širokou třídu úloh, tak pro některé úlohy existují vhodnější (výkonnější) metody. [46][58][61][64][56][38]

2.1 Stručná historie

Počátky evolučních algoritmů bychom mohli najít už v roce 1948, kdy slavný matematik a zakladatel počítačových věd Alan Turing [28] ve své eseji *Inteligentní stroje* představil myšlenku, že výzkum v oblasti umělé inteligence bude v budoucnu spjat s prohledáváním v prostoru celých čísel, a že toto prohledávání bude pravděpodobně založeno na genetickém/evolučním prohledávání. V roce 1950 pak v článku *Computering Machinery and Inteligence* s touto myšlenkou dále pracoval a formuloval tak základní principy evolučních algoritmů. Nejspíše první počítačovou simulaci evolučního procesu pak provedl N. A. Barricelli v roce 1953. V roce 1958 byl následován R. M. Friedbergem, který experimentoval s evolučním algoritmem, který bychom mohli považovat za předchůdce dnešního *genetického programování*. Největší a nejzásadnější dopad měli však následující tři práce, které byly vytvořeny nezávisle na sobě. V roce 1963 Ingo Rechenberg and Hans-Paul Schwefel vytvořili algoritmus *evoluční strategie (ES)*, který použili pro řešení obtížných optimalizačních problémů s reálnými hodnotami parametrů. V roce 1966 Larry Fogel vytvořil algoritmus *evoluční programování (EP)*, který měl sloužit k návrhu konečného automatu pro řízení inteligentního agenta. A nakonec také někdy na přelomu 60. a 70. let John Holland vyvinul *genetický algoritmus* pro navrhování adaptivních systémů. Tyto práce jsou tedy z historického hlediska nejzásadnější pro rozvoj oboru evolučních algoritmů / evolučního počítání. Obor se nadále rozvíjel, kdy byla vyvinuta pestrá škála dalších evolučních algoritmů. Jen pro přehled uveďme roky vzniku/publikování a názvy některých dalších evolučních algoritmů: 1986 – Umělý imunitní systém (AIS), 1992 – Optimalizace mravenčí kolonií (ACO), 1992 – Genetické programování, 1995 – Optimalizace hejnem částic (PSO), 1996/1997 – Diferenciální evoluce (DE), 2005 – Algoritmus umělého včelího roje (ABC) a Optimalizace hejnem světlušek (FA). [52][13][46][61]

2.2 Problematika vymezení evolučních algoritmů

Co se týče vymezení třídy evolučních algoritmů, a tedy identifikace těch algoritmů, které do ní spadají, tak jsou dnes tyto hranice poměrně rozostřené. Klasicky se zde řadí algoritmy, které využívají princip biologické evoluce jako: genetický algoritmus (GA), evoluční programování (EP), evoluční strategie (ES), genetické programování (GP) a diferenciální evoluce (DE). [13] Někteří autoři zařazují do třídy evolučních algoritmů, resp. evolučního počítání i jiné populačně založené metaheuristiky, které nejsou založené přímo na procesu biologické evoluce, ale jsou jinak biologicky inspirovány (např. sociálním chováním). Takovými algoritmy jsou např. optimalizace hejnem částic (PSO, mimochodem tento algoritmus zařazuje to třídy evolučních algoritmů i jeden z jeho autorů), optimalizace mravenčí kolonií (ACO), algoritmus umělého včelího roje (Artificial Bee Colony Algorithm, ABC) , algoritmus umělého imunitního systému aj. [56][60][64][22][54] Jiní autoři tyto nezařazují přímo do třídy evolučních algoritmů, ale do třídy populačně založených metaheuristik, které jsou podobné klasickým evolučním algoritmům. [58][13]

Klasifikaci algoritmů nám ani neusnadňuje fakt, že existují různé hybridní přístupy založené např. na kombinaci GA a PSO [10] nebo kombinaci GA a ABC [63]. Tyto hybridní algoritmy kombinující sociální chování a evoluci, ale i další jiné nové přístupy jako EDA (Odhad distribučními funkcemi) [56] nebo Memetické algoritmy (MA) [34] se v určitých aspektech liší od tradičních evolučních algoritmů. A taky zde není konvence v jejich zařazení. Některými autory jsou tak řazeny mezi evolučními algoritmy, jinými nikoliv.

Jsou tak autoři, kteří se s těmito obtížnostmi klasifikace různých algoritmů vyrovnali tak, že rozdělují evoluční algoritmy na 2 proudy – klasické a novější [56]. Členění různých algoritmů do nějakých tříd tedy není zcela ustálené. Každé dělení má svou určitou logiku, poslední uvedený způsob se jeví jako přijatelný kompromis. V této diplomové práci přistoupíme k podobnému vymezení algoritmů, kdy rozdělíme evoluční algoritmy na konvenční (klasické algoritmy, založené na biologické evoluci) a méně nekonvenční (v tuto chvíli spíše novější přírodou inspirované populačně založené metaheuristiky, které nezapadají do klasického učebnicového členění). Je vhodné připomenout, že i algoritmy, které se neinspirují přímo procesem biologické evoluce, ale jsou postaveny např. na sociálním chování, se mohou snadno schovat pod evoluční terminologii. Protože i takové sociální nebo podobné chování se vyvinulo evolučně [40] a je tedy evolučně podmíněné. Tyto algoritmy tedy těží z výsledků evoluce – a i tím je možné odůvodnit proč je řadit mezi evoluční. Jsou také autoři, kteří uvádějí, že se název *evoluční* odvozuje od toho, že se populace jedinců v čase vyvíjí [46]. Členění evolučních algoritmů na konvenční a méně konvenční (u nich nepanuje obecně konvence kam je řadit) se tedy bude tato diplomová práce dále držet.

2.3 Společné principy

Evoluční algoritmy jsou založeny na postupném vývoji populace jedinců v čase napříč generacemi. V 1. generaci je vygenerována počáteční populace jedinců (např. náhodně nebo pomocí nějaké heuristiky) a tato populace se následně iterativně vyvíjí. V nové generaci je populace z předchozí generace nahrazena novou populací jedinců, přičemž počet jedinců v populaci je obvykle konstantní. Jedinci tvořící novou populaci jsou vybráni z množiny, která obsahuje staré jedince a jedince, kteří byli vygenerováni na základě starých jedinců. Tento výběr nemusí být explicitní a novou populaci mohou implicitně tvořit pouze nově vygenerovaní jedinci. Způsob generování nových jedinců na základě starých jedinců (případně i výběr jedinců do nové populace) má stochastický charakter, kdy se využívá kombinace

prvku náhody a existující informace o zajímavých oblastech prohledávacího prostoru, které reprezentují relativně kvalitní jedinci. Toto je prováděno takovým způsobem, který zajišťuje (měl by zajistit) to, že se průměrná kvalita jedinců napříč generacemi nezhoršuje (ideálně zvyšuje). Kvalita jedince udává kvalitu kandidátního řešení optimalizačního problému, které jedinec reprezentuje a kvalita tohoto řešení je ohodnocena pomocí tzv. fitness funkce (což může být přímo účelová funkce, nebo transformovaná účelová funkce). Vývojem populace dle výše uvedených principů dochází k prohledávání stavového prostoru kandidátních řešení (parametrů účelové funkce), a toto prohledávání konverguje k lepším řešením, resp. v ideálním případě (pokud je algoritmus správně navrhnut) nejlepšímu řešení (globálnímu extrému účelové funkce). Podmínkou ukončení může být např. dosažení stanoveného počtu generací, nebo dosažení stavu, kdy po několik generací nedošlo ke zkvalitnění populace jedinců, nebo kombinace obou podmínek. [46][58][21]

Takovýto obecný princip evolučního algoritmu je zapsán pomocí pseudokódu v algoritmu 1. Pro získání lepší představy, jak by mohl tento algoritmus v nějaké jeho konkrétní realizaci na konkrétním problému fungovat, nám může posloužit obrázek 2.1. Zde je na algoritmu Diferenciální evoluce ukázáno, jak jedinci v populaci během generací postupně při prohledávání konvergují k nejlepšímu řešení – globálnímu minimu účelové funkce. Červené body ležící na povrchu účelové funkce (v 3D prostoru) představují hodnoty této funkce pro jedince v populaci (kteří mají šedou barvu a nacházejí se v prohledávaném 2D prostoru kandidátních řešení). Jako účelová funkce je zvolena Ackleyho funkce, která obsahuje jedno globální minimum a řadu lokálních minim. Tato funkce je jednou relevantních testovacích funkcí, na které se běžně vyhodnocuje robustnost evolučních algoritmů [46][58]

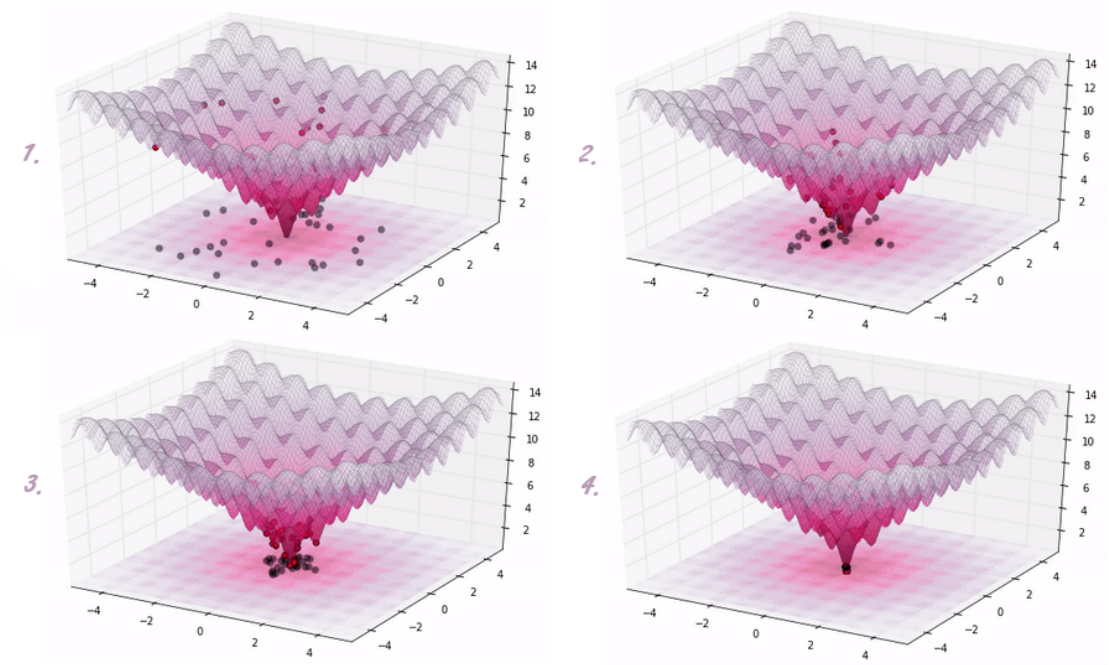
Algoritmus 1: Podoba obecného evolučního algoritmu. Zdroj: [58]

$t \leftarrow 0$;
Inicializuj populaci jedinců reprezentujících kandidátních řešení P_t;
repeat
  Vygeneruj nové jedince P'_t na základě jedinců z P_t;
  Do nové populace P_{t+1} vyber jedince z $P_t \cup P'_t$;
  $t \leftarrow t + 1$;
until *není splněno ukončující kritérium*;

Evoluční algoritmy se liší podle toho, jakým způsobem provádějí generování jedinců (a případně i jejich nahrazování v populaci) a jestli při tom využívají paměť. *Metody bez paměti* jsou ty, které pro tyto účely využívají jen aktuální populaci jedinců. Naopak *metody s pamětí* jsou ty, které pro tyto účely využívají nejen aktuální populaci, ale i informace z předchozích hledání. [46][58]

Při návrhu evolučního algoritmů je potřeba zvolit způsob reprezentace kandidátního řešení (tedy jak bude toto řešení zakódováno) a podobu fitness (účelové) funkce, která vyhodnocuje kvalitu tohoto řešení. Obojí závisí na typu řešeného problému a obojí má přitom dopad na efektivitu evolučního algoritmu. Vyhodnocování řešení v rámci fitness funkce zabere v praxi drtivou většinu výpočetního času, někdy i tolik, že je potřeba přistoupit k meta-modelování a aproximovat model výpočtu pomocí jiného výpočetně méně náročného modelu. [58]

Evoluční algoritmus pracuje primárně s reprezentací řešení – např. se může jednat o matici binárních číslic, která reprezentuje spojení mezi neurony v neuronové sítí. Jedincem se tak myslí obvykle přímo dané řešení, nikoliv jeho reprezentace. V případě, že je potřeba



Obrázek 2.1: Ukázka čtyř vybraných fází algoritmu diferenciální evoluce na problému minimalizace účelové funkce (zde zvolena Ackleyho funkce). Populace jedinců (šedé body, červené body jsou jim odpovídající hodnoty účelové funkce) během prohledávacího procesu postupně konverguje k nejlepšímu řešení – globálnímu minimu účelové funkce. (Obrázek vytvořen na základě zdroje: [43])

ohodnotit kvalitu řešení – např. neuronovou síť, tak se toto řešení získá převodem z jeho reprezentace (např. booleovská matice spojení neuronů). V některých případech řešení a jeho reprezentace splývají. Reprezentace by měla mít tyto vlastnosti: [58]

- **Úplnost** – Všechna potenciální řešení musí být reprezentovatelná.
- **Souvislost** – Každé řešení musí být v prohledávacím prostoru reprezentací dosažitelné.
- **Efektivnost** – Reprezentace musí být navržena tak, aby práce ní byla co nejefektivnější.

V ideálním případě by také měla mít reprezentace také vlastnost *lokality*, která zaručuje, že malé resp. velké změny v reprezentaci řešení způsobí malé resp. velké změny v řešení [13]. Typy reprezentace jsou obecně následující: [58][46][21]

- **Lineární reprezentace** – Jedná se o nějaký řetězec znaků z dané abecedy. Např. vektory binárních číslic, celých čísel, reálných čísel, případně jejich kombinace. Pokud sekvence binárních číslic reprezentuje nějaké číslo, nemusí být při drobných změnách splněna vlastnost lokality, proto se v takových případech používá speciální kódování (např. Grayovo kódování).
- **Nelineární reprezentace** – Jedná se o komplexnější struktury jako např. grafy – často stromy.

Některé evoluční algoritmy mohou být spjaty jen s určitým typem reprezentace řešení (např. genetické programování se stromy). Mezi prostorem řešení a prostorem reprezentací řešení může být mapování: [58]

- **1:1** – Obvyklé mapování, pro některé problémy obtížně dosažitelné.
- **1:N** – Řešení může být reprezentováno více způsoby, což může mít negativní dopad na výkonnost prohledávacího procesu.
- **N:1** – Více řešení může být reprezentováno jedním způsobem, což může mít pozitivní dopad na výkonnost prohledávání. Jedná se o tzv. nepřímé kódování, kde chybí kompletní informace o řešení. Např. booleovská matice spojení neuronů může reprezentovat více možných neuronových sítí – konkrétní získáme dosazením vah.

2.3.1 Evoluční algoritmy založené na biologické evoluci

Evoluční algoritmy založené na biologické evoluci tvoří významnou podmnožinu evolučních algoritmů, proto si jejich společné koncepty popíšeme. Tyto algoritmy simulují evoluční proces druhu organismu, přičemž vychází z neodarwinismu, který je založen na Darwinově evoluční teorii a poznatcích z molekulární biologie. Algoritmy jsou speciálním případem obecného evolučního algoritmu. Pracujeme zde opět s nějakou populací jedinců (kteří reprezentují kandidátní řešení a jsou ohodnoceni fitness funkcí), která se v čase napříč generacemi vyvíjí. Způsob vývoje však simuluje evoluční proces druhu, který je založený na: [52] [46][13][21]

- Generování rozmanitosti v populaci pomocí reprodukce (sexuální či asexuální).
- Selekčním tlaku, který umožňuje jedincům s vyšší fitness, kteří vykazují lepší stupeň adaptace na prostředí / lepší kvalitu, se s větší pravděpodobností účastnit reprodukce a také s větší pravděpodobností přežít do další generace.

Krok generování nových jedinců z obecného evolučního algoritmu tak u těchto evolučních algoritmů založených na biologické evoluci sestává z výběru rodičů a tvorby jejich potomků pomocí variačních operátorů. Výběr jedinců do další generace je pak v podstatě podobný jako u obecného evolučního algoritmu, kdy se obecně volí z množiny potomků a rodičů podle nějakého klíče. Např. se vybírá podle fitness, nebo se vybírají jen potomci, apod. Uvedené principy těchto algoritmů jsou zapsány pomocí pseudokódu 2. [52][46][13][21]

Algoritmus 2: Podoba obecného evolučního algoritmu, který je inspirován principy biologické evoluce. Zdroj: [58]

$t \leftarrow 0$;
Inicializuj populaci jedinců reprezentujících kandidátních řešení P_t;
Ohodnoť kvalitu jedinců P_t;
while *není splněno ukončující kritérium* **do**
    Vyber rodiče P'_t z P_t;
    Vytvoř potomky P''_t reprodukcí rodičů P'_t;
    Ohodnoť kvalitu potomků P''_t;
    Do nové populace P_{t+1} vyber jedince z $P_t \cup P''_t$;
    $t \leftarrow t + 1$;
end

S řešením a reprezentací řešení se u těchto algoritmů pojí tyto pojmy: [55][21]

Genotyp Představuje reprezentaci kandidátního řešení. Jinak se také nazývá chromozom. Pracuje s ním evoluční algoritmus (např. během reprodukce při vytváření potomka). Jako základní manipulační jednoty má tzv. geny, hodnoty těchto genů jsou potom alely a pozice genu v chromozomu je lokus.

Fenotyp Představuje kandidátní řešení. Pracuje s ním fitness funkce.

Mechanismus výběru rodičů pro reprodukci spolu s mechanismem výběru jedinců do další generace určují selekční tlak (jak moc jsou při výběru preferováni lepší jedinci), který ovlivňuje míru exploitace algoritmu. Selekční tlak je motorem evolučního procesu a není dobré když je příliš nízký (což vede na neefektivní prohledávání prostoru) ani příliš vysoký (což snižuje diverzitu populace a zvyšuje pravděpodobnost zaseknutí se v lokálním extrému). Do celé hry však vstupuje i mechanismus reprodukce, kdy pomocí něj můžeme docílit např. agresivnější explorace algoritmu a tím vyvážit např. příliš silnou selekci (s čímž se můžeme setkat např. u evolučních strategií (ES)). Při malém selekčním tlaku a velké reprodukční diverzitě získáváme exploračně orientovaný algoritmus, zatímco při velkém selekčním tlaku a malé reprodukční diverzitě získáváme exploitačně orientovaný algoritmus. Při návrhu mechanismů selekce rodičů, reprodukce a selekce jedinců do další generace nám tak jde především o celkové správné vyvážení explorace a exploitace algoritmu. Počet vybraných rodičů, resp. počet produkovaných potomků může být obecně menší, stejný, nebo větší než je velikost populace, různé varianty evolučních algoritmů mají obvykle vlastní strategii kolik potomků vytvořit a jak provést výběr jedinců do další generace. [58][13][52]

Mechanismus výběru rodičů může být náhodný, nebo deterministický kdy vybíráme jen ty nejlepší rodiče. Může být (a také často bývá) pravděpodobnostní, kdy kvalitnější jedinci mají větší šanci, že vytvoří potomky, čímž vzniká určitý selekční tlak. Pravděpodobnostní charakter dovoluje, aby mohli být pro reprodukci (s jistou malou pravděpodobností) vybráni i vyloženě nekvalitní jedinci. Díky tomuto můžeme při prohledávání prohledávacího prostoru uniknout z lokálního extrému fitness funkce (pokud nemáme jinak ku pomoci reprodukční operátor se silným stupněm produkované diverzity). Možných pravděpodobnostních strategií pro výběr rodičů je několik: [58][13][52][21]

- **Výběr ruletovým kolem** – Pravděpodobnost výběru jedince je úměrná vzhledem k jeho fitness hodnotě. Uvažujeme vnitřní otáčivou část ruletového kola jako koláčový graf, kde určitý jedinec má na tomto koláči podíl úměrný jeho fitness hodnotě. Vnější neotáčivá část rulety obsahuje ručičku, která vybírá jedince. Pokud chceme vybrat μ jedinců, provedeme μ nezávislých roztočení rulety. Viz ukázku 2.2. Tato metoda ale může v některých případech vykazovat předčasnou konvergenci (pokud je na začátku v populaci několik málo velmi dobrých jedinců, tak brzy zaberou celou populaci na úkor horších jedinců a prohledávání se předčasně soustředí jen do určitých oblastí) nebo nízký selekční tlak (pokud jsou fitness hodnoty jedinců velmi podobné).

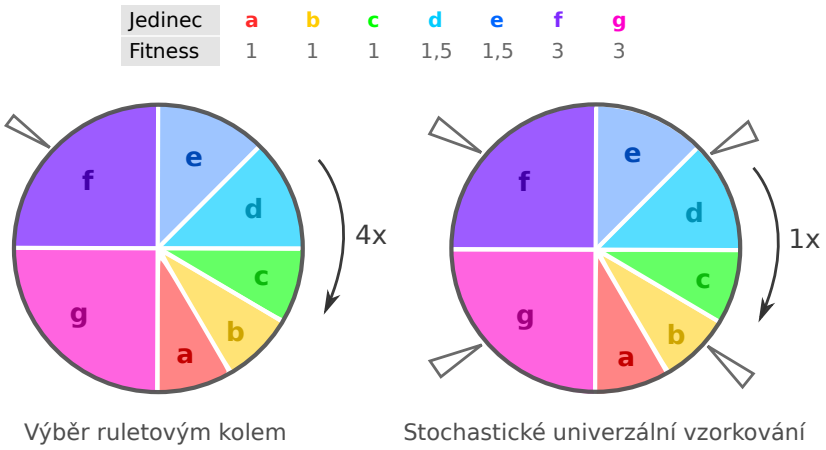
- **Stochastické univerzální vzorkování** – Podobné jako u výběru ruletovým kolem. Na vnější neotáčivé části rulety je však umístěno s rovnoměrným rozestupem tolik ručiček, kolik chceme vybrat jedinců. Tyto jedince pak vybereme pouhým jedním roztočením rulety. Tento způsob odstraňuje problémy klasického výběru ruletovým kolem. Viz ukázku 2.2.

- **Výběr založený na pořadí** – Jedinci jsou seřazeni podle své fitness a podle jejich pořadí je pro každého jedince vypočítána tzv. přeškálované fitness $f_{rescaled}(i)$ podle

vztahu 2.1 [13], která se poté uplatňuje při výběru ruletovým kolem. Je to opět další způsob jak odstranit nevýhody klasického ruletového kola. V daném vztahu 2.1 proměnná $P \in \langle 1.0, 2.0 \rangle$ značí škálovací faktor, který určuje velikost selekčního tlaku), proměnná n pak značí velikost populace (a zároveň udává rank nejhoršího jedince), proměnná $rank(i)$ udává pořadí daného jedince i.

$$f_{rescaled}(i) = 2 - P + 2(P-1)\frac{(rank(i)-1)}{(n-1)} \tag{2.1}$$

- **Výběr turnajem** – Náhodně se vybere k jedinců, kde k je velikost turnaje a z těchto jedinců se vybere nejlepší. Pokud chceme vybrat μ jedinců, opakujeme μ krát turnaj. Čím vyšší hodnota k, tím větší je selekční tlak. Tato strategie opět překonává nevýhody klasického výběru pomocí ruletového kola. Také nevyžaduje absolutní vyjádření fitness hodnoty jedinců, ale jen jejich relativní porovnání kvality.



Obrázek 2.2: Ukázka výběru 4 rodičů pomocí metody *výběru ruletovým kolem* a metody *stochastického univerzálního kódování*. U první metody je potřeba pro výběr 4 rodičů 4krát nezávisle roztočit ruletu, zatímco u druhé metody stačí jedno roztočení a ručičky nám vyberou 4 rodiče. Povšimněme si také, jak velikost výseče (a tedy pravděpodobnost výběru jedince) je úměrná fitness jedince. (Obrázek vytvořen na základě zdroje: [58])

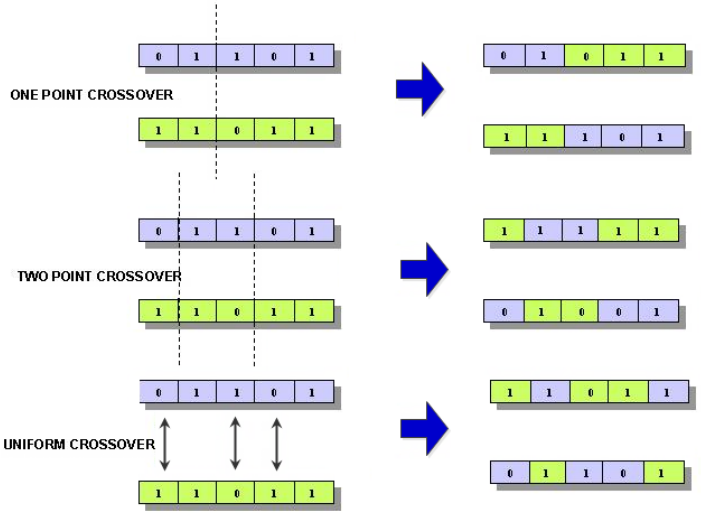
Mechanismus reprodukce používá reprodukční operátory, jinak nazývané variační operátory. Tyto operátory vytvářejí variaci v populaci, kdy se na základě rodičů snažíme vyprodukovat o něco kvalitnější potomky. Těmito operátory jsou křížení a mutace. Tyto reprodukční operátory pracují nad genotypem, který ovlivňuje efektivitu těchto variačních operátorů. Reprodukce může být sexuální, nebo asexuální. V případě sexuální reprodukce je potomek (nebo potomci) vytvořen pomocí křížení obvykle dvou (méně častěji více) rodičů a možné následné mutace takového embrya. Ve případě asexuální reprodukce je potomek vytvořen na základě jednoho rodiče a následné mutace. Asexuální reprodukce má tendenci na rozdíl od sexuální reprodukce prohledávat spíše lokální okolí, záleží však na nastavení agresivity mutace. Z hlediska efektivity asexuální reprodukce je tedy klíčové, aby se očekávaná vzdálenost potomka od rodiče (ovlivněná stupněm mutace) měnila dynamicky podle potřeby, resp. podmínek (tvar plochy fitness funkce). Variační operátory jsou aplikovány obvykle s určitou pravděpodobností, a tedy potomci mohou být i prostou kopií svých rodičů. [58][52]

Křížení je aplikováno na $n > 1$ rodičů za vzniku jednoho či více potomků. Obvykle se aplikuje na 2 rodiče, méně častěji se aplikuje i na více rodičů, což ale nemá biologický ekvivalent v přírodě. Potomci poté obsahují zděděné vlastnosti od svých rodičů. Křížení je stochastické, kdy se náhodně určí části genotypu rodičů, které potomek zdědí. Křížení by mělo mít tyto vlastnosti: [58][21]

- **Dědičnost** – Operátor křížení zajišťuje, to že potomci zdědí znaky od rodičů, tedy stejní rodiče by měli vyprodukovat opět stejné potomky. Z toho důvodu (a také z důvodu, že operátor křížení nezajišťuje) je vhodné křížení doplnit i mutací, která může zajistit potřebnou diverzifikaci.
- **Validita** – Operátor křížení by měl být schopný najít jen ty body stavového prostoru, které reprezentují validní řešení. Tohoto může být obtížné dosáhnout při řešení optimalizačních problémů s omezujícími podmínkami.
- **Další vlastnosti** – Společné vlastnosti rodičů by měli být během křížením zachovány a rozdílnost rodičů (měřena vzdáleností v prohledávacím prostoru) by neměla být menší než rozdílnost rodičů a potomků.

Existují různé typy křížení, podle typu reprezentace (genotypu), uveďme si některé z nich: [13][58][56]

- **Křížení diskrétních / binárních vektorů** – Může být jednobodové, či vícebodové, kdy se náhodně zvolí jeden či více bodů křížení, které rozdělí genotyp na segmenty. Každý potomek poté vznikne složením segmentů z různých rodičů. Křížení také může být uniformní, kdy se pro náhodně vybrané geny 2 rodičů přehodí jejich hodnoty a takto získáme 2 potomky. Pro názornou ukázku viz 2.3.
- **Křížení reálných vektorů** – Může probíhat obdobně jako u diskrétního křížení (n-bodové, uniformní), kdy vzniká více potomků. Také je možné vytvořit z rodičů jednoho potomka tak, že se např. zprůměrují prvky vektorů rodičů, nebo se s těmito reálnými hodnotami provede jiná transformace – např. váhování. Takový způsob se pak nazývá aritmetické křížení.
- **Křížení stromů** – Obvykle probíhá tak, že se náhodně vyberou podstromy obou rodičů a tyto se vymění, čímž vzniknou 2 modifikovaní potomci.



Obrázek 2.3: Ukázka jednobodového, dvoubodového a uniformního křížení binárních vektorů. (Zdroj: [1])

Mutace je aplikována na jednoho jedince, čímž vznikne mutant (potomek), který může být více, či méně odlišný od rodiče podle stupně mutace. Mutace je schopna dodat chybějící genetickou informaci do populace, čímž zajišťuje explorativní schopnosti EA. Mutace je stochastický operátor (stejně jako křížení a na rozdíl od selekce, která může být deterministická), který provádí náhodné změny v genotypu. Mutace by měla mít tyto vlastnosti: [58][21][56]

- **Ergodicita** – Mutační operátor by měl být schopný dosáhnout každého bodu (který reprezentuje řešení) v prohledávacím prostoru. Měl by tak zajišťovat spojitost prohledávacího prostoru, kdy se z každého bodu mohu dostat do jakéhokoliv jiného bodu.
- **Validita** – Mutační operátor by měl být schopný najít jen ty body stavového prostoru, které reprezentují validní řešení. Tohoto může být obtížné dosáhnout při řešení optimalizačních problémů s omezujícími podmínkami.
- **Lokalita** – Měla by platit přímá úměra mezi změnami genotyu a fenotypu, tedy malé změny v genotypu způsobené mutací by se měly ve fenotypu projevit také malými změnami. Pokud by platil pravý opak mohlo by prohledávání vykazovat náhodný charakter.

Existují různé typy mutace, podle typu reprezentace (genotypu), uveďme si některé z nich:[13][56][46]

- **Mutace diskrétního / binárního vektoru** – S určitou pravděpodobností se zmutuje každý prvek (gen) vektoru (genotypu) a to tak, že se hodnota daného prvku nastaví na jinou hodnotu z dané abecedy (což u binárního vektoru znamená prohození 0 za 1 a naopak).

- **Mutace reálného vektoru** – S určitou pravděpodobností se zmutuje každý prvek (gen) vektoru (genotypu) a to tak, že se hodnota daného prvku zmenší či zvětší o nějakou náhodně vygenerovanou hodnotu. Často se používá Gaussovská mutace, kdy se k prvku vektoru přičte hodnota náhodné veličiny s normálním rozložení pravděpodobnosti se středem v nule a standardní odchylkou σ definovanou uživatelem: N(0, σ). Takto se nejčastěji bude hodnota genu měnit o malé hodnoty, ale občas se může objevit velká změna.

- **Mutace stromu** – Mutace stromu může být realizována řadou způsobů, jedna z možných (používána např. v GP) mutací spočívá v náhodném zvolení podstromu a jeho nahrazením jiným náhodně vygenerovaným podstromem.

Mechanismů výběru jedinců do další generace, resp. nahrazení jedinců současné populace může být velké množství, kdy každý vytváří různý stupeň selekčního tlaku. Jelikož selekční tlak může vznikat i při výběru rodičů pro reprodukci, je vhodné jeho sílu vyvážit odpovídajícím selekčním tlakem při výběru jedinců do další generace. Selekce přeživších jedinců je na rozdíl od selekce rodičů často spíše deterministická než stochastická. Velikost populace se pak obvykle napříč generacemi neměnní, tedy vždy přežije stejné množství jedinců. Mezi možné strategie nahrazení populace / výběru jedinců do další generace patří např. následující: [13][58][21][52]

- **Generační obměna** – Potomci nahrazují celou původní generaci. Pokud je potomků více než jedinců původní generace, je možné přistoupit k jejich redukci pomocí různých mechanismů (výběr nejlepších, náhodný výběr, deterministický výběr). Tento způsob zajišťuje menší selekční tlak a je obvyklý např. u GA.

- **Výběr nejlepších** – Nejlepší jedinci ze současné populace a potomků jsou vybráni do další generace. Tento způsob vytváří poměrně silný selekční tlak a je obvyklý např. u ES.

- **Steady-state nahrazení** – Za jednu generaci je vyprodukován jeden nebo několik málo potomků a tito nahradí v populaci nejhorší jedince.

- **Náhodné nahrazení** – Náhodně se vyberou jedinci ze současné populace a z potomků. Tento způsob zajišťuje minimální selekční tlak a může vyvažovat silnou selekci rodičů.

- **Elitismus** – Nejlepší jedinec, nebo několik málo nejlepších jedinců ze současné populace vždy přežijí do další generace. Jedná se pouze o doplněk k dalším způsobům nahrazení populace, který zajistí, že se dobrý genetický materiál neztratí.

2.4 Konvenční evoluční algoritmy

Jak již bylo dříve uvedeno, jako konvenční evoluční algoritmy označujeme ty algoritmy u kterých panuje široká shoda v zařazení mezi evoluční algoritmy. Jedná se o algoritmy s určitou historií a tradicí. Tyto algoritmy spadají obecně do třídy přírodou inspirovaných populačně založených metaheuristik a v rámci této třídy se vyznačují tím, že jsou inspirovány biologickou evolucí, resp. neodarwinismem.

2.4.1 GA – Genetický algoritmus

GA dnes představuje jeden z nejpopulárnějších, resp. nejpoužívanějších evolučních algoritmů. Spolu s ES a evolučním programováním (EP) významným způsobem formoval obor evolučního počítání. Algoritmus byl zpopularizován v 70. letech (jeho kořeny však sahají přibližně do 60. let) Johnem Hollandem, který působil na Univerzitě v Mitchiganu, a to v jeho knize *Adaptation in Natural and Artificial Systems*. Původně měl GA pomoci porozumět adaptacím v přírodních systémech. Pro účely optimalizace, kde nachází dnes především své uplatnění, se GA začal používat o něco později. GA nabývá různých podob a modifikací, zde si představíme jeho kanonickou verzi. GA rozlišuje genotyp a fenotyp. Genotyp má fixní velikost a pro reprezentaci fenotypu se používají tradičně binární vektory i když dnes jsou rozšířené i jiné druhy reprezentace. Obvykle se při selekci rodičů uplatňuje pravděpodobnostní výběr (např. turnaj, ruleta, apod). Reprodukce probíhá sexuálně, kdy se zkříží 2 rodiče, čímž vzniknou 2 potomci, kteří se navíc ještě zmutují. Pravděpodobnost křížení je spíše větší, neboť se na tento variační operátor klade velký důraz a pravděpodobnost, že proběhne mutace je spíše menší, přičemž míra mutace je fixní a poměrně malá. Křížení bývá obvykle n-bodové nebo uniformní, mutace poté obvykle prohazuje náhodně vybrané bity v genotypu. Strategie náhrady populace je generační obměna, kdy novou generaci tvoří jen potomci. Celkový selekční tlak tedy není příliš velký, což vzhledem k poměrně nízké agresivitě mutace vyvažuje explorativní a exploitativní schopnost algoritmu. U GA se uvažuje populace obvykle v řádu desítek až stovek jedinců. Shrnutí základních vlastností klasického GA přehledně reprezentuje tabulka 2.1. Princip činnosti GA je poté ilustrován v pseudokódu 3. [13][58][56][55][21]

Existuje i paralelní verze GA, označována jako distribuované genetické algoritmy, nebo také migrační model. U tohoto modelu existuje více populací, které se evolučně vyvíjí nezávisle na sobě. Občas dochází k migraci, kdy si tyto populace mezi sebou vymění jedince, čímž se podpoří variabilita genetického materiálu a sníží riziko předčasné konvergence k lokálnímu optimu. [46]

Reprezentace	Řetězec bitů
Výběr rodičů	Pravděpodobnostní úměrné fitness (ruleta)
Křížení	Jednobodové křížení
Mutace	Prohození bitu (z 1 na 0 a naopak)
Výběr přeživších	Generační obměna (přežívají jen potomci)

Tabulka 2.1: Nástin vlastností klasického GA (Zdroj: [21])

Algoritmus 3: Podoba klasického GA. (Zdroj: [13])

Urči jak bude zakódováno kandidátní řešení (fenotyp) v (genotypu) a definuj fitness funkci;
Inicializuj počáteční populaci n jedinců reprezentujících kandidátní řešení;
while *není splněno ukončující kritérium* **do**
    Dekóduj všechny jedince n v populaci na odpovídající kandidátní řešení a vypočítej jejich fitness;
    Vyber n jedinců z populace s pravděpodobností úměrnou jejich fitness a umísti je do společného prostoru rodičů;
    repeat
        Vyber náhodně 2 rodiče ze společného prostoru rodičů a s pravděpodobností p_{cross} proveď křížení těchto rodičů, čímž vzniknou 2 potomci;
        Pokud křížení neproběhlo, tak jsou potomci pouhé kopie jejich rodičů;
        S pravděpodobnostní p_{mut} zmutuj každý prvek (gen) potomků (genotypů);
    until *není vygenerováno n potomků*;
    Starou populaci nahraď potomky, čímž vzniká nová generace;
end

2.4.2 ES – Evoluční strategie

Evoluční strategie patří mezi významné evoluční algoritmy. Vznikl v 60. letech na Technické univerzitě v Berlíně za účelem optimalizace tvarů z hlediska jejich aerodynamických vlastností, kde jiné optimalizační metody selhaly. Autory byli tehdejší studenti Ingo Rechenberg, Hans-Paul, Schwefel a Peter Bienert. Od té doby je ES široce používán pro řešení různých optimalizačních problémů. Těžištěm je řešení spojitých optimalizací (kde se pracuje s reálnými parametry), ES se ale také uplatnily i pro kombinatorické a víceúčelové optimalizace nebo také optimalizace s omezujícími podmínkami. Tradiční ES používají reprezentaci v podobě řetězce reálných čísel. Pro selekci rodičů se uplatňuje náhodný výběr s uniformním rozložení pravděpodobnosti. Jako variační operátor byla původně použita pouze mutace (Gaussovská mutace), pozdější varianty zavedly i křížení, které je typicky aritmetické či uniformní a má spíše globálnější charakter (kdy se využívá více než dvou rodičů). U ES se poprvé objevil mechanismus sebe-adaptace, kdy se během optimalizace mění parametry ES podle zpětné vazby prohledávacího procesu nebo se evolučně vyvíjí spolu s genotypy. Typicky se mění směrodatná odchylka σ u Gaussovské mutace, což ovlivňuje stupeň mutace a tedy i stupeň generované diverzity. Způsob strategie náhrady populace je poté deterministický a je určen variantou ES, kdy můžeme rozlišit 2 základní varianty ES: [46][13][21][58][58][55]

- **Varianta** $(\mu + \lambda) - ES$ – U této varianty má populace μ jedinců a během reprodukce je vygenerováno λ potomků. Novou populaci tvoří nejlepší jedinci z množiny potomků a jedinců z původní populace. Selekční tlak je tak poměrně vysoký.
- **Varianta** $(\mu, \lambda) - ES$ – U této varianty má populace μ jedinců a během reprodukce je vygenerováno λ potomků. Novou populaci tvoří nejlepší jedinci z množiny potomků. Musí platit: $\lambda \geq \mu$, kdy potomků je vygenerováno mnohonásobně více (např. jich může být 7x více jak rodičů). Tento nepoměr mezi vygenerovanými a přeživšími potomky zajišťuje poměrně vysoký selekční tlak.

Shrnutí základních vlastností klasického ES přehledně reprezentuje tabulka 2.2.

Reprezentace	Vektor reálných čísel
Výběr rodičů	Náhodné s uniformním rozložením pravděpodobnosti
Křížení	Aritmetické či uniformní
Mutace	Gaussovská mutace
Výběr přeživších	$(\mu + \lambda) - ES$ nebo $(\mu, \lambda) - ES$

Tabulka 2.2: Nástin vlastností klasické ES (Zdroj:[21])

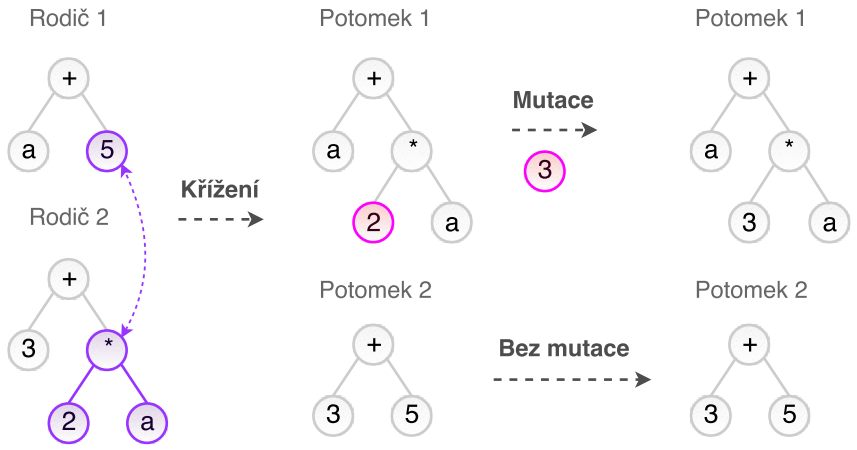
2.4.3 GP – Genetické programování

GP představil v 80. letech John Koza a jeho první aplikací byl návrh programů v jazyce LISP. GP vychází z klasického GA se kterým sdílí mnoho společných principů, ale na rozdíl od něj a jiných EA neřeší tradiční optimalizační úlohy, kde se snažíme najít parametry fitness funkce, které maximalizují zisk. U GP hledáme nejlepší program pro daný problém (např. symbolická regrese), což je možné uvažovat jako speciální případ optimalizace, kde jediným parametrem fitness funkce je právě samotný program. Mapování mezi prostorem reprezentací řešení a prostorem řešení je 1:1. Reprezentace obvykle nabývá podobu nelineárních stromových struktur, které reprezentují program a nemá fixní velikost. Evoluce řešení má tak v podstatě otevřený konec, kdy řešení může mít teoreticky neomezenou složitost. Nekontrolovatelný růst však často vede k tzv. *Bloat* problému, kdy se stromová struktura zvětšuje bez nějakého zlepšení fitness a typicky obsahuje mnoho redundantních bloků. Tento problém se řeší různými metodami (např. penalizací složitých struktur). Stromová struktura obsahuje typicky 2 typy uzlů: *funkce* (nelistové uzly, mají nenulovou aritu, např. matematické funkce, řídící struktury aj.) a *terminály* (listové uzly, mají nulovou aritu, např. proměnné nebo konstanty), které dohromady tvoří základní stavební bloky programu a je potřeba je předem definovat. Výběr rodičů je pravděpodobnostní. Reprodukce je sexuální, kdy se s určitou pravděpodobností aplikuje křížení (u rodičů se s určitou pravděpodobností prohodí náhodně vybrané podstromy) a následná mutace (podstrom se nahradí náhodně vygenerovaným podstromem). Novou populaci tvoří pouze potomci. Populace čítá oproti jiným variantám EA poměrně hodně jedinců (řádově tisíce). Shrnutí základních vlastností klasického GP přehledně reprezentuje tabulka 2.4. Ukázku sexuální reprodukce 2 rodičů pak ukazuje obrázek 2.4. [13][58][21][52][46][56]

Reprezentace	Stromové struktury
Výběr rodičů	Pravděpodobnostní úměrné fitness
Křížení	Prohození podstromů
Mutace	Náhodná změna v stromu
Výběr přeživších	Generační obměna (přežívají jen potomci)

Tabulka 2.3: Nástin vlastností klasické EP (Zdroj: [21])

Existuje také několik řekněme podtypů genetického programování jako kartézské genetické programování nebo gramatická evoluce, které se liší v různých ohledech, např. ve způsobu kódování jedince. [46]



Obrázek 2.4: Ukázka reprodukce u GP. Potomek 1 reprezentuje program, který vyčísluje výraz: $a + 3 * a$, kde a je nějaká proměnná programu.

2.4.4 DE – Difereciální evoluce

Diderenciální evoluci vytvořili Ken Price a Riner Storm v 90. letech a jedná se o jeden z nejúspěšnějších algoritmů v oblasti spojité optimalizace. Biologická inspirace je zde poněkud menší, a i principy reprodukce jsou mírně odlišné od již popsaných. Jedinci v populaci jsou reálné vektory a nad těmito se mj. provádí operace diference vektorů, od které je odvozen název algoritmu. Během jedné generace každý jedinec x_i produkuje potomka pomocí tří náhodně zvolených odlišných jedinců (pomocní rodiče). Tito 3 zvolení jedinci nejdřív vytvoří tzv. *variační vektor* tak, že se k prvnímu z nich (tzv. bázovému vektoru) přičte přeškálovaná diference zbylých dvou vektorů (tzv. *diferenční vektor*) – tento proces je označován jako *diferenciální mutace*. Tento variační vektor v_i se poté zkříží s jedincem x_i a vzniká potomek - tzv. *zkušební vektor* u_i, křížení je náhodné uniformní. Zkušební vektor poté svého rodiče x_i v následující generaci nahradí, pokud je lepší. K vytvoření potomka je tedy zapotřebí 4 rodičů. Výše uvedený popis odpovídá klasické verzi algoritmu $DE/rand/1/bin$. Pro značení konkrétního typu DE je zavedena notace: $DE/x/y/z$, kde x značí způsob výběru bázového vektoru (rand = náhodně), y značí kolik diferenčních vektorů je použito pro vytvoření variantního vektoru a z označuje schéma křížení (bin = binomické, resp. uniformní křížení). Princip činnosti klasického DE je zobrazen v algoritmu 4. Jedním z parametrů, které je potřeba nastavit je CR, který udává stupeň křížení, tedy jak se bude potomek rodiče x_i lišit od tohoto rodiče. Hodnoty parametru se volí z intervalu $\langle 0, 1\rangle$. Krajní hodnoty 0 resp. 1 není vhodné volit, protože poté se evoluce může zastavit (potomci budou téměř kopie rodičů x_i), resp. může mít podobu náhodného prohledávání (potomci budou produktem jen náhodně zvolených pomocných rodičů bez účasti jedince x_i, který by přitom měl být hlavním rodičem). Obecně by se hodnota parametru CR měla snižovat se zvyšující se separabilitou účelové funkce. Dalším z parametrů které je třeba nastavit je F který leží v intervalu $\langle 0, 2\rangle$. Jedná se o škálovací faktor, který ovlivňuje míru diferenciální mutace a na tento parametr je DE mnohem citlivější než na CR a optimálně by se měl snižovat s druhou odmocninou populace N. Proměnná $\Im_r$ pak v algoritmu zajišťuje, že potomek u_i není prostou kopií x_i. Shrnutí základních vlastností klasické DE přehledně reprezentuje tabulka 2.3. [13][58][21][46][56]

Algoritmus 4: Algoritmus *DE/rand/1/bin* pro minimalizaci n-dimenzionální funkce f(x). Zdroj: [56]

Inicializuj populaci jedinců reprezentujících kandidátních řešení $\{x_i | i \in [1, N]\}$;
while *není splněno ukončující kritérium* **do**
  foreach *jedinec* x_i, $i \in [1, N]$ **do** `// každý jedinec generuje potomka`
    $r_1 \leftarrow$ náhodný integer $\in [1, N] : r_1 \neq i$;
    $r_2 \leftarrow$ náhodný integer $\in [1, N] : r_2 \notin \{i, r_1\}$;
    $r_3 \leftarrow$ náhodný integer $\in [1, N] : r_3 \notin \{i, r_1, r_2\}$;
    $v_i \leftarrow x_{r1} + F(x_{r2} - x_{r3})$; `// diferenciální mutace`
    $\Im_r \leftarrow$ náhodný integer $\in [1, n]$;
    foreach *dimenze* $j \in [1, n]$ **do** `// uniformní křížení`
      $r_{CRj} \leftarrow$ náhodný integer $\in [0, 1]$;
      if $(r_{CRj} < CR)$ *or* $(j = \Im_r)$ **then** $u_{ij} \leftarrow v_{ij}$;
      else $u_{ij} \leftarrow x_{ij}$;
    end
    if $f(u_i) < f(x_i)$ **then** $x_i \leftarrow u_i$; `// nahrazení rodiče potomkem`
  end
end

Reprezentace	Vektory reálných čísel
Výběr rodičů	Každý jedinec x_i produkuje potomka s využitím jiných tří uniformně náhodných vybraných jedinců r_1, r_2, r_3
Křížení	Uniformní křížení variačního vektoru a jedince x_i
Mutace	Diferenciální mutace pomocí r_1, r_2, r_3, vzniká varianční vektor
Výběr přeživších	Deterministický, potomek nahradí rodiče pokud je kvalitnější

Tabulka 2.4: Nástin vlastností klasické DE (Zdroj: [21])

Kapitola 3

Umělé neuronové sítě

Umělé neuronové sítě, nebo jen neuronové sítě (dále označovány zkratkou NN – neural networks) je rodina paralelních výpočetních modelů, které jsou inspirovány způsobem fungování mozku a centrální nervové soustavy. Z obecného pohledu provádí nelineární mapování vstupu na výstup a mohou sloužit např. pro klasifikaci, predikci nebo shlukování. Skládají se ze sítě elementárních výpočetních jednotek, nazývaných neurony, které dohromady v rámci sítě vykazují komplexní emergentní chování. Tyto sítě v podstatě tvoří ohodnocené orientované grafy, kde se signál mezi neurony (které představují uzly sítě) šíří po směru hran (které představují spojení mezi uzly/neurony). Hranám jsou přiřazeny určité váhy. Neurony pracují různým způsobem podle typu neuronové sítě, nebo dle své role v rámci dané sítě. Nejdříve pomocí bázové funkce provedou lineární transformaci vstupních signálů/hodnot neuronu a jejich vah. Po lineární transformaci následuje nelineární transformace, kterou provádí aktivační funkce a její výstup tvoří výstup neuronu. Obecně je vstupem sítě vektor hodnot, a stejně tak i výstup. [39][55][13]

Umělé neurony a umělé neuronové sítě abstrahují funkci skutečných neuronů a neuronových sítí a neodpovídají zcela přesně jejich funkci. Např. u biologických neuronů je signál zakódován pomocí frekvence, zatímco u umělých neuronů je signál zakódován pomocí amplitudy, což vede na rychlejší přenos, ovšem tento způsob kódování, by byl u skutečných biologických neuronů obtížně dosažitelný. U skutečných neuronů, na rozdíl od většiny jejich umělých, dochází také během času ke změnám jejich spojení, vah, nebo způsobu zpracování informací. Neurony a neuronové sítě tak vykazují určitou plasticitu. Tyto změny jsou indukovány signály, které neurony zpracovávají, a dochází tak v podstatě k učení, díky čemuž např. si vytváříme vzpomínky, osvojujeme si určité činnosti apod. Aby síť umělých neuronů fungovala, jak potřebujeme, je potřeba ji také tzv. naučit. Toto učení se však soustřeďuje typicky jen na úpravu vah a probíhá jen jednorázově na začátku na nějakém trénovacím vzorku dat. Porovnání biologických neuronů a umělých neuronů ilustruje obrázek 3.2.[55]

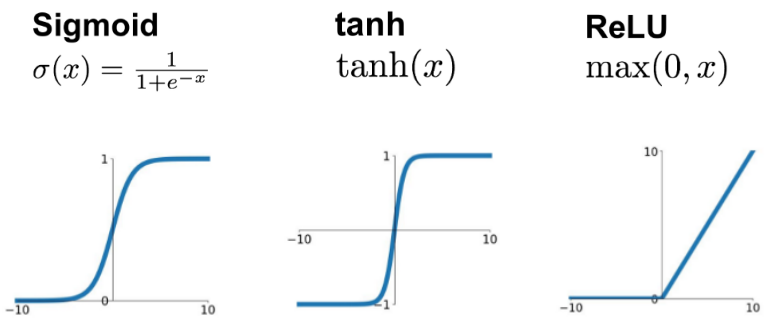
Neuronových sítí existuje velké množství, kde každá má různé zaměření a schopnosti. NN bychom mohli rozlišit podle čtyř jejich aspektů: [55][13]

- **Topologie** – Určuje, jak jsou neurony propojeny mezi sebou. Mezi základní přístupy patří např. *dopředné sítě* nebo *rekurentní sítě*. Dopředné sítě se skládají ze několika vrstev neuronů (obvykle se rozlišují vstupní vrstvy, skryté vrstvy a výstupní vrstvy) které jsou seřazeny za sebou. Výstupy neuronů jedné vrstvy mohou být připojeny pouze na vstupy neuronů následující vrstvy. Výpočet odezvy pro vstup pak probíhá postupně po vrstvách. Mezi tyto sítě patří např. MLP, konvoluční neuronové sítě nebo SOM. Obecnou podobu těchto sítí zachycuje obrázek (D) v obrázku 3.2. Rekurentní

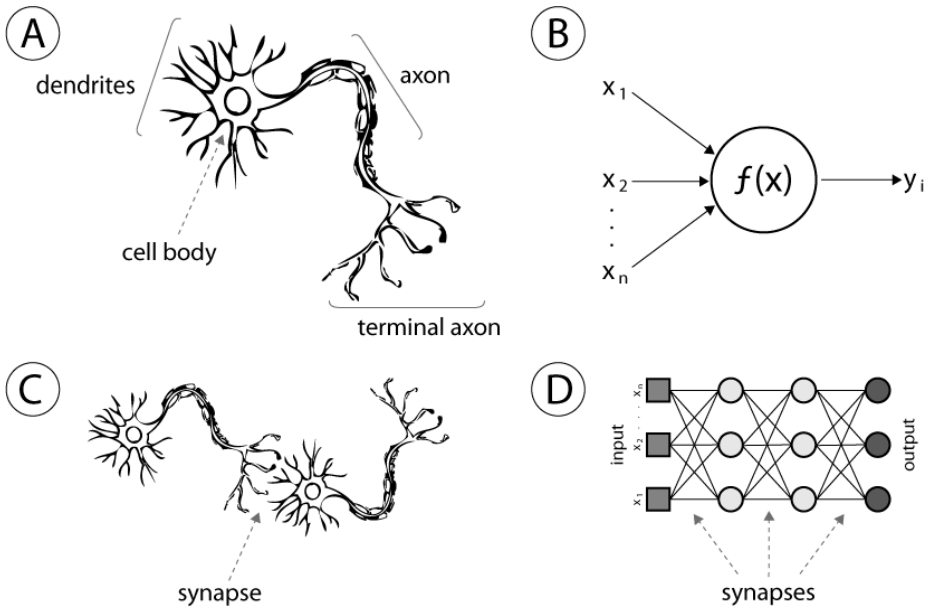
sítě mohou nabývat různých podob, ale u všech se mohou vyskytovat smyčky, kdy je neuron v této smyčce ovlivněn více či méně svým některým předchozím výstupem. Příkladem rekurentních sítí jsou např. Hopfieldovy sítě.

- **Funkce neuronu** – Určuje, jaké bázové a aktivační funkce jsou u neuronů použity. Příkladem bázových funkcí může být suma váhovaných vstupů, nebo vzdálenost mezi vstupem a váhami. Aktivační funkce pak mohou být obecně skokové (např. signum) nebo spojité (např. hyperbolický tangens, nebo sigmoid). Příklady aktivačních funkcí ilustruje obrázek 3.1.

- **Způsob trénování/učení** – Určuje, jakým způsobem se síť trénuje/učí (tedy, jak se provádí proces nastavování vah neuronů, případně se mění topologie sítě). Tento způsob se odvíjí od úlohy, kterou má síť řešit. Základními přístupy jsou *učení s učitelem (supervised learning)* a *učení bez učitele (unsupervised learning)*. Při *učení s učitelem* máme k dispozici kromě vstupních dat i očekávané (správné) odezvy sítě pro tyto data. Skutečné odezvy sítě pak porovnáváme s očekávanými a na základě chyby upravujeme váhy, tak abychom chybu minimalizovali. Tímto naučíme neuronovou síť pro dané trénovací vstupy predikovat odpovídající výstupy, resp. nalezeme model, který postihuje závislost mezi vstupem a výstupem. Snahou je, aby pro nové vstupní hodnoty, na kterých síť nebyla trénována poskytovala správné výstupy. Učení s učitelem použijeme např. pro účely klasifikace nebo regrese. Příkladem známého trénovacího algoritmu je backpropagation, který se používá pro trénování dopředných sítí (i když existuje verze i pro rekurentní sítě). Tento algoritmus se snaží upravovat váhy neuronů ve směru záporného gradiendu chyby (čímž chybu minimalizuje). Toto provádí postupně po vrstvách od výstupních vrstev ke vstupním. Při *učení bez učitele* nemáme ke vstupním datům k dispozici informace o očekávaných výstupem, naopak očekáváme od neuronové sítě, že v datech odhalí skryté vzory. Typicky se tento přístup používá pro úlohy shlukování, kdy chceme neznámá data zařadit podle jejich podobnosti do shluků. Příkladem sítí využívající učení bez učitele jsou sítě SOM a ART.



Obrázek 3.1: Příklady typických aktivačních funkcí. (Vytvořeno na základě zdroje: [30])



Obrázek 3.2: **(A)** Biologický neuron, kde dendrity přijímají vstupní chemické signály od jiných neuronů ve formě neurotransmitterů, což jsou chemické látky přenášející nervové vzruchy mezi dvěma neurony. Počty receptorů neurotransmitterů na dendritech v podstatě určují váhu dané synapse. Chemické signály jsou poté transformovány na elektrické impulzy, které putují až k výběžkům axonu, kde jsou převedeny na chemické signály (neurotransmittery) a přenáší se do dalších neuronů skrze jejich dendrity. Spoj mezi neurony, kde dochází k chemickému přenosu signálů je označován jako synapse. Neurony vykazují plasticitu, kdy se mohou jejich spoje, váhy spojů nebo způsoby zpracování signálů časem měnit na základě zpracovávaných signálů, čímž dochází k učení. **(B)** Schéma umělého neuronu, který má více vstupů (a k nim přiřazené váhy), které nelineárně transformuje na jeden výstup (a ten může být vstupem pro více dalších neuronů). **(C)** Ukázka synapse u biologických neuronů. **(D)** Schéma synapsí mezi umělými neurony, které mají podobu váhovaných hran mezi uzly představující neurony. Dané synapse vytvářejí síť neuronů, zde se konkrétně jedná o typ dopředné neuronové sítě. [55][13] (Zdroj obrázku: [39])

3.1 Konvoluční neuronové sítě

Konvoluční neuronové sítě (CNN) jsou speciálním typem dopředných sítí, které se vyznačují různými typy jednotlivých vrstev. Tyto sítě jsou velmi efektivní zejména v oblasti klasifikace obrazu, kde dosahují ve srovnání s jinými metodami nejlepších výsledků. Na rozdíl od klasických NN dokáží ze vstupu získat potřebné rysy, tedy jsou schopné provést jakési automatické předzpracování. Navíc se v této extrakci rysů zdokonalují učením. Tímto odpadá potřeba vytvářet pro konkrétní vstupní data extraktor příznaků. CNN se při předzpracováním vstupu (což jsou typicky obrazová data) inspirují způsobem, jakým je v mozku zpracovávána obrazová informace buňkami ve vizuálním kortexu. O extrakci příznaků se starají konvoluční a další speciální vrstvy. Tyto příznaky jsou poté vstupem pro dopřednou plně propojenou síť, která může provést např. klasifikaci. Dále v této kapitole uvažujeme CNN jako nástroj pro klasifikaci obrazových dat. [35][27][48]

CNN má několik typů vrstev, které se řadí za sebou a mohou tak vytvářet komplexní architekturu: [27][32]

- **Konvoluční vrstvy** – Primárním účelem konvolučních vrstev je extrakce rysů ze vstupního obrazu pomocí operace konvoluce, která zachycuje lokální závislosti v obraze a je relativně invariantní vůči posunům, šumu v obraze apod. Konvoluční vrstvy směrem od první k poslední postupně detekují rysy od těch vysokoúrovňových (obecnějších) k těm nízkoúrovňovým (specifičtějším). Konvoluční vrstva typicky obsahuje řadu konvolučních filtrů (jinak nazývány také konvolučních jádra nebo detektory rysů), což jsou matice, které obsahují číselné hodnoty (které můžeme považovat za jakési váhy). A pro každý filtr a vstup se provede konvoluce, která je ilustrována pro 2D filtr a 2D vstup na obrázku 3.3. Prvky jádra a odpovídajícího výseku (daného rozměry jádra) vstupu se vynásobí a sečtou, čímž dostaneme odpovídající výstupní hodnotu. Jádro se takto posunuje postupně z levého horního rohu do pravého spodního. Krok posunu jádra je možné nastavit, stejně tak možnou výplň, kdy se původní vstup rozšíří o nulové hodnoty na okraji, aby při konvoluci nedošlo k redukci rozměrů výstupu. Jelikož je v jedné konvoluční vrstvě pomocí více obecně různých filtrů produkováno pro daný vstup více výstupních matic, můžeme je vrstvit za sebe a v podstatě tak vytvořit jakousi trojrozměrnou matici obecně nazývanou jako mapa rysů (feature map), která má kromě rozměrů jako šířka a výška také hloubku danou počtem filtrů. V případě, že tato mapa rysů bude vstupem další konvoluční vrstvy, bude se provádět konvoluce už nikoliv nad 2D vstupem a 2D jádrem, ale nad 3D vstupem a 3D jádrem, toto je ilustrováno na obrázku 3.4. Princip je v podstatě stejný jako u 2D konvoluce, jen zde budeme při výpočtu uvažovat i třetí rozměr.

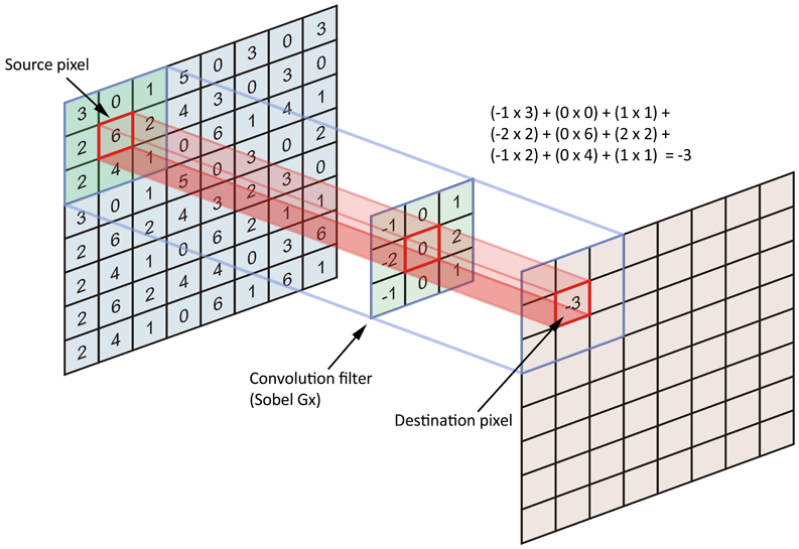
- **Nelineární/Aktivační vrstvy** – Výstupem konvoluční vrstvy je mapa rysů, což je produkt lineární transformace vstupu. Pro rozumnou funkci neuronových sítí a konvoluční sítě nevyjímaje je potřeba zavést nelinearitu. Za každou konvoluční vrstvou tak následuje nelineární vrstva, která aplikací aktivační funkce na každou hodnotu v mapě rysů, provede nelineární transformaci výstupu konvoluční vrstvy. Typicky používanou aktivační funkcí u CNN je $ReLU$, která záporné hodnoty v mapě rysů nastaví na nulu. Tato aktivační funkce je upřednostňována před jinými (jako např. hyperbolický tangens) kvůli tomu, že díky ní probíhá trénování CNN mnohem rychleji.

- **Polling vrstvy** – Polling vrstvy redukují rozlišení mapy rysů s tím, že zachovávají podstatné informace. V podstatě realizují podvzorkování mapy rysů. Zmenšení rozlišení má několik výhod. Redukuje vliv šumu ve vstupním obraze, počet parametrů a

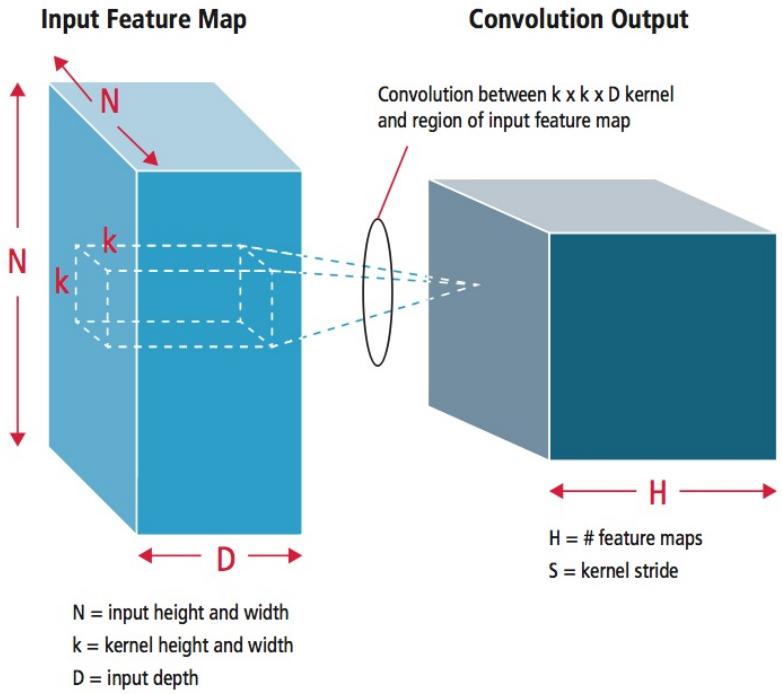
výpočtů a tím zabraňuje přetrénování. Dále činí síť invariantní vůči menším distorzím, transformacím a translacím vstupního obrazu. A prakticky umožňuje detekovat objekty v obraze bez ohledu na to, kde jsou umístěny. Polling vrstva může a nemusí následovat po nelineární vrstvě. Podvzorkování se provádí tak, že se v mapě rysů po vrstvách (po 2D maticích) vždy vezme daná 2D matice a rozdělí se na menší nepřekrývající se části (typicky s rozměry 2x2). Pro danou část se pak vypočítá jedna hodnota která ji bude reprezentovat, tímto dojde k redukci rozměrů mapy rysů na polovinu (hloubka však zůstává pochopitelně stejná, dané vrstvy mapy rysů nezmizí, pouze jejich rozměry se zmenší). Podle toho jakým způsobem se provede výpočet reprezentující hodnoty rozlišujeme *max-polling* a *avg-polling*. U max-polingu se vybere maximální hodnota, zatímco u avg-pollingu se spočítá průměrná hodnota. Toto je lépe ilustrováno na obrázku 3.5.

- **Plně propojené vrstvy** – Kaskáda konvolučních, nelineárních a polling vrstev nám umožní získat potřebné rysy ze vstupního obrazu. Tato 3D mapa rysů je poté zploštěná (v tzv. flatten vrstvě) do 1D vektoru rysů. A dle tohoto vektoru rysů probíhá v plně propojených dopředných vrstvách finální klasifikace daného obrazu. Neurony v těchto vrstvách mají jako bázovou funkci sumu váhovaných vstupů. Aktivační funkce mohou být různé s tím, že aktivační funkce neuronů finální vrstvy v případě klasifikace obrazu je obvykle tzv. softmax funkce, která zajistí, že výstupy neuronů budou reprezentovat pravděpodobnosti daných tříd (které neurony reprezentují) pro daný vstupní vzorek.

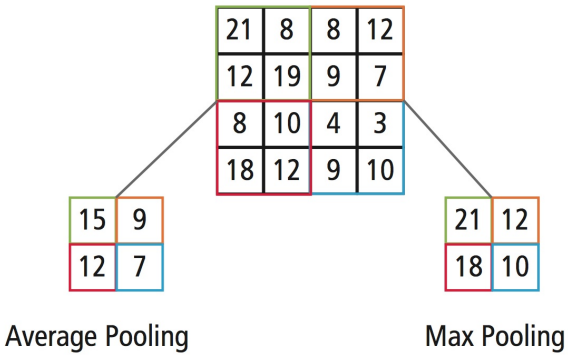
Celkovou finální podobu CNN ilustruje obrázek 3.6. Základní pohled na to, jak funguje biologická předloha, kterou se CNN inspiruje nastiňuje obrázek 3.7. Výkon CNN pak závisí do velké míry na její architektuře, čímž se budeme zabývat později v této práci.



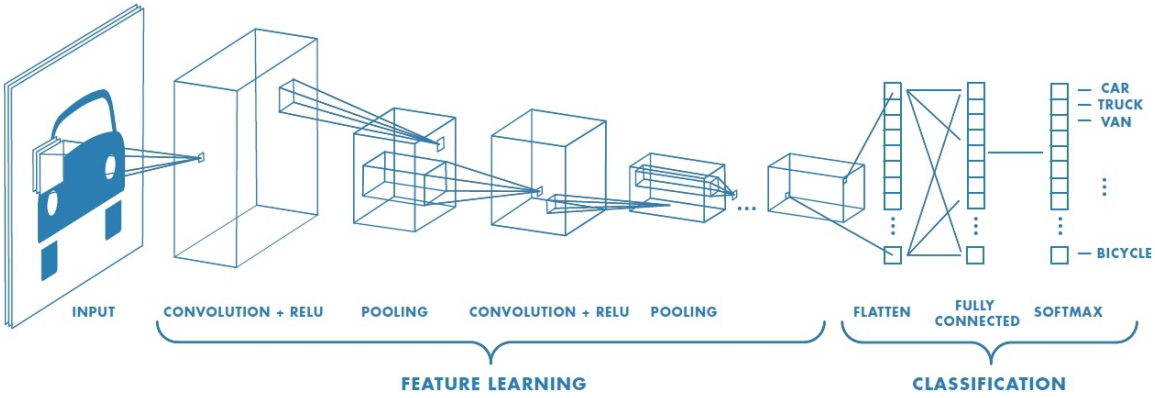
Obrázek 3.3: Ilustrace principu konvoluce 2D jádra a 2D vstupu. (Zdroj: [25])



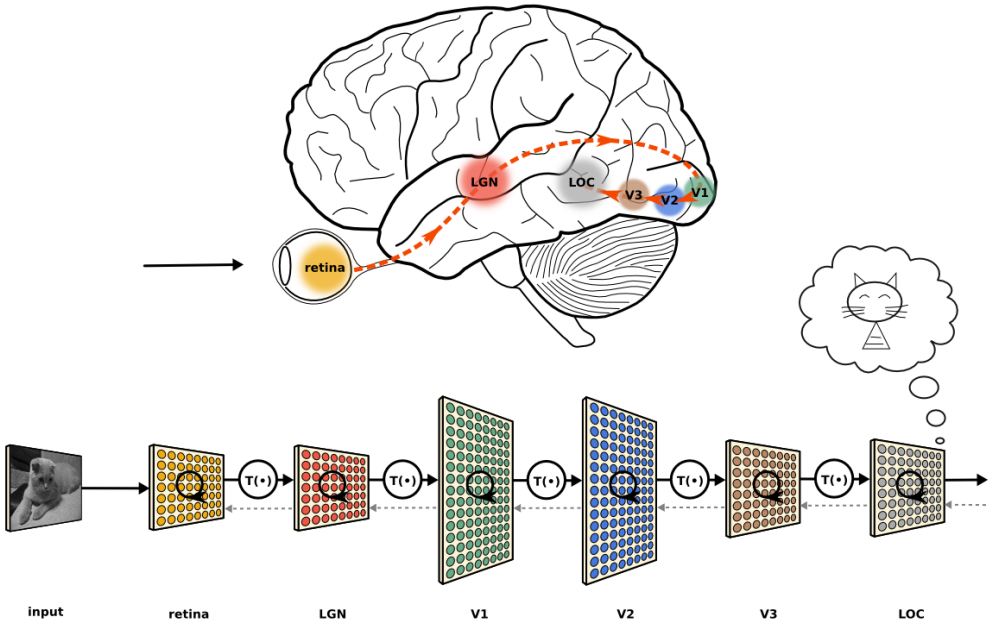
Obrázek 3.4: Ilustrace principu konvoluce 3D jádra a 3D vstupu. (Zdroj: [27])



Obrázek 3.5: Ilustrace principu podvzorkování (zmenšení rozlišení) pomocí metod max-polling a avg-polling. Toto se provede pro každou vrstvu mapy rysů. (Zdroj: [27])



Obrázek 3.6: Konvoluční neuronová síť provádí extrakci rysů pomocí kaskády konvolučních, aktivačních, polling a jiných vrstev. V rámci konvolučních vrstev se počet filtrů a tedy hloubka výstupní mapy rysů liší. Finální mapa rysů je zploštěna na vektor příznaků. Tento vektor je pak vstupem pro klasickou dopřednou plně propojenou síť, která v případě úlohy klasifikace provádí samotnou klasifikaci obrazu a výstupní neurony mají softmax aktivační funkci, která přiřazuje jednotlivým třídám míru pravděpodobnosti. (Zdroj: [3])



Obrázek 3.7: Základní pohled na princip činnosti biologické předlohy CNN, který ukazuje jakým způsobem mozek zpracovává a vyhodnocuje vizuální informaci. Zpracování obrazu a získání rysů obrazu podobné konvoluci, se odehrává v oblastech vizuálního kortexu $v1$, $v2$ a $v3$. Klasifikace, resp. rozpoznání objektu se poté odehrává v rozhodovací oblasti LOC. (Zdroj: [16])

Kapitola 4

Neuroevoluce

Neuroevoluce spočívá v aplikaci evolučních algoritmů při hledání kvalitních neuronových sítí pro řešení daného problému. Zde se zaměříme spíše na principy evoluce dopředných sítí, které jsou ale přenositelné i na jiné typy sítí. Zde je přehled toho, co mohou EA na neuronových sítích vyvíjet (a to najednou, nebo jednotlivě): [55][13]

- **Návrh přenosové funkce neuronů** – Existují různé typy aktivačních funkcí, které mohou dávat lepší či horší výsledky a to třeba i v závislosti na jejich kombinaci.
- **Návrh topologie** – Návrh propojení mezi neurony v rámci fixního počtu neuronů ve fixním počtu vrstev, nebo návrh počtu neuronů a počtu vrstev, nebo obojího.
- **Váhy pro pevnou strukturu sítě** – Dochází tak k optimalizaci vah a nahrazení standardního učicího algoritmu, což je problematické pro větší množství vah.
- **Výběr nebo parametry učícího algoritmu** – Pro optimální naučení sítě mohou hrát roli i parametry učícího algoritmu, které je možné nalézt pomocí EA. Případně pokud je možné použít více učicích algoritmů, je možné nalézt nejlepší z nich (případně i s nejlepšími parametry).

Existují přístupy TWEANN, které umožňují vyvíjet topologií neuronových sítí a zároveň i jejich váhy, kdy jsou inkrementálně vyvíjeny jednoduché modely do komplexnější podoby. [55]

Co se týče reprezentace problému, u neuroevoluce se používá přímé i nepřímé kódování. U přímého kódování jsme schopni získat fenotyp (strukturu sítě, váhy, apod.) přímo z genotypu. U nepřímého kódování máme sadu pravidel, jejichž iterativní aplikací je možné zkonstruovat fenotyp. [13]

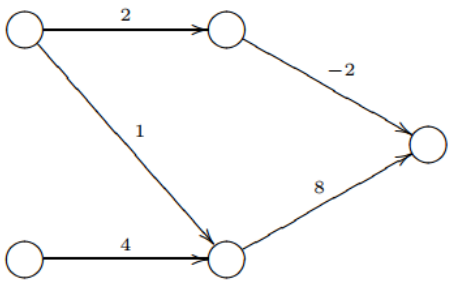
4.1 Přímé kódování

V této kapitole se budeme zabývat způsoby přímého kódování vybraných aspektů neuronových sítí, které chceme navrhnout pomocí EA (např. jejich strukturu, váhy, apod.). Tento způsob návrhu reprezentace je základní krok, dalším krokem je návrh fitness funkce a volba EA, což bude také diskutováno. [13]

V případě, že chceme pomocí EA navrhnout váhy spojů mezi neurony pro fixní topologii, nám v podstatě stačí tyto váhy zakódovat jako vektor čísel. Tyto čísla mohou mít binární podobu, což demonstruje obrázek 4.1 nebo mohou být z reálné domény. Fitness funkci je

poté možné navrhnout tak, že se pro vstupní data nechá zjistit odezva sítě, tato se porovná s očekávanou odezvou a nižší chyba bude značit vyšší fitness. Dle zvolené reprezentace poté může být použit lib. vhodný EA. Například je možné použít GA, a v případě reálné reprezentace může být použit např. také PSO. Návrh vah pomocí evolučních algoritmů je však výhodnější než specializované algoritmy jako backpropagation jen v některých případech, kdy nelze tyto algoritmy použít nebo je jejich použití nevhodné – např. pokud je chybová funkce nediferencovatelná nebo je vysoce multimodální. [13]

Encoding	*Weight*
10100	+2
11000	+1
00000	no connection
10010	+4
00100	−2
10001	+2



Obrázek 4.1: Kódování vah v dopředné síti pomocí vektoru binárních čísel. První bit udává, jestli je je váha záporná (0) nebo kladná (1). Zbývající 4 bity čtené zleva doprava představují absolutní hodnotu váhy. (Zdroj: [13])

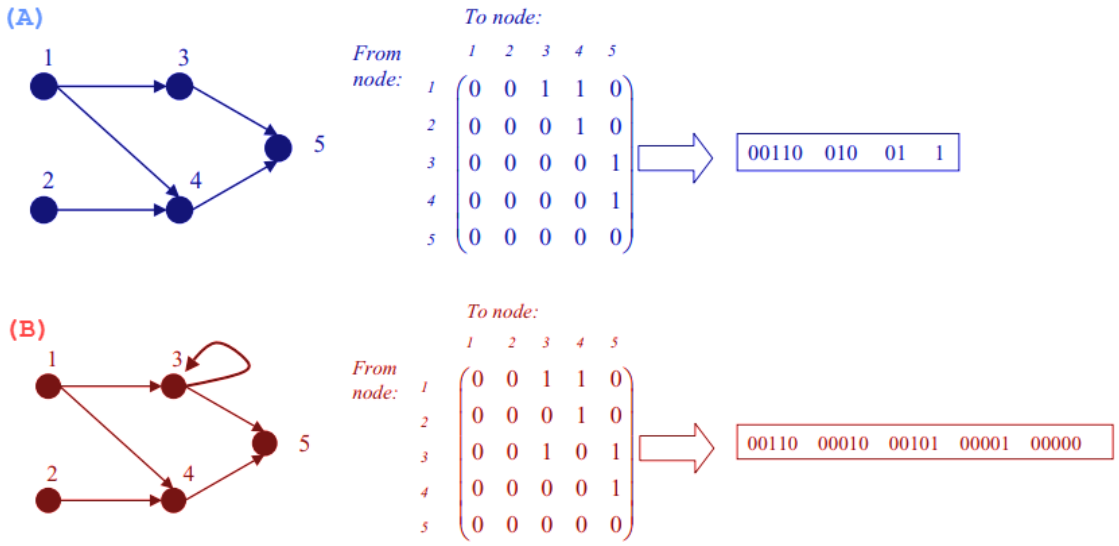
Pokud chceme pomocí EA navrhnout způsob propojení neuronů, můžeme to udělat pomocí matice spojení o velikost $N \times N$, kde N je maximální počet neuronů. Jednička v matici pak značí, že dané spojení dvou neuronů je aktivní, nula naopak značí, že mezi danými neurony spojení není. Obrázek 4.2 demonstruje matice a jim odpovídající sítě podle toho, jestli je naším požadavkem dopředná nebo rekurentní síť. V dané ilustraci vidíme, že z matice můžeme snadno vytvořit binární chromozom. Fitness vyhodnocujeme tak, že z daného chromozomu vytvoříme neuronovou síť s (náhodně) inicializovanými vahami a tuto síť natrénujeme pomocí standardního algoritmu jako je např. backpropagation. U takto natrénované sítě se pro testovací vstupy zjistí jejich odezva, ta se porovná s očekávanou odezvou a nižší chyba bude značit vyšší fitness. Jako evoluční algoritmus můžeme zvolit takový který umí pracovat s binárními řetězci, tedy např. GA. Takovýto EA, který v kontextu neuronových sítí navrhuje způsob propojení mezi neurony a pro nastavení vah používá klasický trénovací algoritmus (např. backpropagatiion) označujeme jako hybridní. [13]

Přímé kódování použité pro návrh propojení neuronů je sice intuitivní, ale není ideální. A je sním spojena řada problémů, které snižují efektivitu hybridního algoritmu, který nad tímto kódováním pracuje: [13]

- **Zašuměná fitness** – Hybridní EA navrhuje strukturu sítě, ovšem kvalita sítě není určena jen její strukturou (propojením neuronů), ale i vahami spojů. Tyto váhy jsou před hodnocením kvality sítě nastaveny náhodně a pomocí trénování (např. algoritmem backpropagation) jsou optimalizovány. Zde ale může stát, že u sítě s dobrou architekturou jsou váhy před trénováním inicializovány nešikovně a nakonec jí není přiřazena velká kvalita. Toto lze redukovat tím, že se pro danou architekturu sítě provede několik ověření fitness (které zahrnuje také několik inicializování vah a natrénování sítě) a výsledná fitness se zprůměruje. Ovšem toto zvyšuje výpočetní náročnost.
- **Redundantní reprezentace a mechanismus křížení** – U přímého kódování struktury NN nastává problém, kdy pro nějaký fenotyp existuje více možných genotypů, re-

spektive že více různých genotypů může reprezentovat funkčně ekvivalentní sítě. Toto samozřejmě snižuje efektivitu prohledávacího procesu a je potřeba tomuto správně uzpůsobit operátor křížení.

- **Přetrénování** – Pokud je N vysoká hodnota, vytvořené sítě mohou být komplexnější než by bylo potřeba a síť se poté přetrénuje a špatně generalizuje. Řešením by mohlo být zavedení penalizaci fitness komplexních sítí.



Obrázek 4.2: **(A)** V případě dopředné sítě pracuje EA jen s horním pravým trojúhelníkem matice spojení, jehož prvky můžeme po řádcích zřetězit do binárního řetězce. **(B)** V případě rekurentní sítě, která může obsahovat smyčky, pracujeme s celou maticí, kterou můžeme celou po řádcích zřetězit do binárního řetězce. (Obrázek vytvořen na základě zdroje: [13])

4.2 NEAT

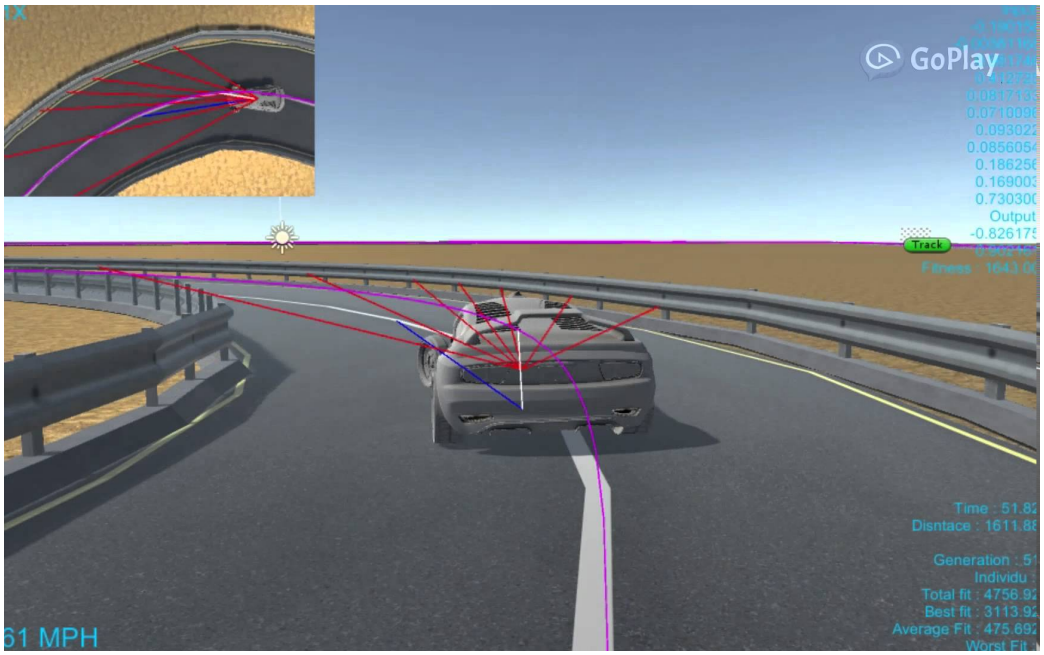
NEAT (neuroevolution of augmenting topologies) je evoluční algoritmus vytvořený v roce 2002 speciálně pro návrh neuronových sítí a s cílem překonat předchozí nevýhody přímého kódování. Vyvíjí zároveň strukturu neuronové sítě i její váhy a to inkrementálním způsobem od jednodušších struktur ke komplexnějším. Používá speciální reprezentaci a operátor křížení. [13]

4.3 Aplikační oblasti

Aplikační oblasti neuroevoluce jsou v podstatě shodné s aplikačními oblastmi neuronových sítí, neboť neuroevoluce je limitována jen tím, kde lze použít neuronové sítě. Obecnými aplikacemi tak mohou být např. klasifikace, predikce nebo shlukování. Konkrétnější aplikace neuroevoluce mohou být např. tyto: [55]

- **Robotika** – Návrh kontrolérů robotů v podobě NN. Příkladem může být navržení kontroléru pro řízení auta v simulátoru závodní hry pomocí neuroevoluce 4.3, toto se překrývá s aplikační oblastí v počítačových hrách.

- **Finanční trhy** – Návrh inteligentního forexového obchodníka založeného na NN.
- **Umělý život** – Studium interakcí umělých organismů s vyvíjející se inteligencí založenou na NN.
- **Počítačové vidění** – Návrh NN pro detekci rysů v obraze.
- **Datová komprese** – Návrh NN pro datovou kompresi.
- **Počítačové hry** – Vyvíjející se inteligence (reprezentována NN) NPC postav ve hrách.
- **Návrh a optimalizace obvodů** – NN může pracovat jako digitální obvod a tento je možné pomocí EA navrhnout nebo optimalizovatt
- **A mnoho dalších**



Obrázek 4.3: Návrh kontroleru pro řízení auta v závodním simulátoru pomocí neuroevoluce. Vzdálenosti od okolních objektů měřené paprsky jsou vstupem neuronové sítě, výstupem jsou akční pohyby vozidla. Po 512 generacích evoluce už kontrolér dokáže poměrně obstojně ovládat vozidlo na dané dráze. (Zdroj: [57])

Kapitola 5

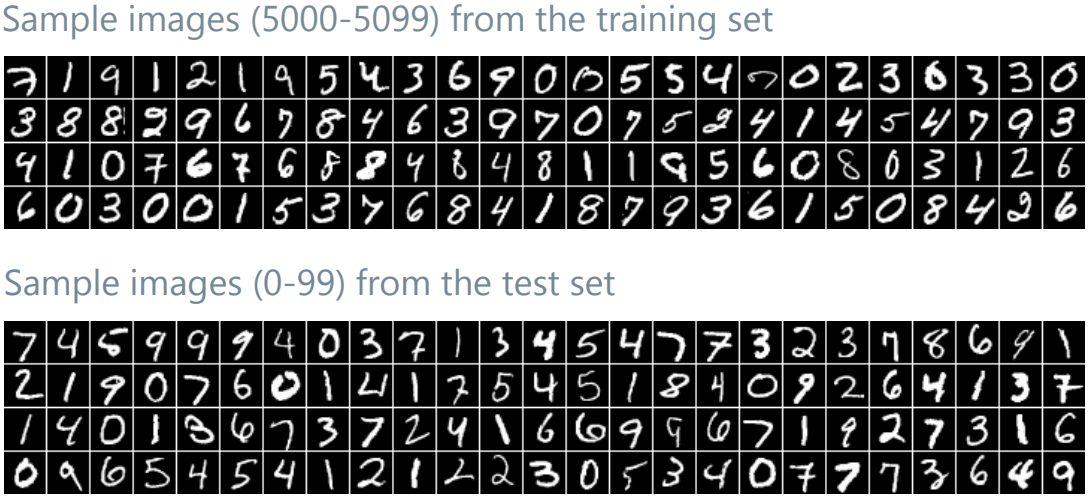
Evoluční návrh architektury CNN pro klasifikaci ručně psaných číslic

Výkon konvolučních neuronových sítí závisí do značné míry na jejich architektuře, tedy jak jsou jednotlivé vrstvy poskládány za sebou, jaké mají parametry apod. Toto se ukazuje především na soutěži v počítačovém vidění *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, kde různé týmy světa soutěží v tom, kdo má nejlepší model schopný klasifikace, detekce apod. Jedná se o velmi prestižní soutěž, která je v daném oboru počítačového vidění přirovnána k Olympijským hrám. V podstatě od roku 2012 tuto soutěž ovládly CNN, kdy v tomto roce poprvé vyhrála CNN AlexNet. A od toho okamžiku, kdy se zjistilo, že jsou CNN velmi slibné, došlo k jejich většímu rozšíření. AlexNet měla oproti dnešním sítím poměrně jednoduchou architekturu s pěti konvolučními vrstvami, třemi max-polling vrstvami, a třemi plně propojenými (dense) vrstvami, přičemž klasifikovala obrázky do 1000 kategorií. Používala i tzv. dropout vrstvy, které se používají jen u trénování, kdy se snaží zabránit přetrénování. Od tohoto roku v dané soutěži vyhrávaly CNN s čím dál komplexnějšími architekturami. [32][19]

V rámci praktické části diplomové práce je snahou nahradit ruční konstrukci a ověřování různých architektur CNN pomocí neuroevolučního algoritmu, který by měl pro zvolený klasifikační problém nalézt automaticky použitelnou architekturu CNN. Tato architektura by ale neměla být složitá a měla by dosahovat podobných výsledků jako state-of-the-art modely. Jeden z tradičních benchmarkových klasifikačních problémů na kterém se ověřují (nejen) CNN je dataset MNIST s ručně psanými číslicemi [36]. Tento tak bude představovat zvolený klasifikační problém.

5.1 Zvolený klasifikační problém – dataset MNIST

Dataset MNIST obsahuje 60 000 trénovacích a 10 000 testovacích vzorků, což jsou černobílé obrázky ručně psaných číslic o rozměrech 28×28 [36]. Na trénovacích vzorcích se model (v našem případě CNN) naučí správně klasifikovat vstupní obrázky do tříd (tedy detekovat jaká číslice je na obrázku). Na testovacích vzorcích se poté vyhodnocuje přesnost klasifikace daného modelu – tedy kolik procent vstupních vzorků bylo klasifikováno správně. Architektura se totiž bude v rámci vyhodnocování její fitness trénovat a následně testovat (a výsledky testování rozhodnou o fitness hodnotě). Způsobu vyhodnocování fitness ale bude probrán do detailu později. Ukázku vzorků z tohoto datasetu představuje obrázek 5.1.



Obrázek 5.1: Ukázka 100 vzorků z trénovací a testovací množiny datasetu MNIST. Vzorky s indexy 5000 až 5099 z trénovací množiny patří do množiny 5000 vzorků s indexy v rozsahu 5000 až 9999, na kterých se bude CNN trénovat v rámci vyhodnocení fitness jedince.

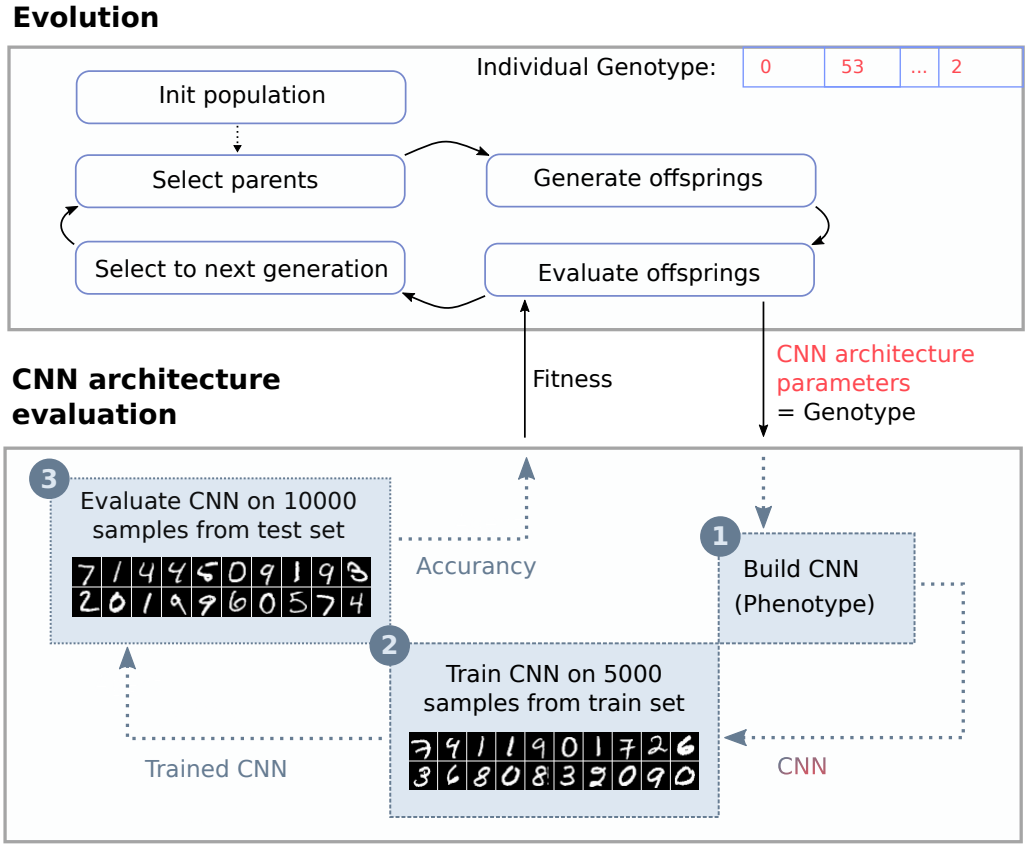
Tento problém tedy představuje zvolený klasifikační problém, pro který budeme hledat optimální architekturu CNN pomocí evolučních algoritmů. Hlavní motivací pro výběr tohoto problému je ten, že se jedná z hlediska výpočetní náročnosti o jeden z nejmenších možných problémů. Dalším důvodem je to, že je to známý benchmarkový problém a dosažené výsledky lze srovnat se state-of-art výsledky a vyhodnotit použitelnost neuroevoluce. Výpočetní náročnost související s řešením tohoto problému je tedy relativně malá v porovnání s jinými problémy, které mají vyšší dimenzionalitu. Protože pro vícerozměrné obrázky je potřeba z hlediska dosažení rozumné přesnosti klasifikace složitější síť a se složitější sítí roste zejména časová náročnost trénování takové sítě.

Úloha návrhu architektury CNN pro tento problém pomocí neuroevoluce však bude z hlediska výpočetního času velmi náročná. Navržený neuroevoluční algoritmus totiž pracuje jako hybridní algoritmus, kdy hledá architekturu pomocí evolučního algoritmu, ale při vyhodnocení kvality dané architektury tuto natrénuje pomocí backpropagation, což je časově náročná operace. Řádné trénování (mnohokrát v cyklu na celé trénovací množině) jedné architektury může zabrat pro netriviální architektury i několik dní. Když by se takto měla vyhodnotit každá architektura (jedinec) v populaci, tak ohodnocení všech jedinců v populaci nám může zabrat až týdny. Proto je tato časová náročnost snížena tím, že trénování architektury bude jen přibližné a proběhne v jednom cyklu na pouhých 5000 vzorcích z celkových 60 000 trénovacích vzorků. Tímto se pro středně složité architektury (jak si ukážeme později v experimentu) dostáváme do řádu minut až desítek minut. Toto zjednodušení trénování sice sníží do určité míry výpočetní náročnost, ale posílí se tím mnohem více role počátečního nastavení vah. Navíc výsledky (tedy natrénované architektury) které získáme, se v přesnosti nebudou moci vyrovnat nejlepším známým výsledkům (které byly získány řádným natrénováním). Nicméně takto nejlepší nalezené architektury v rámci evoluce můžeme vzít a natrénovat řádně (už mimo jakoukoliv evoluci). A na základě řádného natrénování a získaných výsledků, je možné rozhodnout, jestli je nalezená architektura kvalitní pro řešení problému klasifikace psaných číslic. Jak se později ukáže, tak i 5000 trénovací vzorků stačí na nalezení architektury, kterou když řádně natrénujeme dostaneme model, který se v přesnosti klasifikace velmi blíží state-of-the-art modelům.

Dosud vyjmenovaná opatření pro redukci výpočetní náročnosti, jako je volba co nejjednoduššího klasifikačního problému a zjednodušení trénování, však sama o sobě pro provedení experimentu nestačí. Problém je totiž stále velmi časově náročný a proto se fitness všech jedinců v populaci bude počítat v rámci experimentů paralelně. Potom ohodnocení všech jedinců v populaci bude trvat nejvýše tolik času, kolik nejdéle trvajícímu jedinci. Díky tomu už bude možné provést experiment v rozumné délce trvání (1̃0h).

5.2 Koncept neuroevoluce klasifikátoru

V tuto chvíli již máme představu o problému který řešíme a taky částečně o způsobu jak jej řešíme. V této kapitole si popíšeme další principy, s jakými budeme aplikovat neuroevoluci na řešený problém. Zejména se seznámíme s použitými EA, strukturou chromozomu se kterým EA pracují. Základní schéma principu činnosti neuroevoluce na úloze návrhu architektury CNN znázorňuje schéma 5.2. Toto schéma si postupně v rámci této podkapitoly vysvětlíme.



Obrázek 5.2: Schéma aplikace neuroevoluce při evolučním návrhu architektury CNN pro klasifikaci ručně psaných číslic.

Chromozom (Genotyp) se kterým pracují evoluční algoritmy a který kóduje odpovídající architekturu CNN, má obecně podobu, která je ilustrována na obrázku 5.3. V podstatě se jedná o vektor celých čísel s fixní délkou. Nicméně pro konkrétní experiment je možné délku zvolit tak, že se stanoví počet bloků *(C)* a počet bloků *(D)*. Dané bloky pak mohou být podle nastaveného příznaku aktivní nebo neaktivní a tím si evoluce sama určí počet vrstev,

a tedy složitost architektury. Blok *(C)* v sobě skrývá konvoluční vrstvu, kterou může (podle nastaveného příznaku) následovat polling vrstva. Přičemž pro konvoluční vrstvu je možné určit počet filtrů a jejich velikost. Nelineární (aktivační) vrstvy jsou pro aktivní konvoluční vrstvu implicitně ReLU. Pro polling vrstvu je možné určit pomocí celočíselného indexu typ pollingu . Blok *(D)* pak představuje plně propojenou vrstvu pro kterou je možné určit počet neuronů a typ aktivační funkce (která se specifikuje pomocí celočíselného indexu).

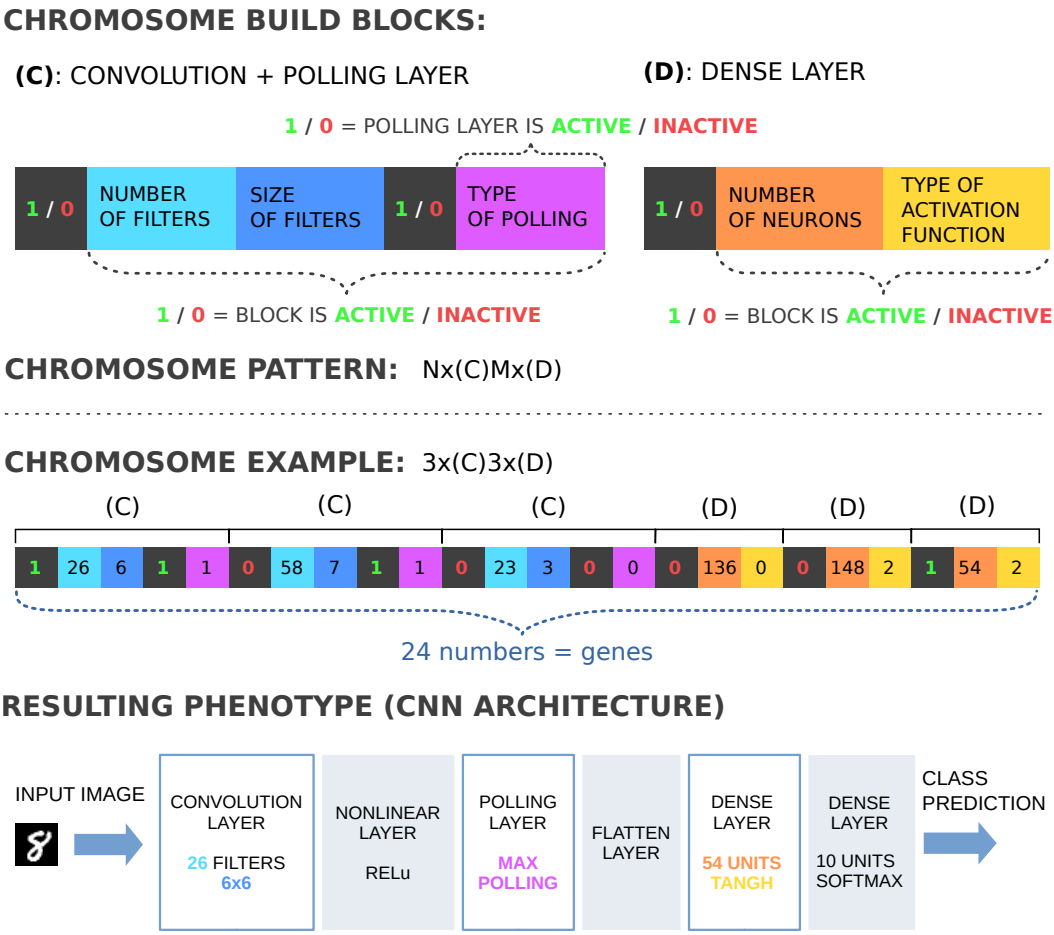
Bloky *(D)* vždy následují až za bloky *(C)* a implicitně se mezi ně umisťuje zplošťující (flatten) vrstva. Za poslední blok *(D)* se poté implicitně umisťuje softmax vrstva, která přiřazuje jednotlivým třídám pravděpodobnost. Třída s nejvyšší pravděpodobností je braná jako výsledek klasifikace. Za oba bloky *(C)* a *(D)* se při trénování sítě implicitně umísťuje tzv. dropout vrstva zbraňující přetrénování. Přičemž u bloku *(C)* se tato vrstva umísťuje, jen pokud je tento blok aktivní. Tato vrstva způsobí, že v každém učícím cyklu je náhodně vybraná část neuronů deaktivovaná. To znamená, že během výpočtu odezvy sítě jsou tyto neurony ignorovány a při zpětném šíření chyby (backpropagation) se jejich váhy neupravují [15]. U bloku *(C)* je míra deaktivace nastavena na 0,1 (10%) a u bloku *(D)* potom na 0,2 (20%).

Princip jak se určuje fitness, byl již trochu nastíněn dříve. Kvůli časové náročnosti se počítá paralelně pro všechny jedince v populaci. A spočívá v natrénování architektury CNN na 5000 vzorcích z trénovací množiny datasetu MNIST (s indexy 5000–5099) a následném otestování na všech vzorcích testovací množiny. Výsledná fitness hodnota pak odpovídá přesnosti klasifikace (accurancy), která se určí podle vzorce 5.1.

$$presnost = \frac{pocet\ spravne\ klasifikovanych\ vzorku}{celkovy\ pocet\ klasifikovanych\ vzorku}. \tag{5.1}$$

Jedná se tedy o hodnotu v rozsahu $\langle 0, 1 \rangle$. A po převodu na procenta vyjadřuje procentuální úspěšnost modelu při klasifikaci. Důležité je zmínit to, že při trénování se v počátku nastaví váhy podle vždy stejného náhodného semínka. Takže stejná architektura sítě bude vždy vykazovat stejnou přesnost klasifikace, což je důležité z hlediska konvergence evoluce, protože i počáteční nastavení vah má na přesnost klasifikace vliv.

Při trénování sítě jsou pak hyperparametry použité optimalizační/minimalizační metody *gradient descent* (gradientní sestup) nastaveny takto: *koeficient učení* je nastavený na hodnotu 0,15 a *velikost dávky* je nastavena na hodnotu 100. Koeficient učení je větší z toho důvodu, aby došlo na 5000 vzorcích alespoň k nějakému uspokojivému natrénování a větší velikost dávky snižuje časovou náročnost trénování. Experimentálně pak bylo zjištěno, že při vyšším koeficientu učení už může dojít (dle podoby trénované architektury) k divergenci při optimalizaci.



Obrázek 5.3: Podoba chromozomu (genotypu) pomocí kterého je zakódována CNN a způsob sestavení fenotypu.

Jako EA jsou v experimentech aplikovány GA a ES. Jejich společné principy a nastavení jsou následující:

- **GA** – Rodiče se vybírají turnajem. Křížení rodičů (pokud je aplikováno) může být uniformní nebo dvoubodové a provádí se s určitou pravděpodobností, u uniformního se navíc s pravděpodobností 50% prohodí hodnota každého prvku chromozomu. V rámci mutace se s určitou pravděpodobností zmutuje každá hodnota genu pomocí náhodného uniformního výběru hodnoty z možného rozsahu hodnot pro daný typ genu. Do další generace přežívají (jsou vybráni) jen potomci. Uplatňuje se elitismus, tedy nejlepší jedinec přežívá vždy do další generace beze změny.
- **ES** – Rodiče se vybírají náhodně. Křížení rodičů (pokud je aplikováno) může být uniformní, kde se s pravděpodobností 50% prohodí hodnota každého prvku chromozomu. V rámci mutace se s určitou pravděpodobností zmutuje každá hodnota genu pomocí náhodného uniformního výběru hodnoty z možného rozsahu hodnot pro daný typ genu. Při výběru jedinců do další generace se uplatňuje strategie plus nebo čárka. U strategie čárka se uplatňuje elitismus (u strategie plus se uplatňuje přirozeně), tedy nejlepší jedinec přežívá vždy do další generace beze změny.

5.3 Experiment s návrhem architektury CNN pro klasifikátor MNIST

Po nezbytném úvodu do řešené problematiky a popisu principů řešení, přichází na řadu už konkrétní praktický experiment. V tomto experimentu bude zkoumána a analyzována schopnost neuroevoluce řešit úlohu s návrhem optimální architektury CNN pro problém klasifikace ručně psaných číslic. V rámci experimentu bude potřeba zvolit vhodné parametry evolučních algoritmů a definovat (omezit) prohledávací prostor architektur CNN, což znamená definovat jak délku chromozomu (počty jednotlivých bloků), tak i rozsah hodnot prvků (genů) těchto bloků. Takové omezení definuje tabulka 5.1. Pro toto omezení bylo experimentálně zjištěno, že vyhodnocení fitness může v nejhorším případě trvat až 1500s. Nejhorší případ je ten, kdy jsou aktivované všechny možné konvoluční vrstvy s maximálním počtem a velikostí filtrů. A navíc nejsou aktivované žádné polling vrstvy, které by redukovali výstupní dimenzionalitu konvolučních vrstev. A navíc jsou aktivované všechny plně propojen vrstvy s maximálním počtem neuronů. Experimentálně však také bylo zjištěno, že podobné architektury se uchytí v populaci jen velmi zřídka a většina nejhorších případů zabere spíše pár stovek sekund. Proto mohl být experiment vůbec proveden.

V experimentu jsou na řešení úlohy aplikovány GA a ES v šesti různých nastaveních. Tyto shodná nastavení jsou uvedena v tabulce 5.2. Celkově tak na experiment připadá 12 výsledků, kterými jsou grafy zobrazující průběh evoluce a úspěšnost při hledání řešení.

Na datovém nosiči jsou poté k dispozici i další zajímavé výsledky z experimentů. Jedná se především o chromozomy, ze kterých je po importu možné sestavit odpovídající model CNN a provádět s tímto modelem experimenty.

Grafy tedy slouží jako podklad pro analýzu a vyhodnocení experimentu. Jeden graf odpovídá konkrétnímu nastavení určitého EA (GA, ES) a pro takto nastavený EA vizualizuje průběh evoluce (závislost fitness na počtu evaluací) pomocí 6 boxplotů s rovnoměrným rozestupem mezi evaluacemi. Počtem evaluací se rozumí počet vyhodnocení fitness ($pocet\,uplynulych\,generaci \cdot pocet\,potomku\,vyhodnocenych\,za\,generaci$). Tyto boxploty jsou vytvořeny pro daný počet evaluací na základě fitness hodnot jedinců ze 30

nezávislých evolučních běhů, aby bylo možné algoritmus statisticky zhodnotit pro různé počáteční nastavení generátoru náhodných čísel. Ve skutečnosti je spuštěno 32 evolučních běhů, ale typicky alespoň jeden nestihne ve vymezeném čase doběhnout (protože objeví složitou architekturu časově náročnou na trénování). Celkový počet evaluací je stanoven na 1380, na základě toho se pro každou variantu EA stanoví konkrétní počet generací jako podíl hodnoty 1380 a počtu potomků vyhodnocených za generaci. Počet evaluací tak není příliš velký, nicméně experiment je časově poměrně náročný (přibližně 80 tisíc jádrohodin), a navíc vývoj v rámci evoluce, jak bude možné pozorovat z grafů, se poměrně rychle ustálí, takže největší smysl má zachytit evoluci právě v ranné fázi, kde dochází k nejzajímavějšímu vývoji. U experimentu jsou grafy omezeny na fitness hodnoty v rozsahu $\langle 0.65, 0.95\rangle$, kde se nachází drtivá většina fitness hodnot jedinců v populaci.

Nad boxploty jsou v grafu zobrazeny i další užitečné udaje, které se k nim váží. Horní řádek mezi nadpisem grafu a horní hranou grafu udává nejvyšší dosaženou přesnost klasifikace (jedná se tedy o maximální možnou fitness hodnotu převedenou na procenta). Tato hodnota je poměrně důležitá a z daného boxplotu lze sice vyčíst, ale jen přibližně.

Vymezení prohledávacího prostoru architektur CNN (vztaženo k chromozomu)	
Parametry / Vymezení	**Vymezení hodnot**
Počet bloků typu (C)	3
Počet bloků typu (D)	3
Počet filtrů v konvoluční vrstvě	$\langle 5, 70\rangle \subset \mathbb{N}$
Velikost filtrů v konvoluční vrstvě	$\langle 2, 10\rangle \subset \mathbb{N}$
Typ podvzorkování v polling vrstvě	$\{averagepolling, maxpolling\}$
Počet neuronů v plně propojené vrstvě	$\langle 100, 700\rangle \subset \mathbb{N}$
Typ aktivační funkce neuronů v plně propojené vrstvě	$\{ReLU, sigmoid, tanh\}$

Tabulka 5.1: Tabulka definuje strukturu chromozomu a rozsahy hodnot jeho prvků. Tímto je v rámci experimentu definován prohledávací prostor architektur CNN.

Genetický algoritmus (GA)						
Parametry/Varianty	**var1**	**var2**	**var3**	**var4**	**var5**	**var6**
Populace	23	23	23	46	46	46
Křížení	Uniformní	Ne	Uniformní	Uniformní	Ne	Uniformní
Pravděpod. křížení	0,7	-	0,9	0,7	-	0,7
Pravděpod. mutace genu	0,1	0,1	0,05	0,1	0,1	0,1
Velikost turnaje	4	4	8	8	4	4

Evoluční strategie (ES)						
Parametry/Varianty	**var1**	**var2**	**var3**	**var4**	**var5**	**var6**
Základní schéma	(23 + 23)	(23, 46)	(23 + 23)	(46 + 23)	(46 + 23)	(46, 92)
Křížení	Ne	Ne	Uniformní	Uniformní	Ne	Ne
Pravděpod. křížení	-	-	0,7	0,9	-	-
Pravděpod. mutace genu	0,1	0,1	0,1	0,05	0,1	0,1

Tabulka 5.2: Tabulka s nastavením ve kterém se liší jednotlivé varianty EA použité v rámci experimentů s evolučním návrhem architektury CNN.

Srovnání variant GA

Všechny uvedené varianty GA (viz 5.4) jsou schopné naleznout poměrně dobré řešení, které má přesnost klasifikace více jak 93%. Z hlediska průběhů se jednotlivé varianty zásadně neodlišují. Nejlepší variantu je objektivně těžké určit, protože v různých kritériích vítězí jiná varianta, záleží tak na na úhlu pohledu. Z hlediska dosažených výsledků je nejlepší varianta 5, která je schopna najít řešení s přesností klasifikace 94,19%. Z hlediska nejvyššího dosaženého mediánu přesnosti klasifikace je poté nejlepší varianta 3. Tato varianta je také nejlepší z hlediska počáteční konvergence. Z hlediska průběhu evoluce je pak nejlepší varianta 6, která vykazuje stabilní růst/konvergenci.

Z hlediska dosažených výsledků je poté nejhorší varianta 2, která dokáže najít řešení s přesností klasifikace 93,80%. Z hlediska kolísavého průběhu evoluce je nejhorší varianta 3. Nejhorší varianta z hlediska rychlosti konvergence a nejvyššího dosaženého mediánu je varianta 1.

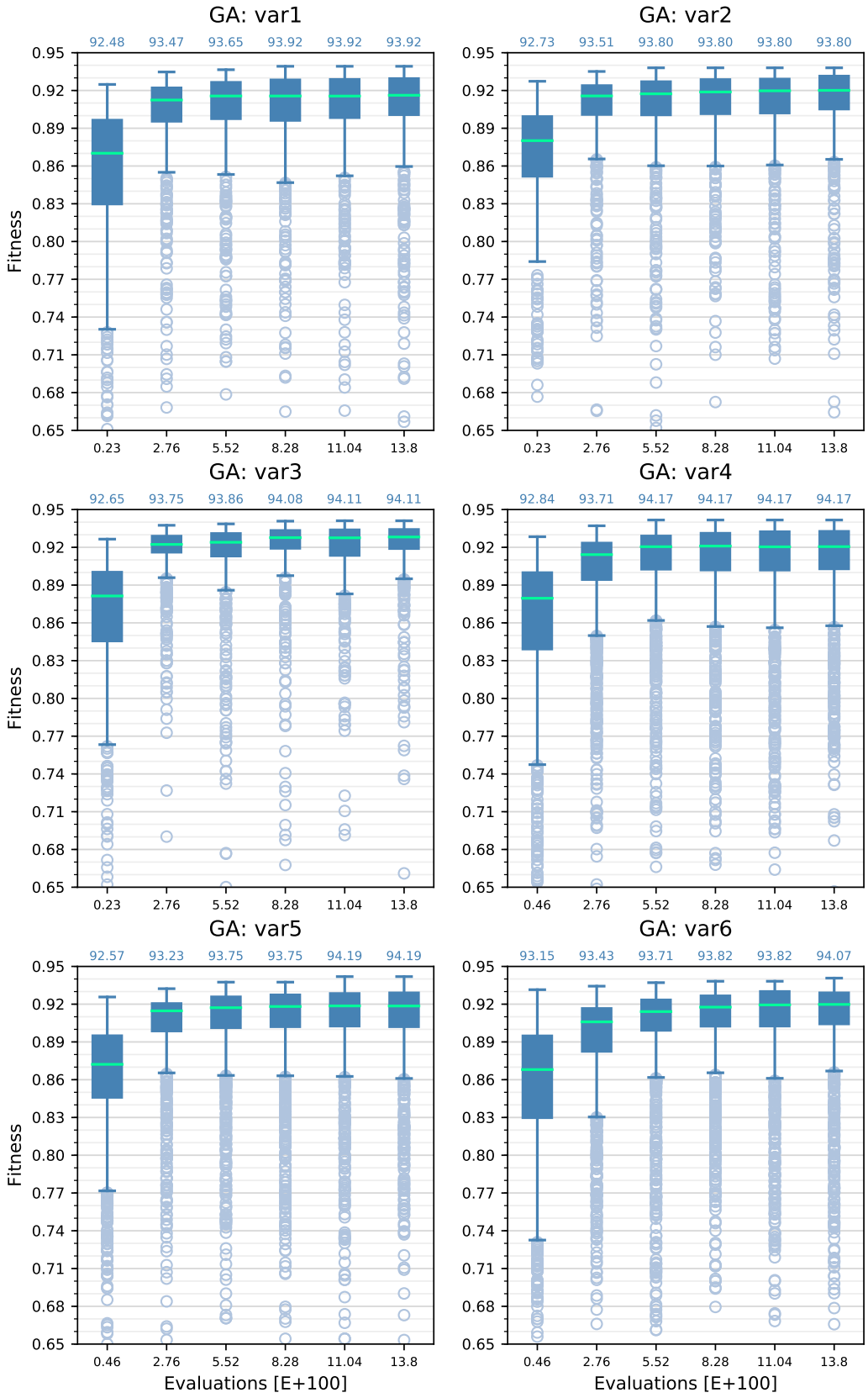
Srovnání variant ES

Všechny uvedené varianty ES (viz 5.5) jsou schopné nalézt poměrně dobré řešení, které má přesnost klasifikace více jak 93%. Z hlediska průběhů se jednotlivé varianty zásadně neodlišují. Nejlepší variantu je objektivně těžké určit, protože v různých kritériích vítězí jiná varianta, záleží tak na na úhlu pohledu. Z hlediska dosažených výsledků je nejlepší varianta 3, která je schopna najít řešení s přesností klasifikace 94,19%. Tato varianta je také spolu s variantou 1 nejlepší z hlediska počáteční konvergence. Z hlediska nejvyššího dosaženého mediánu přesnosti klasifikace je nejlepší variantu obtížné určit. Z hlediska průběhu evoluce vykazují všechny varianty stabilní růst/konvergenci.

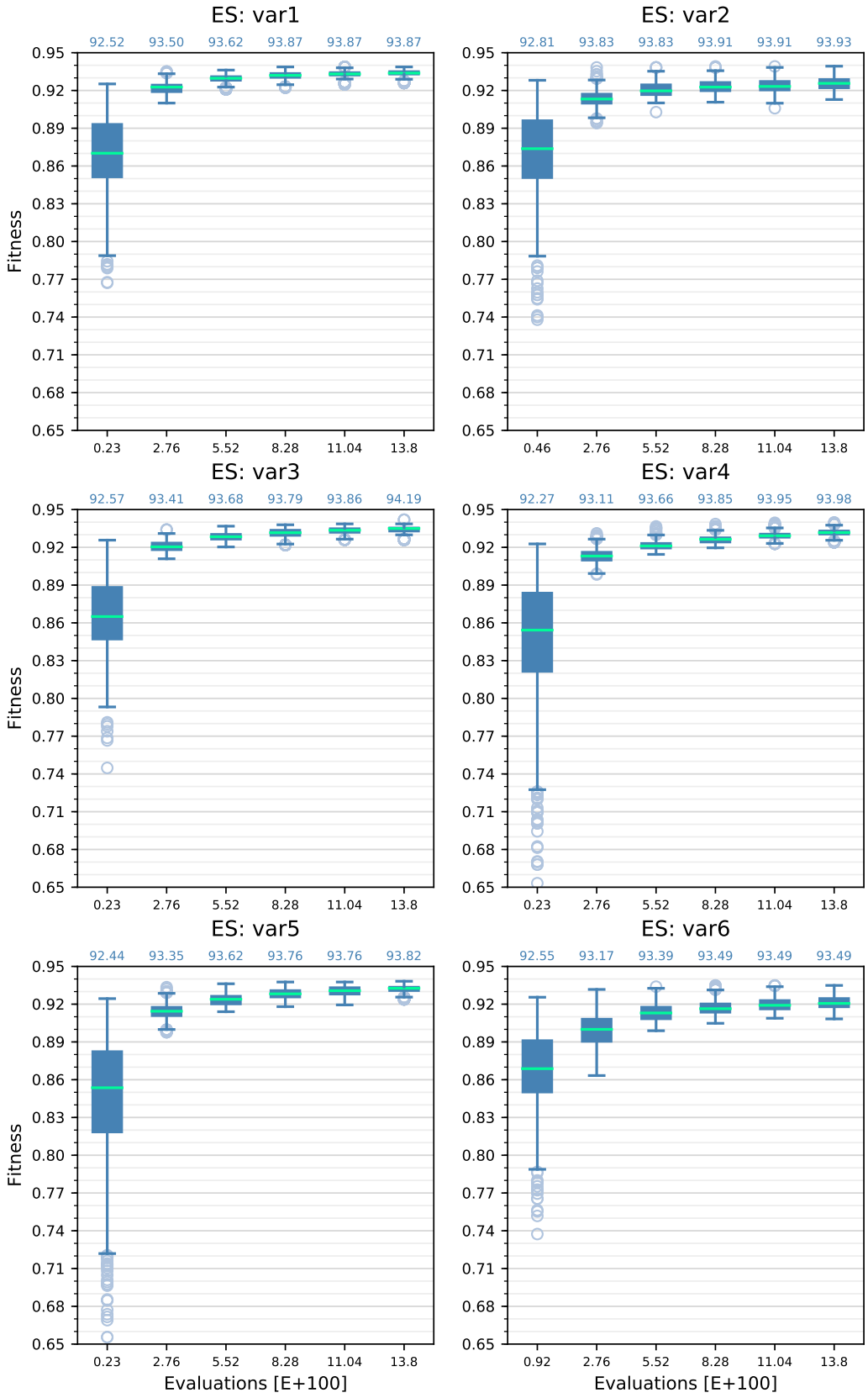
Z hlediska dosažených výsledků je poté nejhorší varianta 6, která dokáže najít řešení s přesností klasifikace 93,49%. Tato varianta je také nejhorší z hlediska rychlosti konvergence a nejvyššího dosaženého mediánu. Objektivně je tuto variantu možné prohlásit za nejslabší.

Srovnání GA, ES a celkové zhodnocení

Všechny algoritmy dokáží najít řešení s přesností klasifikace nad 93%. Nicméně z hlediska počtu variant, které dokáží najít řešení s přesností klasifikace nad 94% vítězí GA kde to dokáží dvě třetiny, zatímco u ES to dokáže jen jedna třetina variant. ES varianty ale vykazují oproti variantám GA mnohem lepší průběh a konvergenci. U ES je mnohem menší rozptyl v kvalitě jedinců a i nejhorší možné případy během evoluce zcela vymizí a je jich podstatně menší množství než u GA, kde po celou dobu evoluce se v populaci vyskytují velmi nekvalitní jedinci. Graf zachycuje jen určitý rozsah fitness hodnot, ale reálně fitness hodnoty klesají u GA i hluboko pod 0,65 (65% přesnost klasifikace) během celé evoluce (nejsou výjimkou i hodnoty 0,10). Z hlediska nejlepších výsledků je zajímavé, že jak GA tak ES nalezly jako nejlepší řešení (architekturu CNN) takové, které má přesnost klasifikace 94,19%. U GA toto dokázala varianta 5 a u ES varianta 3. Toto tak bude zřejmě limit přesnosti klasifikace, kam se lze vůbec dostat s danými 5000 trénovacími vzorky (dávkovaných po 100) a koeficientem učení 1.5. A je zde poměrně velká naděje, že při větším počtu trénovacích vzorků (a odpovídající úpravě koeficientu učení) se tyto nejlepší architektury rovněž vyznamenají, což ostatně ověříme dále v této kapitole.



Obrázek 5.4: Porovnání šesti variant GA na problému evolučního návrhu architektury CNN pro klasifikaci psaných číslic.



Obrázek 5.5: Porovnání šesti variant ES na problému evolučního návrhu architektury CNN pro klasifikaci psaných číslic.

5.4 Výsledky experimentu po řádném natrénování architektur

Experiment ukázal, že je možné použít EA k návrhu vhodné architektury CNN. Nicméně kvůli omezenému počtu trénovacích vzorků se podařilo dosáhnout přesnosti klasifikace nejvýše 94,19%. I toto je poměrně velký úspěch na tom že bylo použito pouze 5000 trénovacích vzorků. Nicméně ambice experimentu jsou mnohem vyšší. Cílem je ukázat, že evolučně navržené architektury (byť pro redukovaný počet trénovacích vzorků) mají potenciál po řádném natrénování se přiblížit state-of-the-art výsledkům.

Důležité je zmínit to, že obecně nemusí platit (a často to taky neplatí), že by architektury s lepší fitness byly po řádném natrénování vždy lepší než architektury s horší fitness. Pokud si totiž architektura vede lépe na 5000 vzorcích, než jiná architektura, nemusí si nutně vést lépe i na zbylých vzorcích trénovací množiny, na kterých bude navíc trénována cyklicky. Toto však spíše nastává pokud nejsou rozdíly ve fitness hodnotách nějak markantní. Obrázky v trénovací množině totiž nejsou zase natolik rozdílné, takže pokud si nějaká architektura povede na 5000 vzorcích výrazně hůř než jiná, tak zřejmě to nebude o moc lepší ani u jiných vzorků.

Nejlepší nalezené architektury z jednotlivých variant EA tak byly natrénovány řádně na 50 a 100 epochách (epochu představuje v našem případě 55000 horních vzorků z trénovací množiny) s koeficienty učení 0,7 a 0,5 a velikostí dávky 100. Celkově tak proběhly 4 subexperimenty. A výsledky jsou velmi zajímavé. Všechny z architektur dosáhly přesnosti nad 99%. Nejlepší dosažena přesnost klasifikace činila 99,49%. A této přesnosti dosáhly shodně 2 architektury:

- **GA: var4 (100 epoch; koeficient učení 0,7)** – Doba tréningu přibližně 11h. Chromozom má podobu: [1, 5, 2, 0, 0, 1, 22, 5, 1, 1, 1, 37, 9, 1, 1, 0, 151, 0, 0, 622, 1, 1, 193, 2]
- **ES: var4 (100 epoch; koeficient učení 0,5)** – Doba tréningu přibližně 39h. Chromozom má podobu: [1, 25, 2, 0, 0, 1, 54, 8, 1, 1, 1, 46, 8, 1, 1, 0, 130, 1, 0, 558, 2, 1, 470, 2]

Z chromozomu vidíme, že obě architektury mají stejný vzor z pohledu aktivních vrstev. Obě mají aktivní 3 konvoluční vrstvy, přičemž jen za posledními dvěma mají aktivní polling vrstvu, která je typu *max-polling*. Obě mají taky jednu plně propojenou vrstvu s aktivační funkcí hyperbolický tangens. Obě se jinak liší ve složitosti z hlediska velikosti a počtu filtrů a neuronů, kdy architektura z *ES: var4* má menší složitost. Pokud bychom zkoumali architektury pro vzájemně prohozené koeficienty učení, tak si obě architektury v klasifikaci pohorší o pár setin procenta. Pořád ale budou dosahovat skvělých výsledků. Vidíme tak, že vhodně nastavený koeficient učení může síť lépe naučit a zvýšit tak přesnost klasifikace. Zároveň ale vidíme, že i jinak složité architektury si mohou vést při klasifikaci podobně, protože roli hrají i počátečně nastavené váhy.

Přesnost klasifikace 99,49% je velmi dobrá a blíží se state-of-the-art výsledku který je 99,79%.[11]

5.5 Základní informace k programové realizaci, vývoji a testování

Experiment byl vyvíjen, spouštěn a testován na superpočítači Salomon.[1] V rámci experimentu bylo využito přibližně 80 tisíc jádrohodin. Experiment je psán v Pythonu s využitím frameworku DEAP[2] pro tvorbu evolučních algoritmů. Pro práci s NN se využívá knihovna TensorFlow[3] od Google. Pro meziprocesorovou komunikaci a tedy i paralelizaci je využita knihovna mpi4py[4] která zpřístupňuje MPI pro Python. Paralelizace počítání fitness jedinců probíhá tak, že proces master realizuje evoluční algoritmus a v případě potřeby vyhodnocení fitness jedinců v populaci zašle k vyhodnocení tyto jedince ostatním pracovním procesům *workers*. Pokud je těchto procesů méně než jedinců, probíhá vyhodnocení fitness semiparalelně, kdy se jedinci vyhodnocují paralelně po dávkách (kde velikost dávky odpovídá počtu *workers*). Počet jedinců ale musí bát v takovém případě násobkem počtu procesů.

Během evoluce se v daných intervalech (po daném počtu generací) zapisují statistiky evoluce do HDF5 souborů (pro danou generaci vznikne v HDF5 souboru zvláštní dataset) z kterých jsou také generovány grafy.

[1] https://docs.it4i.cz/salomon/hardware-overview/
[2] http://deap.readthedocs.io/en/master/
[3] https://www.tensorflow.org/
[4] http://www.mpi4py.scipy.org/docs/

Kapitola 6

Evoluční optimalizace neurokontroléru pro řízení přistání modelu 1. stupně rakety Falcon 9

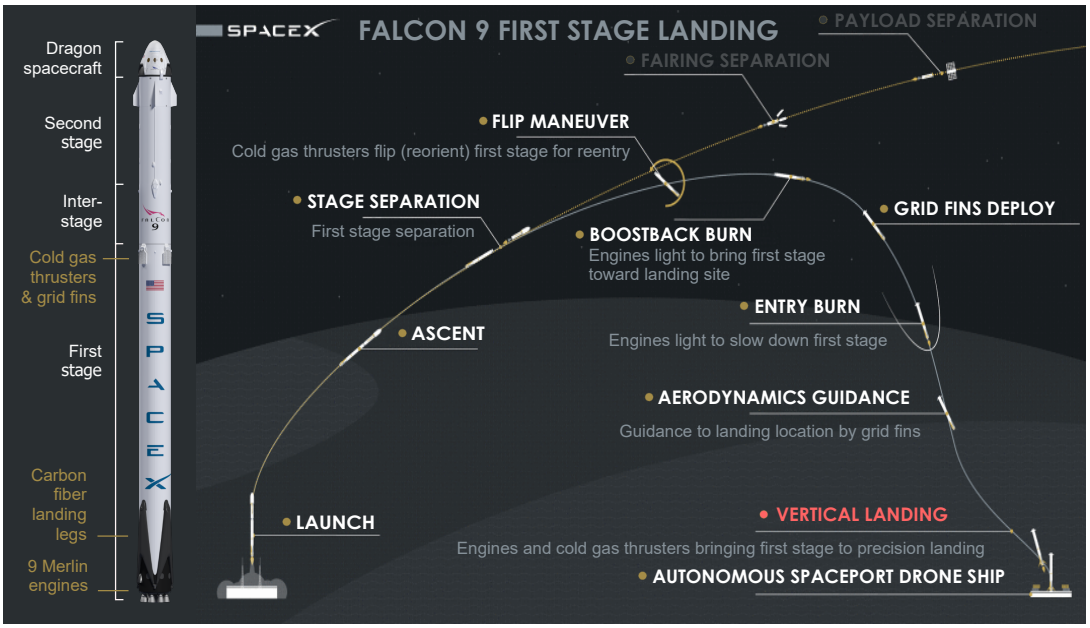
Jednou z významných aplikačních oblastí neuroevoluce je evoluční robotika. Ta spočívá v evolučním návrhu morfologie a kontrolérů nejrůznějších agentů (autonomních robotů – reálných či spíše častěji simulovaných). V kontextu neuroevoluce uvažujeme evoluční návrh kontrolérů, které mají podobu neuronových sítí (nazýváme je poté neurokontroléry), kde vstupy tvoří data ze senzorů agenta a výstupy představují akce agenta. Cílem neuroevoluce v rámci evoluční robotiky je tedy návrh neuronové sítě, která jako kontrolér daného agenta vykazuje požadované chování. [44][20]

Tato kapitola bude zaměřena na experimenty s evoluční optimalizací vah neurokontroléru, který ovládá model 1. stupně rakety Falcon 9 v rámci simulace autonomního přistání tohoto stupně. Základní snahou tak bude najít skrze evoluční algoritmy takové váhy neurokontroléru, že pro dané počáteční podmínky dovede 1. stupeň Falcon 9 (dále bude pro jednoduchost označován také jako raketa) k úspěšnému přistání. Tato kapitola tak představuje další z praktických částí práce.

6.1 Falcon 9

Falcon 9 je dvoustupňová raketa společnosti SpaceX (viz obrázek 6.1), která slouží pro transport nákladu (jako satelity nebo kosmická loď Dragon) na danou oběžnou dráhu. Kosmická loď Dragon [4] je poté schopna dopravovat náklad (a v budoucnu i lidi) na Mezinárodní vesmírnou stanici ISS a následně se bezpečně vrátit zpátky na Zem (a to i s nákladem z ISS) což dnes žádná jiná kosmická loď neumí. Dalším unikátem je to, že zatímco u jiných raket po oddělení 1. stupně tento shoří v atmosféře, 1. stupeň Falcon 9 disponuje systémem bezpečného návratu a přistání. Tímto se stává opětovně použitelný pro další start, čímž je výrazně snížena cena dopravy nákladů do vesmíru. Na 1. stupni je rovněž zajímavé to, že mu mohou až 2 motory z jeho devíti motorů Merlin selhat a přitom stále dokončí svou misi. [5] Během návratu prochází 1. stupeň několika fázemi, které je potřeba autonomně řídit. Vzestup a návrat 1. stupně schematicky zachycuje obrázek 6.1

V rámci této kapitoly budeme experimentovat s návrhem neurokontroléru, který bude autonomně řídit poslední fázi přesného přistání, k čemuž tedy bude využívat hlavní motory a dusíkové trysky. Přistání přitom bude probíhat na autonomní plovoucí plošině, což je



Obrázek 6.1: Obrázek zachycuje na levé straně Falcon 9 a na pravé straně jeho misi s důrazem na část návratu 1. stupně. Návrhem neurokontroleru pro řízení poslední fáze, kterou je precizní vertikální přistání, se zabývá tato kapitola. (Obrázek vytvořen na základě zdrojů [5][9])

ponton s přistávací plochou, který je schopný pomocí motorů udržovat přesnou polohu [41]. U některých misí totiž nemá 1. stupeň dostatek paliva, aby mohl přistát na zemi, a proto takto přistává na oceánu [42]. Schéma takového přistání zachycuje předchozí obrázek 6.1. Fotografie z reálného přistání, které se poprvé podařilo v dubnu 2016, jsou poté zachyceny na obrázku 6.2.

6.2 Simulátor přistání

Simulátor přistání rakety [45] představuje neoficiální prostředí (enviroment) pro OpenAI Gym [14], což je toolkit, který umožňuje srovnání algoritmů pro řešení problémů/úloh posilovaného/zpětnovazebného učení (anglicky reinforcement learning, dále jen RL). RL tvoří celou podoblast strojového učení podobně jako učení s učitelem (supervised learning) a učení bez učitele (unsupervised learnign). Úloha přistání rakety tak má charakter úlohy RL. To znamená že agent (raketa) se pohybuje v diskrétních časových krocích v prostředí a interaguje s okolím, kdy na základě stavu (pozorování) vybírá akci z množiny akcí. Každá akce vynese agentovi určitou odměnu, cílem je pak obvykle maximalizovat kumulativní odměnu. [14] Pro řešení těchto typů úloh/problémů je možné, mimo jiné tradiční metody, použít s úspěchy právě i neuroevoluci. [26]

Při popisu simulátoru budu dále vycházet z jeho zdrojových kódů[1] a také z experimentů s tímto simulátorem. Dokumentace k simulátoru/simulaci v podstatě neexistuje, což pro účely vývoje kontrolérů rakety ovšem není potřeba. V RL úloze, kterou simulace představuje potřebuje znát (jak již bylo výše nastíněno) kontrolér jen specifikaci svých

[1] https://github.com/EmbersArc/gym/blob/master/envs/box2d/rocket_lander.py

Obrázek 6.2: Závěrečná fáze přistání 1. stupně rakety Falcon 9 z dubna 2016 [2], která bude simulována a v rámci níž budou prováděny experimenty, kdy pomocí evoluční optimalizace vah neurokontroléru se bude hledat takový kontrolér, který pro dané poč. podmínky dovede raketu k úspěšnému přistání na plovoucí plošinu. (Obrázek vytvořen na základě zdrojů [7][6])

vstupů/výstupů a zpětnou vazbu (odměnu) a na základě toho se učí správně řídit v našem případě raketu nebo obecně agenta/robota. Ani zde při popisu simulátoru nebudeme zacházet do příliš velkých detailů (a dělat tak tomuto simulátoru úplnou dokumentaci), ale základní parametry simulace je jistě vhodné nastínit, abychom měli představu o řešené úloze. Také je na místě upozornit, že pro potřeby mých experimentů byl simulátor v nezbytných aspektech upraven.

Simulace určitým způsobem zjednodušuje a abstrahuje reálnou situaci. Jedno z těchto zjednodušení je v tom, že probíhá ve 2D, kde je jako fyzikální engine použit Box2D[2]. Dalším zjednodušením je model přistávací situace, který je v měřítku 0,35. Což znamená, že rozměry simulovaného světa (včetně rakety a jiných objektů) jsou přibližně 3krát menší než reálná modelovaná situace. Toto za prvé výrazně redukuje čas simulace a za druhé se tímto přizpůsobujeme použitému fyzikálnímu enginu Box2D, který je optimalizovaný pro menší pohybující se objekty.

Simulace tedy začíná volným nekontrolovatelným pádem rakety pro dané počáteční podmínky (výška, lineární a úhlová rychlost, náklon). Následně přechází raketa na řízené přistání pomocí kontroléru (v našem případě neurokontroléru). Řízení probíhá v každém časovém kroku simulace (což je 60krát za sekundu protože je nastaveno 60 FPS=snímky za sekundu). Vstupem neurokontroléru jsou stavové proměnné, které udávají tyto informace o raketě:

- **pozice x**
- **pozice y (výška)**
- **náklon rakety (úhel)**

[2]https://github.com/pybox2d/pybox2d/wiki/manual

- **indikátor kontaktu levé přistávací nohy s přistávací plochou**
- **indikátor kontaktu pravé přistávací nohy s přistávací plochou**
- **nastavení systému pro řízení velikosti tahu motoru (míra škrcení výkonu motoru)**
- **horizontální rychlost**
- **vertikální rychlost**
- **úhlová rychlost**

Tyto proměnné jsou již v simulátor standardizovány přibližně na na interval -1 až 1. Což samozřejmě pomáhá k lepšímu procesu učení, když data nejsou příliš vychýlená, ale mají nulový střed a jednotkovou varianci [53]. Výstup neurokontroléru se poté liší podle typu simulace. A dle toho se v simulátoru dále zpracovává na konkrétní řídící akce, a ty se převádí na konkrétní fyzikální síly. Způsoby jakými se nakonec řídí raketa ukazuje obrázek. 6.3.

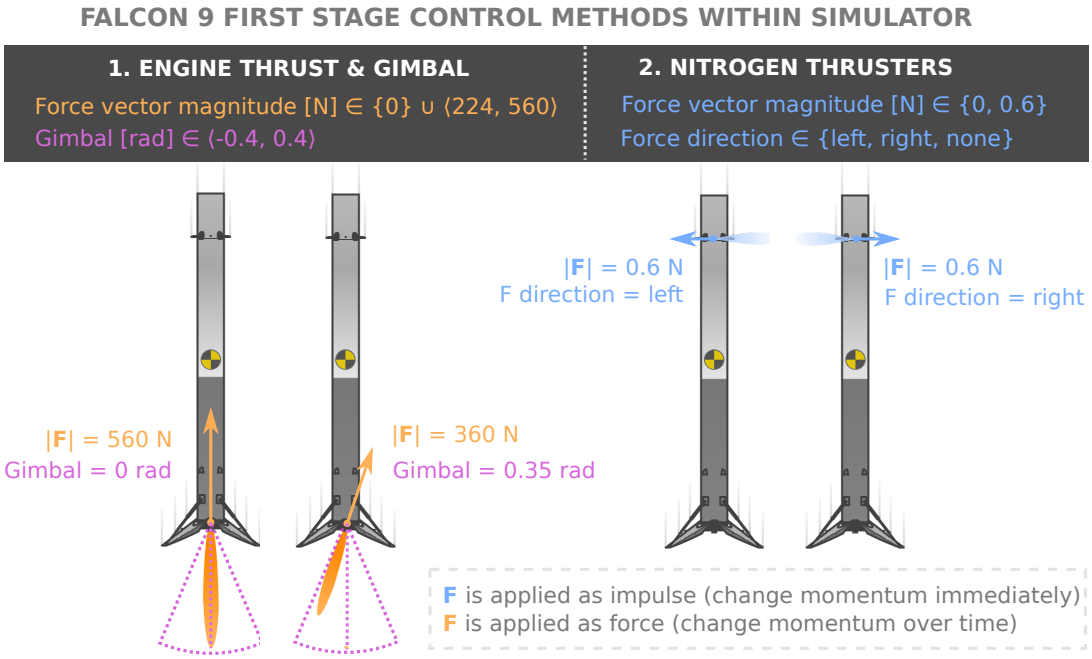
Řízená simulace probíhá nejvýše 1200 kroků simulace, což je při 60 FPS 20 sekund. Pokud je dosaženo tohoto limitu, simulace se předčasně ukončí a přistání tedy neproběhne. Simulace se během řízeného přistání může také ukončit pokud raketa opustí vymezený prostor nebo zkrátka dojde k nárazu či nešetrnému přistání, které nevydrží přistávací nožičky. Naopak pokud dojde k validnímu přistání, simulace se ukončí jako úspěšná. Validní přistáním vyžaduje jednak nízkou rychlost, aby se nezlomili přistávací nožičky a také setrvání na obou nožičkách po dobu 1 sekundy. V reálné situace se mimochodem nožičky vysouvají až těsně před přistáním, což v naší simulace také abstrahujeme.

Z hlediska řízení může simulace probíhat diskrétně nebo spojitě a dle konkrétního způsobu se liší i výstupy neurokontroléru:

- **Spojitá simulace** – V rámci spojité simulace se může najednou aplikovat více řídicích akcí (takže je možné např. zároveň manipulovat s hlavním motorem a dusíkovými tryskami). Také hodnoty těchto řídících akcí nejsou předem dané a určuje je výstup neurokontroléru (který se ale ořízne na interval -1 až 1). Je možné tedy např. zvyšovat tah motoru o libovolně veliké hodnoty v povoleném intervalu. Spojité řízení je v podstatě úloha predikce, kde je třeba, aby neurokontrolér vstupům přiřadil spojité výstupy (celkově to jsou 3 výstupy). Každý takový výstup ovlivňuje potom určitý aspekt řízení rakety: *Vychýlení expanzní trysky motoru doleva/doprava*, *Zvýšení/Snížení tahu motoru*, *Aktivace levé/pravé dusíkové trysky.*
- **Diskrétní simulace** – V rámci diskrétní simulace se aplikuje jen jedna řídicí akce v simulačním kroku (takže není možné např. manipulovat zároveň s dusíkovými tryskami a hlavním motorem) a zároveň hodnota této akce je předem dána (např. se nastaví zvýšení tahu motoru o fixní stupeň). Diskrétní řízení je v podstatě úloha klasifikace, kde je třeba, aby neurokontrolér vstupům přiřadil jednu ze 7 tříd, kde každá třída odpovídá nějaké řídící akci: *Vychýlení expanzní trysky motoru doleva*, *Vychýlení expanzní trysky motoru doprava*, *Zvýšení tahu motoru*, *Snížení tahu motoru*, *Aktivace levé dusíkové trysky*, *Aktivace pravé dusíkové trysky*, *Žádná akce.*

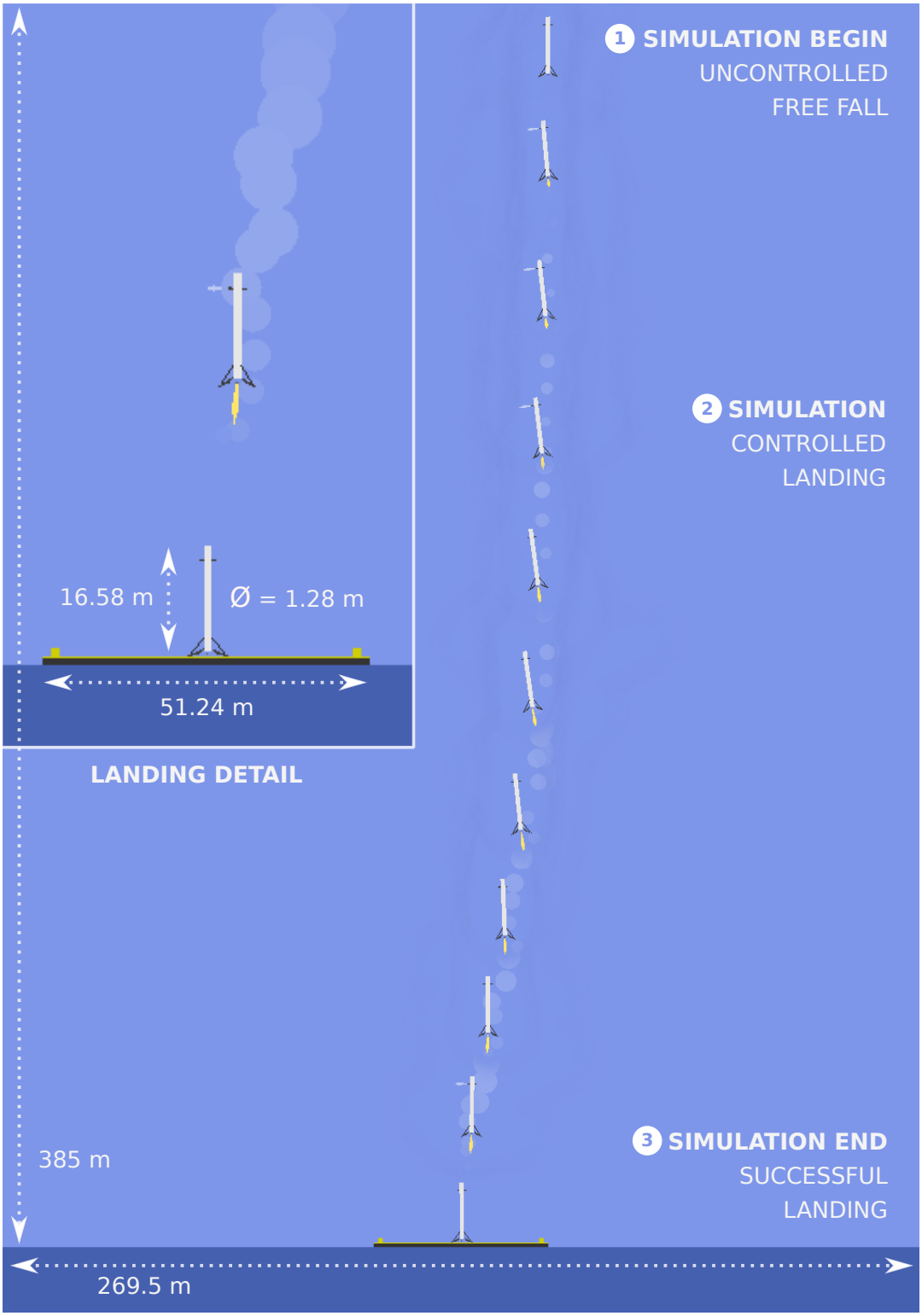
Simulaci je také možné vizualizovat, k tomuto používá simulátor knihovnu Pyglet[3]. Rozměr snímku je potom nastaven jako 1008x1440. O renderování snímku je třeba zažádat

[3]https://bitbucket.org/pyglet/pyglet/wiki/Documentation



Obrázek 6.3: 2 základní způsoby jakými je možné v simulátoru řídit Falcon 9 během přistání (zvyšování/snižování/vektorování tahu raketového motoru a použití dusíkových trysek). Řízení se uplatňuje v každém v časovém kroku simulace. Rozdíl (z hlediska efektu) mezi dusíkovými tryskami a raketovým motorem je v tom, že raketový motor dokáže vyvinout mnohem větší sílu a pomocí této mění postupně hybnost rakety v čase. Trysky poté aplikují sílu jako impulz, kterým mění hybnost rakety okamžitě. [29] V reálné situaci se ještě používají tzv. roštová kormidla [59] (angl. grid fins), ale jejich efekt a význam se snižující se rychlostí klesá (protože jsou založena na aerodynamickém řízení). Navíc v rámci základní 2D simulace bez aerodynamického tření by se jejich efekt simuloval jen těžko. Proto se v simulaci neuvažují. V reálné situaci je také těžiště (černo-žluté kolo uprostřed) umístěno níže. (Obrázek vytvořen na základě zdrojů [51][50])

explicitně, stejně jako je třeba si zažádat explicitně o provedení simulačního kroku. O renderování se také může samo postarat prostředí OpenAI Gym, které pak ukládá posloupnost snímků pomocí programu FFmpeg jako video H.264 v kontejneru MP4. Renderovat také můžeme manuálně a např. pomocí knihovny PIL si snímky uložit ve formátu PNG. Vizualizace úspěšného přistání pak vypadá např následovně: 6.4.



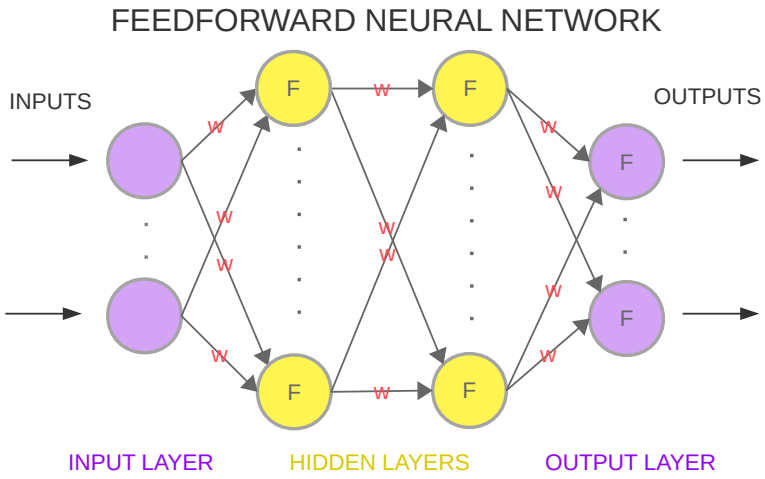
Obrázek 6.4: Vizualizace úspěšného přistání pro dané poč. podmínky s popisem rozměrů simulace. Všimněme si na detailu přistání, že raketa používá jak vektorování tahu hlavních motorů (tryska je vychýlena mírně doleva), tak i levou dusíkovou trysku. Pozn.: časové intervaly mezi jednotlivými pozicemi rakety nejsou stejné.

6.3 Neurokontrolér

Neurokontrolér rakety představuje klasická dopředná neuronová síť. V experimentech, kterými jsme se v této práci zabýval, je použita NN, která se skládá v tomto pořadí z:

1. vstupní vrstvy o počtu 10 neuronů.
2. skryté vrstvy o počtu 20 neuronů.
3. skryté vrstvy o počtu 10 neuronů.
4. výstupní vrstvy o variabilním počtu neuronů dle konkrétního experimentu.

Neurony mezi sousedními vrstvami jsou plně propojeny. Vstupy sítě jsou přivedeny na vstupní vrstvu, která je pouze distribuuje do 1. skryté vrstvy. Neurony ve skrytých vrstvách a výstupní vrstvě poté své vstupy nelineárně transformují pomocí funkce $F(x) = a(b(x))$, kde b je bázová funkce a a je aktivační funkce. Bázová funkce představuje klasický váhovaný součet vstupů a jako aktivační funkce je použit hyperbolický tangens. Podobu takové sítě zachycuje obrázek 6.5. V rámci neuro-evolučních experimentů tak budeme mít fixní architekturu sítě (až na variabilní konfigurace výstupní vrstvy), pro kterou se budou optimalizovat pouze váhy.



Obrázek 6.5: Schéma obecnější podoby neurokontroléru rakety. Červeně jsou vyznačeny váhy, které hledáme pomocí evolučních algoritmů. Ostatní parametry sítě zůstávají fixní.

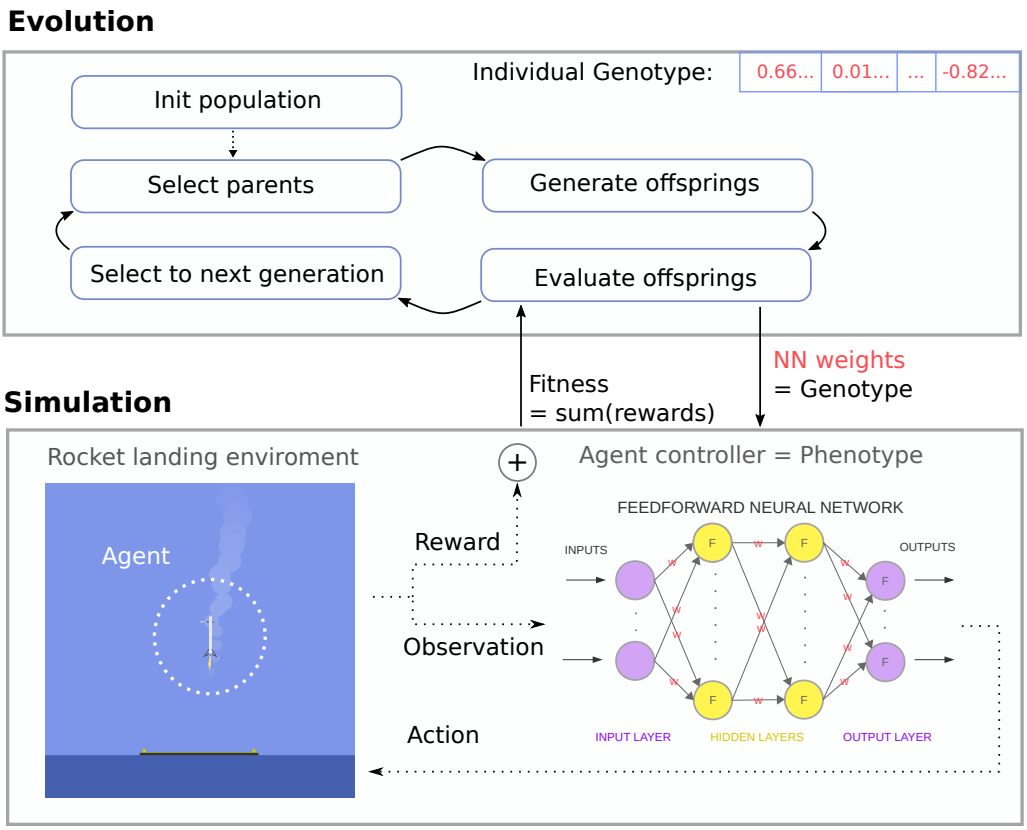
Architektura neuronové sítě má pochopitelně značný vliv při hledání kvalitního neurokontroléru, který je schopný s raketou přistát. Vymezuje nám totiž prohledávací prostor kandidátních řešení, ve kterém pomocí optimalizace vah hledáme ty nejlepší. Pro jednodušší problém, který je řešen např. v rámci experimentu 1, se nicméně ukázalo, že pokud se počet neuronů ve skrytých vrstvách sníží na polovinu, nebo naopak 2krát zvětší oproti výše uvedenému referenčnímu modelu, tak bude evoluce schopna pořád najít v rámci optimalizace vah řešení. Porovnání kvality takto nalezených řešení ale již nebylo předmětem dalšího šetření. Nicméně se tím prokázalo, že takto nastavená architektura bude pro další experimentování poměrně vhodná.

Důvodem, proč vůbec funguje, nebo teoreticky vůbec může fungovat, pro daný problém dopředná síť, je ten, že prostředí, ve kterém se agent pohybuje, je plně pozorovatelné (angl. full observable). To znamená, že agent (neurokontrolér) může zvolit optimální akci jen na základě stavových proměnných, které mu prostředí v daném okamžiku (v daném kroku simulace) poskytne. Takové prostředí se potom označuje jako Markovův rozhodovací proces. Opakem jsou potom částečně pozorovatelné Markovovy rozhodovací procesy. Aby mohl agent v takových prostředích provést optimální akci pro daný simulační krok (ve kterém nemá kompletní informaci o prostředí), je třeba při rozhodování zohlednit i minulé stavy. Za těchto situací (představme si např. že bychom neznali u rakety údaje o její rychlosti) je potom vhodné zvolit jako neurokontrolér rekurentní neuronovou síť. U té se díky smyčkám do rozhodování promítají i minulé stavy, a touto časovou integrací stavů lze postihnout různé časové závislosti a vzory. [31]

Díky aktivačním funkcím *tangh* ve výstupní vrstvě je zajištěno, že neurokontrolér bude mít rozsah výstupů v intervalu $\langle -1.0, 1.0 \rangle$, který předpokládá simulátor. Co se týče hodnot, kterých mohou nabývat váhy, tak je to na optimalizačním algoritmu (EA), avšak ve všech zde prezentovaných experimentech jsou váhy omezeny na interval $\langle -1.0, 1.0 \rangle$.

6.4 Koncept neuroevoluce pro optimalizaci přistávacího kontroleru

V tuto chvíli již máme představu o problému, který řešíme, a částečně i povědomí o tom, jak jej budeme řešit. V této kapitole si toto povědomí rozšíříme, kdy si více popíšeme některé aspekty aplikace neuroevoluce na řešený problém. Zejména se seznámíme s použitými EA, strukturou chromozomu se kterým EA pracují a v neposlední řadě také se způsobem, jakým se vyhodnocuje kvalita kandidátních řešení, což má spolu s vhodným návrhem architektury neuronové sítě zásadní dopad na výsledky. Základní schéma principu činnosti neuroevoluce, jehož komponenty si zde objasníme je v obrázku 6.6.



Obrázek 6.6: Schéma aplikace neuroevoluce při optimalizaci vah neurokontroleru řídícího přistání modelu 1. stupně rakety Falcon 9. Simulace probíhá v simulačních krocích ve kterých agent na základě svých akcí získává odměnu a kumulativní odměna za celou simulaci se použije při výpočtu fitness.

Podoba chromozomu (genotypu) je poměrně jednoduchá a přímočará. Jedná se o vektor reálných čísel, které přímo odpovídají vahám neuronové sítě. Rozsah hodnot, které mohou nabývat prvky chromozomu tedy odpovídá rozsahu vah optimalizované neuronové sítě. V rámci zde uvedených experimentů se jedná o interval $\langle -0.1, 0.1 \rangle$, nicméně v datovém archivu jsou i experimenty s jinými rozsahy. Ve zkratce je možné říct, že zatímco např. u diferenciální evoluce změna rozsahů vah moc nezabrala, u evolučních strategií se v některých případech projevila pozitivně. Díky reálnému chromozomu je problematika přistání rakety převedena na tradiční problém spojité optimalizace. Díky tomu se zde otevírá (i přes omezující podmínky na rozsah hodnot) celá paleta použitelných EA jako je PSO, DE, CMA-ES apod.

Fitness hodnota se u různých experimentů počítá jinak, ale v zásadě se vždy vychází z kumulativní odměny za simulaci. Kumulativní odměna sestává ze součtu dílčích odměn, které simulátor vypočítává pro každý časový krok. Aby celá neuroevoluce správně fungovala, je potřeba toto počítání odměny vyladit. Původní počítání odměn v simulátoru se ovšem ukázalo jako ne příliš vhodné, takže doznalo jistých změn. Způsob jakým se počítají odměny s vyznačenými změnami oproti originálnímu algoritmu ilustruje algoritmus 5.

Hlavním úkolem a cílem neurokontroléru je přistát s raketou rychle, plynule a ekonomicky. Výpočty odměn jsou tak postaveny na tom, že zvýhodňují zlepšování přistávacího profilu během simulačních kroků. To znamená zvýhodnění: postupného snižování rychlosti, zmenšování vzdálenosti od cílového místa a zmenšování náklonu rakety. Jedná se tak o multikriteriální optimalizaci, kde je třeba vážit jednotlivé složky/kritéria. Pokud nějaký kontrolér úspěšně s raketou přistane, tak je ještě extra zvýhodněn. Simulace také dále zvýhodňuje jakýkoliv kontakt nožiček s přistávací plochou. Penalizováno je naopak použití postranních trysek (pokud možno by měl kontrolér používat jen hlavní motor a postranní trysky používat pro vyrovnání co nejméně). Dále je udělena penalizace za každý krok simulace, kdy je raketa ve vzduchu, takže to nutí kontroléry, aby zahájily přistávací manévr co nejdříve a nedrželi raketu příliš dlouho ve vzduchu.

Takto nastavené originální počítání odměny se ale ukázalo jako nevyhovující. Během evoluce měly totiž rakety stále velkou rychlost dopadu a měly tak velký problém s přistáním. Proto jsem zavedl speciálně penalizování rychlosti rakety podle její vertikální vzdálenosti od přistávací plošiny. Tedy čím blíže je raketa zemi, tím menší by měla mít rychlost. Před touto změnou to bylo nedostatečně akcentováno (špatně vyladěno v rámci multikriteriální optimalizace) a po této úpravě počítání začaly rakety už mnohem ochotněji přistávat. Tato změna byla v rámci vyhodnocení odměny nejklíčovější. Další změna je pak v tom, že s klesající vertikální vzdáleností rakety od země je méně penalizováno použití postranních trysek, takže raketa si může blízko země dovolit více manévrovat a tím zvýšit své šance na přistání. Další změna je ve větším zvýhodnění přistání, kdy takový neurokontrolér získá hned 2 body k dobru. Experimentálně bylo zjištěno, že i takto dochází v malém procentu případů k tomu, že kontrolér který s raketou přistane, dostane celkově menší kumulativní odměnu, než ten který nepřistane. Důvodem je to, že kontrolér, který nepřistál má výrazně lepší průběh přistávací křivky než ten který přistál. Poslední změnou je odstranění ořízntutí odměny na interval $\langle -1, 1 \rangle$. Především totiž chceme ve fitness zachytit i drobné nuance v kvalitě jedince a také díky bodovému zvýhodnění přistání na 2b se dostáváme mimo horní jednotkovou hranici.

Algoritmus 5: Způsob jakým se nyní v simulátoru počítá odměna v rámci jednoho simulačního kroku. Kumulativní sumu odměn za simulaci poté používám pro výpočet fitness. Modrou barvou jsou zvýrazněné ty části, které jsem doplnil a šedou barvou ty, které jsme odstranil z originálního algoritmu pro výpočet odměny.

```
// Příprava stavových proměnných
vzdalenost_y ← normalizace vertikální vzdálenosti na interval ⟨0, 1⟩;
vzdalenost_x ← normalizace horizontální vzdálenosti na interval ⟨−0.5, 0.5⟩;
uhel ← normalizace úhlu náklonu rakety na interval ⟨−1, 1⟩;
vzdalenost ← ∛(vzdalenost_y^2 + (vzdalenost_x * 3)^2);
rychlost ← ∛((rychlost_vertikalni / poc_rychlost_vertikalni)^2 + (rychlost_horizontalni / poc_rychlost_vertikalni)^2);
penalizace ← 0.1 / (FPS=snimky za sekundu);
// Výpočet odměny kroku simulace
odmena ← 0;
if použita postranní tryska then odmena ← −penalizace · vzdalenost_y^2 ;
if není konec simulace then
    ohodnoceni_t ← −0.5 · (vzdalenost + (rychlost · (1 + (1 − vzdalenost_y))^4)^2 +
      |uhel|^2)) + 0.1 · pocet_nozicek_v_kontaktu_s_pristavaci_plochou;
    if existuje ohodnoceni_{t−1} then
        odmena ← odmena + (ohodnoceni_t − ohodnoceni_{t−1});
    if pocet_nozicek_v_kontaktu_s_pristavaci_plochou = 0 then
        odmena ← odmena − 0.25 / FPS;
    else if raketa úspěšně přistála then
        odmena ← 1 · 2;
        Nastavení příznaku konce simulace ;
if je konec simulace then
    odmena ←
      odmena + max(−1, −2 · (rychlost + vzdalenost + |uhel| + |uhlova_rychlost|));
Oříznutí odměny na hranici intervalu ⟨−1, 1⟩ pokud se nachází mimo něj;
```

Jako EA jsou v experimentech aplikovány GA, ES a DE. Jejich společné principy a nastavení jsou následující:

- **GA** – Rodiče se vybírají turnajem. Křížení rodičů (pokud je aplikováno) může být uniformní nebo dvoubodové a provádí se s určitou pravděpodobností, u uniformního se navíc s pravděpodobností 5% prohodí hodnota každého prvku chromozomu. V rámci mutace se s určitou pravděpodobností zmutuje každá hodnota genu buď uniformně (pomocí náhodného uniformního výběru hodnoty z možného rozsahu hodnot pro daný typ genu) a nebo pomocí Gaussovské mutace (s určitou standardní odchylkou a provede se oříznutí na interval povolených hodnot vah NN). Do další generace přežívají (jsou vybráni) jen potomci. Uplatňuje se elitismus, tedy nejlepší jedinec přežívá vždy do další generace beze změny.
- **ES** – Samoadaptující se varianta, kde chromozom jedince obsahuje samoadaptující se parametr (*strategy*), který se uplatní v Gaussově mutaci. Rodiče se vybírají náhodně. Křížení rodičů (pokud je aplikováno) může být uniformní, kde se s pravděpodobností 50% prohodí hodnota každého prvku chromozomu nebo aritmetické. V rámci mutace jedince se nejdříve zmutuje *strategy* parametr tak, že se vynásobí hodnotou

$\exp(\frac{1}{\sqrt{dlka\ chromozomu}} \cdot \mathcal{N}(0,1))$. Následně se zmutuje každý gen jedince, tím že se k němu přičte hodnota výsledku násobení $strategy \cdot \mathcal{N}(0,1))$. [12] Pokud se takto zmutovaný gen dostane mimo hranici povolených hodnot vah NN, tak se k této hranici přiblíží na 80% své současné vzdálenosti. Při výběru jedinců do další generace se uplatňuje strategie plus nebo čárka. U strategie čárka se uplatňuje elitismus (u strategie plus se uplatňuje přirozeně), tedy nejlepší jedinec přežívá vždy do další generace beze změny.

- **DE** – Uplatňuje se buď schéma rand/1/bin nebo best/2/bin. S tím, že ostatní parametry jsou variabilní. Variabilní je rovněž způsob oříznutí hodnoty genu nacházející se mimo vymezený interval, kde je možné uplatnit přístupy použité v GA a ES.

Ne všechny výše uvedené varianty EA (např. různé druhy křížení a mutací) však jsou v prezentovaných experimentech nakonec využity.

6.5 Experimenty s optimalizací přistávacího kontroléru

Po nezbytném úvodu do řešené problematiky a popisu principů řešení, přichází na řadu už samotné experimenty. V experimentech zkoumáme a analyzujeme schopnost neuroevoluce řešit úlohu s přistáním rakety. Celkově jsou prezentovány 3 experimenty, v rámci nichž se řešená úloha obecně liší v počátečních podmínkách rakety, různém počtu počátečních podmínek (pro které požadujeme přistání rakety) a různých způsobech ovládání rakety. V každém experimentu jsou na řešení úlohy aplikovány GA, ES a DE v šesti různých nastaveních. Nastavení jsou v rámci všech experimentů stejná, tedy stejnou sady algoritmů ověřujeme pro každý experiment. Toto nám poté umožňuje porovnat jak si algoritmy a jejich nastavení vedou na různých parametrech úlohy. Nastavení nejsou vybrána náhodně, ale jde o výběr nastavení, která se ukázala jako relativně schopná při řešení úlohy v rámci experimentu 1 a tedy je pravděpodobné, že s těmito nastaveními se podaří vyřešit i složitější úlohy v experimentech 2 a 3. Dalším kritériem pro výběr těchto nastavení bylo to, že se od sebe zase tolik neliší (takže pozorný čtenář může vysledovat, jaký vliv má změna toho či onoho parametru na výkonnost algoritmu). Zároveň ale v rámci experimentu 1 vykazují tyto nastavení/varianty na grafech dostatečně variabilní průběhy, takže je na co se dívat. Posledním kritériem při výběru bylo to, aby alespoň jedno nastavení patřilo spíše mezi ty horší. Vybrat daná nastavení, aby splnila všechna na ně kladená kritéria, byla poměrně věda. Tyto shodná nastavení pro všechny experimenty jsou v tabulce 6.1. Celkově tak na jeden experiment připadá 18 výsledků, kterými jsou grafy zobrazující průběh evoluce a úspěšnost při hledání řešení.

Na datovém nosiči jsou poté k dispozici i další zajímavé výsledky z experimentů, které zde není možné prezentovat. Jedná se především o videozáznamy ze simulace přistání, které jsou pořizovány pro ty nejlepší neurokontroléry průběžně během evoluce. Spolu s videozáznamy jsou uloženy i váhy daných neurokontrolérů. Tyto váhy je možné např. importovat do modelu NN a s takto získaným neurokontrolerem dále experimentovat.

Grafy tedy slouží jako podklad pro analýzu a vyhodnocení experimentu. Jeden graf odpovídá konkrétnímu nastavení určitého EA (GA, ES, DE) a pro takto nastavený EA vizualizuje průběh evoluce (závislost fitness na počtu evaluací) pomocí 11 boxplotů s rovnoměrným rozestupem mezi evaluacemi. Počtem evaluací se rozumí počet vyhodnocení fitness ($pocet\ uplynulych\ generaci \cdot pocet\ potomku\ vyhodnocenych\ za\ generaci$). Tyto boxploty jsou vytvořeny pro daný počet evaluací na základě fitness hodnot jedinců ze 48

obecně nezávislých evolučních běhů, aby bylo možné algoritmus statisticky zhodnotit pro různé počáteční nastavení generátoru náhodných čísel. Celkový počet evaluací je stanoven na 105 000, na základě toho se pro každou variantu EA stanoví konkrétní počet generací jako podíl hodnoty 105 000 a počtu potomků vyhodnocených za generaci. U jednotlivých experimentů jsou grafy omezeny na fitness hodnoty v rozsahu $\langle maximalni\,hodnota\,fitness + 1 - 30, maximalni\,hodnota\,fitness + 1\rangle$, kterých nabývá naprosto drtivá většina jedinců.

Nad boxploty jsou v grafu zobrazeny i další užitečné udaje o průběhu a schopnostech daného EA Horní řádek udává kolik procent jedinců v populaci (bráno ze všech 48 evolučních běhů pro daný počet evaluací) představuje úspěšné řešení problému. Spodní řádek poté udává kolik procent ze 48 evolučních běhů nalezlo řešení (tedy v kolika procentech případů je evoluce úspěšná). Dobrý algoritmus by měl mít obě hodnoty relativně vysoké. Pochopitelně už je vůbec úspěch pokud se alespoň nějaké řešení podaří nalézt.

Genetický algoritmus (GA)						
Parametry/Varianty	**var1**	**var2**	**var3**	**var4**	**var5**	**var6**
Populace	70	70	70	140	70	140
Křížení	Ne	Ne	Uniformní	Ne	Ne	Dvou-bodové
Pravděpod. křížení	-	-	0,8	-	-	0,8
Pravděpod. mutace genu	0,05	0,05	0,05	0,05	0,1	0,05
Velikost turnaje	10	5	5	5	5	5
Stand. odchylka σ Gauss. mutace	0,2	0,2	0,2	0,2	0,1	0,4

Evoluční strategie (ES)						
Parametry/Varianty	**var1**	**var2**	**var3**	**var4**	**var5**	**var6**
Základní schéma	(70 + 70)	(70, 140)	(70 + 70)	(70, 140)	(140 + 70)	(140 + 70)
Poč. rozsah parametru *strategy* (samoadaptující se parametr mutace)	0,5—1	0,2—0,7	0,2—0,7	0,2—0,7	0,2—0,7	0,1-–0,5

Difereciální evoluce (DE)						
Parametry/Varianty	**var1**	**var2**	**var3**	**var4**	**var5**	**var6**
Populace	70	70	70	70	140	280
Základní schéma	rand/1/bin	best/2/bin	rand/1/bin	best/2/bin	rand/1/bin	rand/1/bin
F (škálovací faktor mutace)	0,5	0,3	0,3	0,1	0,1	0,05
CR (míra křížení)	0,7	0,7	0,7	0,9	0,9	0,95

Tabulka 6.1: Tabulka s nastavením ve kterém se liší jednotlivé varianty EA použité v rámci experimentů s evoluční optimalizací vah neurokontroléru. Tato nastavení jsou společná pro všechny experimenty.

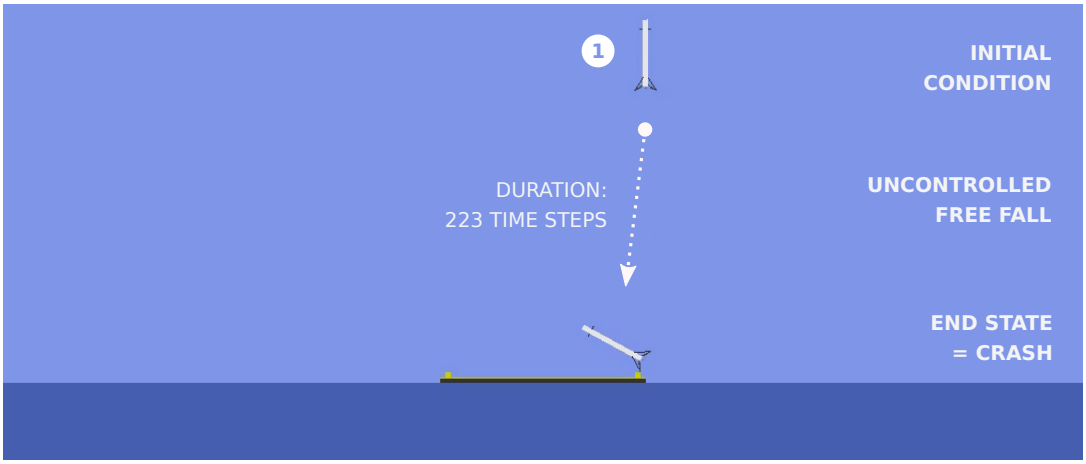
6.5.1 Experiment 1 - jedna počáteční podmínka, spojitá simulace

V prvním experimentu je řešena úloha s přistáním rakety pro jedinou počáteční podmínku a to v rámci spojité simulace. Počáteční podmínka je spíše jednodušší, neboť raketa je poměrně blízko cílového místa přistání a nemá náklon. Nicméně má trochu větší rotaci. Díky spojité simulaci může kontrolér v jeden čas provádět více akcí najednou (řídit tah a vektorizaci motoru a používat postranní trysky) a může řídit míru těchto akcí. Jedná se

tak o základní experiment z hlediska obtížnosti úlohy. Parametry počáteční podmínky jsou přehledně shrnuty v tabulce 6.2. Ukázka dopadu rakety pro danou počáteční podmínku (pokud není raketa nijak ovládána) představuje obrázek 6.7.

Experiment 1 - Počáteční podmínky/stavy rakety	
Parametry / Poč. podmínky	**Podmínka 1**
Horizontální vzdálenost od středu přistávací plochy [m] kladná/záporná = raketa napravo/nalevo od středu	25,62
Vertikální souřadnice / výška [m]	365,75
Úhel náklonu [rad] kladný/záporný = v protisměru / ve směru hodinových ručiček	0
Lineární rychlost vertikální [m/s] kladná/záporná = nahoru/dolů	-80
Lineární rychlost horizontální [m/s] kladná/záporná = napravo/nalevo	-2,28
Úhlová rychlost [rad/s] kladná/záporná = v protisměru / ve směru hodinových ručiček	0,3

Tabulka 6.2: Počáteční podmínka/stav rakety v experimentu 1. V tomto stavu je spuštěna simulace.



Obrázek 6.7: Pro danou počáteční podmínku v rámci experimentu 1 vizualizujeme stav rakety na začátku simulace a na konci simulace (po neřízeném pádu). V rámci experimentu 1 chceme najít pomocí neuroevoluce neurokontrolér, který pro tuto počáteční podmínku dovede raketu k úspěšnému přistání.

Srovnání variant GA

Všechny uvedené varianty GA (viz 6.8) jsou schopné nalézt řešení. Z daných variant vychází objektivně jako nejlepší varianta 4. Je nejlepší po všech stránkách, jak z hlediska průběhu evoluce, tak i výsledků, kdy v 92% případů je schopna nalézt řešení a 25% jedinců v populaci představuje řešení. Nejhorší varianta už není na první pohled tak zřejmá, ale mělo by se s největší pravděpodobností jednat o variantu 6, která nevykazuje žádnou konvergenci a jen 2,7% jedinců v populaci představuje řešení. Tato varianta je ale stále schopná v 77%

případů nalézt řešení. Varianta 3 sice nalezne řešení jen 69% případů, ale oproti variantě 6 vykazuje mnohem lepší konvergenci a mnohem více jedinců v populaci představuje řešení.

Srovnání variant ES

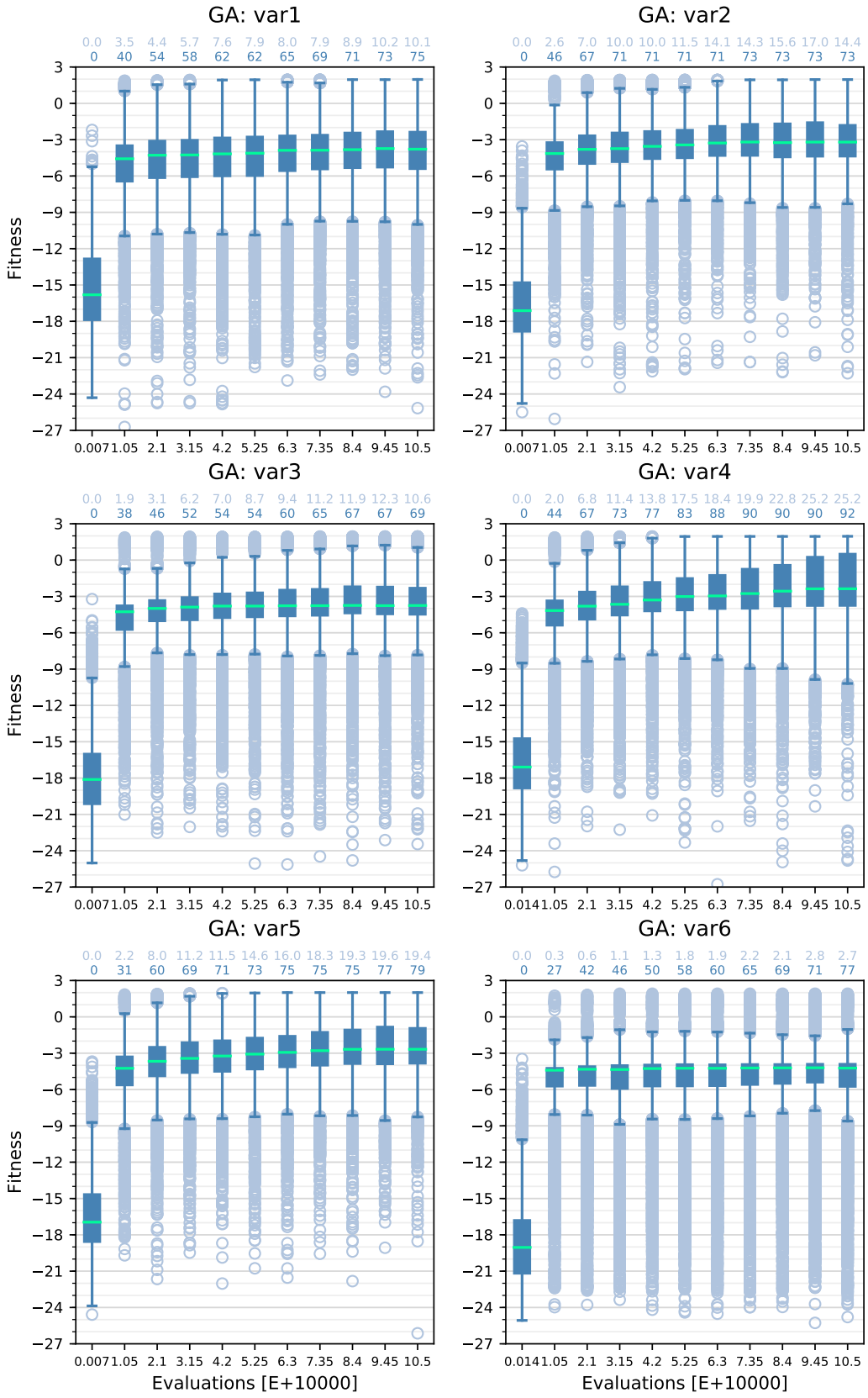
Všechny uvedené varianty ES (viz 6.9) jsou schopné nalézt řešení. Z daných variant vychází objektivně jako nejlepší varianta 6. Je nejlepší po všech stránkách, jak z hlediska průběhu evoluce, tak i výsledků, kdy v 88% případů je schopna nalézt řešení a 52,9% jedinců v populaci představuje řešení. Z hlediska dosažených výsledků je nejhorší varianta 1, která je schopna nalézt řešení jen v 12% případů a jen 0,2% jedinců v populaci u ní představuje řešení. Nicméně na rozdíl od variant 2 a 4 vykazuje její průběh vzestupnou tendenci. Varianty 2 a 4 dosahují o něco lepších výsledků, ale brzy přestávají konvergovat.

Srovnání variant DE

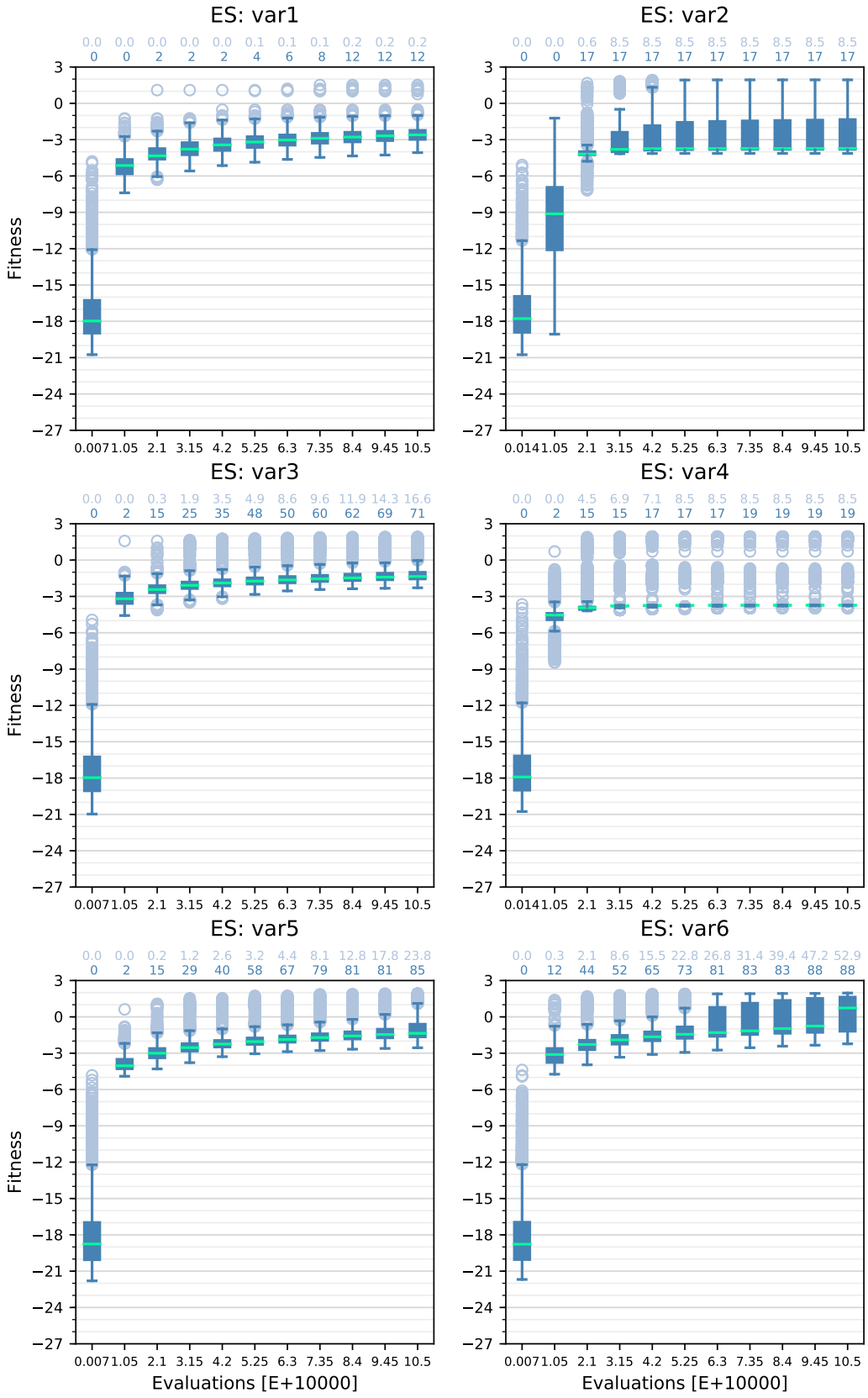
Všechny uvedené varianty DE (viz 6.10) jsou schopné nalézt řešení. Z daných variant vychází jako nejlepší varianty 5 a 6. Varianta 5 dokáže v 94% případů nalézt řešení v čemž si vede nejlépe. Z hlediska množství jedinců v populaci představující řešení vítězí varianta 6 u níž je to přes 63% jedinců. Obě varianty vykazuje dobrý průběh a konvergenci. Mezi nejslabší varianty patří varianta 1 a 4. Varianta 1 oproti variantě 4 vykazuje lepší konvergenci a je schopna nalézt řešení ve více případech, avšak jen 0,2% jedinců v populaci představuje řešení. Varianta 4 poté dokáže nalézt řešení jen ve 4% případů a má statický průběh.

Srovnání GA, ES, DE a celkové zhodnocení

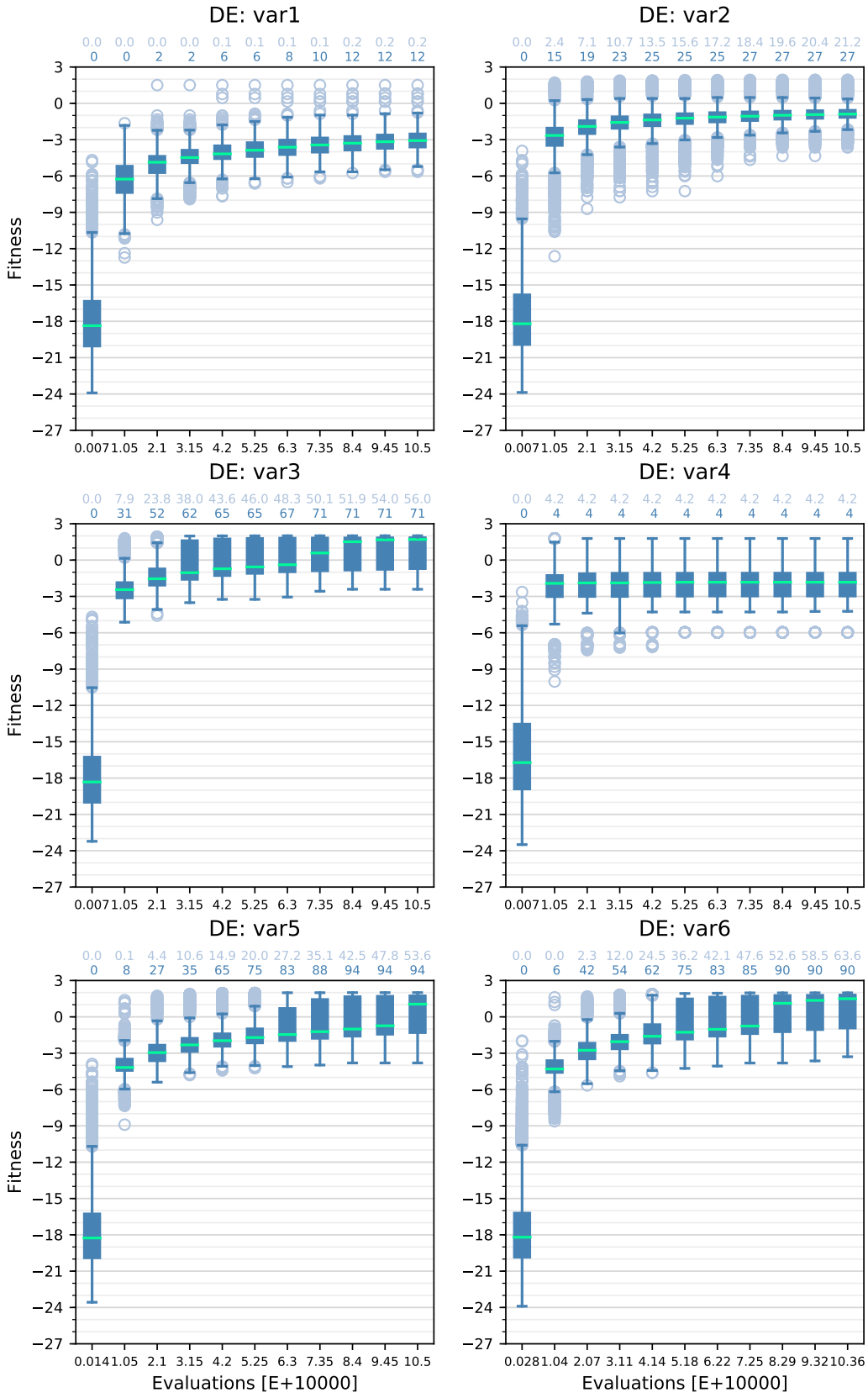
Všechny algoritmy dokáží najít řešení a v rámci této úlohy tedy uspěly a to poměrně uspokojivě. Pokud se v něčem algoritmy výrazně liší, tak zjevně větší míra explorace u GA než u ES či DE. U GA se totiž vyskytuje mnohem více okrajových hodnot, boxploty mají na pohled větší rozsah, taky obvykle nalezne řešení ve větším procentu případů, ale z hlediska procenta jedinců v populaci představující řešení je na tom hůře. Z hlediska výsledků drží prvenství DE, jejíž varianta 5 dokáže najít řešení v 94% případů a varianta 6 obsahuje v populaci 63,6% jedinců, kteří představují řešení.



Obrázek 6.8: Porovnání šesti variant GA na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 1.



Obrázek 6.9: Porovnání šesti variant ES na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 1.



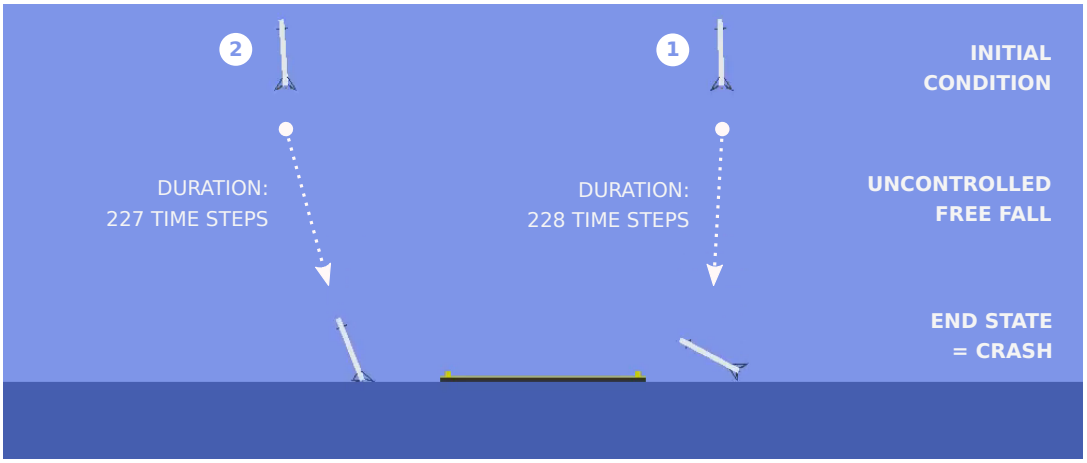
Obrázek 6.10: Porovnání šesti variant DE na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 1.

6.5.2 Experiment 2 - dvě počáteční podmínky, spojitá simulace

Ve druhém experimentu je řešena úloha s přistáním rakety pro dvě různé počáteční podmínky a to v rámci spojité simulace. Počáteční podmínky jsou už v porovnání s experimentem 1 více komplikovanější, především z hlediska vzdálenosti od cílového místa přistání. Navíc je požadavkem aby raketa přistála pro obě dvě podmínky, což činí úlohu mnohem náročnější a neurokontrolér už musí mít obecnější charakter. Díky spojité simulaci může kontrolér stejně jako v experimentu 1 v jeden čas provádět více akcí najednou (řídit tah a vektorizaci motoru a používat postranní trysky) a může řídit míru těchto akcí. Z pohledu obtížnosti se tak celkově jedná o pokročilejší experiment. Parametry počáteční podmínky jsou přehledně shrnuty v tabulce 6.3. Ukázka dopadu rakety pro dané počáteční podmínky (pokud není raketa nijak ovládána) představuje obrázek 6.11.

Experiment 2 - Počáteční podmínky/stavy rakety		
Parametry / Poč. podmínky	**Podmínka 1**	**Podmínka 2**
Horizontální vzdálenost od středu přistávací plochy [m] kladná/záporná = raketa napravo/nalevo od středu	44,59	-64,31
Vertikální souřadnice / výška [m]	365,75	365,75
Úhel náklonu [rad] kladný/záporný = v protisměru / ve směru hodinových ručiček	0,025	0,062
Lineární rychlost vertikální [m/s] kladná/záporná = nahoru/dolů	-77,78	-77,96
Lineární rychlost horizontální [m/s] kladná/záporná = napravo/nalevo	-0,66	4,81
Úhlová rychlost [rad/s] kladná/záporná = v protisměru / ve směru hodinových ručiček	0,28	0,11

Tabulka 6.3: Počáteční podmínka/stav rakety v experimentu 2. V tomto stavu je spuštěna simulace.



Obrázek 6.11: Pro dané počáteční podmínky v rámci experimentu 2 vizualizujeme stav rakety na začátku simulace a na konci simulace (po neřízeném pádu). V rámci experimentu 2 chceme najít pomocí neuroevoluce neurokontrolér, který pro tyto dvě různé počáteční podmínky dovede raketu k úspěšnému přistání.

Z pohledu výpočetní náročnosti bude potřeba v rámci vyhodnocení fitness provést simulaci pro obě počáteční podmínky, což časově experiment velmi prodraží a to přibližně 2krát. Fitness hodnota se počítá jako průměrná kumulativní odměna na základě kumulativních odměn získaných ze dvou simulací pro 2 různé počáteční podmínky. Navíc pokud raketa v obou simulacích přistála, připočte se k průměrné kumulativní odměně jednička.

Srovnání variant GA

Ne všechny varianty GA (viz 6.12) jsou schopné nalézt řešení, konkrétně jedna z variant nenalezne žádné řešení. Z daných variant vychází objektivně jako nejlepší varianta 4. Je nejlepší po všech stránkách, jak z hlediska průběhu evoluce, tak i výsledků, kdy v 29% případů (v experimentu 1 92%) je schopna nalézt řešení a 3.1% jedinců v populaci (v experimentu 1 25%) představuje řešení. Nejhorší varianta je zcela evidentně variantu 6, která není schopná najít žádné řešení a navíc to vypadá jakoby konvergovala k horším fitness hodnotám. Druhá nejhorší je varianta 3, která nalezne řešení v 17% případů (v experimentu 1 69%). Je tak vidět, že varianta 4, která excelovala v experimentu 1 podává nejlepší výsledky i v této náročnější úloze. Podobně varianty 3 a 6, které v experimentu 1 propadly, jsou nejhoršími variantami i zde. A pokud se budeme dívat pozorně všimneme si, že je to tak i s ostatními variantami. Zdá se tedy, že v případě GA je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze.

Srovnání variant ES

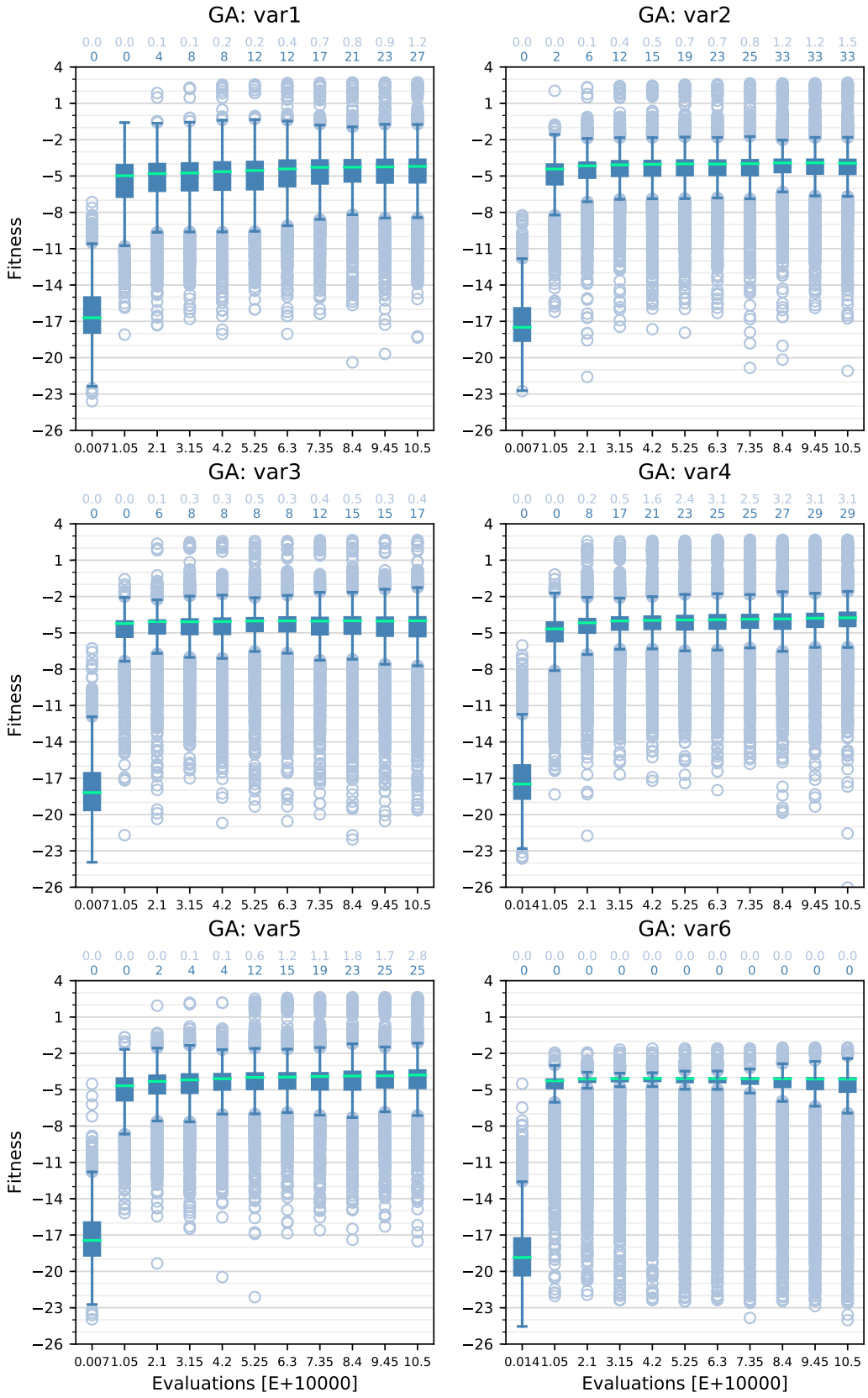
Ne všechny varianty ES (viz 6.13) jsou schopné nalézt řešení, konkrétně 3 varianty (polovina) nenalezne žádné řešení. Z daných variant vychází jako nejlepší varianty 3 a 6. Varianta 6 je ve 4% případů (v experimentu 1 88%) schopna nalézt řešení a 1,3% jedinců v populaci (v experimentu 1 52,9%) představuje řešení. Varianta 3 má 2,1% jedinců v populaci (v experimentu 16,6%) představující řešení. Z hlediska dosažených výsledků jsou jednoznačně nejhorší varianty 1, 2 a 4, které nejsou schopné nalézt žádné řešení. Je tak vidět, že varianta 6, která excelovala v experimentu 1 podává nejlepší výsledky i v této náročnější úloze. Podobně varianty 1, 2, 4, které v experimentu 1 propadly, jsou nejhoršími variantami i zde. Zdá se tedy, že v případě ES je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze.

Srovnání variant DE

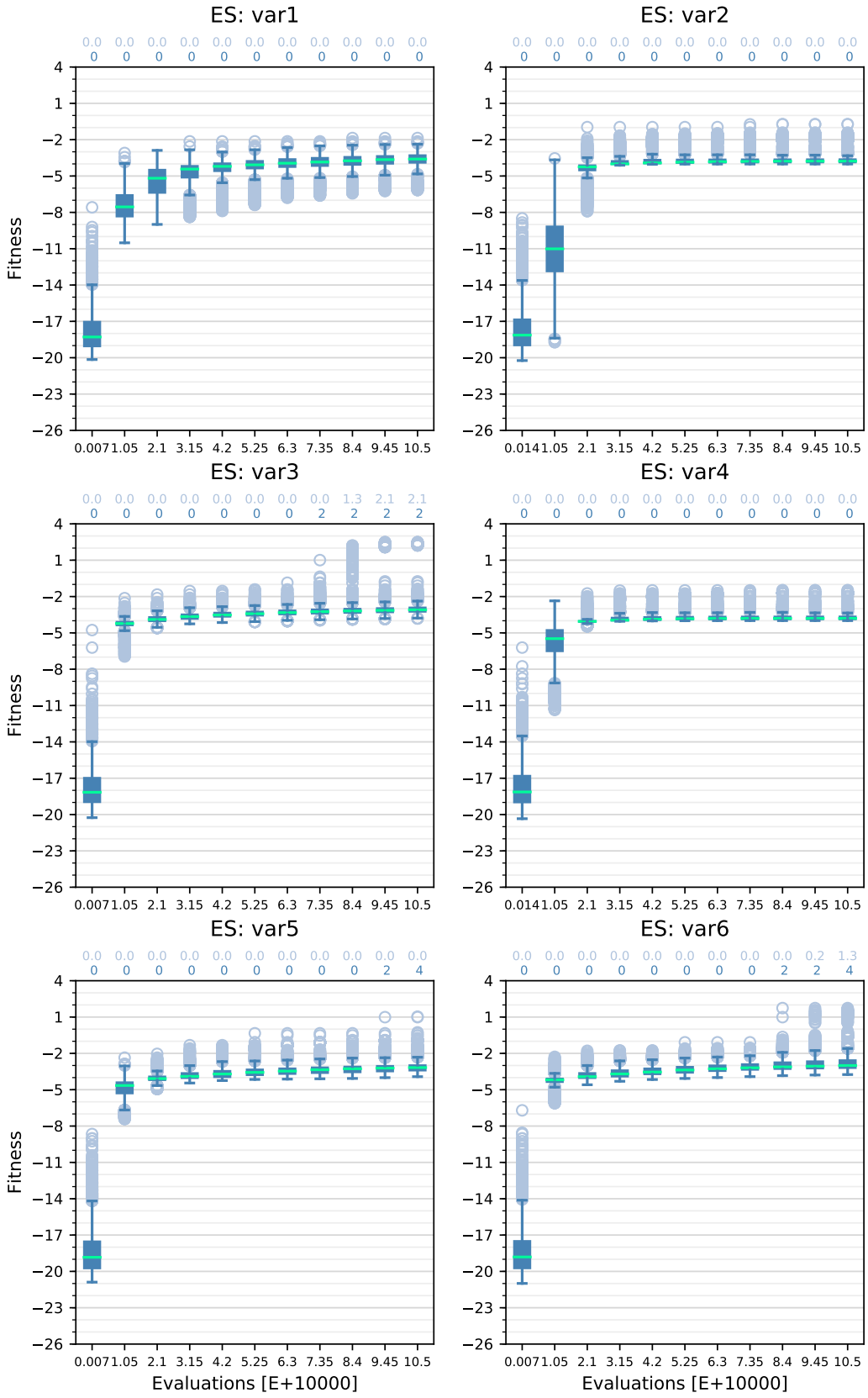
Ne všechny varianty DE (viz 6.14) jsou schopné nalézt řešení, konkrétně 3 varianty (polovina) nenalezne žádné řešení. Z daných variant vychází jako nejlepší jednoznačně varianta 6, která dokáže v 8% případů (v experimentu 1 90%) nalézt řešení, v čemž si vede nejlépe. A stejně tak si vede nejlépe z hlediska množství jedinců v populaci představující řešení, kde má 6,3% (v experimentu 1 63,6%). Mezi nejslabší varianty patří varianta 1, 2 a 4, které nedokáží najít řešení. Přičemž např. varianta 1 vykazuje oproti variantě 4 lepší konvergenci (stejně jako v experimentu 1). Je tak vidět, že varianta 6, která excelovala v experimentu 1 podává nejlepší výsledky i v této náročnější úloze. Podobně varianty 1, 2, 4, které v experimentu 1 propadly, jsou nejhoršími variantami i zde. Zdá se tedy, že v případě DE je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze.

Srovnání GA, ES, DE a celkové zhodnocení

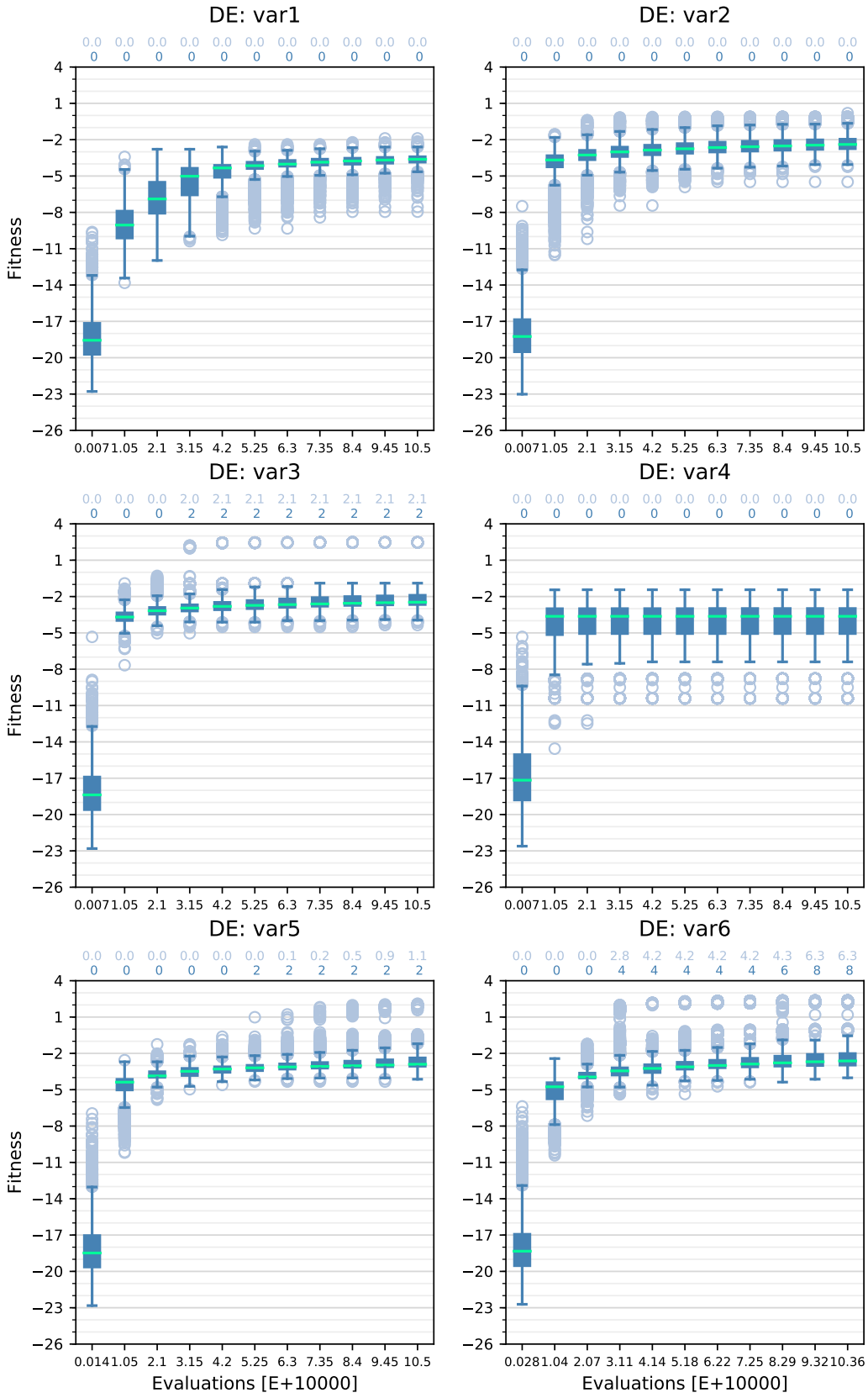
V tomto složitějším experimentu už algoritmy nedokáží vždy najít řešení. Z hlediska řešení fungují nejlépe GA, kde jen jedna varianta nedokázala najít řešení, zatímco u ES a DE měla problém polovina variant. Zajímavým zjištěním, je to, že algoritmy, které se ukázaly jako nejlepší při řešení jednodušší úlohy v experimentu 1 zůstávají nejlepší i při řešení náročnější úlohy v experimentu 2. A podobně algoritmy, které se neosvědčilY v jednodušší úloze, naprosto selhaly i ve složitější úloze. Zdá se tedy, že při hledání optimálního nastavení EA pro složitější úlohu stačí najít vhodné nastavení na jednodušší úloze a pak by mělo relativně fungovat i na složitější úloze. Vypadá to totiž tak, že i když jsou úlohy rozdílné, tak z pohledu EA zase ne tolik, aby bylo potřeba měnit parametry. Tímto se potvrdila správnost metodiky, kdy bylo nastavení EA odladěno na jednodušším experimentu a poté použito i na složitějších experimentech. Kdyby se na složitějších experimentech aplikovalo nějaké náhodné nastavení, je docela pravděpodobné, že by nebylo optimální a evoluce by nebyla schopna najít žádné řešení. Z hlediska výsledků tentokrát zvítězil GA, jehož varianta 4 dokáže najít řešení v 29% (v experimentu 1 92%) a obsahuje v populaci 3,1% jedinců (v experimentu 1 25,2%), kteří představují řešení.



Obrázek 6.12: Porovnání šesti variant GA na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 2.



Obrázek 6.13: Porovnání šesti variant ES na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 2.



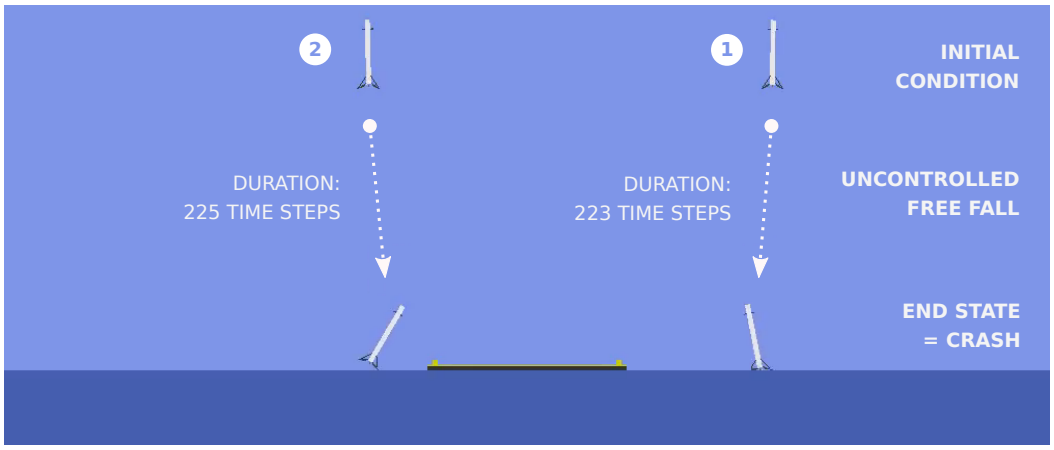
Obrázek 6.14: Porovnání šesti variant DE na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 2.

6.5.3 Experiment 3 - dvě různé počáteční podmínky, diskrétní simulace s omezeným řízením

Ve třetím experimentu je řešena úloha s přistáním rakety pro dvě různé počáteční podmínky a to v rámci diskrétní simulace s omezeným řízením. Počáteční podmínky jsou v porovnání s experimentem 1 více komplikovanější, především z hlediska vzdálenosti od cílového místa přistání. V porovnání s experimentem 2 jsou podmínky přibližně stejně náročné. Stejně jako u experimentu 2 je zde požadavek, aby raketa přistála pro obě dvě podmínky, což činí úlohu z pohledu počátečních podmínek srovnatelně náročnou s úlohou v rámci experimentu 2 a řádově náročnější než u experimentu 1. Parametry počáteční podmínky jsou přehledně shrnuty v tabulce 6.4. Ukázka dopadu rakety pro dané počáteční podmínky (pokud není raketa nijak ovládána) představuje obrázek 6.15.

Experiment 3 - Počáteční podmínky/stavy rakety		
Parametry / Poč. podmínky	**Podmínka 1**	**Podmínka 2**
Horizontální vzdálenost od středu přistávací plochy [m] kladná/záporná = raketa napravo/nalevo od středu	63,27	-40,79
Vertikální souřadnice / výška [m]	365,75	365,75
Úhel náklonu [rad] kladný/záporný = v protisměru / ve směru hodinových ručiček	-0,011	0,025
Lineární rychlost vertikální [m/s] kladná/záporná = nahoru/dolů	-79,9	-78,64
Lineární rychlost horizontální [m/s] kladná/záporná = napravo/nalevo	-1,4	1,28
Úhlová rychlost [rad/s] kladná/záporná = v protisměru / ve směru hodinových ručiček	0,053	-0,2

Tabulka 6.4: Počáteční podmínka/stav rakety v experimentu 3. V tomto stavu je spuštěna simulace.



Obrázek 6.15: Pro dané počáteční podmínky v rámci experimentu 3 vizualizujeme stav rakety na začátku simulace a na konci simulace (po neřízeném pádu). V rámci experimentu 3 chceme najít pomocí neuroevoluce neurokontrolér, který pro tyto dvě různé počáteční podmínky dovede raketu k úspěšnému přistání.

Na rozdíl od předchozích experimentů není tentokrát simulace spojitá, nýbrž diskrétní. To znamená, že v jeden čas může neurokontrolér provádět jen jednu akcí (řídit tah nebo vektorizovat tah motoru nebo používat postranní trysky) a nemůže řídit míru těchto akcí (protože míra je nastavena fixně). Toto by ještě nemuselo být nutně o řád náročnější, než je úloha z experimentu 2 (je to jen zkrátka jinak postavená úloha). Aby se náročnost zvedla o řád nahoru tak se omezí možnost řízení rakety jen na hlavní motor a nebude tedy možné používat postranní trysky. Toto samozřejmě řízení citelně zkomplikuje. Z pohledu obtížnosti se tak celkově jedná o nejvíce náročnější experiment.

V rámci vyhodnocení fitness bude potřeba (stejně jako u experimentu 2) spustit simulaci 2krát pro obě počáteční podmínky, což časově experiment prodraží přibližně 2krát. Fitness hodnota se počítá opět stejně jako u experimentu 2 – tedy jako průměrná kumulativní odměna na základě kumulativních odměn získaných ze dvou simulací pro 2 různé počáteční podmínky. Navíc pokud raketa v obou simulacích přistála, připočte se k průměrné kumulativní odměně jednička.

Srovnání variant GA

Ne všechny varianty GA (viz 6.16) jsou schopné nalézt řešení, konkrétně jedna z variant nenalezne žádné řešení. Z daných variant vychází objektivně jako nejlepší varianta 4, případně 1. Varianta 4 obsahuje v populaci 1,2% případů (v experimentu 2 3.1%), což je nejvíce. Varianta 1 je zase v 15% případů (v experimentu 2 27%) schopna nalézt řešení, což je nejvíce. Nejhorší varianta je zcela evidentně varianta 6, která není schopná najít žádné řešení a navíc to vypadá jakoby konvergovala k horším fitness hodnotám. Druhá nejhorší je varianta 3, která nalezne řešení v 6% případů (v experimentu 2 17%). Je tak vidět, že varianty 4, která excelovala v experimentu 1 a 2 podává nejlepší výsledky i v nejnáročnější úloze v rámci experimentu 3. Podobně varianty 3 a 6, které v experimentu 1 a 2 propadly, jsou nejhoršími variantami i zde. A pokud se budeme dívat pozorně všimneme si, že je to tak přibližně i s ostatními variantami. Zdá se tedy, že v případě GA je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze.

Srovnání variant ES

Ne všechny varianty ES (viz 6.17) jsou schopné nalézt řešení, konkrétně 3 varianty (polovina) nenalezne žádné řešení. Z daných variant vychází jako nejlepší jednoznačně varianta 6, která je v 6% případů (v experimentu 2 4%) schopna nalézt řešení a 4,9% jedinců v populaci (v experimentu 2 1,3%) představuje řešení. Z hlediska dosažených výsledků jsou jednoznačně nejhorší varianty 1, 2 a 4, které nejsou schopné nalézt žádné řešení. Je tak vidět, že varianta 6, která excelovala v experimentu 2 podává nejlepší výsledky i v této náročnější úloze (a dokonce i lepší výsledky). Podobně varianty 1, 2, 4, které v experimentu 1 propadly, jsou nejhoršími variantami i zde. A pokud se budeme dívat pozorně všimneme si, že je to tak přibližně i s ostatními variantami. Zdá se tedy, že v případě ES je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze. Velkým překvapením je pouze to, že v náročnější úloze si vede varianta 6 lépe než v teoreticky méně náročné úloze. Ostatní varianty ES si nicméně vedou hůře, tak jak by se čekalo. Zdá se však, že to s tou náročností nemusí být vždy tak jednoznačné. Protože může existovat algoritmus, který je vhodně přizpůsoben na řešení třeba právě náročné úlohy.

Srovnání variant DE

Ne všechny varianty DE (viz 6.18) jsou schopné nalézt řešení, konkrétně 4 varianty (dvě třetiny) nenaleznou žádné řešení. Z daných variant vychází jako nejlepší varianta 6, která dokáže v 4% případů (v experimentu 2 8%) nalézt řešení, v čemž si vede nejlépe. A stejně tak si vede nejlépe z hlediska množství jedinců v populaci představující řešení, kterých má 4,2% (v experimentu 2 6,3%). Mezi nejslabší varianty patří varianty 1, 2, 4 a 5, které nedokáží najít řešení. Přičemž např. varianta 1 vykazuje oproti variantě 4 lepší konvergenci (stejně jako v experimentu 2). Je tak vidět, že varianta 6, která excelovala v experimentu 1 a 2 podává nejlepší výsledky i v této náročnější úloze v rámci experimentu 3. Podobně varianty 1, 2, 4, které v experimentu 1 a 2 propadly, jsou nejhoršími variantami i zde. Zdá se tedy, že v případě DE je rozumné hledat vhodné nastavení/varianty v jednodušší úloze, protože pak budou fungovat přibližně odpovídajícím způsobem i v náročnější úloze.

Srovnání GA, ES, DE a celkové zhodnocení

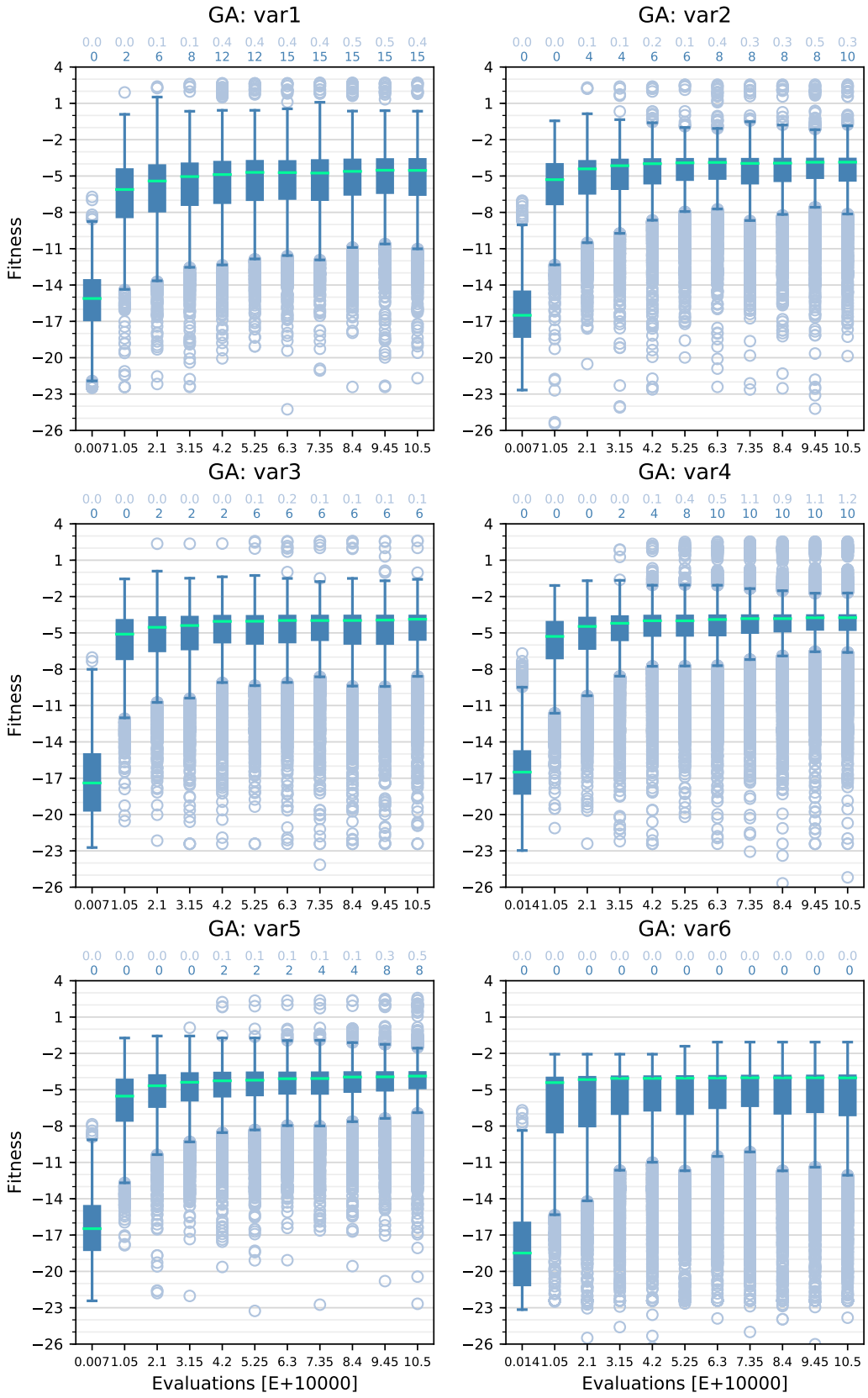
V tomto složitějším experimentu už algoritmy nedokáží vždy najít řešení. Z hlediska řešení fungují nejlépe GA, kde jen jedna varianta nedokázala najít řešení, zatímco u ES měla problém polovina a u DE dokonce dvě třetiny variant. Opět se navíc potvrdilo (stejně jako u experimentu 2), že algoritmy, které se ukázaly jako nejlepší při řešení jednodušší úlohy v experimentu 1 a 2, zůstávají nejlepší i při řešení náročnější úlohy v experimentu 3. A podobně algoritmy, které se neosvědčili v jednodušší úloze naprosto selhali i ve složitější úloze. Zdá se tedy, že při hledání optimálního nastavení EA pro složitější úlohu stačí najít vhodné nastavení na jednodušší úloze a pak by mělo relativně fungovat i na složitější úloze. Z hlediska výsledků v této úloze (i když ne úplně přesvědčivě) zvítězil algoritmus ES, jehož varianta 6 dokáže najít řešení v 6% (v experimentu 2 4%) a obsahuje v populaci 4,9% jedinců (v experimentu 2 1,3%), kteří představují řešení. To, že tato varianta řeší náročnější problém úspěšněji, než méně náročný, je velmi zajímavý objev.

6.6 Základní informace k programové realizaci, vývoji a testování

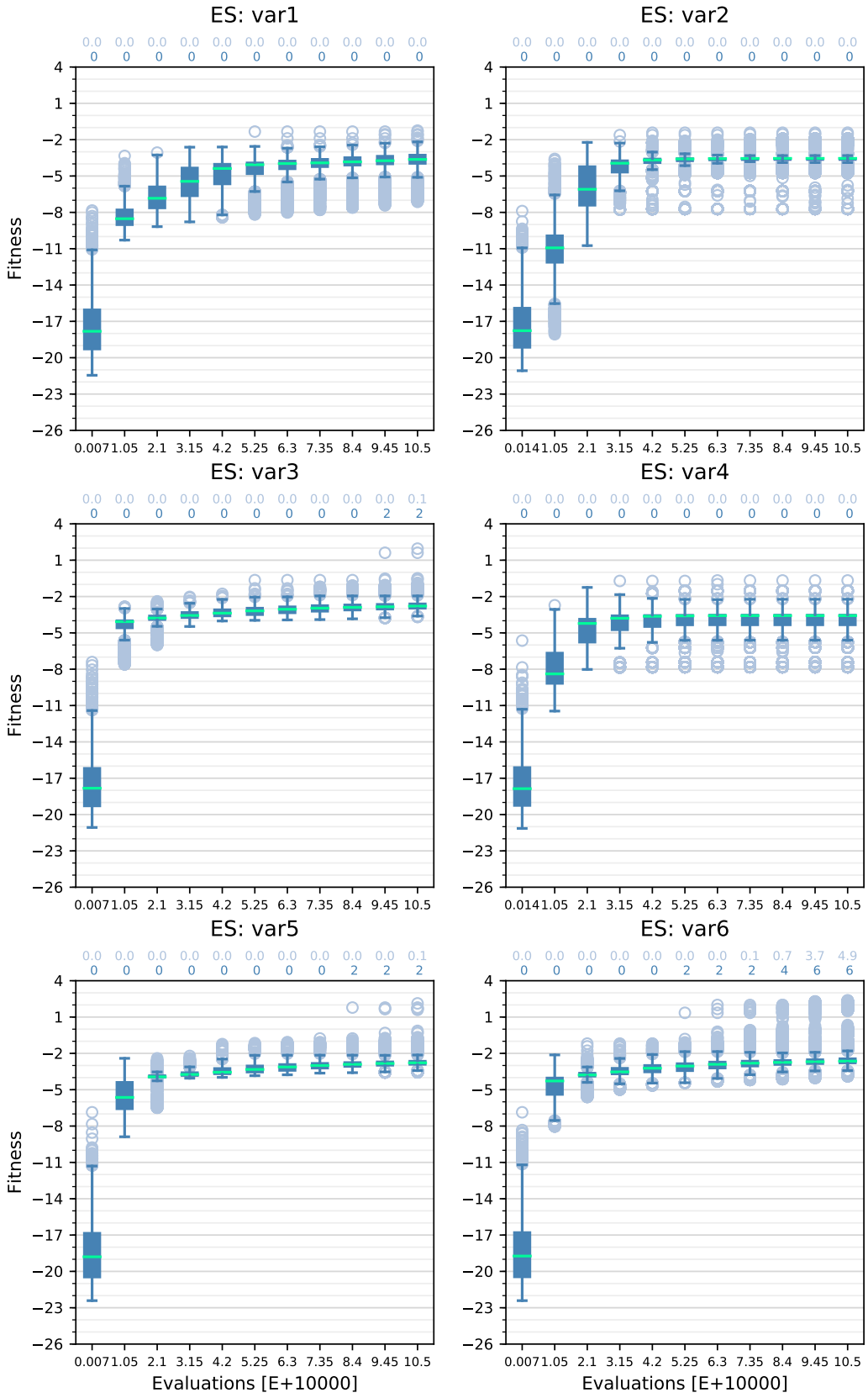
Experimenty byly vyvíjeny, spouštěny a testovány na superpočítači Salomon. V rámci experimentů bylo využito přibližně 90 tisíc jádrohodin. Experimenty jsou psány v Pythonu s využitím frameworku DEAP pro tvorbu evolučních algoritmů. Pro tvorbu NN se využívá framework Keras[4], který v pozadí využívá Tensorflow. Pro meziprocesorovou komunikaci je využita knihovna mpi4py která zpřístupňuje MPI pro Python.

Během evoluce se v daných intervalech (po daném počtu generací) zapisují statistiky evoluce do HDF5 souborů (pro danou generaci vznikne v HDF5 souboru zvláštní dataset) z kterých jsou také generovány grafy. Také se ukládají videa ze simulace, díky čemuž je možné sledovat postupné zlepšování. Pro dané videa se ukládají ve formátu HDF5 i váhy NN, takže je možné je později načíst do modelu NN a dále s takovým modelem experimentovat.

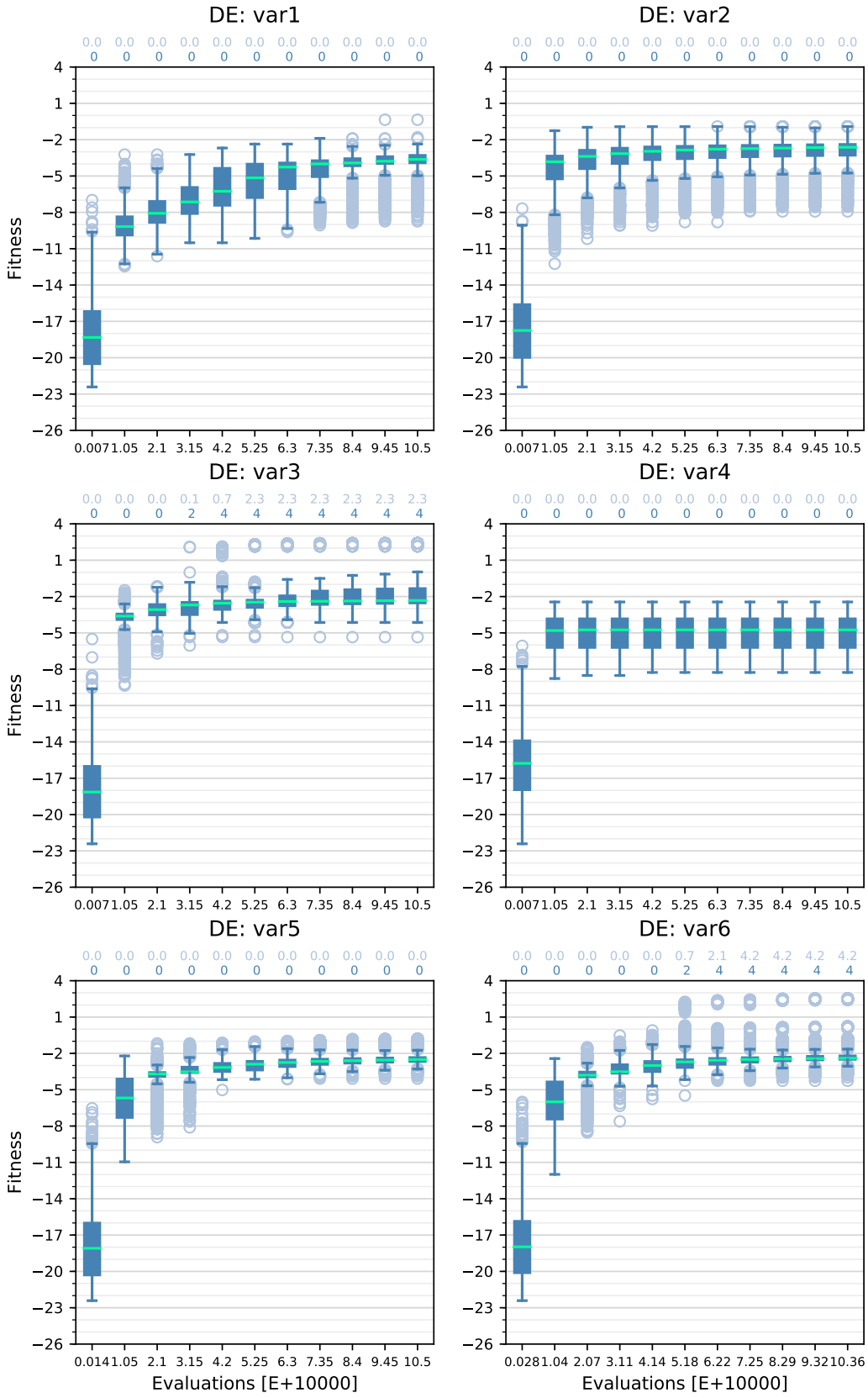
[4] https://keras.io/



Obrázek 6.16: Porovnání šesti variant GA na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 3.



Obrázek 6.17: Porovnání šesti variant ES na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 3.



Obrázek 6.18: Porovnání šesti variant DE na problému evoluční optimalizace vah neurokontroléru v rámci experimentu 3.

Kapitola 7

Závěr

Na teoretické rovině se práce zabývala evolučními výpočetními technikami, neuronovými sítěmi a následně konceptem a aplikací neuroevoluce, kdy se evoluční výpočetní techniky uplatňují při návrhu a učení neuronových sítí. Takto získané znalostí byly prerekvizitou k pochopení praktické části práce, která je jádrem diplomové práce a představuje hlavní přínos autora.

Praktická část měla za cíl ukázat uplatnění neuroevoluce na dvou typově odlišných úlohách. První úloha měla za cíl evoluční návrh architektur CNN, které by pro problém klasifikace ručně psaných číslic datasetu MNIST dosahovaly (po řádném natrénování) vysoké přesnosti. Cílem druhé úlohy pak bylo optimalizovat neurokontrolér (jeho váhy) pro úspěšné přistání 1. stupně rakety Falcon 9 v rámci 2D simulace, a to pro různé parametry simulace a počáteční podmínky.

Obě úlohy byly výpočetně velmi náročné a tak byly řešeny pomocí superpočítače. Celkově bylo pro řešení úloh spotřebováno přes 170 tisíc jádrohodin a to pro každou úlohu přibližně rovnoměrně (první úloha přibližně 80 tisíc, druhá úloha přibližně 90 tisíc). První úloha navíc pro svou enormní časovou náročnost vyžadovala paralelizaci na úrovni vyhodnocování fitness hodnot jedinců v populaci.

U první úlohy byl proveden jeden experiment, v rámci něhož byl omezen prohledávací prostor architektur CNN na takové, které nemají příliš velkou složitost. V tomto experimentu byly použity jako evoluční algoritmy GA a ES. Každý evoluční algoritmus byl ověřen v šesti různých nastaveních/variantách. Všechny varianty nalezly architektury, které měli při trénování na 5000 vzorcích (redukovaná trénovací množina používaná během evoluce při vyhodnocování fitness) přesnost klasifikace nad 93%. Nejlepší architektury pak i 94,19%. Když se potom nejlepší nalezené architektury (v rámci evoluce) natrénovaly (již mimo evoluci) řádně na celé (horních 55 000 vzorků) trénovací množině ve 100 cyklech, dosáhly všechny přesnost klasifikace více jak 99%. Nejlepší architektury pak dosáhli přesnost 99,49%, což je poměrně blízko nejlepší známé přesnosti klasifikace 99,79% pro MNIST dataset [11]. Neuroevoluce tak v dané úloze poměrně dobře uspěla a ukázalo se, že je možné v rámci evoluce pracovat jen s redukovaným počtem trénovacích vzorků, díky čemuž lze neuroevoluci v této oblasti vůbec aplikovat.

U druhé úlohy byly provedeny celkem tři experimenty, které se lišili počátečními podmínkami a parametry simulace (diskrétní, spojitá). V těchto experimentech byly použity jako evoluční algoritmy GA, ES a DE. Každý evoluční algoritmu byl ověřen v šesti různých nastaveních/variantách stejných pro všechny experimenty. Některé varianty pak byly schopné nalézt řešení (neurokontrolér schopný přistát s raketou pro dané počáteční podmínky) ve všech experimentech. V každém experimentu, ať již byl náročnější více nebo

méně neuroevoluce uspěla a dokázala najít neurokontrolér se kterým 1. stupeň Falcon 9 přistál pro dané počáteční podmínky. Z experimentů tedy vyplynulo, že je neuroevoluce schopna navrhnout kontrolér, který:

- Přistane jak pro jednu počáteční podmínku, tak dvě počáteční podmínky.
- Přistane jak ve spojité (v jeden simulační krok lze aplikovat více řídicích akcí najedou) tak i v diskrétní simulaci (v jeden simulační krok lze aplikovat jen jednu možnou řídicí akci).
- Přistane i když nemůže používat k vyrovnání postranní trysky.
- Vykazuje obecnější chování, pokud byl optimalizován na přistání pro 2 podmínky. To znamená, že pro jinou dostatečně odlišnou počáteční podmínku (na kterou nebyl optimalizován) sice nejspíš s raketou nepřistane, ale řízení rakety připomínající přistání tam bude s největší pravděpodbností vidět.

Neuroevoluce tak v obou úlohách uspěla, kdy zaprvé nalezla velmi kvalitní architektury CNN a zadruhé optimalizovala váhy neurokontroleru, tak že tento neurokontrolér dokázal s 1. stupněm Falcon 9 hladce přistát. V těchto ohledech se tak podařilo posunout vědu zase o něco dále. Zase víme něco více o schopnostech neuroevoluce a způsoby jak lze neuroevoluci aplikovat.

Na provedený základní výzkum by bylo možné dále mnoha způsoby navázat. U první úlohy připadá v úvahu aplikace principu evolučního návrhu architektury CNN na jiný problém než je dataset MNIST. Když se ukázalo, že pro menší problém neuroevoluce funguje, je možné pokročit k řešení o něco složitějšího a výpočetně jistě náročnějšího problému. Také by mohlo být přínosné dovolit evoluci nastavovat více parametrů architektury sítě. U druhé úlohy je nasnadě optimalizovat kontrolér pro 3 a více počátečních podmínek, případně v rámci experimentu nastavovat podmínky náhodně, za účelem nalezení obecného kontroléru schopného přistát pro lib. počáteční podmínku. Toto je nejspíše absolutní, ale v této chvíli z hlediska výpočetní náročnosti nejspíše nedosažitelná meta. Dále by mohlo být zajímavé zkoumat vliv složitosti architektury neurokontroléru na schopnost přistání. Možností, kam ve výzkumu pokračovat a směřovat je tedy celá řada, včetně např. toho, že by se místo 2D simulace ověřila schopnost neuroevoluce na 3D simulaci, což by ale bylo výpočetně mnohem náročnější. Také v tuto chvíli o žádné takové simulaci nevím, takže by ji bylo potřeba vytvořit jako součást případného dalšího výzkumu.

Literatura

[1] *Multi-Objective Genetic Algorithm (MOGA)* [online]. SAS IP, Inc. [cit. 2018-01-20]. Dostupné z: <https://www.sharcnet.ca/Software/Ansys/15.0.7/en-us/help/wb_dx/dxBEMtemp11.html>.

[2] *CRS-8 Launch and Landing* [online]. Space Exploration Technologies Corp., April 08, 2016 [cit. 2018-05-22]. Dostupné z: <http://www.spacex.com/news/2016/04/09/crs-8-launch-and-landing>.

[3] *Convolutional Neural Network* [online]. The MathWorks, Inc., ©1994-2018 [cit. 2018-01-17]. Dostupné z: <https://www.mathworks.com/discovery/convolutional-neural-network.html>.

[4] *DRAGON* [online]. SPACE EXPLORATION TECHNOLOGIES CORP, ©2017 [cit. 2018-04-27]. Dostupné z: <http://www.spacex.com/dragon>.

[5] *FALCON 9* [online]. SPACE EXPLORATION TECHNOLOGIES CORP, ©2017 [cit. 2018-04-30]. Dostupné z: <http://www.spacex.com/falcon9>.

[6] *FALCON 9 FIRST STAGE AFTER LANDING ON DRONESHIP "OF COURSE I STILL LOVE YOU"* [online]. SPACE EXPLORATION TECHNOLOGIES CORP, ©2017 [cit. 2018-04-27]. Dostupné z: <http://www.spacex.com/gallery/2016-0>.

[7] *FALCON 9 FIRST STAGE LANDING ON DRONESHIP "OF COURSE I STILL LOVE YOU"* [online]. SPACE EXPLORATION TECHNOLOGIES CORP, ©2017 [cit. 2018-04-27]. Dostupné z: <http://www.spacex.com/gallery/2016-0>.

[8] *LunarLander-v2* [online]. OpenAI, ©2018 [cit. 2018-05-22]. Dostupné z: <https://gym.openai.com/envs/LunarLander-v2/>.

[9] *THE WHY AND HOW OF LANDING ROCKETS* [online]. SPACE EXPLORATION TECHNOLOGIES CORP, JUNE 25, 2015 [cit. 2018-04-30]. Dostupné z: <http://www.spacex.com/news/2015/06/24/why-and-how-landing-rockets>.

[10] Ali, A. F.; Tawhid, M. A. : A hybrid particle swarm optimization and genetic algorithm with population partitioning for large scale optimization problems. *Ain Shams Engineering Journal*, roč. 8, č. 2, 2017: s 191 – 206, ISSN 2090-4479, https://doi.org/10.1016/j.asej.2016.07.008. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S2090447916301101>

[11] Benenson, R. *What is the class of this image ?* [online]. Last updated on 2016-02-22 [cit. 2018-05-22]. Dostupné z: <http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#4d4e495354>.

[12] Bidlo, M. : Aplikované evoluční algoritmy: Základní pojmy a techniky EA. [cit. 2018-05-20]. Dostupné z: <https://wis.fit.vutbr.cz/FIT/st/course-files-st.php.cs?file=%2Fcourse%2FEVO-IT%2Flectures%2F02-TechnikyEA.pdf&cid=11382>

[13] Brabazon, A.; O'Neill, M.; McGarraghy, S. *Natural Computing Algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN 3662436302, 9783662436301.

[14] Brockman, G.; Cheung, V.; Pettersson, L.; aj. : OpenAI Gym. 2016, arXiv:1606.01540. Dostupné z: <https://arxiv.org/pdf/1606.01540>

[15] Brownlee, J. *Dropout Regularization in Deep Learning Models With Keras* [online]. Machine Learning Mastery, June 20, 2016 [cit. 2018-05-22]. Dostupné z: <https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/>.

[16] Cipollini, B. *DEEP NEURAL NETWORKS HELP US READ YOUR MIND* [online]. NeuWrite San Diego, 2015 [cit. 2018-01-18]. Dostupné z: <https://neuwritesd.org/2015/10/22/deep-neural-networks-help-us-read-your-mind/>.

[17] CodeReclaimers. *algorithm on LunarLander-v2* [online]. OpenAI, ©2018 [cit. 2018-05-22]. Dostupné z: <https://gym.openai.com/evaluations/eval_FbKq5MxAS9GlvB7W6ioJkg/>.

[18] Desell, T. : Large Scale Evolution of Convolutional Neural Networks Using Volunteer Computing. *CoRR*, roč. abs/1703.05422, 2017, 1703.05422. Dostupné z: <http://arxiv.org/abs/1703.05422>

[19] Deshpande, A. *The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3)* [online]. Adit Deshpande, August 24, 2016 [cit. 2018-01-19]. Dostupné z: <https://adeshpande3.github.io/adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html>.

[20] Doncieux, S.; Bredeche, N.; Mouret, J.-B.; aj. : Evolutionary Robotics: What, Why, and Where to. *Frontiers in Robotics and AI*, roč. 2, 2015: str. 4, ISSN 2296-9144, 10.3389/frobt.2015.00004. Dostupné z: <https://www.frontiersin.org/article/10.3389/frobt.2015.00004>

[21] Eiben, A.; Smith, J. *Introduction to Evolutionary Computing*. Second edition. Natural Computing Series, Springer-Verlag Berlin Heidelberg, 2015. Dostupné z: <https://doi.org/10.1007/978-3-662-44874-8>. ISBN 978-3-662-44873-1, 978-3-662-44874-8, 10.1007/978-3-662-44874-8.

[22] Elbeltagi, E.; Hegazy, T.; Grierson, D. : Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, roč. 19, č. 1, 2005: s 43 – 53, ISSN 1474-0346, https://doi.org/10.1016/j.aei.2005.01.004. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S1474034605000091>

[23] Gomez, F. J.; Miikkulainen, R. Active Guidance for a Finless Rocket Using Neuroevolution. 2003. Dostupné z: <http://nn.cs.utexas.edu/?gomez:gecco03>.

[24] Graupe, D. *Principles of Artificial Neural Networks*. 2nd. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2007. ISBN 9812706240.

[25] Hartley, T. *When Parallelism Gets Tricky: Accelerating Floyd-Steinberg on the Mali GPU* [online]. Arm Limited, ©1995-2018 [cit. 2018-01-18]. Dostupné z: <https://community.arm.com/graphics/b/blog/posts/when-parallelism-gets-tricky-accelerating-floyd-steinberg-on-the-mali-gpu>.

[26] Heidrich-Meisner, V.; Igel, C. : Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms*, roč. 64, č. 4, 2009: s 152 – 168, ISSN 0196-6774, https://doi.org/10.1016/j.jalgor.2009.04.002, special Issue: Reinforcement Learning. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0196677409000364>

[27] Hijazi, S. L.; Kumar, R. Using Convolutional Neural Networks for Image Recognition. 2015. Dostupné z: <https://ip.cadence.com/uploads/901/cnn_wp-pdf>.

[28] Hodges, A. *Alan Turing: Creator of modern computing* [online]. BBC, ©2017 [cit. 2017-11-03]. Dostupné z: <http://www.bbc.co.uk/timelines/z8bgr82>.

[29] game design initiative : Physics Engines. [cit. 2018-05-22]. Dostupné z: <https://www.cs.cornell.edu/courses/cs3152/2014sp/lectures/19-PhysicsEngines.pdf>

[30] Jadon, S. *Introduction to Different Activation Functions for Deep Learning* [online]. A Medium Corporation, March 2016 [cit. 2018-05-22]. Dostupné z: <https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deep-learning-9689331ba092>.

[31] Juliani, A. *Simple Reinforcement Learning with Tensorflow Part 6: Partial Observability and Deep Recurrent Q-Networks* [online]. Medium Corporation, Oct 7, 2016 [cit. 2018-05-22]. Dostupné z: <https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-6-partial-observability-and-deep-recurrent-q-68463e9aeefc>.

[32] Karn, U. *An Intuitive Explanation of Convolutional Neural Networks* [online]. the data science blog, August 11, 2016 [cit. 2018-01-17]. Dostupné z: <https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/>.

[33] Koppejan, R.; Whiteson, S. : Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary Intelligence*, roč. 4, č. 4, Dec 2011: s 219–241, ISSN 1864-5917, 10.1007/s12065-011-0066-z. Dostupné z: <https://doi.org/10.1007/s12065-011-0066-z>

[34] Krasnogor, N. *Memetic Algorithms*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. Dostupné z: <https://doi.org/10.1007/978-3-540-92910-9_29>. ISBN 978-3-540-92910-9, 10.1007/978-3-540-92910-9_29.

[35] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. 2012. Dostupné z: <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>.

[36] LeCun, Y.; Cortes, C.; Burges, C. J. *THE MNIST DATABASE of handwritten digits* [online]. [cit. 2018-01-19]. Dostupné z: <http://yann.lecun.com/exdb/mnist/>.

[37] Lehman, J.; Miikkulainen, R. : Neuroevolution. *Scholarpedia*, roč. 8, č. 6, 2013: str. 30977, 10.4249/scholarpedia.30977, revision #133684. Dostupné z: <http://www.scholarpedia.org/article/Neuroevolution>

[38] Mach, M. *Evolučné algoritmy : Prvky a princípy*. 1st. Edícia vedeckých spisov Fakulty elektrotechniky a informatiky TU Košice, 2009. ISBN 978-80-8086-123-0.

[39] Maltarollo, V.; Honório, K.; Silva, A. : Applications of Artificial Neural Networks in Chemical Problems. In *Artificial Neural Networks - Architectures and Applications*, editace K. Suzuki, kapitola 10, Rijeka: InTech, 2013, 10.5772/51275. Dostupné z: <http://dx.doi.org/10.5772/51275>

[40] McGlynn, T. : How Does Social Behavior Evolve. *Nature Education Knowledge*, roč. 3, č. 10, 01 2010: str. 69. Dostupné z: <https://www.nature.com/scitable/knowledge/library/how-does-social-behavior-evolve-13260245>

[41] Melechin, P. *Vše o autonomních přistávacích plošinách ASDS* [online]. ElonX [cit. 2018-05-22]. Dostupné z: <http://www.elonx.cz/autonomni-pristavaci-plosiny/>.

[42] Melechin, P. *Proč SpaceX někdy přistává na moři a jindy na pevnině?* [online]. ElonX, Aktualizováno 12. 4. 2018 [cit. 2018-05-22]. Dostupné z: <http://www.elonx.cz/proc-spacex-nekdy-pristava-na-mori-jindy-na-pevnine>.

[43] Mier, P. R. *yabox/README.md* [online]. GitHub, Inc., 10.8. 2017 [cit. 2017-12-19]. Dostupné z: <https://github.com/pablormier/yabox/blob/master/README.md>.

[44] Nelson, A. L.; Barlow, G. J.; Doitsidis, L. : Fitness functions in evolutionary robotics: A survey and analysis. *Robotics and Autonomous Systems*, roč. 57, č. 4, 2009: s 345 – 370, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2008.09.009. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0921889008001450>

[45] Niederberger, S. *OpenAI Gym with custom SpaceX Rocket Lander environment for box2d* [online]. ©2018 [cit. 2018-05-22]. Dostupné z: <https://github.com/EmbersArc/gym>.

[46] Oplatková, Z.; Ošmera, P.; Šeda, M.; aj. *Evoluční výpočetní techniky - principy a aplikace*. 1st. BEN - technická literatura, 2008. ISBN 80-7300-218-3.

[47] Pretorius, C. J.; du Plessis, M. C.; Gonsalves, J. W. : Neuroevolution of Inverted Pendulum Control: A Comparative Study of Simulation Techniques. *Journal of Intelligent & Robotic Systems*, roč. 86, č. 3, Jun 2017: s 419–445, 10.1007/s10846-017-0465-1. Dostupné z: <https://doi.org/10.1007/s10846-017-0465-1>

[48] Rein, D. *Convolutional Neural Networks for Image and Video Processing : Introduction* [online]. 09.Februar 2017 [cit. 2017-21-11]. Dostupné z: <https://wiki.tum.de/display/lfdv/Introduction>.

[49] Risi, S.; Togelius, J. : Neuroevolution in Games: State of the Art and Open Challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, roč. 9, č. 1, March 2017: s 25–41, ISSN 1943-068X, 10.1109/TCIAIG.2015.2494596.

[50] Ross, J. *SpaceX Falcon 9* [online]. Jon Ross., ©2018 [cit. 2018-05-01]. Dostupné z: <https://zlsadesign.com/infographic/vehicle/spacex-falcon9-control/>.

[51] Ross, J. *SpaceX rockentry nomenclature* [online]. Jon Ross., ©2018 [cit. 2018-05-01]. Dostupné z: <https://zlsadesign.com/infographic/trajectory/spacex-falcon9-booster-hoverslam/>.

[52] Rozenberg, G.; Bck, T.; Kok, J. N. *Handbook of Natural Computing*. 1st. Springer Berlin Heidelberg, 2012. Dostupné z: <https://doi.org/10.1007/978-3-540-92910-9>. ISBN 978-3-540-92909-3, 978-3-540-92910-9, 10.1007/978-3-540-92910-9.

[53] Saitta, S. *Standardization vs. normalization* [online]. July 10, 2007 [cit. 2018-05-22]. Dostupné z: <http://www.dataminingblog.com/standardization-vs-normalization/>.

[54] Sharma, M.; Chandra, S. Application of Artificial Bee Colony Algorithm for numerical optimization technique. June 2015. 10.1109/IADCC.2015.7154905.

[55] Sher, G. I. *Handbook of Neuroevolution Through Erlang*. 1st. New York: Springer, New York, NY, 2013. Dostupné z: <https://doi.org/10.1007/978-1-4614-4463-3>. ISBN 978-1-4614-4463-3, 978-1-4614-4462-6, https://doi.org/10.1007/978-1-4614-4463-3.

[56] Simon, D. *Evolutionary Optimization Algorithms : Biologically-Inspired and Population-Based Approaches to Computer Intelligence*. First edition. John Wiley & Sons, Inc., 2013. ISBN 978-0-470-93741-9.

[57] Surya, M. *Neuroevolution for Racing Game AI* [online]. YouTube, 14. 2. 2016 [cit. 2018-01-19]. Dostupné z: <https://youtu.be/_1TOKKgAock>.

[58] Talbi, E. *Metaheuristics: From Design to Implementation*. First edition. Hoboken, New Jersey: John Wiley & Sons, Inc., 2009. ISBN I978-0-470-27858-1.

[59] Toufar, P. *Až vyrazí SpaceX do vesmíru s posádkou na novém Falconu 9* [online]. 13. května 2018 [cit. 2018-05-22]. Dostupné z: <https://technet.idnes.cz/spacex-elon-musk-falcon-9-block-5-crew-dragon-fki-/tec_vesmir.aspx?c=A180511_223231_tec_vesmir_kuz>.

[60] Vasant, G.-W. W., Pandian; Dieu, V. N. *Handbook of Research on Modern Optimization Algorithms and Applications in Engineering and Economics*. 1st. Advances in computational intelligence and robotics (ACIR) book series, Hershey, PA, USA: IGI Global, 2016. ISBN 1466696443, 9781466696440.

[61] Yang, X. : Metaheuristic Optimization. *Scholarpedia*, roč. 6, č. 8, 2011: str. 11472, 10.4249/scholarpedia.11472, revision #91488.

[62] You, Z.; Pu, Y. : The Genetic Convolutional Neural Network Model Based on Random Sample. roč. 8, 11 2015: s 317–326.

[63] Yuan, Y.; Zhu, Y. A hybrid artificial bee colony optimization algorithm. Aug 2014. ISSN 2157-9555, 10.1109/ICNC.2014.6975884.

[64] Zhang, J.; Chen, W.-N.; Zhan, Z.-H.; aj. : A survey on algorithm adaptation in evolutionary computation. *Frontiers of Electrical and Electronic Engineering*, roč. 7, č. 1, Mar 2012: s 16–31, ISSN 1673-3584, 10.1007/s11460-012-0192-0. Dostupné z: <https://doi.org/10.1007/s11460-012-0192-0>

Příloha A

Obsah přiloženého paměťového média

Na paměťovém médiu, které je dodávané s prací je následující

- **Elektronická verze práce ve formátu pdf** - V kořenovém adresáři pojmenovaná jako *dip.pdf*
- **Zdrojový text práce v LaTeXu** - V kořenovém adresáři pojmenován jako *dip.zip*
- **Dokumentace k programům, které byly použity k provedení experimentů s CNN a raketou** - V kořenovém adresáři pojmenovaná jako *doc.pdf*
- **Zdrojové kódy programu, který byl použit k provedení experimentů s CNN** - V adresáři */src/CNN-mnist/*.
- **Zdrojové kódy programu, který byl použit k provedení experimentů s raketou** - V adresáři */src/falcon-landing/*.
- **Výstupy provedených experimentů a popis experimentů** - V adresáři */Experiments/*.
- **Některé potřebné knihovny a datové soubory** - V adresáři */support/*. Jedná se např. o dataset MNIST, nebo použitý simulátor.

Příloha B

Oficiální webová stránka projektu

Oficiální webová stránka pro tento projekt (diplomovou práci) se nachází na adrese: https://bitbucket.org/herech/dp-neuroevolution.
Skrze tuto stránku může v budoucnu proběhnout zveřejnění např. některých podpůrných materiálů, které jsou dostupné na paměťovém médiu (zdrojové kódy, zajímavé výsledky apod.) a tedy pro širší veřejnost jinak obtížně dostupné.