VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

KOEVOLUČNÍ ALGORITMY A KLASIFIKACE

COEVOLUTIONARY ALGORITHMS AND CLASSIFICATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MARTIN HURTA**
AUTHOR

VEDOUCÍ PRÁCE **Ing. MICHAELA DRAHOŠOVÁ, Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2020/2021

Zadání diplomové práce

Student: **Hurta Martin, Bc.**
Program: Informační technologie a umělá inteligence Specializace: Inteligentní systémy
Název: **Koevoluční algoritmy a klasifikace**
Coevolutionary Algorithms and Classification
Kategorie: Umělá inteligence
Zadání:

1. Seznamte se s problematikou genetického programování, koevolučních algoritmů a evolučního návrhu klasifikátorů.
2. Navrhněte program umožňující provádět evoluční návrh klasifikátoru pomocí genetického programování s využitím koevoluce.
3. Program z bodu 2 implementujte a ověřte jeho funkčnost na zadaných úlohách.
4. Porovnejte dosažené výsledky s výsledky jiných přístupů.
5. Zhodnoťte dosažené výsledky.

Literatura:

- ŠIKULOVÁ Michaela a SEKANINA Lukáš. Acceleration of Evolutionary Image Filter Design Using Coevolution in Cartesian GP. *Lecture Notes in Computer Science*. 2012, roč. 2012, č. 7491, s. 163-172. ISBN 978-3-642-32936-4. ISSN 0302-9743.
- Dle pokynů vedoucí práce.

Při obhajobě semestrální části projektu je požadováno:

- Splnění bodů 1 a 2 zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Drahošová Michaela, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 19. května 2021
Datum schválení: 30. října 2020

Abstrakt

Cílem této práce je automatizovaný návrh programu pro detekci projevů dyskineze z pohybových dat pacientů. K návrhu programu je využito kartézské genetické programování, které bylo z důvodu urychlení procesu návrhu doplněno o koevoluci prediktorů fitness s proměnlivou velikostí, která umožňuje vyhodnocení kvality kandidátních řešení na pouhé části trénovacích dat. Vzniklé řešení dosahuje srovnatelné schopnosti rozlišení mezi třídami (AUC) s existujícím řešením při dosažení v průměru trojnásobného zrychlení procesu návrhu oproti variantě bez prediktorů fitness. Experimenty s metodami křížení prediktorů neukázaly významný rozdíl mezi zvolenými metodami. Zajímavých výsledků však bylo dosaženo při experimentech s celočíselnými datovými typy vhodnými pro implementaci v hardwaru, kdy u datového typu o osmi bitech bez znaménka (uint8_t) bylo dosaženo nejenom srovnatelné schopnosti rozlišení mezi třídami (pro významné projevy dyskineze AUC = 0,93 shodně jako pro existující řešení) a zlepšení rozlišovací schopností u chodících pacientů (AUC = 0,80 oproti AUC = 0,73 u existujícího řešení), ale navíc v průměru téměř devítinásobného zrychlení návrhu oproti variantě bez prediktorů fitness využívající datový typ float.

Abstract

The aim of this work is to automatically design a program that is able to detect dyskinetic movement features in the measured patient's movement data. The program will be developed using Cartesian genetic programming equipped with coevolution of fitness predictors. This type of coevolution allows to speed up a design performed by Cartesian genetic programming by evaluating a quality of candidate solutions using only a part of training data. Evolved classifier achieves a performance (in terms of AUC) that is comparable with the existing solution while achieving threefold acceleration of the learning process compared to the variant without the fitness predictors, in average. Experiments with crossover methods for fitness predictors haven't shown a significant difference between investigated methods. However, interesting results were obtained while investigating integer data types that are more suitable for implementation in hardware. Using an unsigned eight-bit data type (uint8_t) we've achieved not only comparable classification performance (for significant dyskinesia AUC = 0.93 the same as for the existing solutions), with improved AUC for walking patient's data (AUC = 0.80, while existing solutions AUC = 0.73), but also nine times speedup of the design process compared to the approach without fitness predictors employing the float data type, in average.

Klíčová slova

strojové učení, klasifikace, evoluční algoritmy, genetické algoritmy, genetické programování, kartézské genetické programování, koevoluční algoritmy, prediktor s proměnlivou velikostí, dyskineze

Keywords

machine learning, classification, evolutionary algorithm, genetic algorithm, genetic programming, cartesian genetic programming, coevolutionary algorithm, adaptive fitness predictor, dyskinesia

Citace

HURTA, Martin. *Koevoluční algoritmy a klasifikace*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Michaela Drahošová, Ph.D.

Koevoluční algoritmy a klasifikace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením paní Ing. Michaely Drahošové, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Martin Hurta
17. května 2021

Poděkování

Rád bych poděkoval Ing. Michaele Drahošové, Ph.D. za velkou pomoc při vedení mé práce, cenné rady a věcné připomínky, které mi pomohly při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Pozorováním přírody získáváme přístup k té největší sadě řešení problémů světa. Všechna jsou zcela volně dostupná a většina prošla v průběhu milionů let řadou generací vývoje. Je tedy logické, že oboru informačních technologií není inspirace přírodou cizí a řešení logistiky podle vzoru mravenčích kolonií nebo umělá inteligence řízená neuronovými sítěmi podobnými těm v naší hlavě jsou běžnými a úspěšně používanými metodami.

Právě neuronové sítě jsou dnes velmi oblíbenou technikou pro implementaci umělé inteligence a strojového učení včetně řešení problému klasifikace obrazu, textu nebo pohybu, kterého se týká tato práce. Jejich nevýhodou je ale výpočetní náročnost, která znesnadňuje jejich využití ve stále se zvětšující oblasti nízkoenergetických a vestavěných zařízení, a právě zde nastupuje opět přírodou inspirované genetické programování, které mnohdy umožňuje nalezení kompaktnějších řešení a tedy i energeticky a paměťově méně náročných, a otevírá bránu jednoduchým a zároveň inteligentním zařízením [9].

Příroda s sebou ale nepřináší jen to dobré a zde leží problém řešený touto prací – detekce dyskineze u pacientů s Parkinsonovou nemocí. Parkinsonova nemoc je úzce spojena s poruchou tvorby neuropřenašeče dopaminu, což vede u pacientů ke zpomalení pohybů, ztuhlosti, třesu či ke kompletnímu omezení pohybových schopností. Těmto projevům nemoci je bráněno pomocí umělého nahrazení dopaminu, kdy však při vyšším dávkování dochází k dyskinezi, která se projevuje nedobrovolnými trhavými pohyby a často až agresivními svalovými křečemi. Detekce těchto projevů je tedy klíčová pro správné dávkování léků. Tato práce se proto zabývá automatizovaným návrhem co nejjednodušší a energeticky úsporné detekce dyskineze. Návrh je prováděn za pomocí kartézského genetického programování (CGP) z dat získaných ze šesti pohybových senzorů umístěných na těle pacienta [10].

Hlavním problémem použití CGP je náročnost získání onoho jednoduchého řešení, které spočívá především v nutnosti vyhodnocení kvality řady generací populace kandidátních programů. Proto tato práce přichází s využitím koevoluce kandidátních řešení programů a prediktorů jejich fitness hodnoty a tím urychlení celého procesu.

Následující kapitola 2 obsahuje obecný popis strojového učení a vyhodnocení kvality klasifikátoru. Další kapitola 3 se již zaměřuje na evoluční algoritmy a jejich podmnožinu genetických algoritmů. Jejich důležitá část genetické programování je popsána v kapitole 4, po které je vysvětlen princip koevolučních algoritmů v kapitole 5. Data využitá při řešení úlohy jsou představena v kapitole 6 společně se zhodnocením aktuálního stavu jejich klasifikace. Kapitola 7 se zabývá návrhem klasifikátoru a principem jeho funkčnosti. Popis implementace řešení je obsažen v kapitole 8 a je následovaný ověřením funkčnosti a popisem experimentů a jejich vyhodnocením v kapitole 9. Poslední kapitola 10 shrnuje práci společně s dosaženými výsledky a navrhuje možnosti dalšího pokračování v práci.

Kapitola 2

Strojové učení a klasifikace

Strojové učení je rozsáhlým oborem umělé inteligence, který má za úkol nalezení skrytých vzorů v trénovacích datech libovolného původu a následovné využití těchto schopností při řešení problému predikce či například klasifikace dat nových. Tohoto je možné dosáhnou díky tomu, že téměř všechny kompletně nenáhodná data jistý vzor obsahují a právě pro ten je algoritmem vytvořen model.

Klasifikace, která je náplní této práce, je problémem, kdy odezvou modelu pro libovolný vstup je kvalitativní informace o třídě, do které určitý vstup patří [8]. Řešený problém může být binárního typu, kdy výstup klasifikátoru udává pravděpodobnost příslušnosti ke klasifikované třídě nebo problémem více tříd, kdy zpravidla pro každou třídu existuje jedna výstupní hodnota představující míru příslušnosti do dané třídy a výsledek klasifikace je určen podle nejvyšší z těchto hodnot. Evoluční techniky použitelné k návrhu takového klasifikátoru jsou popsány v následujících kapitolách.

Tato kapitola slouží pouze k jednoduchému vysvětlení základů strojového učení v souvislosti s klasifikací, společně se základním rozdělením algoritmů podle typu procesu učení a následovným popsáním vyhodnocení kvality vzniklého modelu včetně vysvětlení ROC křivky (Receiver Operating Characteristic, operační charakteristiky přijímače) a související hodnoty AUC (Area Under Curve, plochou pod křivkou). Pokud není uvedeno jinak, všechny informace v této kapitole pocházejí z Burkov [2] a Segaran [13].

2.1 Dělení modelů podle druhu učení

Existuje celá řada algoritmů strojového učení, které mají různé vhodné případy použití a jednoduchost výsledného řešení od jednoduchých rozhodovacích stromů po neuronové sítě, jejichž rozhodovací proces je zcela skryt. Všechny tyto algoritmy lze dělit podle velké řady kritérií nebo použití. Hlavním je ale způsob jejich učení, který lze rozdělit do čtyř základních skupin.

Učení s učitelem

Při učení s učitelem má algoritmus k dispozici trénovací data, která jsou sestavena z dvojic vstupních objektů a cílových výstupů. Úkolem algoritmu je na základě těchto dat vytvořit model, který je schopen pro libovolný vstupní vektor predikovat jeho třídu.

Učení bez učitele

Učení bez učitele se od předchozího typu liší v absenci informace o třídách jednotlivých trénovacích vektorů příznaků. Vytvořený model pak podle zamýšleného účelu může data například přidělovat do klastrů dat s podobnými vlastnostmi, zmenšovat dimenzi odstraněním z některých příznaků vektoru nebo detekovat vzorky příliš odlišné od vzorků průměrných.

Kombinace učení s učitelem a bez učitele

Učení částečně s učitelem je kombinací předchozích dvou typů, kdy je označena pouze malá část vektorů příznaků. Úkol tohoto algoritmu je stejný jako v případě učení s učitelem, ale za značeného snížení nákladů, které často obnáší označování trénovacích dat.

Zpětnovazebné učení

Při zpětnovazebním učení je algoritmus součástí prostředí jehož stav je vstupním vektorem. Algoritmus provádí akce, které ovlivňují prostředí a přinášejí různou míru odměny. Cílem algoritmu je se naučit funkci převádějící vektor stavu prostředí na akci, která maximalizuje předpokládanou průměrnou odměnu.

2.2 Vyhodnocení schopnosti klasifikátoru rozlišit mezi třídami

Po vytvoření modelu je potřeba ověření jeho kvality. Za tímto účelem je datová sada rozdělena na dvě části například v poměru 9:1. Větší část se nazývá trénovací sadou a probíhá na ní proces učení modelu. Druhá menší část, pak slouží k ověření schopnosti modelu generalizovat, tedy pracovat korektně s dosud neviděnými daty.

U klasifikačních modelů, kterým se zabývá i tato práce, je však možné získat i mnohem přesnější údaje než pouhou schopnost generalizace a ty jsou uvedeny dále.

Matice záměn

Matice záměn je tabulkou zobrazující schopnost modelu predikovat prvky jednotlivých tříd. Osa y popisuje skutečné třídy a osa x třídu predikovanou modelem. Tabulka 2.1 zobrazuje výsledky modelu, který má za úkol klasifikovat obrázky obsahující kočku z testovací sady obsahující sto fotografií koček a sto fotografií psů. Z tabulky můžeme vyčíst, že se modelu podařilo úspěšně rozpoznat 75 koček, tedy TP = 75 (True Positive, skutečně pozitivní) a také úspěšně rozpoznat 85 fotografií bez kočky, tedy TN = 85 (True negative, skutečně negativní). Co se týká selhání modelu, tak v FN = 25 (False negative, falešně negativní) případů kočku chybně nerozpoznal a naopak v FP = 15 (False positive, falešně pozitivní) případech ji viděl i v obrázcích psů. Tyto hodnoty nám kromě prvotního ujištění, že model funguje lépe než v případě náhodného tipu, také naznačují mírnou tendenci spíše predikovat absenci kočky a hlavně nám poslouží k výpočtu následujících hodnot.

	Kočka (predikovaná)	Pes (predikovaný)
Kočka (skutečná)	75 (TP)	25 (FN)
Pes (skutečný)	15 (FP)	85 (TN)

Tabulka 2.1: Tabulka záměn modelu pro klasifikaci obrázků obsahujících kočku.

Preciznost, citlivost a přesnost

Dvěma nejvíce používanými metrikami pro popis kvality modelu jsou preciznost (anglicky precision nebo positive predictive value – PPV) a citlivost (anglicky sensitivity, recall nebo true positive rate – TPR). Preciznost udává poměr správných pozitivních predikcí následně:

$$Preciznost = \frac{TP}{TP + FP}. \tag{2.1}$$

Citlivost nás zase informuje o poměru správně predikovaných hodnot z cílové třídy:

$$Citlivost = \frac{TP}{TP + FN}. \tag{2.2}$$

Pro náš příklad z tabulky 2.1 by pak tyto hodnoty byly: preciznost = 0,83 a citlivost = 0,75. V praxi si je často potřeba vybrat mezi zvýšením jedné či druhé hodnoty, což závisí na příkladu použití. Například systém zachycení nevyžádané pošty vyžaduje velkou preciznost pro zamezení odstranění důležité komunikace, zatímco systém detekující výbušniny v zavazadlech si může dovolit menší preciznost, ale musí nalézt všechny výbušniny a tedy mít vysokou citlivost.

Stejně jako pro matici záměn jsou i tyto hodnoty zjistitelné i pro klasifikační problémy o více třídách, kde se počítají zvlášť pro každou třídu proti sloučení ostatních negativních tříd.

Obecnou schopnost klasifikátoru rozlišit mezi třídami modelu můžeme nakonec vystihnout hodnotou přesnosti (anglicky accuracy – ACC), která nám udává celkový poměr správně predikovaných vzorků, který je pro náš klasifikátor koček roven hodnotě 0,8 a je vypočítán takto:

$$Přesnost = \frac{TP + TN}{TP + TN + FP + FN}. \tag{2.3}$$

V případě větší důležitosti některé ze skupin na základě použití je možné upravit jejich význam pomoci vynásobení konstantou.

Plocha pod křivkou ROC

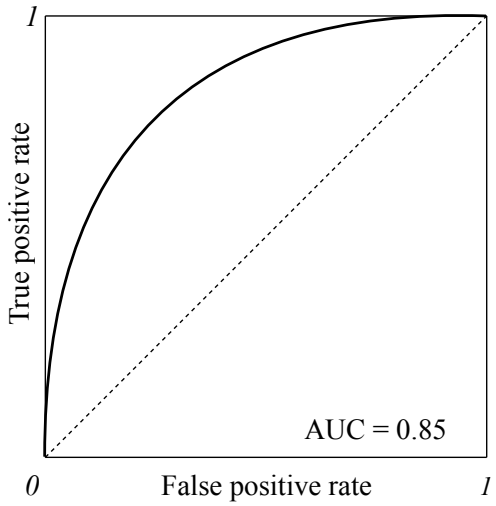
ROC křivka je často používanou metodou popisu kvality klasifikátorů, které umožňují na svém výstupu zobrazit míru jistoty klasifikace do jednotlivých tříd. Jedná se o vykreslení poměru TPR (citlivost) na ose y proti FPR (anglicky False Positive rate, míra falešně pozitivních), které je vypočítáno následně:

$$FPR = \frac{FP}{FP + TN}, \tag{2.4}$$

na ose x. Toto vykreslení je provedeno spočítáním těchto hodnot v rozsahu rozhodovací hranice jak je zobrazeno na obrázku 2.1.

Při rozhodovací hranici nula získáme hodnoty TPR i FPR rovny jedné, která tvoří pravý horní roh grafu. Při hranici rovné jedné naopak není klasifikován žádný objekt a obě hodnoty tedy poklesnou na nulu v levém dolním rohu. Tato křivka umožňuje jednoduchou volbu hranice podle potřeb použití a navíc poskytuje hodnotu plochy pod touto křivkou zvanou AUC (Area Under the ROC Curve, česky plocha pod křivkou operační charakteristiky přijímače).

AUC popisuje obecnou rozhodovací schopnost modelu při jeho ideálním použití pomocí jedné číselné hodnoty. Při tvorbě modelu je chtěné se dostat nad hodnotu 0,5 značící čistě



Obrázek 2.1: Příklad ROC křivky s hodnotou AUC = 0.85. Převzato z [2].

náhodné rozhodování. Hodnota pod 0,5 pak značí zásadní chybu a model predikující třídy opačně.

V případě modelů klasifikujících do více tříd jsou hodnoty TPR a FPR spočítány pro každou z nich zvlášť se sloučením ostatních tříd do negativních vzorků. ROC křivka je pak vykreslena pro každou třídu zvlášť.

Kapitola 3

Evoluční a genetické algoritmy

Jednou z variací strojového učení jsou také evoluční algoritmy. Tyto algoritmy jsou inspirované Darwinovou evoluční teorií a po jejím vzoru obsahují populaci řešení, která se během řady generací po malých krocích vyvíjí a díky přežití silných jedinců se vlastnosti jedinců v populaci postupně zlepšují. Velkou výhodou těchto algoritmů je, že bez podrobnější znalosti problému jsou díky své adaptační schopnosti schopné vést k nalezení inovativních a zároveň kvalitních řešení.

Evoluční algoritmy jsou hojně využívány především v optimalizačních problémech. Jejich využití však není limitováno pouze na ně a umožňuje například i hledání parametrů splňujících omezující podmínky, hledání zajímavých vzorků dat při dolování znalostí nebo optimalizaci spustitelných objektů.

Tato kapitola pokračuje obecným popisem evolučních algoritmů, následovným zaměřením na jednotlivé části v podobě reprezentace řešení, popisu fitness funkce a genetických operátorů selekce, křížení a mutace. Kapitola je zakončena popisem genetických algoritmů, na které navazuje kapitola o genetickém programování. Pokud není uvedeno jinak všechny informace v této kapitole jsou přebrány z De Jong [5], Sekanina [14] a Whitley [17].

3.1 Princip fungování evolučních algoritmů

Obecný princip těchto algoritmů spočívá v inicializaci první generace jedinců představujících kandidátní řešení problému, kdy je možné využít jak náhodného vygenerování, tak i jiných systematických způsobů. Každému z nich je následně pomocí fitness funkce přiřazena hodnota fitness popisující kvalitu onoho řešení. V každé iteraci evoluce jsou pak vybráni na základě ní jedinci použití k vytvoření potomků a následně jedinci k odstranění před započetím další iterace. V tomto procesu evoluce je pokračováno, dokud není dosaženo dostatečně kvalitního řešení nebo maximálního počtu iterací a jako řešení je navrácen jedinec s nejvyšší dosaženou hodnotou fitness.

V otázce velikosti populace zatím nebylo dokázáno, že by variabilní velikost přinášela při hledání řešení lepších výsledků a je proto definována konstantami M a K. Konstanta M udává velikost samotné populace a konstanta K počet nově vytvořených potomků v každé iteraci, přičemž velikost populace je vždy na konci iterace opět zredukována na velikost M.

Samotná volba konstant M a K závisí na složitosti řešeného problému přičemž vyšší čísla pomáhají k větší paralelnosti a větší míře prozkoumání prohledávacího prostoru, ale také zvyšují výpočetní nároky a k jejich vhodnému nastavení tedy pomáhá znalost složitosti řešeného problému.

3.2 Reprezentace řešení

Vnitřní reprezentace řešení má formu řetězce o pevně dané velikosti a je provedena formou genotypu či fenotypu. Reprezentace genotypem se podobá přírodní reprezentaci DNA a skládá se z řetězců znaků definované abecedy, které kódují parametry řešení na které se převádí. Oblíbenými abecedami jsou například přirozená čísla nebo binární kódování. Tato reprezentace usnadňuje následovnou práci s genetickými operátory, jelikož mohou být použity nezávisle na řešeném problému. Reprezentace pomocí fenotypu naopak popisuje řešení problému přímo jeho chováním například pomocí hodnot parametrů. Následující operace musejí být proto přímo vytvořeny pro konkrétní problém.

3.3 Hodnota fitness

Hodnota fitness udává kvalitu jednotlivých řešení. Základní varianta hrubé fitness udává hodnotu přirozenou pro doménu řešeného problému. Tuto hodnotu lze normalizovat jejím podílem se sumou hrubé fitness všech jedinců populace nebo také standardizovat převedením do podoby, kdy nejlepší řešení dosahuje nulové hodnoty. Dále ji lze omezit do intervalu [0, 1] obrácenou hodnotou součtu standardizované hodnoty a čísla jedna.

Na celkové vyšší výpočetní náročnosti procesu evoluce algoritmů má nejvyšší podíl právě vyhodnocováním fitness nově vzniklých jedinců v každé iteraci algoritmu. Dnes již běžná dostupnost vícejádrových systémů tedy nabádá k paralelní úpravě této části algoritmu vyhodnocením fitness na oddělených vláknech, jádrech či systémech. Dalším způsobem je rozdělení populace na paralelně řešené části. Mírnější možností je vytvoření lokálních oblastí jedinců, kteří jsou rozděleni na výpočetní uzly a reagují mezi sebou, nebo ve větší míře pak přímo vytvořením samostatných oddělených populací s občasnou migrací mezi nimi.

3.4 Genetické operátory

Selekční mechanismy

V základním evolučním algoritmu se nacházejí dvě příležitosti k výběru jedinců a využití fitness hodnoty k selekčnímu tlaku na populaci. První příležitostí je výběr rodičovských prvků následovaný možností při výběru jedinců k odstranění z populace. Správné množství tohoto tlaku je velmi důležité, jelikož přílišný tlak a volby pouze nejlepších jedinců vedou k rychlé konvergenci, avšak mohou vést pouze k lokálnímu optimu. Příliš malý tlak umožňuje algoritmu lépe prozkoumat prohledávací prostor a nalézt co nejvýhodnější řešení, avšak za cenu pomalé konvergence.

V praxi se proto většinou používá větší forma selekčního tlaku jen v jedné z příležitostí a druhá se ponechává více stochastickou. Mezi nejpoužívanější procesy výběru patří následující formy výběrů.

- Deterministická selekce – Výběr pouze N nejlepších jedinců.
- Turnajová selekce – N opakování výběru K jedinců a výběru nejlepšího z nich.
- Proporcionální selekce – Pravděpodobnost výběru jedince je úměrná jeho hodnotě fitness.
- Selekce podle pořadí – Pravděpodobnost výběru jedince je úměrná umístění v populaci při seřazení podle hodnoty fitness.

Turnajová a proporcionální selekce odpovídá přirozenému výběru v přírodě. Deterministická selekce a selekce podle pořadí zase výběru během umělého šlechtění. Nejvyšší formu tlaku přináší pochopitelně deterministická selekce, následovaná turnajovou selekcí, která však již svým principem přináší do výběru stochastičnost. Nejmenší selekční tlak pak přinášejí selekce proporcionální a selekce podle pořadí, která svým principem řeší problémy příliš vysoké pravděpodobnosti zvolení výrazně lepšího řešení nebo stagnace v případě podobné hodnoty fitness napříč populací.

Poslední vlastností k rozhodnutí ve vztahu k selekci je určení, zda má být selekce jedinců pro novou populaci provedena mezi pouze nově vytvořenými potomky a tím efektivně odstraněním generace minulé nebo selekcí napříč všemi jedinci což dále zvyšuje selektivní tlak a vede k hladovějšímu algoritmu.

Tvorba nové generace

Při tvorbě nové generace jedinců chceme v duchu evoluce získat řešení nová, avšak podobná aktuálně existujícím. Tento proces může podobně jako v reálném světe nastat asexuálně pouze pomocí mutace nebo křížením dvou rodičů. Asexuální rozmnožování jednoho rodiče pak vede k hledání v prohledávacím prostoru poblíž rodičovského jedince a tedy především k lokální optimalizaci.

Genetická operace křížení

Při křížení dochází ke kombinaci částí rodičovských řešení a navíc k případné mutaci. Využití odlišných částí od více rodičovských prvků vede ke globálnímu hledání optima, u kterého však může být problém nalézt nejideálnější optimum. Právě tento druh rozmnožování specifikuje podmnožinu genetických algoritmů, u kterých dochází ke křížení genotypů rodičů a u kterých je dále popsán.

Genetická operace mutace

Operace mutace pozměňuje každý gen mutovaného jedince s předem nastavenou malou pravděpodobností, která umožňuje postupnou evoluci po malých krocích. U binární reprezentace genomu je mutace jednoduchou změnou bitu. U ostatních reprezentací je třeba dbát na udržení platnosti hodnot genu, což je například u reálných hodnot dosahováno intervaly povolených hodnot, ze kterých je prováděno generování. Příklad obou typů mutace chromozomu je zobrazen na obrázku 3.1.

(a) Binární mutace.

(b) Mutace celých čísel.

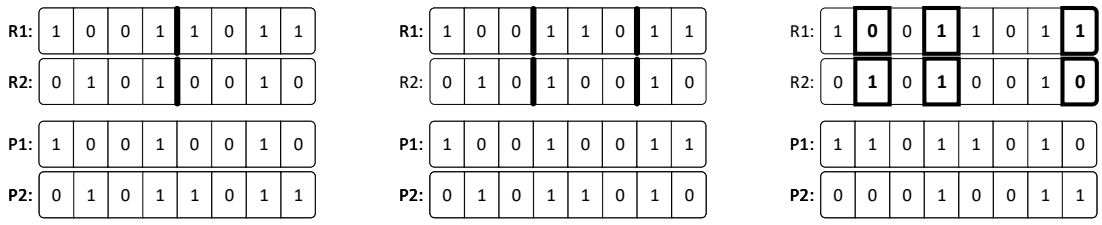
Obrázek 3.1: Příklad mutace chromozomu: a) binárního, b) přirozených čísel s intervalem hodnot [0, 100]

3.5 Genetické algoritmy

Genetické algoritmy jsou variantou algoritmů evolučních, se kterými sdílí většinu svého chování. Hlavní charakteristiky se odvíjejí od vnitřní reprezentace pomocí genotypu a využití sexuální formy reprodukce. Genotyp připouští jak klasickou formu skládající se z binárních hodnot tak i dnes běžně užívané reálné hodnoty. Genetické algoritmy také dbají na vyvážení selekce, křížení a mutace a zpravidla aplikují méně evolučního tlaku a volí kombinaci nepřesahujících generací a mírnějších změn při jejich tvorbě.

Křížení je hlavním prvkem genetických algoritmů. Často používaným přístupem je jednobodové křížení při kterém je náhodně vygenerován bod křížení, ve kterém jsou prohozeny části rodičovských genů umístěných za tímto bodem. Tento způsob křížení však vede k jevu, kdy kombinace umístěné na začátku a konci chromozomů nebývají přeneseny společně do potomka, což může komplikovat hledání co nejvýhodnějšího řešení.

Z toho důvodu bývá někdy upřednostňováno vícebodové křížení. Při kterém je provedeno prohození genů rodičovských prvků u každého vygenerovaného bodu. Další rozšířenou metodou je uniformní křížení při kterém jsou prohazovány pouze individuální geny rodičovských prvků. Ukázky všech způsobů křížení jsou zobrazeny na obrázku 3.2.



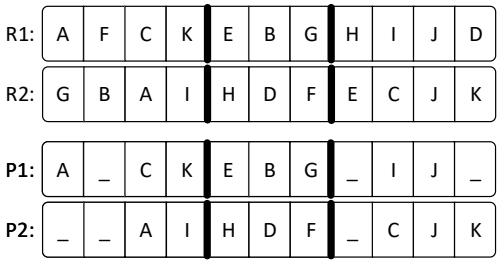
(a) Jednobodové křížení. (b) Dvoubodové křížení. (c) Uniformní křížení.

Obrázek 3.2: Varianty křížení chromozomů R1 a R2: a) jednobodové, b) dvoubodové, c) uniformní.

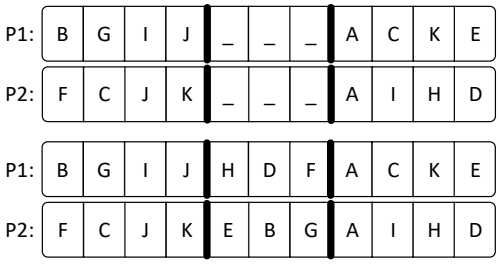
Kromě těchto základních metod křížení existuje celá řada dalších včetně metod zaměřených na problémy vyžadující obsažení každého prvku z množiny možných hodnot právě jednou. Mezi tyto metody patří například i metody Ordered Crossover (OX) popsaný v Goldberg [7] a Non-Wrapping Ordered Crossover (NWOX) představený v [4]. Tyto metody dosahují dobrých výsledků například při řešení problému obchodního cestujícího a jsou si v principu velmi podobné [1].

Nejprve jsou vytvořeni dva potomci zkopírováním obou rodičů. Následně jsou náhodně vybrány dva body křížení a a b a hodnoty umístěny v tomto rozmezí každého rodiče jsou v druhém potomku nahrazeny volnými místy. U metody OX jsou pak všechny hodnoty v potomcích posunuty doprava tak, aby byla první hodnota umístěna na indexu $b + 1$ a vytvořilo se tímto volné místo pro vložení hodnot z druhého rodiče v rozsahu a až b. U metody NWOX jsou takto doprava posunuty pouze hodnoty od indexu a, čímž dochází k omezení míry posunu hodnot. Porovnání výsledků obou metod lze vidět na obrázku 3.3.

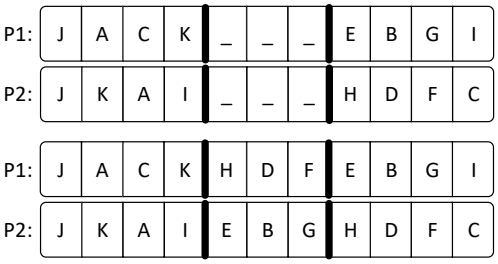
Operace křížení je i u genetických algoritmů často doplněna mutací, avšak zpravidla v menší míře než u obecných evolučních algoritmů. Pro chromozom délky N je doporučena pravděpodobnost mutace jednotlivých genů 1/N.



(a) Rodičovské chromozomy společně s jejich stavem po odstranění prvků z rozmezí a až b, které je shodné pro obě metody.



(b) Metoda OX.



(c) Metoda NWOX.

Obrázek 3.3: Znázornění metod pro křížení chromozomu OX a NWOX. Část a) obsahuje původní rodičovské chromozomy s vyznačenými místy křížení a ukazuje společný stav po kroku odstranění hodnot obsažených v střední části druhého rodiče, b) ukazuje posun a výsledek při použití metody OX a c) při použití metody NWOX.

Kapitola 4

Genetické programování

Genetické programování je jednou z variant evolučních algoritmů a zaměřuje se na tvorbu programů ve formě spustitelných struktur, které se pokouší řešit zadané problémy. Obvyklá reprezentace řešení u genetických algoritmů o pevně dané délce bez hierarchie se však pro reprezentaci programu obvykle nehodí. Pevná délka omezuje maximální délku programu, která není dopředu známá a absence hierarchie neumožňuje důležité prvky programování ve formě cyklů, podprocesů nebo rekurze. Z tohoto důvodu jsou tradiční formou chromozomu u genetického programování stromové struktury. Použití však není omezeno pouze na ně a dále jsou používány i struktury lineární nebo grafové.

Základní popis algoritmu evoluce genetické programování se příliš neliší od obecného postupu evolučních algoritmů. Po inicializaci počáteční populace jedinců představujících kandidátní řešení následuje cyklus představující průběh evoluce, v kterém je pokračováno dokud není nalezeno dostatečně kvalitní řešení nebo není dosažen limit počtu generací. Prvním krokem iterace cyklu je spuštění jednotlivých programů pro výpočet jejich hodnoty fitness. Na základě hodnoty fitness a částečné náhodnosti je zvolena část populace do role rodičů nových jedinců. Ti jsou vytvořeni na základě křížení rodičů a případné mutace. Nakonec je rozhodnuto o ponechání části jedinců do nové generace.

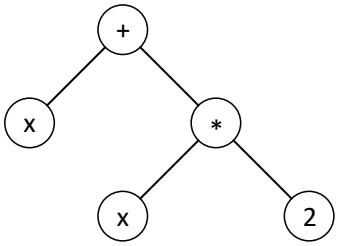
Tato kapitola se zabývá popisem dvou hlavních variant genetického programování využívajících reprezentaci v podobě stromové struktury a kartézské mřížky, kde jsou popsány jejich vlastnosti, způsoby inicializace populace a genetické operátory. Kapitola je pak zakončena popisem výpočtu fitness při využití genetického programování pro klasifikaci. Pokud není uvedeno jinak, všechny informace v této kapitole pocházejí z Miller [11], Sekanina [14] a Vanneschi [16].

4.1 Stromové genetické programování

Jak již bylo řečeno, nejrozšířenější formou genetického programování je stromové genetické programování. To využívá reprezentaci programu pomocí hierarchické stromové struktury zploštělé do podoby řetězce.

Reprezentace chromozomu

Stromové struktury se skládají z kombinace funkčních a terminálních symbolů, které představují listy stromu. Funkční bloky mají fixní počet vstupů daný jejich aritou a operaci, která může být libovolného typu ať už aritmetického, binárního nebo složitějších podmínek a iterací. Příklad jednoduchého stromu je k vidění na obrázku 4.1.



Obrázek 4.1: Příklad jednoduchého stromu provádějícího operaci y=(x+(x*2)).

Množiny funkcí a terminálů jsou také popsány dvěma vlastnostmi: uzavřeností a dostatečností. Uzavřenost vyžaduje, aby všechny funkce programu byly schopny přijmout všechny možné terminály a výstupy jiných funkcí. Tato vlastnost zaručuje, že program v průběhu svého běhu nebude moct skončit na nevalidní operaci. Operace je možné pro splnění téhle podmínky i upravit do podoby chráněné varianty jako například definování konstantního výstupu při dělení nulou. Další možností je pro stromy obsahující operace a terminály různých typů jejich explicitní definice a možnosti navázání funkcí jen na výstupy stejného typu.

Dostatečnost zaručuje, že množina terminálů a funkcí je schopna popsat hledané řešení problému. Tato vlastnost však není pro řadu problémů jednoduše zjistitelná a její dosažení vyžaduje předchozí znalost problému.

Inicializace populace

Dynamická délka chromozomu genetického programování dělá inicializaci řešení složitějším procesem, ve kterém již nestačí pouhé naplnění kapacity řetězce. Běžně používány jsou proto metody Grow, Full a Ramped Half-and-Half.

Metoda Grow začíná umístěním náhodné funkce jako kořene stromu a náhodných prvků na její vstupy a opakování přidávání náhodných terminálů nebo funkcí dokud nedojde k dosazení terminálů do všech listových uzlů nebo maximální povolené hloubky stromu d - 1, kdy jsou již doplněny pouze terminály. Velikost těchto stromů silně závisí na poměru množství funkcí a terminálů. Při velkém množství terminálů jsou stromy zpravidla krátké a v opačném případě dosahují zase shodně nejvyšší hloubky d.

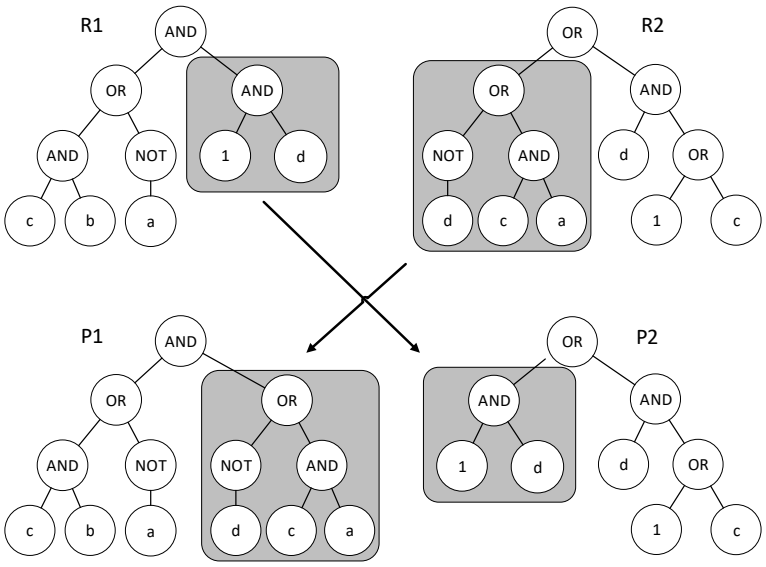
Metoda Full plní uzly až do hloubky d - 1 pouze pomocí funkcí a terminály v tomto případě tvoří pouze listové uzly. Tato metoda tedy vede k populaci pouze shodně hlubokých stromů a omezuje různorodost populace.

Naopak metoda Ramped Half-and-Half se snaží o co nejbohatší sbírku řešení pomocí vytvoření části stromů o velikost 1/d o hloubce jedna, části o velikosti 1/d o hloubce dva atd. a v každé této skupině naplnění poloviny stromů metodou Grow a poloviny metodou Full.

Operace křížení

Standardní křížení u stromového genetického programování probíhá výběrem dvou rodičů R_1 a R_2. Následně jsou v obou jedincích náhodně vybrány uzly k provedení křížení. Klasickou cestou je upřednostnění interních uzlů před listovými, ale pro zabránění bobtnání řešení je možné upravit pravděpodobnosti pro znevýhodnění také uzlů poblíž kořene a lis-

tových uzlů nebo využít shodné pozice uzlů v obou stromech. Po vybrání uzlů ke křížení je provedeno vytvoření dvou potomků prohozením podstromů jak je ukázáno na obrázku 4.2.



Obrázek 4.2: Rekombinace stromů pro vytvoření nových potomků.

Operace mutace

Klasická operace mutace probíhá u genetických programů s velmi malou pravděpodobností volbou náhodného uzlu, kde je opět možné znevýhodnit uzly poblíž kořene a listů. Vybraný uzel je společně s jeho podstromem odstraněn a nahrazen nově vygenerovaným uzlem do maximální hloubky d. Příklad jednoduché mutace je zobrazen na obrázku 4.3.

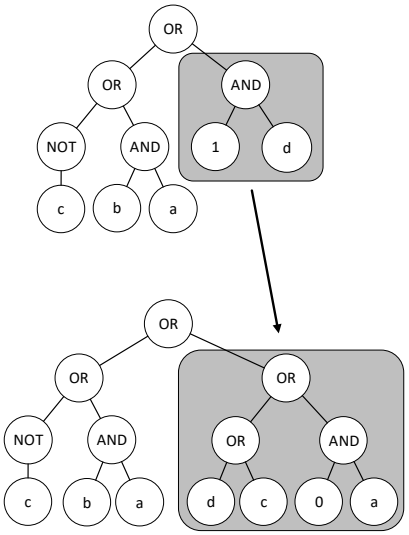
Existuje však celá řada variant mutací jako bodová mutace nahrazující pouhé jednotlivé uzly novými stejného typu a případně arity, permutace, která navzájem prohazuje podstromy uzlu nebo smršťující mutace ponechávající ze řešení pouhý podstrom vybraného uzlu.

4.2 Kartézské genetické programování

Kartézské genetické programování je formou genetického programování, kde jsou řešení reprezentována formou acyklického orientovaného grafu ve tvaru mřížky. Algoritmus evoluce je velmi podobný variantě evolučních strategií pojmenované ES $(1 + \lambda)$, kdy se nová generace řešení skládá z jednoho nejlepšího jedince předchozí generace a z λ nových jedinců, tradičně čtyř. V případě více jedinců se shodnou nejlepší hodnotou fitness je pak pro udržení diverzity důležité zvolit jedince vzniklého teprve v této generaci.

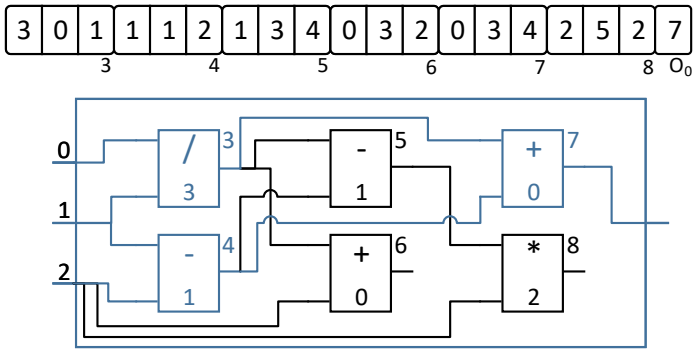
Reprezentace chromozomu

Graf v této reprezentaci genetického programování je definován hned několika parametry. Základními parametry jsou počet sloupců n_c a počet řádků n_r grafu. Dále počet primárních vstupů grafu n_i a počet primárních výstupů n_o. Každý z uzlů grafu se skládá z n_n



Obrázek 4.3: Příklad mutace stromového prvku.

vstupů, funkce z množiny funkcí Γ o velikosti n_f a jednoho výstupu. Vstupy uzlů mohou být připojeny buď na primární vstupy grafu nebo k výstupům uzlů z předešlých sloupců až do vzdálenosti dané parametrem L-back. Příklad grafu a jeho chromozomu je zobrazen na obrázku 4.4.



Obrázek 4.4: Příklad zakódování kartézského programu s parametry $n_c = 3$, $n_r = 2$, $n_i = 3$, $n_o = 1$, $n_n = 2$, L-back = 3 a Γ = {+ (0), - (1), * (2), / (3)}. Vrchní polovina obrázku zobrazuje chromozom řešení, pod kterým se nalézá jeho grafická podoba.

Chromozom řešení se skládá z Λ_{CGP} hodnot daných vzorcem

$$\Lambda_{CGP} = n_r n_c (n_n + 1) + n_o. \tag{4.1}$$

Každý z $n_r n_c$ genů chromozomu je reprezentován pomocí n_n indexů uzlů, které jsou připojeny na jeho vstup, a jedné hodnoty navíc určující funkci daného uzlu. Posledních n_o hodnot označuje indexy uzlů, které jsou připojeny k výstupům grafu. Indexy uzlů užívané

v popisu chromozomu jsou číslovány počínaje primárními uzly a poté po sloupcích počínaje prvním řádkem.

Tento způsob kódování vede k tomu, že i když velikost chromozomu je fixní, velikost fenotypu nikoliv. Všechny primární vstupy ani uzly řešení nemusí vést k výstupům grafu a tedy nemusí být ani použity. U uzlů, které jsou součástí fenotypu, je zase možné pro některé funkce využít jen některé z jejich vstupů (například při změně binární funkce uzlu $x_1 + x_2$ na unární funkci $abs(x_1)$).

Operace mutace

Na rozdíl od běžného genetického programování nepoužívá kartézská varianta ke tvorbě nové populace operaci křížení, ale pouze mutace. Ta je pro daný počet genů provedena s určitou pravděpodobností pro jeho jednotlivé hodnoty, které nahradí hodnotou novou. Tato mutace může kromě změny funkce uzlu vést i ke změně struktury a toho, které uzly jsou aktivní a které ne. Mutace neaktivních uzlů vedou k neutrálním změnám, které se neprojevují na fenotypu chromozomu, ale které jej mohou výrazně změnit při případné pozdější aktivaci.

4.3 Funkce fitness pro symbolickou regresi

Příkladem jednoho z možných využití genetického programování je symbolická regrese, jejímž cílem je aproximace dat v zadaném intervalu. Vyhodnocování těchto programů probíhá na trénovací množině vzorků o velikosti N. Hodnota fitness zde může být definována jako průměrná odchylka výstupu kandidátního programu S a výstupu požadovaného od plně funkčního řešení C následně:

$$f(i) = \sum_{j=1}^{N} |S(i,j) - C(j)| \,. \tag{4.2}$$

Cílem evolučního procesu je tedy pomocí selekčního tlaku dosáhnout minimalizace této odchylky na trénovací množině.

Dostatečná velikost použité trénovací sady je velice důležitá, jelikož v případě příliš malé sady hrozí vznik řešení, které není schopné dostatečně generalizovat, tedy pracovat s dosud neviděnými daty. Příliš velká sada ovšem vede k vysokým výpočetním nárokům této náročné části algoritmu a nastavení tak vyžaduje nalezení ideální velikosti vzhledem ke složitosti problému.

Kromě hlavního využití pro symbolickou regresi je genetické programování využívané při řešení řady problémů z oborů programování, návrhu analogových a číslicových obvodů nebo mechanických systémů. Kromě těchto způsobů je možné využít genetického programování i k řešení klasifikačních úloh, které ve svém principu nejsou nepodobné právě symbolické regresi.

Kapitola 5

Koevoluční algoritmy

Koevoluční algoritmy podobně jako evoluční vycházejí z principu Darwinovy evoluční teorie. Evoluční algoritmy ovšem jedince své populace hodnotí přímo a individuálně na základě výsledku fitness funkce. Toto není případem koevolučních algoritmů. Ty hodnotí své jedince na základě interakcí s ostatními jedinci a tím jim přidělují takzvanou subjektivní fitness.

Objektivní funkce fitness v tradičních evolučních algoritmech je obvykle chápána jako funkce $f : G \to \mathbb{R}$, na jejímž základě lze pro každý genotyp $g_1, g_2 \in G$ určit, který z nich plní zadanou úlohu lépe tak, že porovnáme $f(g_1)$ a $f(g_2)$. Tento vztah g_1 a g_2 zůstává po celou dobu evoluce stejný. V případě koevolučních algoritmů však může být v jednom okamžiku evoluce $f(g_1) > f(g_2)$ a v dalším okamžiku evoluce, kdy dojde k ohodnocení g_1 a g_2 na základě interakce s jinými jedinci, může nastat situace, kdy $f(g_1) < f(g_2)$. Tato fitness se proto nazývá fitness subjektivní.

Tato kapitola je rozdělena do tří částí. První část se zabývá popisem samotných koevolučních algoritmů, vysvětluje myšlenku více populací a práci s nimi. Následně jsou popsány také typy problémů řešitelných tímto přístupem a podoba samotných řešení. Druhá část se zabývá evaluací fitness hodnoty jedinců těchto populací a třetí část využitím populace prediktorů s proměnlivou velikostí. Pokud není uvedeno jinak, všechny informace v této kapitole pocházejí z Popovici [12] a v podkapitole o prediktoru s proměnlivou velikostí z Drahošová [6].

5.1 Reprezentace

Reprezentace koevolučních algoritmů je podobná té z původních evolučních algoritmů. Přesto zde existuje pár rozdílů vzniklých z použití více populací, které se tato podkapitola snaží alespoň z části zachytit.

Jedinci

Samotná podoba jedinců v koevolučních algoritmech se příliš neliší od jejich podoby v evolučních algoritmech. Stále jsou reprezentováni chromozomem složeným z určitých hodnot a stále jsou pomocí genetických operací vyvíjeni v další kvalitnější generace. Jedinci populací zde však představují entity typů domény řešeného problému a zpravidla je tedy počet typů jedinců shodný s počtem typů domény. Všechny typy však nemusí být řešeními problému, ale mohou například zastávat pouze roli testů kandidátních řešení (symbolická regrese má například 2 domény: kandidátní matematické vztahy a testy).

Populace

Koevoluční algoritmy můžeme rozdělit na jednopopulační a vícepopulační. V základní podobě probíhá vyhodnocování kvality jedinců vždy s ostatními jedinci, kdy v případě pouze jedné populace samozřejmě s jedinci té samé populace. V případě více populací pak s jedinci z populací ostatních. Koevoluční algoritmy většinou obsahují alespoň jednu populaci pro doménový typ a jednopopulační zpravidla právě jednu. Pro lepší pochopení principu je níže uveden algoritmus 1 obsahující pseudokód jednopopulační verze. Vícepopulační se pak liší pouze v sekvenčním provedení inicializace a následně jednotlivých iterací algoritmu pro všechny populace.

Algoritmus 1: Pseudokód jednopopulačního koevolučního algoritmu.

Result: Nejlepší jedinec populace
P := inicializace_populace();
H := výběr_hodnotitelů(P);
foreach *Jedince* $I \in P$ **do**
  $I.fitness$ = ohodnocení_na_základě_interakce(I, H)
end
$i := 0$;
while *nejlepší_jedince.fitness < požadovaná kvalita* `AND` $i <$ *limit počtu generací* **do**
  R := výběr_rodičů(P);
end
H := výběr_hodnotitelů(P);
N := vytvoření_nových_jedinců(R);
foreach *Jedince* $I \in N$ **do**
  $I.fitness$ = ohodnocení_na_základě_interakce(I, H)
end
nejlepší_jedince := $\{i \in P | i.fitness > \forall x'.fitness \in P\}$;
P := výběr_jedinců_do_další_generace($P + N$);
i++;

Algoritmy však nejsou omezeny pouze na interakci s aktuálními jedinci, ale mohou využít archivů, které umožňují uchovat a využít i řešení z generací předchozích. Pro vícepopulační algoritmy také existuje možnost paralelní evoluce, která ovšem vyžaduje řešení problémů komunikace mezi populacemi.

Archiv jedinců

Archiv slouží jako typ paměti, která sahá napříč generacemi a umožňuje uložení a práci s uloženými jedinci populací. Archiv často obsahuje řešení problému, nebo je sám řešením, díky ukládání nejlepších nalezených jedinců a možnosti dalšího zkoumání prohledávacího prostoru bez strachu ze zhoršení nalezených řešení.

Další z využití archivu je jeho použití k výpočtu fitness jedinců aktuální populace na základě jejich interakce s těmito nejlepšími řešeními.

Problém a jeho řešení

Řešení problému koevolučního algoritmu může být získáno z řady následujících míst:

- Jedinec libovolné populace – Například v případě více populací stejného typu.

- Jedinec určité populace – Jedinec zkoumané populace.
- Jedinec z archivu – Archiv obsahující nejlepší jedince.
- Celá populace nebo archiv – Například hledání Pareto optimálního složení jedinců.
- Část populace nebo archivu – Například hledání ideálního týmu.

Problémy řešené pomocí koevolučních algoritmů se dají rozdělit na testovací a kompoziční problémy. U testovacích problémů je kvalita jedinců vyhodnocena na základě interakce s populací testů. Kompoziční problémy lze naopak chápat jako snahu dosáhnout ideální spolupráce jedinců v týmu pro dosažení nejlepšího výsledku. Doménu, ve které řešíme jistý problém, pak nazýváme kvůli fitness funkci postavené na interakci interaktivní doménou. Pro další práci bude nyní vhodné definovat několik pojmů.

- Interaktivní doména se skládá z jedné a více funkcí metrik ve formě: $p : X_1 \times X_2 \times ... \times X_n \to R$.
- Každé i z $1 \leq i \leq n$ je doménovou rolí.
- X_i je množina entit doménové role i.
- Prvek $x \in X_i$ je entita.
- N–tice $x_1, x_2, ..., x_n \in X_1 \times ... \times X_n$ je interakce.
- Hodnota $p(x_1, x_2, ..., x_n) \in R$ je výstupem interakce.
- Seřazená množina R je výstupní množinou.

Samotný výběr řešení problému se pak skládá z následujících částí.

- Neprázdná množina $I \subset \{1, ..., n\}$ složená z n doménových rolí.
- Množina $\mathcal{P}$ kandidátních řešení agregovaných z entit doménových rolí I.
- Koncept řešení, který v kandidátních řešeních identifikuje podmnožinu řešení $\mathcal{S} \subset \mathcal{P}$ nebo je seřazuje dle kvality.

Jinými slovy je vytvořena množina kandidátních řešení, která jsou následně rozdělena na skutečná řešení a neřešení nebo seřazena dle hodnoty jejich subjektivní fitness. Navíc je možné, že řešením může být množina více entit. Ty mohou být buď všechny z jedné doménové role, potom $I = (1)$ nebo z více, případně i všech $I = (1, 2, ..., n)$. Komplement I obsahující doménové role nepoužité v řešení označujeme jako $\hat{I}$.

Kompoziční problémy

V kompozičním problému jsou k tvorbě řešení použity všechny doménové role, tedy $|I| = n$ a $|\hat{I}| = 0$.

Používanou praktikou výběru řešení kompozičního problému je koncept ideálního týmu, formálně definovaný následně:

$$\mathcal{S} = \{ \bar{x} \in \mathcal{P} | \forall \bar{x}' \in \mathcal{P}. p(\bar{x}) \leq p(\bar{x}') \Rightarrow p(\bar{x}) = p(\bar{x}') \}, \tag{5.1}$$

kde je řešením množina entit taková, že libovolná jiná množina z kandidátních řešení nemá vyšší hodnotu fitness.

Úlohy založené na testu

Úloha založené na testu může být podle své povahy řešena jak jednopopulačním algoritmem tak i jeho vícepopulační alternativou. Jednopopulační koevoluční algoritmus je vhodný například při hledání vhodné herní strategie. Každý z jedinců tak může chápat zbytek populace jako testy, proti kterým může vyzkoušet své schopnosti.

Příkladem problému k řešení pomocí dvojce populací může být například klasifikace, kdy první populace obsahuje kandidátní klasifikátory a druhá populace algoritmy vybírající stále složitější data ke klasifikaci. Řešením v úlohách založených na testu je jedinec pouze jedné doménové role $|I| = 1$, jelikož všechny ostatní populace zde pracují jako testovací množiny tohoto řešení.

Pro klasický testovací problém s interaktivní doménou dvou množin entit X_1 a X_2, která má jednu metriku $p : X_1 \times X_2 \to R$ a kde X_1 představuje komponentu řešení a X_2 testy, lze použít řadu způsobů výběru řešení $\mathcal{S}$ z množiny kandidátních řešení $\mathcal{P}$.

Jedním ze způsobů je koncept nejlepšího nejhoršího případu, který lze použít v případech, kdy je potřeba se vyvarovat nejhoršího možného případu. Výsledkem je komponenta, jejíž nejmenší hodnota z interakce se všemi testy je nejvyšší mezi ostatními kandidátními řešeními, což je formálně definováno takto:

$$\mathcal{S} = \{\, x \in \mathcal{P} | \forall x' \in \mathcal{P}. \min_{t \in X_2} p(x,t) \leq \min_{t \in X_2} p(x',t) \Rightarrow \min_{t \in X_2} p(x,t) = \min_{t \in X_2} p(x',t) \,\}\,. \tag{5.2}$$

Dalším způsobem je maximalizace všech výstupů, která vrací entitu, které se podaří dosáhnout nejvyšší hodnoty v rámci interakce se všemi testy. Tato entita však v řadě případů nemusí vůbec existovat. Ta je definována takto:

$$\mathcal{S} = \{\, x \in \mathcal{P} | \forall x' \in \mathcal{P} \forall t \in X_2. [p(x,t) \leq p(x',t) \Rightarrow p(x,t) = p(x',t)] \,\}\,. \tag{5.3}$$

Maximalizace očekávané užitečnosti (MEU) navrací řešení, které maximalizuje výstup interakce s libovolně zvoleným testem. Tento postup můžeme chápat i jako maximalizaci sumy subjektivní fitness oproti všem testům daný následně:

$$\mathcal{S} = \{\, x \in \mathcal{P} | \forall x' \in \mathcal{P}. E(p(x, X_2)) \leq E(p(x', X_2)) \Rightarrow E(p(x, X_2)) = E(p(x', X_2)) \,\}\,. \tag{5.4}$$

5.2 Evaluace

Výpočet fitness hodnoty na základě množství interakcí jedinců přináší nové problémy k řešení. Ty se skládají z rozhodnutí o způsobu interakce mezi jedinci, způsobu agregace výsledků interakcí do hodnoty fitness a nakonec provedení komunikace mezi nezávislými populacemi.

Interakce

Při interakci mezi populacemi je důležité se rozhodnout o její míře. Varianta plné interakce jedinců se všemi jedinci ostatních populací je totiž z výpočetního hlediska velmi nákladná a zpravidla je možné využít pouze podmnožiny těchto interakcí. Z tohoto důvodu jsou interakce většinou řešeny pomocí dvou jiných přístupů: zaměřeného na jedince a zaměřeného na populaci. Přístup zaměřený na jedince prochází postupně jednotlivé jedince, kdy u každého provede všechny potřebné interakce a na základě jejich výsledku spočítá fitness.

Častou formou tohoto přístupu je nalezení nejlepšího jedince první populace, který poté jako jediný slouží k hodnocení všech jedinců populace druhé.

Přístup zaměřený na populaci využívá topologii, která zajišťuje, že každý jedinec bude v procesu výpočtu fitness využit alespoň jednou a to například v podobě eliminačního turnaje, který omezí počet interakcí a přitom poskytne obrázek o kvalitě jedinců.

Agregace

V případě, kdy jedinec získá v průběhu více interakcí vícero ohodnocení, je potřeba tato čísla agregovat do jeho finální hodnoty fitness. Mezi jednodušší přístupy patří jednoduché použití minima, průměru nebo maxima, které mohou být vybrány podle vlastností řešeného problému. Druhým způsobem je využití více těchto hodnot pro vytvoření n-tice fitness hodnocení, které přináší výhody například v případě hledání ideálního týmu nebo Paretova optima, ale vyžaduje úpravu evolučních algoritmů pro schopnost vyhodnocení a porovnání těchto hodnot.

Komunikace

Při využití více paralelně běžících populací je potřeba rozhodnout o parametrech jejich vzájemné komunikace, kterou ovlivňují parametry koordinace, toku a frekvence.

Koordinace komunikace asynchronních populací, jejichž vývoj probíhá nezávisle na sobě, probíhá pomocí sdílené paměti. Do té samostatně zapisují informace o svém stavu a získávají informace o druhé populaci. Tok těchto populací je vždy paralelní a z principu umožňuje běh více generací populací najednou. Synchronní populace jsou podřízený společnému řízení, které určuje okamžiky jejich komunikace. Synchronní populace mohou být buď paralelní, kdy vývoj obou populací běží odděleně po určitý čas, kdy jsou obě zastaveny a proběhne komunikace, nebo sekvenční, kdy je v každý okamžik prováděn vývoj pouze jedné populace, které se střídají.

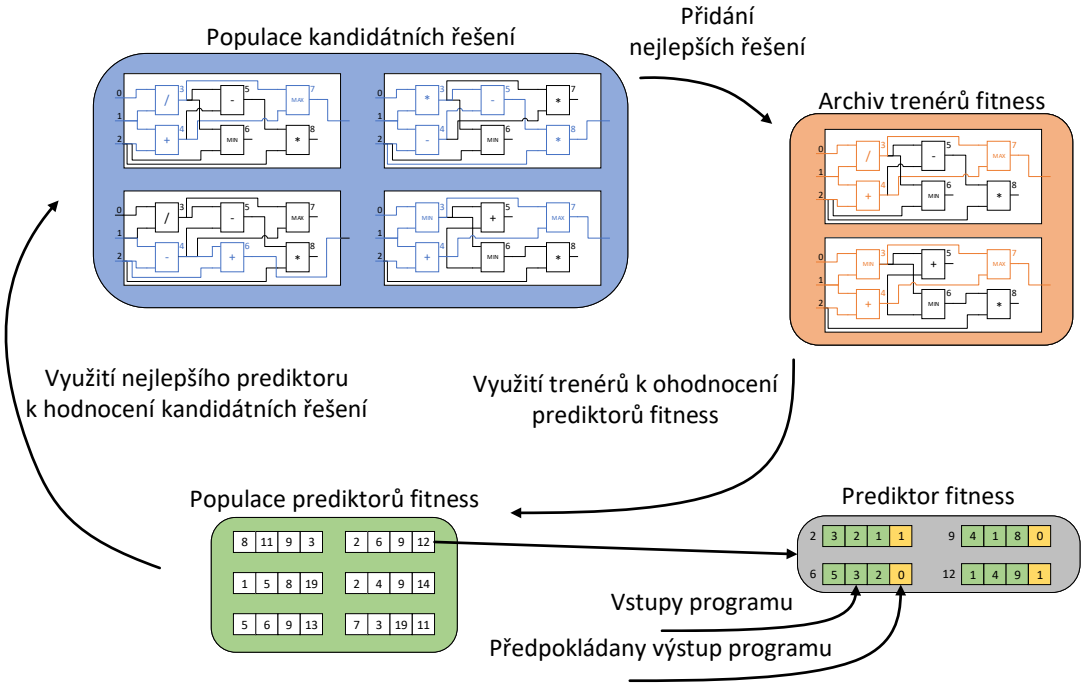
Komunikační frekvence určuje počet evolucí proběhlých v jednotlivých populacích mezi opakováním vzájemné komunikace. Ten může být buď pro všechny populace uniformní nebo se může vzájemně lišit.

5.3 Prediktor s proměnlivou velikostí

Problém klasifikace lze chápat jako jistou formu koevolučního algoritmu, kdy první populaci tvoří kandidátní řešení klasifikátoru a druhou populaci trénovací sada, kdy k ohodnocení klasifikátorů dochází na základě interakce s prvky populace druhé v podobě výpočtu objektivní fitness. V rámci snahy o zrychlení výpočtu fitness můžeme populaci trénovacích vzorků upravit na populaci podmnožin trénovací sady, kde se snažíme na základě interakce zvolit nejlepší podmnožinu, která poté slouží k nahrazení zdlouhavého výpočtu objektivní fitness subjektivní hodnotou vypočtenou pouze na této podmnožině.

Populaci těchto podmnožin nazýváme populací prediktorů fitness, jelikož se snaží co nejpřesněji predikovat objektivní hodnotu fitness klasifikátorů. K určení nejpřesnějšího prediktoru použitého k hodnocení je však potřeba přidělit fitness hodnotu nejprve samotným prediktorům. Toho je docíleno pomocí interakce s archivem trenérů fitness, který obsahuje část nejlepších programů nalezených v průběhu posledních generací klasifikátorů se známou hodnotou objektivní fitness. Prediktory jsou ohodnoceny na základě rozdílu mezi objektivní fitness trenérů a subjektivní hodnotou přidělenou prediktory. Nejlepším prediktorem je pak

ten s nejmenší odchylkou, a který tedy ohodnocuje nejpřesněji. Schéma spolupráce těchto populací je zobrazeno na obrázku 5.1.



Obrázek 5.1: Schéma využití prediktorů fitness při evoluci kandidátních programů a jejich hodnocení pomocí archivu trenérů fitness.

Velikost těchto prediktorů je rozhodujícím faktorem výpočetní náročnosti ohodnocení kvality kandidátních řešení a tedy i celého procesu hledání řešení. Kromě volby vhodné podmnožiny trénovacích dat, které je zajištěno pomocí křížení nejlepších prediktorů, je tedy důležitá volba i nejmenší velikosti prediktoru, která stále vede k dostatečně přesným predikcím.

Postup volby této proměnlivé velikosti byl navržen v [6]. Velikost prediktorů *readLength* je měněna ve dvou případech. Vždy když dojde k nalezení nové nejvyšší subjektivní fitness kandidátních řešení a naopak v případě, že ke změně velikosti prediktoru nedojde po zvolený počet generací G_{update}. Nová velikost prediktorů závisí na odchylce prediktoru fitness I a na detekované fázi evoluce v podobě rychlosti evoluce v dané následně:

$$v = \frac{\Delta f}{\Delta G}, \tag{5.5}$$

kde Δf je rozdíl objektivní fitness aktuálního nejlepšího programu a nejlepšího programu při minulé hodnotě *readLength* a ΔG je počet generací od poslední změny hodnoty *readLength*. Kladná hodnota rychlosti v značí zlepšující se nejlepší řešení a záporná naopak zhoršující se kvalitu řešení.

Chyba predikce prediktoru fitness I je vypočtena jako podíl subjektivní fitness $\hat{f}$ a objektivní hodnoty fitness f následně:

$$I = \frac{\hat{f}}{f}. \tag{5.6}$$

Nová velikost prediktoru je vypočtena vynásobením aktuální velikosti *readLength* a konstanty C, jejíž hodnota je vybrána podle následující tabulky 5.1.

Priorita	Podmínka	Koeficient C
1	$I > I_{thr}; I_{thr} = 2.7$	1.20
2	$\|v\| \leq 0.001$	0.90
3	$v < 0$	0.96
4	$0 < v \leq 0.1$	1.07
5	$v > 0.1$	1.00

Tabulka 5.1: Tabulka výběru konstanty C v závislosti na hodnotě parametrů I a v. Převzato z [6].

Tato proměnlivá velikost prediktoru dopomáhá rychlejšímu nalezení co nejvýhodnějšího řešení problému díky automatizovanému přizpůsobování velikosti prediktoru v průběhu řešení úlohy. V případě zvětšující se chyby predikce se velikost prediktoru za účelem zvýšení přesnosti predikce zvětší. Naopak v případě, že kandidátní řešení dosáhne lokálního optima (není zaznamenáno zlepšení fitness po předem zvolený počet generací) se velikost prediktoru sníží, čímž umožní kandidátnímu řešení opustit lokální optimum.

Kapitola 6

Pohybová data pacientů a existující řešení jejich klasifikace

Data použitá v této práci k detekci projevů dyskineze byla pořízena v rámci Lonesovi studie [10]. Tato kapitola se zabývá základním popisem těchto dat a podmínkami jejich vzniku a pokračuje zhodnocením výsledků dosažených v této studii, která poslouží jako výchozí bod této práce a k porovnání kvality dosaženého řešení.

6.1 Popis pohybových dat

K záznamu dat pacientů bylo využito celkem šesti zařízení umístěných na končetinách, hrudi a hlavě pacienta. Zařízení a jejich umístění lze vidět na obrázku 6.1. Každé z těchto zařízení bylo vybaveno tříosým akcelerometrem a tříosým gyroskopem. Akcelerometr umožňuje snímat informace o změně rychlosti pohybu a úhlu vychýlení k zemskému povrchu a gyroskop informace o prostorové orientaci a úhlové rychlosti [15]. Oba tyto údaje byly zaznamenávány s frekvencí 100 Hz.

Obrázek 6.1: Ukázka zařízení použitého k zaznamenání pohybových dat společně s příkladem umístění na tělo pacienta. Převzato z [10].

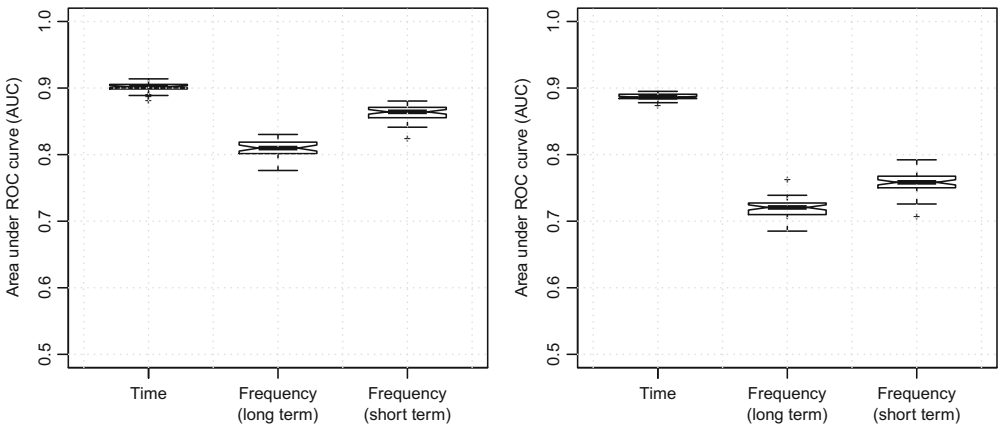
Data pro tuto úlohu byla získána v průběhu dvou klinických studií na celkem dvaceti třech pacientech. První byla provedena na šesti pacientech, kteří byli sledování po dobu šesti hodin. Druhá studie byla provedena na sedmnácti pacientech s měřením o délce dvou hodin. Pacienti obsažení v obou studiích trpěli obdobnou mírou dyskineze, přičemž první

skupina byla v průměru mírně starší a s větší mírou projevů dyskineze. V rámci každé studie bylo provedeno měření na stejném setu senzorů, který se však lišil mezi studiemi, což umožňuje zajištění robustnější detekce příznaků.

Měření byla provedena nezávisle na čase od posledního podání léků a pacienti byli v průběhu měření instruováni k volnému pohybu po místnosti dokumentované infračervenou kamerou. Tyto záznamy následně umožnily trojici doktorů rozdělit pohybová data na části s různou mírou projevů dyskineze podle jednotné stupnice hodnocení dyskineze (UDysRS) od 0 (neprojevující se) až po 4 (paralyzující a zabraňující některým pozicím či pohybům).

6.2 Zhodnocení aktuálního stavu a identifikace problémů

V rámci předešlé snahy o klasifikaci těchto dat v rámci Lonesovy práce [10] bylo provedeno několik zjištění, které je možné dále využít ke zlepšení vlastního návrhu řešení. Kromě navrženého řešení pomocí upravené verze CGP pojmenované IRCGP (implicitní kontextová reprezentace kartézského genetického programování) představené v Cai [3], která stavěla na snaze reprezentovat chromozom způsobem umožňujícím provádění operace křížení, bylo dále vyzkoušeno také využití krátkodobých a dlouhodobých spektrálních klasifikátorů. U všech tří typů klasifikace pak bylo vyzkoušeno použití dat z akcelerometru a gyroskopu a výsledky lze pozorovat na obrázku 6.2.

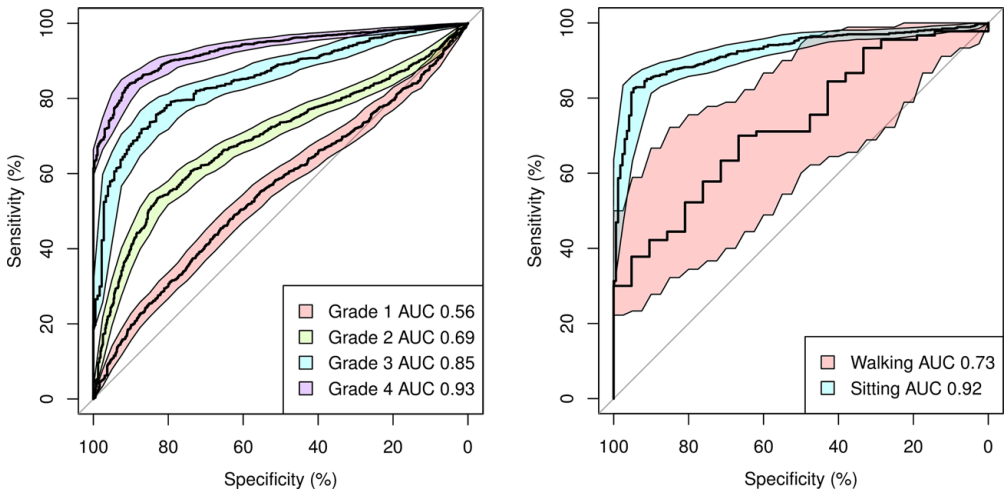


(a) Accelerometry data (b) Rotational data

Obrázek 6.2: Porovnání kvality jednotlivých způsobů klasifikace navržených v Lones [10] a srovnání výsledků získaných při použití dat z (a) akcelerometru a (b) gyroskopu. Jako trénovací data byla pro lepší separovatelnost použita pouze data bez projevů dyskineze a s projevy závažné dyskineze tříd 3 a 4. Data odpovídající třídám 1 a 2 byla pro trénování vynechána. Převzato z [10].

Výsledky ukázaly, že informace o zrychlení získaná z akcelerometru jsou při detekci dyskineze vhodnějším zdrojem informací než rotační data z gyroskopu a zároveň, že navržené řešení zpracovávající tyto vstupní data pomocí plovoucího okna o velikosti 32 vzorků je schopné při detekci závažných projevů dosáhnout hodnoty AUC 0,9.

Dalším zjištěním studie byl problém klasifikace dyskineze u pohybujících se pacientů, kde je průměrná hodnota AUC pouhých 0,73 oproti 0,92 u nepohybujících se pacientů. Dalším problémem je klasifikace nízkých úrovní dyskineze, které jsou navzájem hůře oddělitelné a například u drobných příznaků na úrovni 1 je dosaženo AUC pouze 0,56.



(a) Dyskinesia grades (b) Patient activities

Obrázek 6.3: (a) Hodnoty AUC pro rozpoznání jednotlivých závažností projevů dyskineze. (b) Srovnání průměrné AUC na datech pořízených u sedících a chodících pacientů. Převzato z [10].

Dosažené výsledky, zobrazené na obrázku 6.3, potvrzují schopnost řešení klasifikovat data a zároveň ponechávají další možnosti ke zlepšení především u slabších projevů dyskineze a během pohybu pacientů.

Kapitola 7

Návrh

Cílem této kapitoly je popsat návrh klasifikátoru, který bude schopen na základě pohybových údajů z akcelerometru a gyroskopu rozhodnout o míře dyskineze u pacientů s Parkinsonovou nemocí. Na klasifikátor je kromě přesnosti kladen nárok i v podobě omezení výpočetní náročnosti, která by umožňovala provádět klasifikaci přímo ve snímací jednotce, kde bude navrženo využití operací pracujících s celými čísly oproti modelu představenému v Lonesovi [10], který pracuje s reálnými čísly. U procesu učení, který je u evolučních algoritmů velmi zdlouhavý, pak bude pro urychlení tohoto procesu navrženo využití koevolučních technik.

Kvalita výsledného klasifikátoru bude hodnocena z pohledu schopnosti rozlišit mezi třídami, konkrétně v podobě AUC, a výpočetní náročnosti v porovnání s řešením navrženým v Lonesově práci [10] a dále z pohledu doby potřebné k trénování klasifikátoru při využití koevoluce prediktorů fitness v porovnání s variantou bez využití prediktorů fitness.

7.1 Předzpracování pohybových dat

Prvním důležitým krokem práce je předzpracování dat o pohybu pacientů popsaných v kapitole 6.1. Jednotlivé vzorky dat jsou uloženy v souborech ve formátu CSV. Vlastnosti jednotlivých vzorků jsou uloženy v názvech souborů a jsou popsány v tabulce 7.1.

Název souboru: Trial1_Description607_Walking_LeftLeg_Dyskinesia-1-chor_Tremor-none_ Bradykinesia-undefined_State-on_Rigidity-undefined_Instabiliy-undefined.csv	
Parametr	Popis
Trial1	1. klinická studie
Description607	Vzorek číslo 607
OpeningOrClosingHandMovements	Činnost - chůze
LeftLeg	Jednotka umístěná na levé noze
Dyskinesia-1-chor	Mírná úroveň dyskineze
Tremor-none	Třes - žádný
Bradykinesia-undefined	Bradykineze - nedefinována
State-on	Pacient je ve stavu, kdy vnímá a reaguje
Rigidity-undefined	Ztuhlost - nedefinována
Instabiliy-undefined	Nestabilita - nedefinována

Tabulka 7.1: Příklad názvu souboru vstupních dat a popis parametrů uložených v něm.

Pro využití dat tedy bude nejprve potřeba zpracovat názvy souborů a extrahovat parametr obsahující informaci o míře projevů dyskineze a jejím zařazení do jedné ze čtyř kategorií závažnosti.

Data v těchto souborech jsou ve formátu zobrazeném na obrázku 7.1. Každý řádek souboru představuje data pořízená při jednom měření, která jsou zaznamenávána s frekvencí 100 Hz a skládají se ze šesti hodnot rozdělených čárkami. První tři hodnoty jsou záznamy tříosého akcelerometru v osách x, y a z a jsou následovány třemi hodnotami tříosého gyroskopu.

```
1838,2181,2770,1884,1801,1812
1804,2178,2781,1887,1805,1815
1829,2176,2773,1890,1808,1819
1824,2192,2779,1888,1807,1824
1826,2198,2763,1886,1804,1830
```

Obrázek 7.1: Ukázka formátu pohybových dat pacientů.

Na základě výsledků předešlé studie Lonese [10] budou využita data o zrychlení, která mají lepší diskriminační schopnost a taktéž budou shodně upravena na celkovou magnitudu zrychlení pomocí vzorce:

$$|a| = \sqrt{(|a_1|^2 + |a_2|^2 + |a_3|^2)}. \tag{7.1}$$

Pro využití v jednoduchých vestavěných zařízeních je vhodná reprezentace využívající pevnou řádovou čárku nebo celá čísla. Z tohoto důvodu budou vypočtené magnitudy uloženy pro další zpracování nejenom v základní podobě využívající plovoucí řádovou čárku shodně jako v Lones [10], ale také v datových typech pro celá čísla o různých rozsazích. Pro uložení magnitud do typů o výrazně menších rozsazích bude pak nutné vstupní data upravit pomocí následujícího vzorce:

$$X_d = \frac{X_f - min(X_f)}{max(X_f) - min(X_f)} * max(d), \tag{7.2}$$

kde X_d značí novou omezenou hodnotu, X_f hodnotu původní, $min(X_f)$ a $max(X_f)$ značí minimimální a maximální hodnotu nalezenou ve vstupních datech a $max(d)$ maximální možnou hodnotu nového datového typu. Minimální a maximální hodnoty přitom budou zjišťovány pouze z trénovacích dat pro zamezení ovlivnění testovacími daty.

7.2 Model klasifikátoru projevů dyskineze

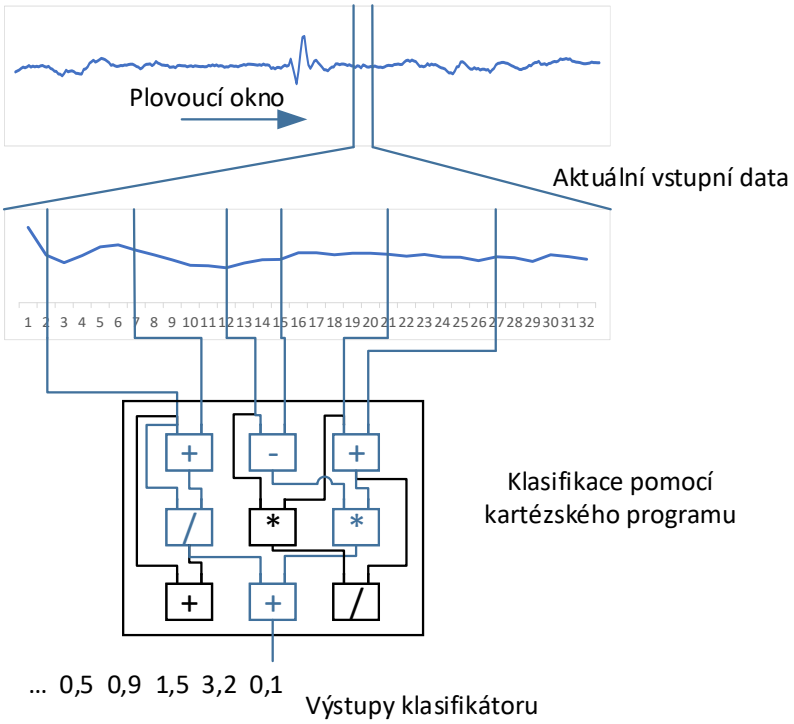
K evolučnímu návrhu klasifikátoru bylo rozhodnuto využít technik genetického programování, které jsou vhodné právě pro hledání spustitelných struktur jako v tomto případě. Dále bylo rozhodnuto o využití formy reprezentace kartézským programem, která byla již opakovaně využita při řešení problému predikce i klasifikace při dosažení dobrých výsledků [11]. CGP je navíc díky svému původnímu zaměření pro návrh číslicových kombinačních obvodů vhodné pro návrh úsporného řešení vhodného k použití ve snímací jednotce [14].

Vstupní data budou na vstup klasifikátoru přikládána v podobě vektoru 32 hodnot, který bude vytvořen pomocí plovoucího okna znázorněného na obrázku 7.2. Ze vstupních dat o celkové velikosti L je tak tímto způsobem vytvořeno celkově $L-31$ vstupů. Výstupem

klasifikátoru je postupně $L-31$ hodnot, ze kterých je vytvořen průměr, jehož výše rozhoduje o detekovaném stupni dyskineze.

Volba dalších parametrů klasifikátoru bude jedním z cílů experimentů, které budou zaměřené na rychlost konvergence při hledání řešení a jeho výslednou kvalitu. Počáteční parametry budou zvoleny podle Lonesovy práce [10], tedy velikost mřížky 6 x 6 s parametrem L-back rovným šesti. Množina funkcí bude upravena pro využití s libovolným datovým typem a rozšířena na množinu $\Gamma = \{+, -, \times, \div, avg, max, min, identita, /2, /4, sin, cos\}$.

Jak již bylo zmíněno, další úprava bude spočívat v nahrazení výpočtů s desetinnými čísly pomocí celočíselných variant o velikosti osmi a šestnácti bitů vhodnějších pro rychlé výpočty v hardwaru, u kterých bude zkoumán jejich vliv na rychlost učení a kvalitu klasifikace.



Obrázek 7.2: Ukázka zpracování posloupnosti dat pomocí plovoucího okna.

7.3 Trénování pomocí CGP

Ke trénování populace kandidátních řešení budou pro zajištění větší robustnosti klasifikátoru použity vstupní data pouze tříd nula (N) a tříd tři a čtyři (P). Trénování pak bude probíhat po vzoru evoluční strategie $1 + 4$, kdy do další generace vždy postupuje nejlepší řešení aktuální generace a čtyři jeho upravené varianty.

Fitness funkce

Po prvotním náhodném vytvoření kandidátních řešení bude provedeno jejich ohodnocení na základě fitness funkce dané hodnotou AUC, která umožňuje co nejpřesněji zhodnotit schopnost rozlišení tříd bez potřeby definice hodnoty prahu. Při výpočtu AUC budou zaznamenány výstupy programu pro všechny trénovací data společně s jejich cílovou třídou. Tyto dvojce budou následně sestupně seřazeny dle výstupní hodnoty a jejich průchodem a posu-

nem po ose x při nalezení prvku třídy P a po ose y u vzorků třídy N dojde k pomyslnému vykreslení křivky ROC. Plocha pod touto křivkou pak bude vypočítána lichoběžníkovou metodou danou vzorcem:

$$\sum_{k=1}^{N} \frac{f(x_{k-1}) + f(x_k)}{2} \Delta x_k, \tag{7.3}$$

který počítá plochu jako sumu součtů sousedních funkčních hodnot, které jsou děleny dvěma a násobeny velikostí kroku, který je v tomto případě roven podílu jedné a počtu vstupů třídy P.

Volba rozhodovacích prahů

S výpočtem hodnoty AUC bude úzce souviset i výpočet prahů pro detekci jednotlivých tříd dyskineze. Ty budou zvoleny dodatečně po dokončení procesu učení klasifikátoru, a to zvlášť pro každou z tříd. Během tohoto procesu budou trénovací data rozdělena do čtyř skupin obsahujících vždy pouze danou kategorii projevů a data bez projevů dyskineze. Na těchto skupinách dat bude provedena klasifikace a výpočet fitness, kdy bude práh nalezen na křivce ROC pomocí metody geometrického průměru daného funkcí:

$$práh = argmax(\sqrt{TPR * (1 - FPR)}), \tag{7.4}$$

kde je hledán práh maximalizující TPR a zároveň minimalizující FPR a tím nabízející vyváženou schopnost klasifikace.

Mutace

Z první náhodně vytvořené generace bude výše uvedeným způsobem vybráno nejlepší řešení, ze kterého bude vytvořeno pět kopií. Tyto kopie budou upraveny pomocí Goldmanovy mutace, která provádí mutace náhodných částí chromozomu do doby než narazí na aktivní gen, kdy proces mutace ukončí. Při výběru nejlepšího řešení v následujících generacích bude navíc uplatněna podmínka, kdy v případě více řešení se shodnou nejlepší fitness bude upřednostněno řešení vzniklé v této generaci.

7.4 Koevoluce prediktorů fitness

K urychlení vyhodnocení kvality klasifikátorů je navrhnuto použití prediktorů fitness, které umožňují urychlení vyhodnocení kvality klasifikátorů díky použití pouhé části trénovací sady. Tyto prediktory budou navíc mít proměnlivou velikost, která zaručí hodnocení klasifikátorů na ideálním množství vstupních dat.

Prediktory fitness

Prediktory fitness budou na začátku inicializovány náhodným pořadím všech trénovacích dat, ze kterých bude použita část definovaná velikostí prediktoru. Následovný vývoj prediktorů bude probíhat podle principu genetických algoritmů, kdy do další generace vždy postoupí čtvrtina nejlepších řešení, čtvrtina bude náhodně vygenerována a zbývající polovina bude získána pomocí turnajového výběru mezi dvojicemi prediktorů a modifikována za pomoci křížení a mutace.

Pro možnost výběru nejlepších prediktorů je nutná funkce udávající jejich fitness a to na základě schopnosti predikovat subjektivní fitness klasifikátorů co nejblíže jejich objektivní hodnotě vypočtené na celé trénovací sadě. Nejlepší z prediktorů je pak uložen do jednočlenného archivu nejlepšího prediktoru a je použit při hodnocení kandidátních řešení klasifikátoru.

Jelikož je potřeba zajistit, že každý ze vstupů bude v prediktoru obsažen právě jednou, bude porovnáno několik metod křížení zajišťujících tuto vlastnost. Vyzkoušeno bude jednobodové i dvoubodové křížení, které však vyžaduje následovné odstranění duplikovaných vstupů a doplnění chybějících a dále metody Ordered Crossover (OX) a Non-Wrapping Ordered Crossover (NWOX), které zajišťují obsažení každého vstupu právě jednou automaticky.

Po operaci křížení bude následovat mutace, která bude s danou pravděpodobností prohazovat pořadí náhodných vstupů.

Archiv trenérů fitness

Hodnocení přesnosti predikování fitness pomocí prediktorů nebude probíhat na populaci klasifikátorů, ale na archivu trenérů fitness. Tento archiv bude na začátku běhu inicializován náhodnými řešeními klasifikátorů, kterým je vypočtena objektivní hodnota fitness. Pět nejlépe ohodnocených jedinců zároveň poslouží jako úvodní populace kandidátních klasifikátorů.

Archiv trenérů fitness se skládá ze dvou částí. První část zůstává konstantní po celou dobu běhu programu kvůli udržení rozmanitosti řešení v archivu. Druhá část je průběžně obměňována vždy při nalezení řešení klasifikátoru s novou nejvyšší subjektivní hodnotou fitness. Tomuto řešení je následně vypočtena objektivní fitness pro zjištění, zda je celkově nejlepším řešením a je použito k nahrazení nejstaršího jedince v druhé části archivu.

Koevoluce populací a změna velikosti prediktorů

K evoluci prediktorů dojde vždy při nalezení kandidátního řešení klasifikátoru s novou nejvyšší hodnotou subjektivní fitness nebo po uplynutí maximálního počtu generací bez evoluce prediktorů.

Samotnou evoluci prediktorů fitness bude však ještě předcházet nalezení aktuálního nejlepšího prediktoru na sadě trenérů prediktorů fitness a výpočet jejich nové velikosti podle způsobu navrženého v Drahošová [6] a popsaného v kapitole 5.3, kdy po výpočtu aktuální rychlosti evoluce a chyby predikce podle vzorců 5.5 a 5.6 bude podle pravidel z tabulky 5.1 rozhodnuto o velikosti konstanty C, jíž bude vynásobena aktuální velikost prediktoru fitness. Následně bude provedena evoluce a nalezeno nejlepší řešení pro použití k hodnocení kandidátních klasifikátorů.

Kapitola 8

Implementace

Tato kapitola ve stručnosti popisuje základní informace o implementaci programu dle předešlého návrhu. Začíná výpisem použitých technologií a nástrojů včetně zdůvodnění jejich použití. Následně je popsána struktura vytvořeného programu, výstupu a způsobu řešení paralelizace. Kapitola je ukončena popisem pomocných skriptů sloužících k provádění následovných experimentů a jejich vyhodnocení.

8.1 Použité nástroje a technologie

Při výběru programovacího jazyka k vytvoření programu bylo především z důvodu rychlosti, která jak již bylo řečeno je hlavním problémem procesu učení genetických algoritmů, rozhodnuto o jazyce C++20. Nejnovější verze jazyka byla kromě důvodu aktuálnosti zvolena pro možnost využít všechny schopnosti jazyka, ze kterých bylo využito především třídy `std::span` popsané dále.

Ke tvorbě programu v tomto jazyce bylo využito vývojového prostředí CLion[1] od české společnosti JetBrains ve verzi 2020.3.3 z důvodů podpory jazyka C++20, osobním zkušenostem s vývojem v tomto nástroji, možnosti jeho volného využití pro studijní účely a možnosti využití programu CMake[2] pro jednoduchou přenositelnost mezi platformami. K verzování byl využit nástroj Git[3] ve spojení s webovou službou GitHub[4].

Vývoj proběhl z velké části na operačním systému Windows 10 s následnými úpravami pro Linuxové systémy, jmenovitě Ubuntu 20.04 a Red Hat Enterprise Linux 7, využívaný na superpočítači Barbora, který je součástí IT4Innovations národního superpočítačového centra spravovaného Technickou univerzitou v Ostravě a který byl využit pro pozdější běh experimentů. Program je tedy otestován pro překlad a běh na těchto operačních systémech, s pouhými drobnými omezeními při spuštění na Windows 10 v podobě měření času běhu algoritmu.

[1] https://www.jetbrains.com/clion/
[2] https://cmake.org/
[3] https://git-scm.com/
[4] https://github.com/

8.2 Struktura programu

Vytvořený program se skládá ze sedmi hlavních tříd:

- `main` - Hlavní třída programu.
- `MoveData` - Zpracování pohybových dat.
- `Classifier` - Klasifikátor projevů dyskineze.
- `Chromosome` - Chromozom kandidátního řešení.
- `PredictorsPopulation` - Populace prediktorů fitness.
- `FitnessPredictor` - Prediktor fitness.
- `Archive` - Archiv trenérů prediktorů fitness.

Třída `main` zajišťuje zpracování argumentů příkazové řádky a spuštění odpovídajících částí programu, kdy je možné buď spustit proces učení klasifikátoru zadáním jeho parametrů, cesty ke vstupním datům a případně parametrů koevoluce a požadavku pro paralelní zpracování nebo spustit pouhé vyhodnocení uloženého klasifikátoru ze souboru.

Načtení uloženého řešení ze souboru ve formátu XML je pak implementováno za pomocí knihovny RapidXml 1.13[5], která umožňuje jednoduché zpracování XML souborů a je poskytována pod licencí MIT. Knihovnu lze tedy využívat neomezeně pod podmínkou přiložení textu licence.

Pohybová data jsou zadána formou cesty k textovému souboru, který na každém řádku obsahuje název jednoho ze souborů pohybových dat umístěných ve shodném adresáři. Takto jsou pak nečteny zvláště data trénovací a testovací. Třída `MoveData` po načtení dle návrhu tyto data uloží do vektorů o datových typech `uint8_t`, `int8_t`, `uint16_t`, `int16_t`, `float` a `double`, ze který je použit typ zadaný uživatelem, přičemž využití libovolného datového typu je umožněno díky využití šablon funkcí.

Samotný proces evoluce klasifikátoru pak probíhá ve třídě `Classifier`. Ta obsahuje vektor prvků třídy `Chromosome` představujících kandidátní řešení. Ty obsahují i metody pro vytvoření výsledného programu a jeho spuštění.

Proces koevoluce je pak umožněn díky třídě `PredictorsPopulation` obsahující kromě populace prediktorů fitness i metody jejich evoluce včetně hodnocení a aktualizace délky. Prediktory fitness reprezentované třídou `FitnessPredictor` se skládají z vektorů celých čísel označujících indexy vstupních dat a pro ohodnocení kandidátních řešení klasifikátoru je vždy vygenerován vektor vstupních dat o dané velikosti.

Jelikož je každé ze vstupních dat umístěno ve vektoru různé délky, který bude načítán pomocí plovoucího okna, bylo potřeba zajistit rychlé a postupné čtení tohoto vektoru. Toho bylo docíleno díky třídě `std::span` umožňující bezpečné a rychlé předání části souvislé sekvence vektoru.

8.3 Paralelizace

Pro urychlení procesu učení klasifikátoru byly paralelizovány části programu představující největší zátěž a zároveň k tomu vhodné. K paralelizaci bylo využito třídy `std::thread`, která umožňuje jednoduchou tvorbu a spouštění vláken.

[5] http://rapidxml.sourceforge.net/

K nalezení nejdéle prováděných částí programu vhodných k paralelizaci byl využit nástroj společnosti Intel VTune[6], který umožňuje jednoduše nalézt části nejvíce zpomalující chod programu i ověřit kvalitu výsledné paralelizace.

Kromě načtení rozsáhlých vstupních dat byly paralelizovány především části programu provádějící vyhodnocení kvality kandidátních řešení klasifikátoru a prediktorů fitness a pak prvotní přidělení objektivní fitness všem trenérům fitness umístěným v archivu.

8.4 Pomocné skripty

Kromě samotného programu provádějícího evoluci klasifikátoru projevů dyskineze u pacientů s Parkinsonovou nemocí byla vytvořena také dvojce skriptů v jazyce Python ve verzi 3.8.6.

Skript `RunExperiments.py` slouží k automatickému opakovanému spouštění programu a ukládání výsledků do souborů pojmenovaných dle nastavených parametrů programu.

Skript `ParseResult.py` pak slouží k automatickému vyhodnocení uložených výsledků. Po zadání cesty k adresáři s výsledky automaticky spočítá průměrnou kvalitu jednotlivých nastavení parametrů, pro každé nastavení nalezne řešení s nejlepší fitness na trénovacích datech, vykreslí krabicové grafy pro přesnost klasifikace testovacích dat a pro časovou náročnost učení. Pro náhodné veličiny fitness na trénovacích datech a čas učení provede také mezi všemi nastaveními Mann-Whitneyův U test pro ověření shodnosti střední hodnoty.

[6] https://www.intel.com/software/products/vtune/

Kapitola 9

Ověření funkčnosti a experimenty

Po dokončení implementace navrženého programu byly provedeny experimenty ověřující jeho vlastnosti, které jsou popsány v této kapitole. Kapitola začíná ověřením funkčnosti řešení a následným hledáním co nejvýhodnějších parametrů. Po jejich nalezení je provedeno zhodnocení zrychlení procesu učení oproti variantě bez využití prediktorů fitness a porovnání schopnosti rozlišit mezi třídami oproti řešení navrženému v Lones [10]. Dále je provedeno porovnání několika metod křížení využitého při evoluci prediktorů fitness a kapitola je završena popisem experimentů zaměřených na využití odlišných datových typů.

Při experimentování byly využity dvě datové sady, zvláště pro trénování a testování klasifikátoru. Parametry těchto sad lze vidět v tabulce 9.1. Pokud není uvedeno jinak, byly výsledky všech experimentů získány pomocí třiceti spuštění programu.

	Trénovací data	Testovací data
Třída 0	1173	1588
Třída 1	442	895
Třída 2	575	628
Třída 3	240	179
Třída 4	25	361

Tabulka 9.1: Složení trénovací a testovací sady použité při experimentech. Hodnoty jsou již sníženy o data neobsahující potřebné minimum 32 záznamů.

Vzhledem k citlivé povaze klasifikovaných dat byl vývoj prováděn na reprezentativním zlomku datové sady a experimenty popsané v této kapitole byly spouštěny na celé datové sadě vedoucí této práce Ing. Michaelou Drahošovou Ph.D.

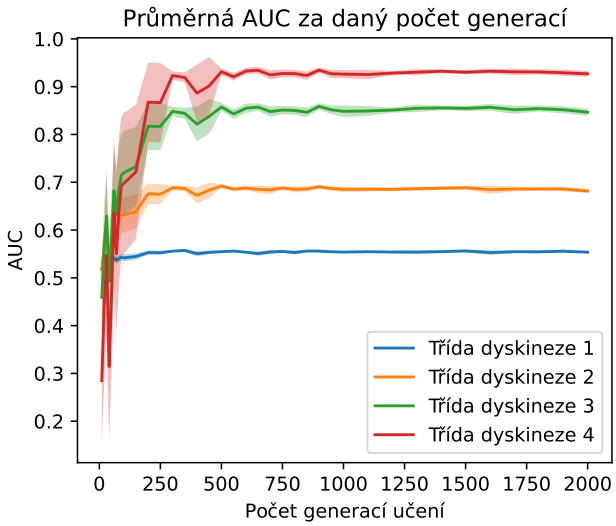
9.1 Experiment 1: Ověření funkčnosti řešení bez prediktorů fitness

Pro ověření funkčnosti řešení byly nejprve provedeny experimenty se základní variantou řešení nevyužívající koevoluci s prediktory fitness. Parametry kartézského genetického programování byly nastaveny na počáteční hodnoty uvedené v tabulce 9.2. S těmito parametry byl algoritmus spuštěn s parametrem maximálního počtu generací v rozmezí deseti až dvou tisíc, kdy výsledky lze vidět na obrázku 9.1. Vybarvené plochy podél křivek značí intervaly spolehlivosti s konfidenční hladinou 95 %.

Parametry	Zvolené hodnoty
Velikost plovoucího okna	32
Tvar kartézské mřížky	4x4, 5x5, 6x6, 1x16, 1x25, **1x36**, 1x40
L-back	Vždy plné propojení
Velikost populace kandidátních řešení	5
Datový typ použitý při výpočtu	float

Tabulka 9.2: Parametry prvního experimentu ověřujícího řešení bez prediktorů fitness a porovnávajícího rozměry kartézské mřížky. Tučně zvýrazněná hodnota označuje nejlepší nalezenou na základě výsledků experimentů.

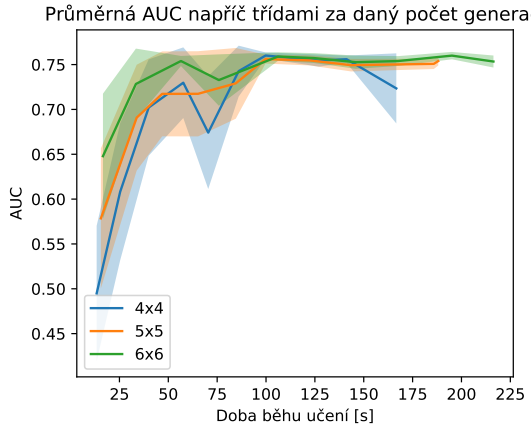
K výraznému zlepšení ve schopnosti klasifikovat testovací data dochází zhruba u 200 generací s průměrnou AUC pro jednotlivé kategorie 0,55, 0,68, 0,82 a 0,87. K získání stabilních výsledků dochází při zhruba 500 generacích s průměrnou AUC 0,55, 0,69, 0,86 a 0,93. Tyto výsledky jsou navíc srovnatelné s výsledky uvedenými v Lones [10] a značí tedy funkčnost implementovaného řešení.



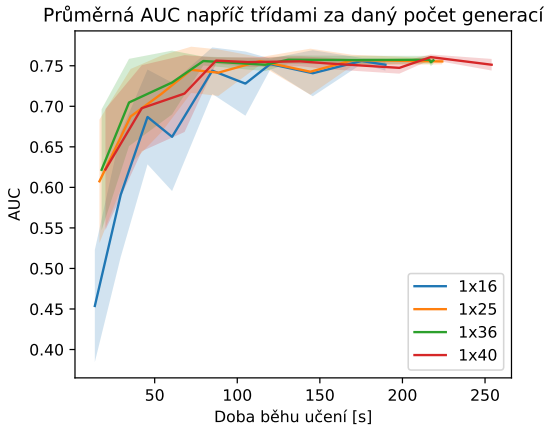
Obrázek 9.1: Vliv počtu generací učení na schopnost klasifikovat testovací data pro základní variantu programu s velikostí mřížky CGP 6x6 a L-back = 6.

Průměrná doba běhu učení počítaná pomocí CPU času při 200 generacích činila 33 vteřin a 106 vteřin pro 500 generací. Ve snaze dosáhnout srovnatelných výsledků za kratší dobu byly vyzkoušeny i další varianty CGP mřížky jako 4x4, 5x5, 1x16, 1x25 nebo 1x36. Vývoj fitness všech v průběhu doby učení lze vidět na obrázku 9.2.

Výsledky ukázaly, že nejrychleji jsou schopny dosáhnout řešení o srovnatelné kvalitě konfigurace 6x6 a 1x36. U varianty 6x6 je prvních výsledků o požadované kvalitě dosaženo za 56 vteřin, stabilně pak za 106 vteřin. U varianty 1x36 dochází k nalezení stabilních výsledků za 80 vteřin. Právě hodnota 80 vteřin posloužila později při hledání co nejvýhodnějších parametrů pro variantu využívající koevoluci s prediktory fitness jako časová hranice pro zajímavá nastavení.



(a) Čtvercový tvar mřížky CGP.



(b) Lineární tvar mřížky CGP.

Obrázek 9.2: Vývoj fitness několika vybraných parametrů CGP mřížky v průběhu učení.

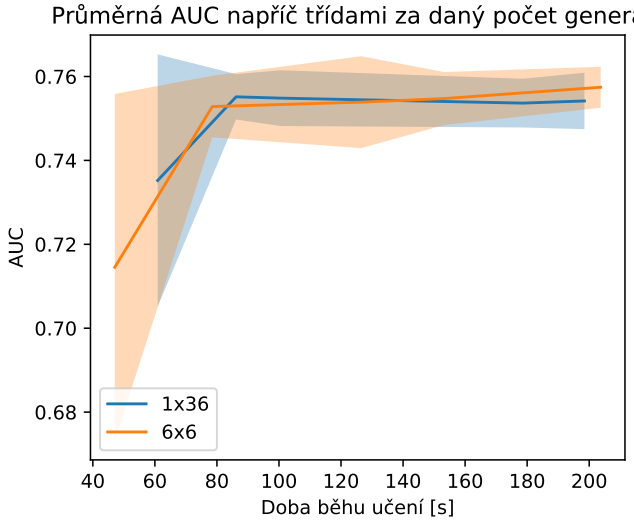
9.2 Experiment 2: Ověření funkčnosti řešení s prediktory fitness

K ověření funkčnosti řešení využívajícího koevoluci s prediktory fitness byly využity původní parametry CGP 6x6 a na základě výsledků z grafu 9.2 také 1x36. Parametry koevoluce pak byly zvoleny na základě zkušeností získaných v průběhu vývoje na hodnoty zobrazené v tabulce 9.3.

Parametry	Zvolené hodnoty
Velikost plovoucího okna	32
Tvar kartézské mřížky	6x6, **1x36**
L-back	Vždy plné propojení
Velikost populace kandidátních řešení	5
Datový typ použitý při výpočtu	float
Pravděpodobnost mutace genů prediktoru	0,01
Velikost populace prediktorů	8
Počáteční velikost prediktoru	100
Minimální velikost prediktoru	10
Maximální počet generací bez evoluce prediktorů	1/10 počtu generací
Práh chyby predikce	1,0
Velikost archivu trenérů fitness	20
Gentický operátor křížení prediktorů fitness	Jednobodové křížení

Tabulka 9.3: Parametry druhého experimentu ověřujícího řešení s prediktory fitness a porovnávající tvar kartézské mřížky. Tučně zvýrazněná hodnota označuje nejlepší nalezenou na základě výsledků experimentů.

I zde se potvrdila podobnost kvality obou nastavení mřížky, jak lze vidět na obrázku 9.3, s opětovně stabilnějšími výsledky u varianty 1x36, která byla schopná dosáhnout průměrné fitness 0,76 za dobu 86 vteřin oproti 203 vteřinám u varianty 6x6 způsobených větším rozptylem AUC.



Obrázek 9.3: Porovnání parametrů CGP mřížky pro řešení využívající prediktory fitness s úvodním nastavením ostatních parametrů.

Řešení s nejlepším průměrem fitness napříč všemi třídami na trénovacích datech získané u varianty 1x36 na 200 generacích dosáhlo pro testovací data AUC pro jednotlivé kategorie 0,55, 0,68, 0,84 a 0,92. Což je srovnatelný výsledek s řešením prezentovaným v Lonesovi [10] a opět tedy potvrzuje funkčnost řešení.

9.3 Experiment 3: Nastavení parametrů koevoluce

Navrhované řešení algoritmu zahrnuje řadu parametrů, které je potřeba při hledání co nejvýhodnějšího řešení správně nastavit. Z časových důvodů však nebyly některé parametry zkoumány a velikost plovoucího okna rovná 32 byla převzata z Lones [10]. Velikost populace pak byla nastavena podle evoluční strategie 1 + 4 na pět jedinců a pravděpodobnost mutace genů prediktorů fitness byla stanovena na 0,01.

Nastavení všech parametrů lze vidět v tabulce 9.4. Tvar kartézské mřížky byl určen podle výsledků předchozího experimentu 9.2. Zkoumány tedy byly zbylé parametry týkající se prediktorů fitness. Při jejich hledání byly voleny co nejvýhodnější hodnoty na základě schopnosti stabilně nacházet řešení, jejichž průměr AUC přes všechny čtyři kategorie činil alespoň 0,75 a tohoto výsledku byly dosaženo do času 80 vteřin. Počet generací kandidátních řešení byl vždy určován pro jednotlivé nastavení ostatních parametrů tak, aby bylo zaručeno stabilní nacházení kvalitních řešení a zároveň minimalizována doba učení.

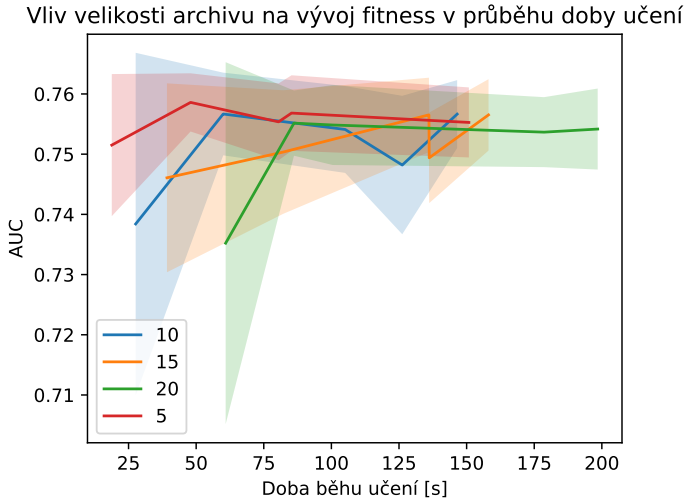
Prvním zkoumaným parametrem byla velikost archivu trenérů, která výrazně ovlivňuje čas učení. Zde byly porovnány varianty s velikostí 5, 10, 15 a 20, kdy vyšší hodnoty vedly již k příliš dlouhé době učení. Zde bylo zjištěno, že zvyšující se velikost archivu vede k nutnosti provedení většího počtu generací pro dosažení obdobných výsledků na testovacích datech a nejlepší a zároveň nejrychlejší je tedy využití minimální velikosti archivu o pěti jedincích. Výsledky lze vidět na obrázku 9.4.

Při porovnání různých velikostí populace prediktorů fitness byly z důvodu způsobu vytváření nové generace, kdy čtvrtina řešení jsou nejlepší řešení předchozí generace, čtvrtina je náhodně vygenerována a zbývající polovina je získána pomocí turnajového výběru a ná-

Parametry	Zvolené hodnoty
Velikost plovoucího okna	32
Velikosti kartézské mřížky	1x36
L-back	36
Velikost populace kandidátních řešení	5
Datový typ použitý při výpočtu	float
Pravděpodobnost mutace genů prediktoru	0,01
Velikost populace prediktorů	4, **8**, 12, 16
Počáteční velikost prediktoru	100
Minimální velikost prediktoru	10
Maximální počet generací bez evoluce prediktorů	1/10 počtu generací
Práh chyby predikce	1,0
Velikost archivu trenérů fitness	**5**, 10, 15, 20
Gentický operátor křížení prediktorů fitness	Jednobodové křížení

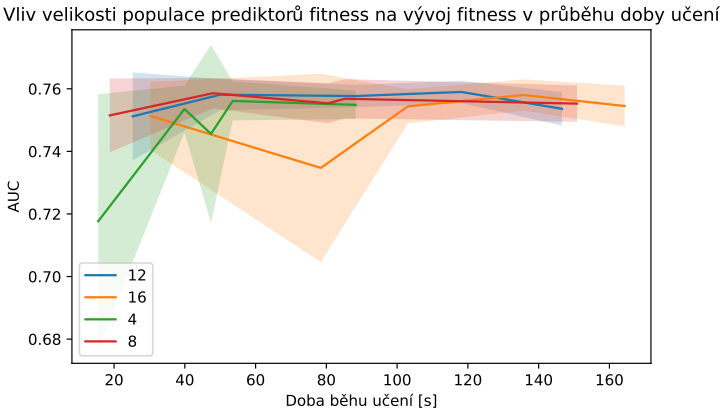
Tabulka 9.4: Úvodní nastavení parametrů pro experiment 3. Tučně zvýrazněné hodnoty označují nejlepší nalezené hodnoty na základě výsledků experimentů.

sledné modifikace za pomoci křížení a mutace, testovány násobky čtyř. Zde se ukázalo, že ideálních výsledků lze dosáhnout při populaci o velikosti osmi jedinců, jelikož menší populace není schopna dosáhnout požadované kvality a u větší populace dochází z důvodu větší výpočetní náročnosti k nalezení řešení za delší dobu, jak lze vidět na obrázku 9.5.



Obrázek 9.4: Porovnání vlivu velikosti archivu na vývoj fitness.

Extrémně důležitým se ukázala správná volba prahu chyby pro evoluci prediktorů fitness. Příliš malý práh vede k rychlému zvětšování velikosti prediktoru na maximální velikost a získávání kvalitních výsledků za cenu příliš pomalého procesu učení. Vysoký práh vede naopak k velmi rychlým, ale nekvalitním výsledkům. Jeho míra byla volena experimentálně společně s velikostí prediktorů a bylo rozhodnuto o hodnotě 1,014. Jako počáteční velikost prediktoru byla zvolena hodnota 300 z celkového počtu 1438 trénovacích vektorů použitých pro ohodnocení kandidátních řešení v přístupu bez koevoluce. Kolem této hodnoty se v této úloze po většinu doby velikost prediktorů pohybovala.



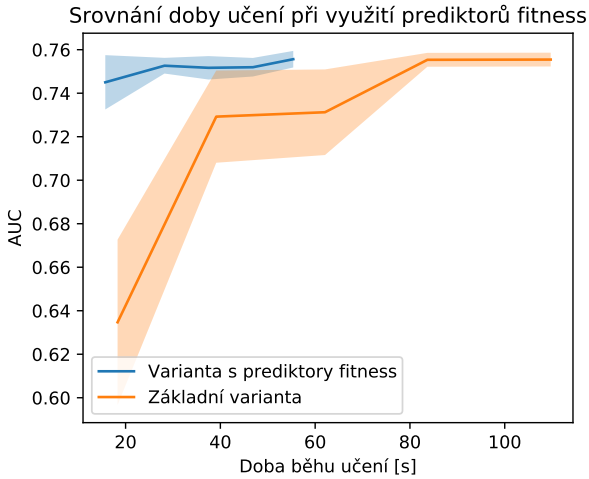
Obrázek 9.5: Porovnání vlivu velikosti populace prediktorů fitness na vývoj fitness.

Jako co nejvýhodnější počet generací kandidátních programů bez výměny prediktorů fitness se pak ukázala pětina celkového počtu generací, což při nastavení 500 generací kandidátních řešení znamená maximálně 100 generací bez evoluce prediktorů fitness. U příliš časté evoluce prediktorů se ukázalo, že vede ke zhoršení kvality řešení z důvodu přílišného zaměřování na aktuální stav řešení.

Po nalezení všech parametrů byly ještě provedeny pokusy o drobné změny a lokální optimalizaci parametrů, kdy bylo zjištěno, že mírně lepších výsledků je možné dosáhnout s archivem o velikosti šesti trenérů fitness.

9.4 Experiment 4: Zrychlení procesu učení

Po nalezení co nejvýhodnějších parametrů, zobrazených v tabulce 9.5, pro základní variantu programu i navrženou variantu využívající prediktory fitness bylo možné zjistit dosažené zrychlení procesu učení. Pro obě varianty byl počet spuštění algoritmu s jedním nastavením rozšířen na sto, aby bylo dosaženo přesnějších výsledků, které lze vidět na obrázku 9.6.

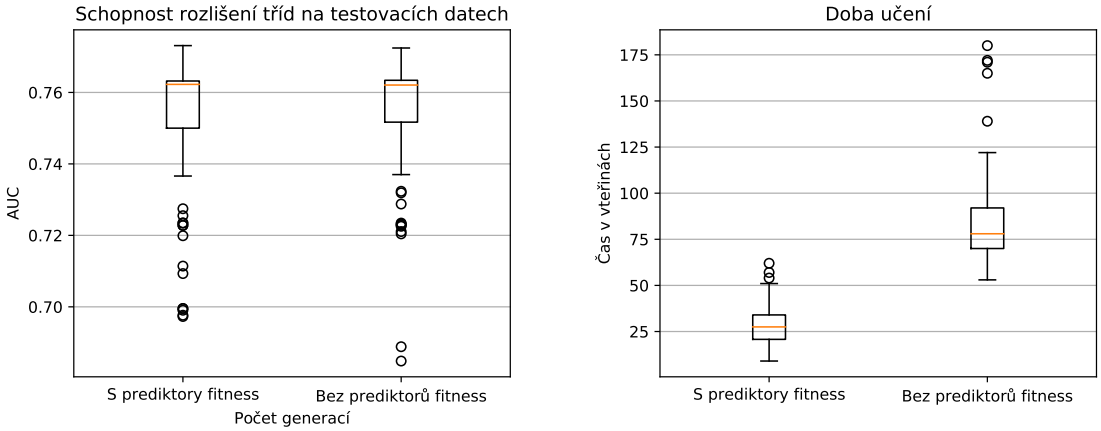


Obrázek 9.6: Porovnání průměrné schopnosti klasifikátorů rozlišit mezi třídami (AUC) a doby učení základní varianty využívající pouze CGP a varianty s prediktory fitness.

Parametry	S prediktory fitness	Bez prediktorů fitness
Velikost plovoucího okna	32	32
Velikosti kartézské mřížky	1x36	1x36
L-back	36	36
Velikost populace kandidátních řešení	5	5
Datový typ použitý při výpočtu	float	float
Počet generací	200	400
Pravděpodobnost mutace genů prediktoru	0,01	
Velikost populace prediktorů	8	
Počáteční velikost prediktoru	300	
Minimální velikost prediktoru	10	
Maximální počet generací bez evoluce prediktorů	1/5 počtu generací	
Práh chyby predikce	1,014	
Velikost archivu trenérů fitness	6	
Gentický operátor křížení prediktorů fitness	Jednobodové křížení	

Tabulka 9.5: Nejideálnější nalezené parametry obou variant programu

Výsledky ukazují značné zrychlení doby učení potřebné pro dosažení srovnatelných výsledků. Ty nastávají u 400. generace základní varianty a 200. generace varianty s prediktory fitness. Srovnatelná schopnost obou řešení klasifikovat testovací data lze jednoduše pozorovat na krabicovém grafu 9.7a a byla potvrzena i pomocí Mann-Whitney U testu při hladině pravděpodobnosti 0,05.



(a) Schopnost klasifikátorů rozlišit mezi třídami (AUC).

(b) Doba učení.

Obrázek 9.7: Porovnání schopnosti klasifikátorů rozlišit mezi třídami (AUC) a doby učení pro nejlepší nalezené parametry pro variantu programu s a bez využití prediktorů fitness.

Urychlení procesu učení těchto dvou srovnatelně kvalitních řešení lze pak dobře vidět na obrázku 9.7b, který ukazuje, že využití prediktorů fitness vedlo v medianu, ale i v průměru,

ke trojnásobnému zrychlení. Přesné výsledky porovnání si pak lze prohlédnout níže v tabulce 9.6.

Varinata	Hodnota	AUC	Čas [s]
S prediktory fitness	Min	0,70	9
	Průměr	0,75	28
	Median	0,76	28
	Max	0,77	62
Bez prediktorů fitness	Min	0,68	53
	Průměr	0,76	84
	Median	0,76	78
	Max	0,77	180

Tabulka 9.6: Základní statistické charakteristiky popisující dobu učení obou variant a schopnost klasifikátoru rozlišit mezi třídami (AUC) na testovacích datech.

9.5 Experiment 5: Porovnání kvality s existujícím řešením

Při porovnání řešení s řešením navrženým v Lones [10] bohužel není možné srovnat dobu učení z důvodu jejího neuvedení. Porovnat však lze alespoň kvalitu řešení společně s jeho velikostí a složitostí.

Při shodné volbě řešení s nejvyšší fitness na trénovacích datech a porovnání jeho schopnosti klasifikovat dosud neviděná data testovací sady bylo dosaženo hodnot srovnatelných s výsledky řešení Lonese [10], jehož výsledky lze vidět i na obrázku 6.3. Porovnání lze vidět v tabulce 9.7.

	Třída 1	Třída 2	Třída 3	Třída 4	Průměr
Navržené řešení	0,56	0,69	0,86	0,94	0,76
Existující řešení	0,56	0,69	0,85	0,93	0,76

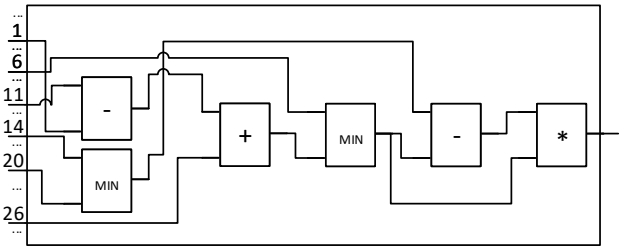
Tabulka 9.7: Porovnání schopnosti navrženého klasifikátoru rozlišit mezi jednotlivými třídami (AUC) .

Srovnatelných výsledků bylo dosaženo také v otázce schopnosti klasifikace zvláště pacientů sedících a pacientů chodících uvedených v tabulce 9.8. U chodících pacientů byla dosažena fitness 0,70 mírně zaostávající za Lonesovými [10] 0,73. U sedících pacientů byla dosažena srovnatelná hodnota 0,93 oproti Lonesovým [10] 0,92.

	Chodící pacienti	Sedící pacienti
Navržené řešení	0,70	0,93
Existující řešení	0,73	0,92

Tabulka 9.8: Porovnání schopnosti navrženého a existujícího řešení klasifikovat zvláště data sedících a chodících pacientů.

Získané řešení lze vidět na obrázku 9.8. Oproti předešlému řešení se skládá ze šesti místo osmi funkčních bloků a především neobsahuje náročné operace dělení.



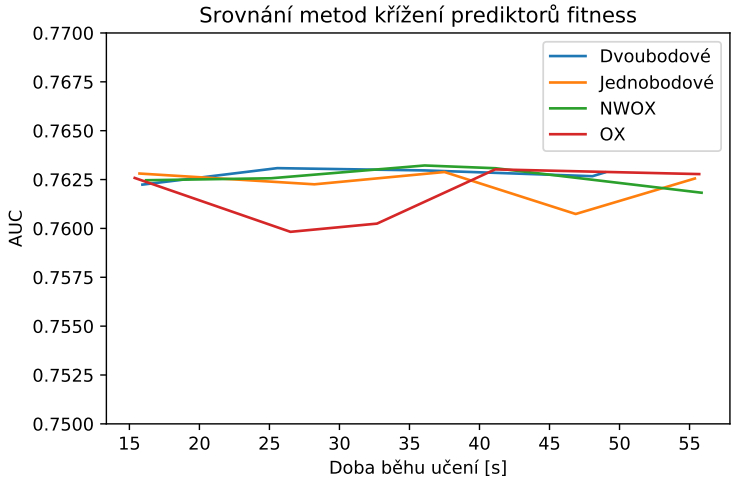
Obrázek 9.8: Aktivní část nejlepšího nalezeného řešení varianty využívající koevoluci prediktorů fitness.

9.6 Experiment 6: Porovnání metod křížení prediktorů fitness

Základní varianta navrženého procesu učení navržená v Drahošová [6] využívá při křížení prediktorů fitness jednobodové křížení s bodem křížení v rámci aktivní části. Obsažení každého trénovacího vektoru právě jednou je pak zajištěno průchodem a odstraněním opakovaných výskytů a doplněním chybějících na konec chromozomu. Pro zvýšení efektivity byly navrženy ještě alternativní metody křížení v podobě dvoubodové varianty a metod Ordered Crossover (OX) a Non-Wrapping Ordered Crossover (NWOX).

U všech alternativních variant bylo provedeno 100 spuštění s maximálním množství generací v rozmezí 100 až 500. Ostatní parametry byly zvoleny podle nejlepších nalezených parametrů pro základní variantu algoritmu a sledována byla schopnost nalezených řešení rozlišit mezi třídami s přihlédnutím na čas nutný k jejich získání.

V průběhu experimentů nebylo nalezeno řešení lepší než u již prezentovaného řešení při využití základního jednobodového křížení. Kvalita ostatních řešení se pak ukázala srovnatelná s pouhým drobnými rozdíly v rozmezí 200 a 400 generací, kdy u 500. generace dochází již k nalezení shodně kvalitních prediktorů u všech variant, jak lze vidět na obrázku 9.9.

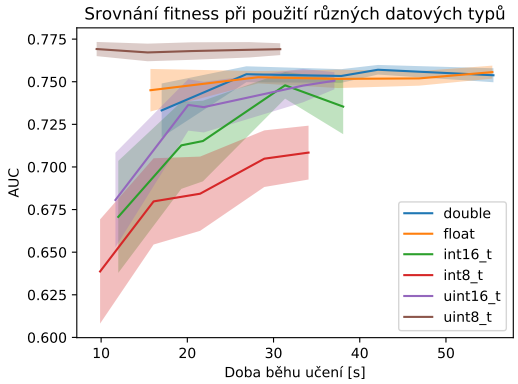


Obrázek 9.9: Porovnání medianu fitness řešení nalezených v průběhu času jednotlivými variantami křížení prediktorů fitness.

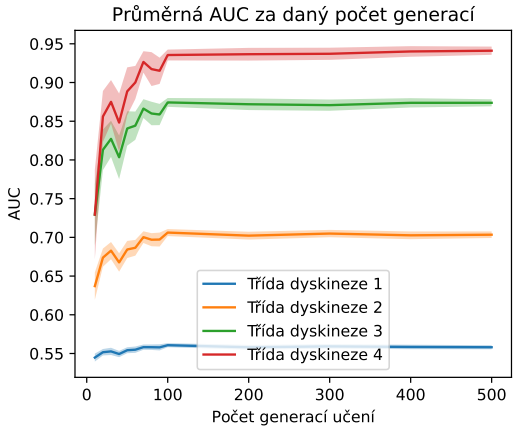
Metoda OX se ukázala jako nejméně vhodná a to pravděpodobně z důvodu velkých posunů způsobených tímto křížením, které výrazně zaměňuje aktivní a neaktivní části chromozomů. Upravená varianta NWOX se naopak ukázala býti vhodná a to pravděpodobně právě díky zamezení velkých posunů prvků v rámci chromozomu. Podobných výsledků pak dosáhly i metody jednobodového a dvoubodového křížení.

9.7 Experiment 7: Porovnání datových typů

Kromě základní varianty využívající při výpočtech čísla s plovoucí desetinnou čárkou implementované datovým typem float byly provedeny experimenty i s celočíselnými datovými typy uint8_t, uint16_t, int8_t a int16_t vhodnějšími pro implementaci v hardwaru a datovým typem double zajišťujícím vyšší přesnost výpočtů.



(a) Porovnání průměrné fitness řešení nalezených v průběhu čas při použití různých datových typů.



(b) Vývoj fitness řešení při využití datového typu uint8_t.

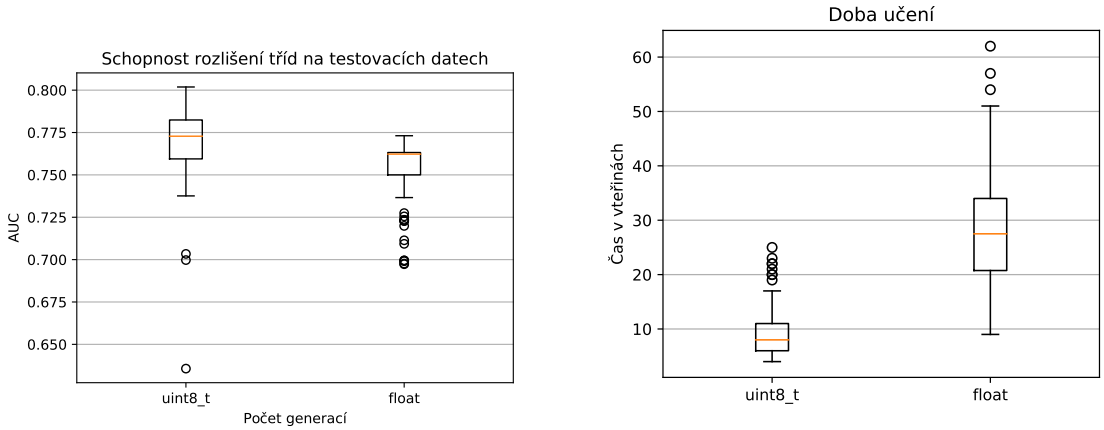
Obrázek 9.10: Porovnání schopnosti klasifikátoru rozlišit mezi třídami při využití různých datových typů.

Výsledky zobrazené na obrázku 9.10a ukázaly, že datový typ double nepřináší žádnou výraznou změnu kvality ani času běhu učení. K mírnému zhoršení kvality pak dochází při využití typů int16_t a uint16_t s velkým poklesem kvality u typu int8_t. Překvapivou se ukázala varianta uint8_t, která dosáhla nejlepších výsledků ze všech variant a to za výrazně kratší dobu učení.

Při bližším pohledu na vývoj kvality průměrného řešení varianty s datovým typem uint8_t na obrázku 9.10b lze vidět, že k nalezení řešení o stabilní kvalitě dochází po zhruba 100 generacích, které této variantě trvají v průměru pouhých 9 vteřin, což je v průměru téměř devítinásobné a v medianu více než desetinásobné, zrychlení oproti variantě bez prediktorů fitness s datovým typem float.

Při porovnání s řešením využívajícím standardní datový typ float a výsledky prezentovanými v podkapitole 9.5 zjistíme, že je toto řešení schopno dosahovat kvalitnějších výsledků. To lze pozorovat na následujícím obrázku 9.11a, který zobrazuje výsledky získané na 100 bězích návrhu klasifikátoru s 200 generacemi pro variantu s datovým typem float a 100 generacemi pro variantu s datovým typem uint8_t. Porovnání doby učení, které bylo zrychleno více než třikrát, lze pak vidět na obrázku 9.11b.

Řešení s nejlepší fitness dosaženou na trénovacích datech pak nabývá na testovacích datech hodnot fitness pro jednotlivé míry dyskineze 0,58, 0,72, 0,84 a 0,93, což ukazuje na srovnatelné schopnosti řešení klasifikovat příznaky nemoci. U dat sedících pacientů pak byla dosažena AUC 0,92 a 0,80 u pacientů chodících, což je výrazné zlepšení jak oproti variantě využívající datový typ float s hodnotou 0,70 tak oproti řešení prezentovanému v Lones [10] s hodnotou AUC 0,73.

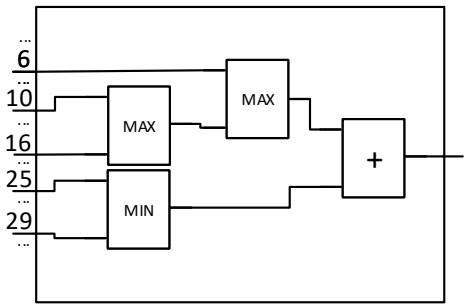


(a) Schopnost klasifikátorů rozlišit mezi třídami (AUC).

(b) Doba učení.

Obrázek 9.11: Porovnání schopnosti klasifikátorů rozlišit mezi třídami (AUC) a doby učení při použití datového typu uint8_t po 100 generacích učení a datového typu float po 200 generacích.

Podobu řešení tvořenou 4 aktivními uzly lze vidět níže na obrázku 9.12. Kompletní porovnání schopnosti klasifikátorů rozlišit mezi třídami lze pak vidět v příloze B.



Obrázek 9.12: Nejlepší nalezené řešení pro variantu využívající datový typ uint8_t.

Kapitola 10

Závěr

V rámci této práce bylo navrhnuto řešení problému automatizovaného návrhu programu pro detekci projevů dyskineze z pohybových dat pacientů. Navržený způsob evolučního návrhu klasifikátoru využívá kartézské genetické programování, které bylo z důvodu urychlení procesu návrhu řešení doplněno o koevoluci prediktorů fitness s proměnlivou velikostí, která umožňuje vyhodnocení kvality kandidátních řešení na pouhé části trénovacích dat.

Po implementaci návrhu, vycházejícího z nastudované literatury, byla provedena řada experimentů, která ověřila schopnosti klasifikátoru rozlišit mezi třídami (AUC) na srovnatelné úrovni s existujícími řešeními. Dále bylo experimentálně ověřeno, že využití koevoluce s prediktory fitness s proměnlivou velikostí umožňuje, při zachování schopnosti klasifikace, v průměru téměř trojnásobné zrychlení procesu učení oproti využití pouhého kartézského genetického programování.

Experimenty s genetickým operátorem křížení použitého u prediktorů fitness neukázaly významnější rozdíl mezi zvolenými metodami. Experimenty s datovými typy naopak ukázaly, že využití datového typu uint8_t místo výchozího datového typu float, použitého v existujícím řešení, vede k dosažení stabilnějších výsledků o srovnatelné schopnosti rozlišení mezi třídami (AUC) a výrazně lepší schopnosti klasifikace dat pohybujících se pacientů. Využití datového typu uint8_t navíc vedle v průměru téměř k devítinásobnému zrychlení procesu učení oproti základní variantě bez prediktorů fitness využívající datový typ float.

Díky tvorbě této diplomové práce jsem získal možnost lépe poznat a především si oblíbit obor evolučních algoritmů v jehož studiu bych rád dále pokračoval. Měl jsem totiž možnost vyzkoušet si, že i přes výrazně větší míru pozornosti věnovanou neuronovým sítím, skýtají přírodou inspirované algoritmy řadu zajímavých algoritmů k objevování, mezi které bezesporu spadá i myšlenka genetického programování či pro mě dříve zcela neznámé spojení vývoje vícero populací.

V rámci dalšího pokračování práce je navrhnuto například využití informace o umístění senzoru jako dalšího vstupu klasifikátoru pro umožnění jednoduššího přizpůsobení klasifikace pro různé části těla. Dále kombinace vstupních dat nejenom z akcelerometru ale také z gyroskopu. V neposlední řadě pak především hlubší prozkoumání vlastností řešení využívajícího datový typ uint8_t a příčiny vyšší úspěšnosti klasifikace při jeho použití.

Literatura

[1] Abdoun, O. a Abouchabaka, J. A Comparative Study of Adaptive Crossover Operators for Genetic Algorithms to Resolve the Traveling Salesman Problem. *International Journal of Computer Application*. Ithaca: Cornell University Library. 2012, sv. 31, č. 11, s. 49–57. ISSN 2331-8422.

[2] Burkov, A. *The Hundred-Page Machine Learning Book*. Quebec, Canada: Andriy Burkov, 2019. ISBN 978-1-9995795-0-0.

[3] Cai, X., Smith, S. L. a Tyrrell, A. M. Positional independence and recombination in cartesian genetic programming. In: *Proceedings of the 9th European Conference on Genetic Programming*. Berlin, Heidelberg: Springer, 2006, sv. 3905, s. 351–360. EuroGP'06. ISBN 03029743.

[4] Cicirello, V. A. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*. Association for Computing Machinery, červenec 2006, sv. 2, s. 1125–1132. GECCO '06. ISBN 1595931864.

[5] De Jong, K. Generalized Evolutionary Algorithms. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing*. Berlin, Heidelberg: Springer, 2012, s. 625–635. ISBN 978-3-540-92909-3.

[6] Drahošová, M., Sekanina, L. a Wiglasz, M. Adaptive Fitness Predictors in Coevolutionary Cartesian Genetic Programming. *Evolutionary Computation*. MIT Press. 2019, sv. 27, č. 3, s. 497–523. ISSN 1063-6560.

[7] Goldberg, D. E. *Genetic Algorithms in Search, Optimization and Machine Learning*. 1. vyd. Boston, MA, United States: Addison-Wesley Longman Publishing Co., Inc., 1989. ISBN 978-0-201-15767-3.

[8] James, G., Witten, D., Hastie, T. a Tibshirani, R. *An introduction to statistical learning : with applications in R*. 1. vyd. New York, NY: Springer, 2013. Springer texts in statistics. ISBN 978-1-4614-7137-0.

[9] Jašíčková, K. *Klasifikace obrazu pomocí genetického programování*. Brno, CZ, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Sekanina, L.

[10] Lones, M., Alty, J., Cosgrove, J., Duggan Carter, P., Jamieson, S. et al. A New Evolutionary Algorithm-Based Home Monitoring Device for Parkinson's Dyskinesia. *Journal of Medical Systems*. New York: Springer US. 2017, sv. 41, č. 11, s. 1–8. ISSN 0148-5598.

[11] Miller, J. F. *Cartesian Genetic Programming.* 1. vyd. Berlin, Heidelberg: Springer-Verlag, 2011. Natural Computing Series. ISBN 978-3-642-17309-7.

[12] Popovici, E., Bucci, A., Wiegand, R. P. a De Jong, E. D. Coevolutionary Principles. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing.* Berlin, Heidelberg: Springer, 2012, s. 987–1033. ISBN 978-3-540-92909-3.

[13] Segaran, T. *Programming collective intelligence : building smart web 2.0 applictions.* 1. vyd. Sebastopol, CA: O´Reilly, 2007. ISBN 978-0-596-52932-1.

[14] Sekanina, L. *Evoluční hardware : od automatického generování patentovatelných invencí k sebemodifikujícím se strojům.* 1. vyd. Praha: Academia, 2009. Gerstner ; sv. 4. ISBN 978-80-200-1729-1.

[15] Tichá, P. *Akvizice a klasifikace pohybu.* Brno, CZ, 2017. Diplomová práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav biomedicínského inženýrství. Vedoucí práce Janoušek, O.

[16] Vanneschi, L. a Poli, R. Genetic Programming — Introduction, Applications, Theory and Open Issues. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing.* Berlin, Heidelberg: Springer, 2012, s. 709–739. ISBN 978-3-540-92909-3.

[17] Whitley, D. a Sutton, A. M. Genetic Algorithms — A Survey of Models and Methods. In: Rozenberg, G., Bäck, T. a Kok, J. N., ed. *Handbook of Natural Computing.* Berlin, Heidelberg: Springer, 2012, s. 637–671. ISBN 978-3-540-92909-3.

Příloha A

Příklad výstupu programu

```
<Run number="0">
<Training>
<NewBest Generation="0" AUC="0.79" PredictorLength="500"/>
<NewBest Generation="6" AUC="0.73" PredictorLength="450"/>
<NewBest Generation="10" AUC="0.84" PredictorLength="648"/>
<NewBest Generation="33" AUC="0.91" PredictorLength="904"/>
<NewBest Generation="336" AUC="0.92" PredictorLength="1256"/>
</Training>
<Time CPU="126" Elapsed="38"/>
<Thresholds One="6.47128" Two="8.17902" Three="14.2702" Four="19.872"/>
<TrainResults>
<Result Level="1" AUC="0.74" TP="350" TN="718" FP="455" FN="92" TPR="0.79" TNR="0.61" F1="0.56"/>
<Result Level="2" AUC="0.85" TP="514" TN="775" FP="398" FN="61" TPR="0.89" TNR="0.66" F1="0.69"/>
<Result Level="3" AUC="0.91" TP="213" TN="917" FP="256" FN="27" TPR="0.89" TNR="0.78" F1="0.60"/>
<Result Level="4" AUC="0.96" TP="24" TN="980" FP="193" FN="1" TPR="0.96" TNR="0.84" F1="0.20"/>
<ActivityResult SittingAUC="0.94" WalkingAUC="0.88"/>
</TrainResults>
<TestResults>
<Result Level="1" AUC="0.57" TP="246" TN="1206" FP="382" FN="649" TPR="0.27" TNR="0.76" F1="0.32"/>
<Result Level="2" AUC="0.69" TP="214" TN="1290" FP="298" FN="414" TPR="0.34" TNR="0.81" F1="0.38"/>
<Result Level="3" AUC="0.83" TP="74" TN="1444" FP="144" FN="105" TPR="0.41" TNR="0.91" F1="0.37"/>
<Result Level="4" AUC="0.91" TP="212" TN="1501" FP="87" FN="149" TPR="0.59" TNR="0.95" F1="0.64"/>
<ActivityResult SittingAUC="0.89" WalkingAUC="0.90"/>
</TestResults>
<Solution>
<Node Index="48" First="5" Second="26" Function="5"/>
<Node Index="59" First="15" Second="18" Function="5"/>
<Node Index="66" First="59" Second="48" Function="1"/>
<Node Index="68" First="66" Second="-1" Function="-1"/>
</Solution>
</Run>
```

Příklad výstupu programu ve formátu XML. První část obsahuje informace o průběhu evoluce v podobě průběhu nejlepší fitness a aktuální velikosti prediktoru fitness. Tato část je ukončena informací o času běhu učení. CPU čas je počítán napříč vlákny a pouze v momenty, kdy procesor provádí běh programu. Uběhnutý čas je pak čas vnímaný uživatelem.

Následují část je uvozena nalezenými prahy detekce jednotlivých tříd dyskineze a pokračuje informacemi o kvalitě řešení na trénovacích i testovacích datech pro všechny tyto úrovně. Pro schopnost klasifikace každé z úrovní jsou vypsány základní statistiky v podobě hodnot AUC, TP, TN, FP, FN, TPR, TNR a F1. Kromě toho jsou vypsány i průměrné AUC získané zvlášť při klasifikaci dat pořízených za chůze a při sezení pacientů na datech kategorie nula (N) a tři a čtyři (P).

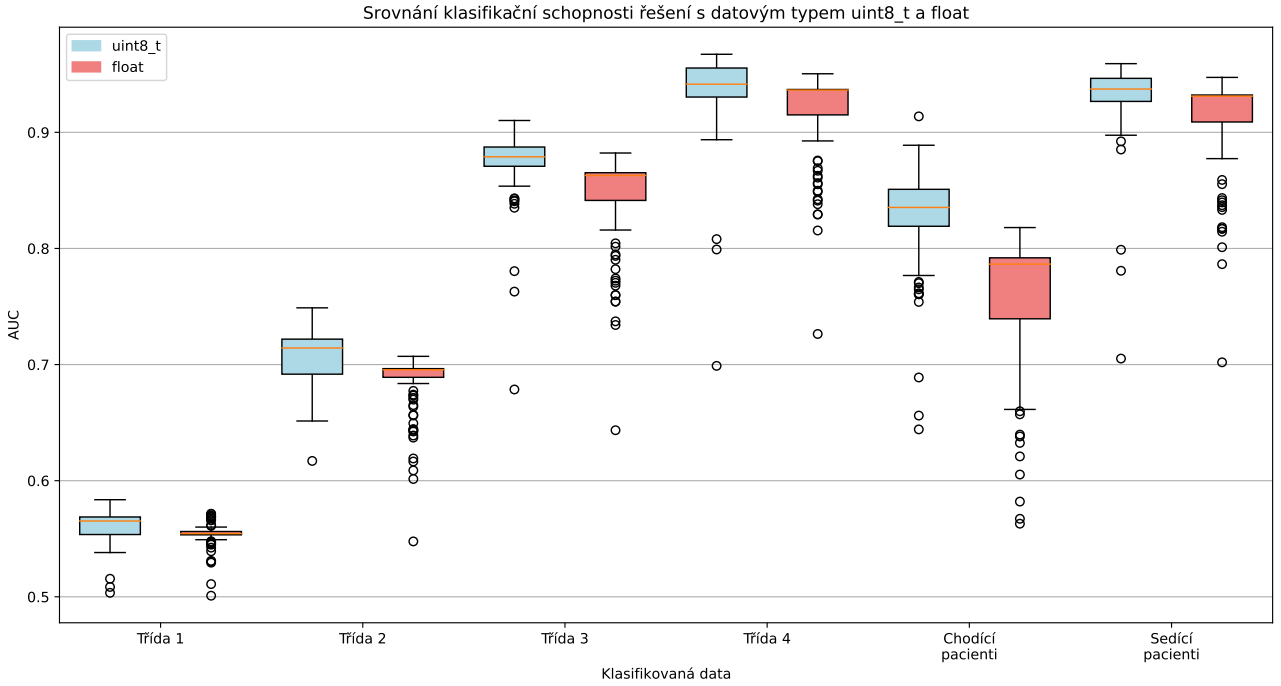
Výstup je zakončen výpisem aktivních uzlů řešení s jejich indexy, funkcemi a indexy vstupů.

Příloha B

Výsledky nejlepších nalezených parametrů

Varianta		Řešení s datovým typem uint8_t	Řešení s datovým typem float
Třída 1	Min	0,50	0,50
	Průměr	0,56	0,55
	Median	0,57	0,55
	Max	0,58	0,57
Třída 2	Min	0,62	0,55
	Průměr	0,71	0,68
	Median	0,71	0,70
	Max	0,75	0,71
Třída 3	Min	0,68	0,64
	Průměr	0,87	0,84
	Median	0,88	0,86
	Max	0,91	0,88
Třída 4	Min	0,70	0,73
	Průměr	0,94	0,92
	Median	0,94	0,94
	Max	0,97	0,95
Chodící pacienti	Min	0,64	0,56
	Průměr	0,83	0,76
	Median	0,84	0,79
	Max	0,91	0,82
Sedící pacienti	Min	0,71	0,70
	Průměr	0,93	0,91
	Median	0,94	0,93
	Max	0,96	0,95

Tabulka B.1: Srovnání schopnosti rozlišit mezi třídami (AUC) výsledných řešení s datovým typem uint8_t a float pro testovací data jednotlivých tříd a zvláště pro rozdělená data sedících a chodících pacientů.



Obrázek B.1: Srovnání klasifikačních schopností řešení využívajících datový typ uint8_t a float.