Metody optimalizace a A.I. pro měření v průmyslové praxi

Bc. et Bc. Jana Šimečková

Diplomová práce
2024

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Akademický rok: 2023/2024

Studijní program: Informační technologie
Forma studia: Kombinovaná
Specializace/kombinace: Softwarové inženýrství (knIT-SWI)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení: **Bc. et Bc. Jana Šimečková**
Osobní číslo: **A22577**

Téma práce: **Metody optimalizace a A.I. pro měření v průmyslové praxi**
Téma práce anglicky: **Optimization Methods and A.I. for Measurements in Industrial Practice**
Jazyk práce: **Čeština**

Vedoucí práce: **doc. Ing. Michal Pluháček, Ph.D.**
Ústav informatiky a umělé inteligence

Zásady pro vypracování:

1. Vypracujte literární rešerši na dané téma.
2. Proveďte měření a sběr dat na reálném zařízení pro analýzu a syntézu modelu.
3. Za pomocí metod umělé inteligence syntetizuje model sledovaného systému.
4. Na modelu systému proveďte komparativní analýzu vybraných optimalizačních metod.
5. Proveďte srovnání výsledků optimalizace na modelu a reálném systému.

Seznam doporučené literatury:

1. ZELINKA, Ivan; SNÁŠEL, Václav a ABRAHAM, Ajith (ed.), 2013. Handbook of Optimization. Online. Intelligent Systems Reference Library. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-30503-0. Dostupné z: https://doi.org/10.1007/978-3-642-30504-7.
2. ŽEGKLITZ, Jan a POŠÍK, Petr, 2021. Benchmarking state-of-the-art symbolic regression algorithms. Genetic Programming and Evolvable Machines. Roč. 22, č. 1, s. 5-33. ISSN 1389-2576. Dostupné z: https://doi.org/10.1007/s10710-020-09387-0.
3. ZELINKA, Ivan, 2009. Evoluční výpočetní techniky: principy a aplikace. Praha: BEN – technická literatura. ISBN 978-80-7300-218-3.
4. PERES, Ricardo Silva; JIA, Xiaodong; LEE, Jay; SUN, Keyi; COLOMBO, Armando Walter et al., 2020. Industrial Artificial Intelligence in Industry 4.0 – Systematic Review, Challenges and Outlook. IEEE Access. Roč. 8, s. 220121-220139. ISSN 2169-3536. Dostupné z: https://doi.org/10.1109/ACCESS.2020.3042874.
5. JAVAID, Mohd; HALEEM, Abid; SINGH, Ravi Pratap a SUMAN, Rajiv, 2022. Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study. Journal of Industrial Integration and Management. Roč. 07, č. 01, s. 83-111. ISSN 2424-8622. Dostupné z: https://doi.org/10.1142/S2424862221300040.

Podpis studenta: Datum:

Podpis vedoucího práce: Datum:

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne

Jana Šimečková v.r.
podpis studenta

ABSTRAKT

Tato diplomová práce se zaměřuje na aplikaci analytického programování, evolučních algoritmů a dalších metod umělé inteligence a strojového učení v kontextu optimalizace a modelování v průmyslové praxi. Práce je založena na pečlivé analýze relevantních zdrojů a jejím primárním cílem je provést odbornou rešerši na dané téma v teoretické části a aplikovat popsané metody v části praktické, včetně zhodnocení provedených experimentů. V rámci teoretické části byly podrobně prozkoumány různé aspekty evolučních výpočetních technik, analytického programování, neuronových sítí a dalších metod využitých při zpracování dat. Praktická část práce se skládá z popisu dat získaných z fyzikálního měření, analýzy možností syntézy modelu popisující tato data a optimalizace nastavení parametrů měření pomocí evolučních výpočetních technik. Byly provedeny rozsáhlé experimenty a analýzy, které přispěly k hlubšímu porozumění dané problematiky a poskytly cenné poznatky pro další výzkum v oblasti aplikace umělé inteligence a strojového učení v průmyslové praxi.

Klíčová slova: analytické programování, umělá inteligence, evoluční algoritmy, strojové učení, optimalizace

ABSTRACT

This master's thesis focuses on the application of analytical programming, evolutionary algorithms, and other methods of artificial intelligence and machine learning in the context of optimization and modeling in industrial practice. The work is based on a meticulous analysis of relevant sources, with its primary aim being to conduct an expert literature review on the given topic in the theoretical part and to apply the described methods in the practical part, including the evaluation of conducted experiments. In the theoretical section, various aspects of evolutionary computing techniques, analytical programming, neural networks, and other data processing methods were thoroughly examined. The practical part of the work consists of a description of data obtained from physical measurements, an analysis of the possibilities of synthesizing a model describing this data, and the optimization of parameters settings using evolutionary computing techniques. Extensive experiments and analyses were carried out, contributing to a deeper understanding of the issue and providing valuable insights for further research in the field of artificial intelligence and machine learning application in industrial practice.

Keywords: analytical programming, artificial intelligence, evolutionary algorithms, machine learning, optimization

Na tomto místě bych moc ráda poděkovala vedoucímu mé práce docentu Michalu Pluháčkovi za odborné vedení práce a cenné rady. Za přínosné konzultace a zprostředkování doménové znalosti děkuji doktoru Michalu Lackovi. Velký dík patří rovněž Martinu Gaidosovi za veškerou podporu, kterou mi poskytl během studií v osobním životě.

Prohlašuji, že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Tato diplomová práce prezentuje detailní a multidisciplinární výzkum, zaměřující se na aplikaci analytického programování, evolučních algoritmů a dalších metod AI a strojového učení v rámci optimalizace a modelování v průmyslové praxi. Práce je založena na pečlivé analýze relevantních zdrojů a jejím primárním cílem je provést odbornou rešerši na dané téma v teoretické části a aplikovat popsané metody v části praktické, včetně zhodnocení provedených experimentů. V rámci rešerše je hlavní pozornost věnována tématu evolučních výpočetních technik, analytickému programování, neuronovým sítím a dále popisu metod využitých při zpracování dat v praktické části. Praktická část práce se skládá z popisu dat získaných z fyzikálního měření, analýzu možností syntézy modelu popisujícího tato data a optimalizace parametrů měření, jejíž výsledky jsou porovnány s reálným systémem.

V rámci této práce je klíčovým aspektem analýza a porozumění datům získaným z fyzikálního měření. Dostatečně velká část dat pro zreprodukování popsaných experimentů je uložena v elektronické podobě společně s prací. Tato data jsou klíčové pro syntézu modelu a jejich správné pochopení je nezbytné pro úspěšné provedení zadání práce. K tomu bylo nutné přistoupit k důkladné statistické analýze a předzpracování dat, které zahrnuje škálování, melting, clustering a další úpravy, které umožňují vytvořit dataset vhodný pro další zpracování.

Práce se dále věnuje aplikaci konceptů analytického programování a evolučních algoritmů. Tyto metody jsou implementovány v programovacím jazyce Python s využitím vhodných knihoven. Algoritmy jsou použity pro rozsáhlou analýzu možností syntézy modelu popisujícího naměřená data. Výsledky mnoha experimentů jsou shrnuty do celku tvořícího odbornou analýzu daného problému, včetně interpretace získaných výsledků a navržení dalších možných postupů v návaznosti na tento výzkum. Společně s analytickým programováním jsou zkoumány také možnosti syntézy modelu pomocí tréninku neuronové sítě.

Získané výsledky ze syntézy modelu jsou poté využity pro optimalizaci parametrů pomocí evolučních výpočetních technik, výsledky jsou porovnány s reálnými daty a zhodnoceny.

I. TEORETICKÁ ČÁST

1 OPTIMALIZACE ÚČELOVÉ FUNKCE

Optimalizace je proces, který se používá k nalezení nejlepšího možného výsledku v rámci daných omezení. Optimalizační metody se široce využívají v mnoha různých oblastech, mohou mít aplikace například ve statistice, ekonomii, fyzice či inženýrství. V průmyslu je optimalizace základním nástrojem pro zvýšení efektivity a snížení nákladů. Optimalizace může pomoci průmyslovým podnikům zlepšit jejich konkurenceschopnost zvýšením produktivity nebo například snížením produkovaného odpadu a spotřeby energie.

Z matematického pohledu je optimalizace typicky proces, při kterém se hledají takové hodnoty, pro které daná účelová funkce nabývá minimální či maximální hodnoty, dle zadání. Účelová funkce (UF) je matematická funkce, která kvantifikuje cíl, kterého se snažíme dosáhnout v rámci optimalizačního problému. Každé množině hodnot parametrů je pomocí ní přiřazena hodnota, která reflektuje kvalitu toho, jak dobře toto nalezené řešení splňuje stanovené cíle. [1]

Formálně lze optimalizační problémy formulovat následovně:

Nechť existuje vektor argumentů $x = (x_1, x_2, .., x_D)$, přičemž *D* je dimenzí řešeného problému a je dáno počtem optimalizovaných argumentů UF. Argumenty vektoru *x* jsou omezeny dolní a horní hranicí, *y* a *z*, jako $y <= x_n <= z$ pro $n = 1, .., D$. Dále definujme účelovou funkci $f_{cost}(x)$, jejíž hodnotu je úkolem minimalizovat či maximalizovat a její funkční omezení $g_n(x)$ pro $n = 1, .., m$, kde *m* je počtem omezení. Při řešení optimalizačního problému je hledáno takové *x*, pro které je hodnota UF minimální nebo maximální, vzhledem k funkčním omezením a k omezením argumentů.

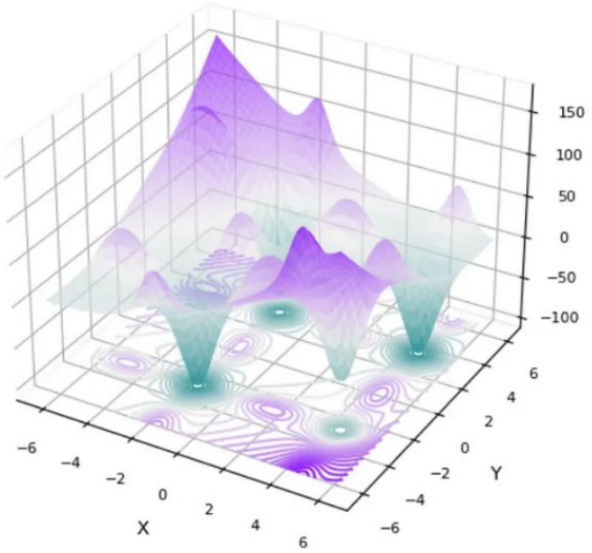
Na optimalizaci řešení UF lze pohlížet také jako na geometrický problém. Množina všech řešení UF tvoří *D*-dimenzionální prostor s návratovou hodnotou UF v dimenzi *D+1*. V tomto *D+1*dimenzionálním prostoru je pak hledáno požadované minimum či maximum.

Když hovoříme o minimu či maximu funkce je myšlen extrém globální. Globální extrém je hodnota funkce, která je buď největší (maximální) nebo nejmenší (minimální) na celém definičním oboru funkce.

Pro nalezení extrémů funkce se obecně nejprve hledají body, kde derivace funkce $f'(x)$ je rovna nule nebo neexistuje. Tyto body se nazývají kritické body nebo kandidáti na extrémy. Pokud je při zkoumání okolních bodů kandidátů na extrémy derivace funkce $f'(x)$ kladná na levé straně kritického bodu a záporná na druhé straně, naznačuje to, že v tomto bodě se nachází lokální maximum. Naopak, pokud je derivace kladná na pravé straně a záporná na druhé straně, naznačuje to, že v tomto bodě se nachází lokální minimum. Lokální extrém je miniminem či maximem na určitém definovaném intervalu funkce. Pokud chceme nalézt extrém globální je třeba provést důkladnou analýzu celého prostoru řešení funkce. [2]

Algoritmy pro hledání lokálního extrému jsou založeny na zkoumání gradientu funkce. [3] Pokud má funkce více extrémů může být využit multistart přístup, při kterém je vykonáno několik lokálních prohledávání s různými počátky. [4]

Algoritmy umožňující globální prohledávání prostoru poskytují větší pravděpodobnost nalezení globálního extrému. Tyto algoritmy mohou být klasifikovány do dvou skupin. Jedná se o stochastické evoluční algoritmy a algoritmy deterministické. [5] Evoluční algoritmy využívají oproti lokálnímu prohledávání informace o prostoru z celé populace bodů rozmístěných v rámci něj a v rámci jednotlivých iterací jsou dále posouvány. Tento přístup je dále popsán v kapitole 2. Deterministické algoritmy jsou často zaměřeny na úzkou skupinu problémů. [6] Jedním z populárních deterministických přístupů je algoritmus DIRECT. [7]



$$f(x,y) = \sin(x) * \exp((1-\cos(y))^2) + \cos(y) * \exp((1-\sin(x))^2) + (x-y)^2$$

Obrázek 1 Na trojrozměrném grafu, který vykresluje řešení rovnice pod ním, lze vidět příklady extrémů funkce. Vyobrazená funkce má vedle globálních extrému také mnoho lokálních minim i maxim. [8]

2 EVOLUČNÍ ALGORITMY

Evoluční algoritmy (EA) označují přístup řešení komplexních problémů pomocí technik inspirovaných evoluční teorií Charlese Darwina. Evoluce v přírodě je proces, který neustále samovolně probíhá a umožňuje diverzifikaci života. Tento proces je poháněn genetickými mutacemi u jedinců, které se postupně hromadí a ve výsledku mění vlastnosti jednotlivých druhů. Přirozeným výběrem pak dochází k selekci jedinců s vlastnostmi, které jim lépe umožňují přežít v daném prostředí a předat více svých genů do dalších generací. [9]

Při implementaci evolučních algoritmů dochází k simulaci těchto procesů, přičemž všechny možné varianty mutací představují prohledávaný prostor problému, jednotlivá řešení generovaná při běhu programu (body v prohledávaném prostoru) representují jedince a tvoří populaci s pomocí níž dochází změnami generovanými algoritmem k přibližování se k nejlepšímu řešení dané úlohy. Stejně jako v přírodě i u implementace evolučních algoritmů je využíváno křížení, rekombinace a mutace. Tyto děje umožňují rychlejší vznik změn u jedinců a zvyšování diverzity, což ve výsledku urychluje průběh evoluce. Kvalitu nalezeného řešení (jedince) označujeme opět analogií k studiu evoluce v přírodě jako fitness. [10], [11]

2.1 Průběh

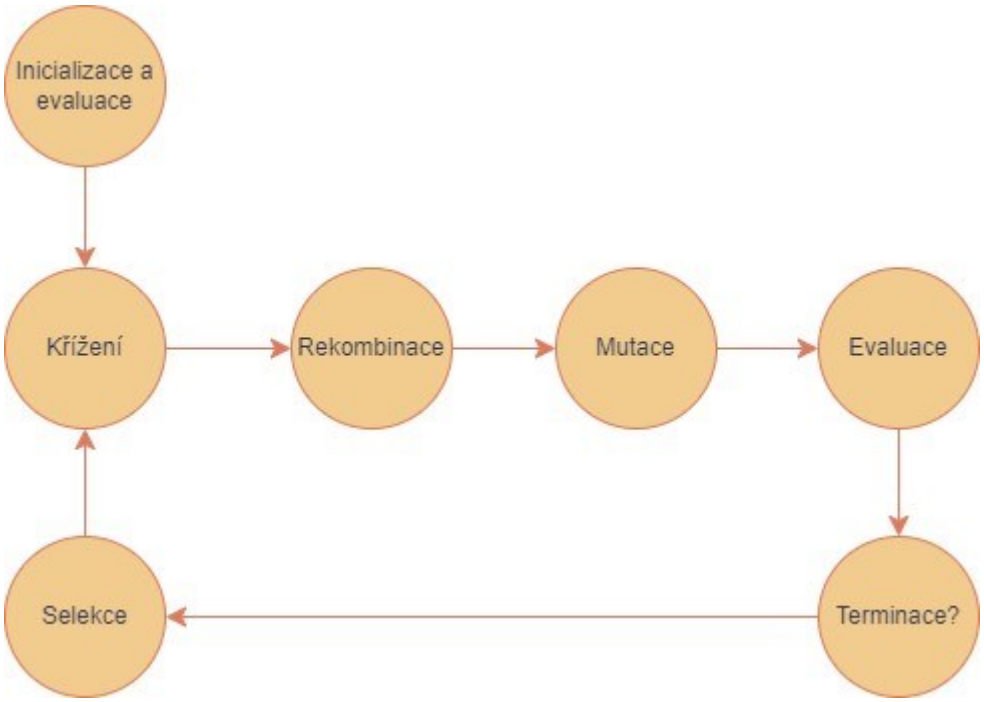
Před začátkem vlastního prohledávání dochází k inicializaci populace. Inicializace populace je typicky provedena náhodně, aby byl pokryt co největší prohledávaný prostor. Například, pokud hledáme nejkratší cestu mezi městy v grafu, každý jedinec může představovat různé permutace měst.

Pak dle zvoleného přístupu dochází k výběru jedinců ke křížení a k vlastnímu křížení. Křížení je proces, kde jsou vybráni dva nebo více jedinců (rodiče) a jsou kombinováni, aby vytvořili nové jedince (potomky). Křížení může být provedeno různými způsoby. Například, pokud máme dva rodiče reprezentující cesty *[A, B, C, D, E]* a *[E, D, C, B, A],* křížením můžeme získat potomka *[A, B, C, B, A]*.

Po procesu křížení může následovat proces známý jako rekombinace. Rekombinace je další mechanismus, který pomáhá vytvářet diverzitu v populaci tím, že umožňuje výměnu genetického částí mezi dvěma nebo více jedinci. Například, pokud máme dva jedince reprezentující různé cesty v grafu, jako jsou *[A, B, C, D, E]* a *[E, D, C, B, A]*, rekombinace by mohla zahrnovat výměnu sub-sekvencí mezi těmito jedinci. Výsledkem by mohly být nové cesty, jako *[A, B, C, B, A]* a *[E, D, C, D, E]*.

Poté může dojít k mutaci, což je náhodná změna v jedinci. Mutace pomáhá zvyšovat diverzitu populace a zabránit uvíznutí v lokálních minimech. Například, v případě hledání nejkratší cesty, mutace může zahrnovat náhodnou změnu jednoho města v cestě za jiné.

Nakonec je pro nové jedince vypočítána hodnota fitness funkce, ta hodnotí, jak dobře jedinec řeší daný problém. Například, v případě hledání nejkratší cesty, jedinci s kratšími cestami budou mít lepší fitness hodnotu. Pak dle zvoleného přístupu jsou vybráni jedinci pro další křížení. Tento proces je opakován iterativně, dokud není nalezeno dostatečně dobré řešení nebo dokud nejsou splněna jiná kritéria pro ukončení. [11]



Obrázek 2 Toto schéma zobrazuje typický průběh EA začínající inicializací populaci a evaluací jedinců, čemuž až do splnění zastavující podmínky následuje iterativně za sebou křížení, rekombinace, mutace, evaluace nově vzniklých jedinců a selekce jedinců do další generace. [10]

EA jsou algoritmy stochastické. To znamená, že nezaručují nalezení nejlepšího možného řešení problému. Při běhu se mu snaží co nejvíce přiblížit. Výsledek je určen zastavující podmínkou, která hledání ukončuje. Může se jednat například o maximální počet ohodnocení jedinců, anebo určení maximálního času běhu algoritmu. [11]

Mezi nevýhody spojené s těmito algoritmy patří nutnost ladění parametrů specifických pro problém a omezení velikosti prostoru řešeného problému. [5]

2.2 Typy evolučních algoritmů

EA představují širokou škálu přístupů k řešení optimalizačních problémů. Každý z nich má své jedinečné vlastnosti a je optimalizován pro řešení specifických typů problémů. Výběr správného algoritmu závisí na povaze konkrétního problému, který se snažíme řešit.

Genetické algoritmy (GA) jsou jedním z nejznámějších typů evolučních algoritmů. Byly navrženy a poprvé představeny Johnem Hollandem v roce 1975. [12] Genetické algoritmy používají operátory jako je křížení, mutace a selekce, inspirované přírodními procesy, k vyhledávání optimálních řešení ve velkých prohledávacích prostorech. Jedinci mohou být reprezentováni binárně, číselně, či jinými znaky. Křížení v GA může probíhat různými způsoby, například jednobodovým křížením, kde je genetický materiál rodičů rozdělen na dvě části, poté vyměněn a spojen dohromady, nebo uniformním křížením, kde každý gen je náhodně vybrán z jednoho z rodičů. Dalšími typy jsou křížení dvoubodové a vícebodové. Mutace zahrnuje náhodné změny v hodnotách prvků jedinců. Ke křížení i k mutaci jedinců dochází při běhu algoritmu s danou pravděpodobností. Selekce je založena na fitness funkci, může probíhat pomocí několika pravidel – Pravděpodobností selekce, Výběr dle pořadí, Výběr pomocí turnaje. [13] Důležitým pojmem je elitismus. Jedná se o proces výběru určitého počtu nejlepších jedinců do další populace (bez křížení), aby nedocházelo k jejich ztrátě. Zvolen je jen určitý počet, aby nedocházelo na druhou stranu ke ztrátě diverzity. [14]

Genetické algoritmy (GA) jsou všeobecně uznávány pro svou schopnost efektivně prozkoumávat velké prostory řešení. GA jsou také relativně snadno implementovatelné a přizpůsobitelné. Nicméně, GA mohou být pomalé a náročné na výpočetní zdroje, zvláště pro velmi velké a složité problémy.

Genetické programování (GP), které bylo poprvé představeno Johnem Kozaem v roce 1992, je dalším typem evolučního algoritmu. [15] GP se liší od GA tím, že jedinci v populaci jsou části algoritmu či matematematické funkce, reprezentované jako stromy. GP je schopné řešit velmi komplexní problémy, které mohou být mimo dosah GA. GP je zvláště silné v automatickém programování a strojovém učení, kde je schopné generovat a optimalizovat kód. Nicméně, GP může být také velmi náročné na výpočetní zdroje a může vyžadovat sofistikovanější implementaci než GA.

Diferenciální evoluce (DE), kterou v roce 1997 představili Rainer Storn a Kenneth Price, je dalším typem evolučního algoritmu. [16] DE je silná v kontinuálních optimalizačních problémech, které vyžadují co nevíce přesné a robustní řešení. Nicméně, DE může být méně

efektivní pro problémy s diskrétními nebo kombinatorickými prostory řešení. Tento algoritmus je zvlášť rozebrán v kapitole 4.

Evoluční strategie (ES) jsou další typ evolučního algoritmu, který byl navržen v Německu v 60. letech 20. století P. Bienertem, I. Rechenbergem a H. P. Schwefelem. [17], [18], [19], [20]. ES se liší od jiných evolučních algoritmů tím, že používají specifické operátory mutace a rekombinace, které jsou navrženy tak, aby byly efektivní v kontinuálních prohledávacích prostorech. Jedinci v ES jsou reprezentováni typicky čísly z oboru reálných čísel. Operátor křížení není využit. Existují čtyři hlavní typy evolučních strategií:

$(1 + 1) - ES$: Tento typ evoluční strategie pracuje s jedním rodičem a jedním potomkem. V každé generaci je rodičovský jedinec mutován k vytvoření potomka a poté je na základě jejich fitness hodnot vybrán buď rodič, nebo potomek pro přechod do další generace.

$(\mu + \lambda) - ES$: Tento typ evoluční strategie pracuje s μ rodiči a λ potomky. V každé generaci jsou rodiče mutováni k vytvoření potomků a poté jsou na základě jejich fitness hodnot vybráni nejlepší jedinci z celkové populace rodičů a potomků pro přechod do další generace.

$(\mu, \lambda) - ES$: Tento typ evoluční strategie je podobný $(\mu + \lambda) - ES$, ale s tím rozdílem, že do další generace přecházejí pouze potomci.

$(\mu/\rho + \lambda) - ES$: Tento typ využívá μ rodičů k vytvoření λ potomků, přičemž ρ rodičů je použito pro rekombinaci. Potomci jsou vybíráni z celkové populace rodičů a potomků.

Evoluční programování (EP), které bylo poprvé představeno Lawrencem Fogelem, je jednou z prvních forem evolučních algoritmů. [21] EP bylo původně navrženo jako metoda pro evoluci konečných automatů, které byly používány k modelování chování biologických organismů. Z biologického úhlu pohledu se EP zaměřuje na evoluci z hlediska fenotypu.

Kooperativní ko-evoluce (CCE), kterou představil Mitchell Potter, je evoluční algoritmus, který umožňuje evoluci více sub-populací, které spolupracují na řešení složitých problémů. CCE se liší od jiných evolučních algoritmů tím, že umožňuje evoluci komplexních řešení pomocí interakce jednodušších podproblémů. [22]

Kooperativní ko-evoluce s diferenciální evolucí (CoDE) je další nový typ evolučního algoritmu. CoDE kombinuje principy kooperativní ko-evoluce a diferenciální evoluce k řešení komplexních optimalizačních problémů. V CoDE je populace rozdělena do sub-populací, které kooperativně ko-evolvují k řešení daného problému. Tento přístup umožňuje CoDE efektivně řešit problémy s velkým počtem proměnných. [23]

Estimační distribuční algoritmy (EDA) jsou typem EA, které se začaly objevovat koncem 90. let. Jedním z prvně představených EDA byl algortimus známý jako "Population Based Incremental Learning" (PBIL). [24] Na rozdíl od tradičních evolučních algoritmů, které využívají operátory jako je křížení a mutace k generování nových jedinců, EDA se snaží odhadnout distribuci kandidátních řešení a generovat nové jedince na základě této distribuce. Tento přístup je založen na myšlence, že kvalitní řešení jsou často soustředěna v určitých oblastech prohledávaného prostoru a že odhadem těchto oblastí můžeme efektivněji generovat kvalitní jedince.

Mezi evoluční algoritmy řadíme také Particle Swarm Optimization (PSO) a Self-Organizing Migrating Algorithm (SOMA). [25], [26] Tyto algoritmy nejsou přímo inspirovány evolucí a nevyužívají operátorů křížení, mutace a selekce. Společným prvkem s ostatními EA ale je využití populace jedinců a jejich změn pro prohledávání prostoru řešení. Zároveň jsou také inspirovány procesy probíhajícími v přírodě. Oba algoritmy jsou zvláště podrobně rozebrány v kapitolách 3 a 5.

2.2.1 Hybridní evoluční algoritmy

Hybridní evoluční algoritmy představují pokročilý koncept v oblasti optimalizačních technik. Tyto algoritmy kombinují různé evoluční algoritmy nebo integrují evoluční algoritmy s jinými technikami, což jim umožňuje využít výhod různých přístupů a dosáhnout lepších výsledků.

Příkladem algoritmu, který kombinuje různé evoluční algoritmy je hybridní algoritmus PSO a DE. Příkladem využití je práce autorů M. F. Tasgetiren, Y. C. Liang a M. Sevkli, kteří jej využívají pro řešení tzv. Job Shop Scheduling Problem. [27] Algoritmy jsou adaptovány pro kombinatorické optimalizace a jsou vylepšeny pomocí lokálního prohledávání (local search) založeného na variable neighborhood search.

Kombinování EA s jinými heuristickými přístupy je velmi hojně používáno. EA jsou velmi výkonné v lokalizování oblasti výskytu extrému, ale pro nalezení konkrétních hodnot extrému mohou být jiné techniky výkonnější. [28]

V práci napsané John Doe a Jane Smith, je aplikován hybridní algoritmus Grey Wolf Optimization a Particle Swarm Optimization (GWO-PSO) na problém optimálního reaktivního výkonu v elektrických sítích. [29]

Dalšími příklad heuristik kombinovanými s EVT je Simulated Annealing (SA). Například ve studii, kterou provedli autoři John Doe a Jane Smith, je zkoumána hybridizace PSO, SA a DE. [30]

Hybridní evoluční algoritmy mohou také kombinovat EVT s technikami strojového učení, například s neuronovými sítěmi nebo rozhodovacími stromy. [31]

Další zajímavou oblastí jsou hybridní evoluční algoritmy, které kombinují evoluční algoritmy s fuzzy logikou. Příkladem je práce od autorů F. Valdez, P. Melin a O. Castillo. [32] Fuzzy logika může být využita k modelování nejistoty a neurčitosti v evolučních algoritmech, což může vést k robustnějším a adaptivnějším řešením. Dále také k efektivnějšímu zpracování výsledků EA.

2.3 Využití v praxi

Evoluční algoritmy představují významný nástroj v mnoha oblastech výzkumu a praxe. Jejich univerzálnost a schopnost efektivně prozkoumávat velké prostory je činí ideálními pro řešení široké škály problémů.

EA jsou široce používány v oblasti strojového učení pro optimalizaci modelů a algoritmů. Například, genetické algoritmy, mohou být použity k procesu výběru rysů (feature selection). Tento proces hledá optimální sadu rysů pro trénování modelu, což může vést k významnému zlepšení výkonu modelu například odstraněním irelevantních, redundantních a šumových data. Příklad práce, která se tímto zabývá je práce C. J Tu, L. Y. Chuang, J. Y. Chang, a V. Yang. V tomto článku je pro výběr rysů použita metoda PSO a SVM (Support Vector Machines) slouží jako fitness funkce pro PSO. [33]

Bioinformatika je další oblast, kde nalezneme široké využití evolučních algoritmů. EA jsou často využívány k analýze genové exprese a sekvenování DNA, což jsou klíčové úkoly v moderní biologii. V práci autorů W. E. Harta, N. Krasnogora, D. A. Pelta a J. Smith jsou EA využity pro předpověď struktury proteinů. Tento problém je klíčový pro pochopení funkce proteinů a jejich interakcí v buňce. [34]

V oblasti robotiky jsou evoluční algoritmy používány k optimalizaci návrhu a řízení robotů. Například PSO je často používáno k vývoji algoritmů pro řízení pohybu robotů. Jedním z příkladů je práce od E. Masehian a D. Sedighizadeh, ve které autoři použili PSO k řešení problému plánování dráhy robota. Vedle tradičního PSO, využívají také algoritmus NPSO

(New or Negative PSO), v němž se jako vodítko nepoužívá gbest, ale nejslibnější směr je určen na základě negativní polohy nejhorší částice. [35]

V oblasti telekomunikací se evoluční algoritmy používají k optimalizaci a řízení telekomunikačních sítí. Například, genetické algoritmy a PSO se často používají k optimalizaci umístění antén v bezdrátových sítích nebo k řízení přenosových rychlostí v datových sítích. Jako příklad můžeme uvést práci od autorů JY Wang, JB Wang, X. Song, M. Chen a J. Zhang. [36] Dalším příkladem z oblasti telekomunikací je práce od autorů N. Jin a Y. Rahmat-Samii, v níž autoři použili PSO za účelem optimalizace parametrů pro efektivnější návrh antén. [37]

EA jsou využívány také v rámci oblastí mimo informační technologie. Jednou z takových oblastí je například ekonomie a finanční inženýrství. GP může být využito k analýze dat z dvojitých aukčních trhů, které tvoří složité a dynamické prostředí s mnoha neznámými proměnnými. O tomto využití lze najít další informace například v kapitole M. Andrewse a R. Pragera z knihy Advances in Genetic Programming. [38]

V oblasti finančního inženýrství se evoluční algoritmy používají k optimalizaci a řízení finančních portfolií. Například, GA a PSO se často používají k optimalizaci alokace aktiv v portfoliu nebo k predikci finančních trhů. Jako příklad můžeme uvést práci od autorů MR Hassana, B. Natha a M. Kirley, kde autoři použili GA k předpovědi finančních trhů. [39]

V energetice jsou evoluční algoritmy používány k optimalizaci a řízení energetických systémů. Například, diferenciální evoluce a genetické programování se často používají k optimalizaci provozu a plánování údržby elektráren nebo k řízení spotřeby energie v budovách. Jako příklad můžeme uvést práci od autorů N. Nomana a H Iba, kde autoři použili diferenciální evoluci k řešení problému economic load dispatch (nejlevnější plán výroby elektřiny vzhledem k ostatním daným faktorům) v elektrárenských systémech. [40]

Evoluční algoritmy jsou často používány pro analýzu a manipulaci s daty v různých oblastech. Příkladem je práce PP Gujar a P. Desai. V této práci autoři používají GP k řešení problému deduplikace záznamů, což je běžný problém v oblasti správy dat. [41]

M. L. Wong, W. Lam, K. S. Leung, P. S. Ngan a J.C.Y. Cheng ve své práci využívají EA k hledání vzorců v medicínských databázích. Tento přístup umožnil autorům identifikovat klíčové vzorce a vztahy v datech, které by mohly být jinak obtížně detekovatelné. Toto může být užitečné nejen v oblasti přírodních věd a zdravotnictví, ale v mnoha dalších různých oblastech, včetně marketingu nebo sociálních věd. [42]

V oblasti dopravy a logistiky jsou evoluční algoritmy často používány k řešení složitých plánovacích a optimalizačních problémů. Například, GA a PSO se používají k řešení problému obchodního cestujícího, který se týká nalezení nejkratší možné cesty procházející všemi městy v síti. Jako příklad můžeme uvést práci od autorů B. M. Bakera, M. A. Ayechewa. V této studii autoři použili GA k řešení problému směrování dopravy, což je klíčový problém v oblasti logistiky a dopravy. [43]

Další oblastí využívající EA je výroba a průmyslové inženýrství, kde se evoluční algoritmy často používají k optimalizaci výrobních procesů a plánování údržby. Například, genetické algoritmy a diferenciální evoluce se často používají k řešení problémů jako je plánování výroby, plánování údržby a plánování rozvrhů. Jako příklad můžeme uvést práci od autorů Z. Tajbakhsha, P. Fattahi a J. Behnamiana, v níž využívají GA a PSO k plánování výrobního systému. [44]

Tyto příklady ilustrují širokou škálu aplikací, kde jsou evoluční algoritmy úspěšně používány. Díky své schopnosti efektivně prozkoumávat velké a komplexní prostory řešení, evoluční algoritmy představují silný nástroj pro řešení široké škály problémů v různých oblastech.

3 PSO

PSO je zkratkou pro Particle Swarm Optimization. Jedná se o metodu založenou na teorii rojů. Optimalizace probíhá iterativní snahou zlepšit kandidátské řešení pohybem jedinců (jednotlivých nalezených řešení) v rámci prohledávaného prostoru pomocí jednoduchých matematických vzorců. Směrování jedinců k nejlepšímu řešení probíhá na základě ohodnocení nalezeného řešení daného jedince a ohodnocení nalezených řešení jedinců sousedních. PSO bylo úspěšně aplikováno v mnoha oblastech, včetně optimalizace nelineárních funkcí a tréninku neuronových sítí. Inspirací pro PSO byla simulacemi chování ptáků v rámci hejna. [45] [46]

3.1 Vývoj algoritmu

Algoritmus byl vyvinut v roce 1995 Jamesem Kennedym, sociálním psychologem, a Russellem Eberhartem, inženýrem elektrotechniky. [25] Původně chtěli modelovat sociální chování, konkrétně chování hejna ptáků nebo roje hmyzu. Tyto simulace byly vytvořeny za účelem zkoumání toho, jaké mechanismy umožňují synchronizovaný, ale nepředvídatelný pohyb většího množství zvířat za současných změn směru pohybu či shlukování. Simulace jsou založeny na udržování optimálních vzdáleností zvířat mezi sebou a jedinci v bezprostředním sousedství. První verze algoritmu byla postavena na nejbližších sousedech a náhodné proměnné zvané "šílenství".

Později byla přidána funkce "kukuřičného pole", která představovala cíl, ke kterému se ptáci snaží dostat. Jedinci byli programováni tak, aby se snažili minimalizovat vzdálenost od tohoto cíle. Toto vedlo k vytvoření konceptu "pbest" a "gbest", kde "pbest" označuje poslední nejlepší pozici, kde se jedinec nacházel a "gbest" představuje nejlepší dosaženou pozici v rámci celého hejna.

Nejlepší v tomto kontextu je myšleno z hlediska ohodnocení pozice, v případě simulace hejna zvířat se může jednat o místo s nalezením největším množství potravy. V případě matematické optimalizace pak o extrémy dané funkce.

Algoritmus byl postupně zjednodušen, vylepšen a byly odstraněny některé nadbytečné prvky, což vedlo k standardní verzi PSO. Tento algoritmus se ukázal jako velmi efektivní pro optimalizaci široké škály funkcí a je v současnosti široce používán v mnoha oblastech.

Jednou z výhod PSO je jeho jednoduchost a efektivita, protože nevyžaduje žádné složité operace, jako je křížení nebo mutace, které se používají v genetických algoritmech. Místo toho se spoléhá na spolupráci a soutěživost mezi částicemi k nalezení optimálního řešení.

3.2 Jedinci

V kontextu PSO jsou jedinci základními jednotkami, které se pohybují v prostoru řešení. Každý jedinec, často nazývaný "částice", je potenciální řešení optimalizačního problému. Tito jedinci jsou rozhodující pro úspěch celého algoritmu, protože jejich pohyby a interakce určují, jak dobře a efektivně bude algoritmus schopen najít optimální řešení. Každý jedinec má následující atributy:

Pozice: Každý jedinec je charakterizována svou polohou v prostoru řešení. Poloha je vektor, jehož dimenze odpovídá dimenzi problému, který je třeba optimalizovat. Například, pokud je problém dvourozměrný, poloha jedince bude určena dvěma souřadnicemi *[x, y]*.

Rychlost: Rychlost jedince určuje, jak rychle a v jakém směru se pohybuje v prostoru řešení. Rychlost je také vektor a je aktualizován v každé iteraci algoritmu na základě současné polohy jedince, jeho nejlepší dosažené polohy a nejlepší dosažené polohy celé populace.

Nejlepší dosažená pozice (pbest): Každý jedinec má uloženu svou nejlepší dosaženou pozici, tedy pozici, ve které byla dosažena nejlepší hodnota fitness funkce. Tato pozice je důležitá pro aktualizaci rychlosti jedince.

Globálně nejlepší dosažená pozice (gbest): Kromě své vlastní nejlepší dosažené pozice (pbest), jedinci také berou v úvahu nejlepší dosaženou pozici celé populace, která se nazývá gbest. Tato hodnota je sdílena mezi všemi jedinci a je aktualizována, pokud některý jedinec dosáhne lepší hodnoty fitness funkce než aktuální gbest.

Jedinci v PSO používají strategii hledání založenou na spolupráci a učení se od ostatních. Tím, že kombinují své vlastní osobní zkušenosti (pbest) s informacemi získanými od ostatních jedinců (gbest), jsou schopni efektivně navigovat prostorem řešení a hledat optimální řešení.

Pomocí parametrů c_1 a c_2 je možné ovlivňovat, jak moc se jedinci budou chovat spíše individuálně a prohledávat prostor spíše ve svém okolí, anebo sociálně a táhnout ke společnému cíli. Vysoké upřednostnění sociálního chování z matematického pohledu může snadno vést k uvíznutí hejna v lokálním extrému v rámci prohledávaného prostoru. [25]

Fitness funkce: Každý jedinec je také hodnocen pomocí fitness funkce, která je specifická pro daný optimalizační problém. Hodnota fitness funkce pro daného jedince určuje, jak "dobré" je jeho aktuální řešení. Fitness funkce je klíčová pro určení pbest a gbest.

3.3 Průběh algoritmu

Prvním krokem v procesu PSO je inicializace. Tento proces zahrnuje vytvoření populace částic na náhodných pozicích v prostoru řešení. Každá částice je charakterizována svou polohou a rychlostí, které jsou dynamické a mění se v průběhu procesu optimalizace. Inicializace je klíčová, protože určuje, kde a jak rychle se částice pohybují při hledání optimálního řešení.

Po inicializaci následuje proces hodnocení. V tomto kroku je každá částice ohodnocena na základě fitness funkce. Tato funkce je určena problémem, který se snažíme optimalizovat, a poskytuje kvantitativní míru toho, jak "dobré" je aktuální řešení dané částice. Hodnocení je důležité pro určení, které částice dosáhly nejlepších řešení a jak se mají částice pohybovat v dalších iteracích.

Po hodnocení se aktualizují nejlepší hodnoty. Každá částice si udržuje svou nejlepší dosaženou pozici (pbest). Dále je sledována globální nejlepší pozice ze všech částic (gbest). Tato pozice je sdílena mezi všemi částicemi a je aktualizována, pokud některá částice dosáhne lepší hodnoty fitness funkce než aktuální gbest.

Následuje klíčový krok PSO – aktualizace rychlosti a pozic. Rychlost každé částice se aktualizuje na základě několika faktorů. Prvním faktorem je současná rychlost částice (v_{ij}^t). Tento faktor zajišťuje, že pohyb částice je kontinuální a že částice neudělá náhlý skok z jednoho bodu do druhého. Druhým faktorem je vzdálenost mezi současnou pozicí částice a její osobní nejlepší pozicí $\left(pBest_{ij} - x_{ij}^t\right)$. Tento faktor vede částici k tomu, aby se pohybovala směrem k nejlepším řešením, která dosud našla. Třetím faktorem je vzdálenost mezi současnou pozicí částice a globálně nejlepší pozicí($gBest_j - x_{ij}^t$). Tento faktor vede částici k tomu, aby se pohybovala směrem k nejlepším řešením nalezeným jakoukoli částicí v populaci. Každý z těchto faktorů je zvážen pomocí konstant ($c_1\ a\ c_2$) , které určují, jak silný vliv má každý faktor na aktualizaci rychlosti, a náhodně generovanými váhami od 0 do 1 ($r_1\ a\ r_2$). Kombinací výše zmíněných částí výpočtu rychlosti částice dostáváme tuto rovnici:

$$v_{ij}^{t+1} = v_{ij}^t + c_1 * r_1 * \left(pBest_{ij} - x_{ij}^t\right) + c_2 * r_2 * (gBest_j - x_{ij}^t)$$

Po aktualizaci rychlosti se aktualizuje i pozice částice. Nová pozice je vypočítána přičtením aktualizované rychlosti k současné pozici. To znamená, že částice se pohybuje o krok rovný její rychlosti ve směru určeném aktualizovanou rychlostí. Aktualizaci pozice můžeme matematicky vyjádřit takto: $x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}$.

Toto je moment, kdy částice "letí" prostorem řešení a hledají optimální řešení. Aktualizace rychlosti a pozic umožňují částicím prozkoumat nové oblasti prostoru řešení a zároveň se soustředit na oblasti, které se ukázaly jako úspěšné v minulých iteracích.

Je důležité zdůraznit, že aktualizace rychlosti a pozic zahrnuje také náhodné složky. Tyto náhodné složky zajišťují, že částice nejsou zcela určeny svými předchozími pohyby a stávajícími nejlepšími řešeními. Díky tomu může algoritmus prozkoumat nové oblasti prostoru řešení a potenciálně najít lepší řešení, než která byla dosud nalezena.

Celý proces se poté opakuje po určitý počet iterací, nebo dokud není dosaženo určité úrovně přesnosti. Tímto způsobem PSO kombinuje hledání nových oblastí prostoru řešení s využitím informací získaných v předchozích iteracích, což umožňuje efektivní a robustní hledání optimálního řešení.

3.4 Rozšířené varianty algoritmu

Jedním z příkladů vylepšené varianty PSO je Weighted Particle Swarm Optimization (WPSO), který zavádí nový parametr zvaný "váha setrvačnosti". Tato váha určuje, do jaké míry je pohyb částice ovlivněn její předchozí rychlostí. To umožňuje algoritmu efektivněji řešit problém nalezení kompromisu mezi zaměřením na průzkum nových oblastí (explorací) a využitím nejlepších dosud nalezených řešení (exploitací). Váha setrvačnosti může být konstantní nebo se může měnit v průběhu času. [47]

Dalším příkladem varianty PSO je heterogenní PSO (HPSO). Ve standardním PSO a většině jeho modifikací, následují jednotlivé částice stejné chování a implementují stejná pravidla pro aktualizaci rychlosti a pozice. To znamená, že částice vykazují stejné charakteristiky hledání. HPSO umožňuje částicím následovat různé chování vyhledávání vybrané z poolu chování, čímž může opět efektivně řešit problém nalezení kompromisu mezi zaměřením na exploraci a exploitaci.

Různá chování mohou zahrnovat model upřednostňující sociální či kognitivní chování, anebo měnit jejich poměr za běhu. Dále může být například implementována časově proměnná setrvačnost s konstantními akceleračními faktory.

HPSO může fungovat buď ve statickém módu (sHPSO), kde jsou chování náhodně přiřazena částicím během inicializace a nemění se během procesu hledání, anebo v dynamickém módu (dHPSO), kde mohou částice během procesu hledání měnit své chování náhodně. Chování částic je v obou případech inicializováno náhodným výběrem z poolu chování. [48]

Varianta PSO nazvaná Multi-Leader Particle Swarm Optimization (MLPSO), zavádí do algoritmu koncept více "leaderů" (vedoucích částic). Místo toho, aby se všechny částice snažily následovat jediné globální nejlepší řešení, v MLPSO existuje několik leaderů, kteří vedou různé skupiny částic k různým oblastem vyhledávacího prostoru. Každá částice se dynamicky rozhoduje o svých "leaderech" na základě teorie her. Nejlepší leader pak bere v úvahu pozice a výsledky ostatních leaderů při aktualizaci své vlastní pozice, aby zlepšil svou kvalitu v každé generaci. [49]

Další vylepšenou variantou PSO je Comprehensive Learning Particle Swarm Optimization (CLPSO). Jedním z klíčových rozdílů mezi CLPSO a standardním PSO je způsob, jakým částice aktualizují své rychlosti. Ve standardním PSO algoritmu se částice učí od svého dosavadního nejlepšího řešení (pbest) a od nejlepšího řešení v celé populaci (gbest). Avšak v CLPSO, se částice učí od různých částic a jejich pbest pro různé dimenze, což vede k silnější globální vyhledávací schopnosti a lepšímu zachování diverzity populace. [50]

Heterogeneous Comprehensive Learning Particle Swarm Optimization (HCLPSO) je algoritmus, který také využívá strategie CL, a navíc rozděluje celkovou populaci na dvě subpopulace, přičemž každá subpopulace je zaměřena buď na exploitaci nebo exploraci.

Průzkumná subpopulace je navržena tak, aby se učila pouze od svých vlastních nejlepších výsledků, což znamená, že není ovlivněna částicemi z subpopulace zaměřené na exploitaci. Ta se naopak učí od nejlepších zkušeností všech částic v celé populaci, včetně těch z průzkumné subpopulace. Tento přístup umožňuje, aby průzkumná subpopulace udržela diverzitu, i když subpopulace zaměřená na exploitaci dospěje k předčasné konvergenci. [51]

Quantum-behaved Particle Swarm Optimization (QPSO) je dalším algoritmem založeným na PSO, který je inspirován pravidly kvantové mechaniky. V kvantové fyzice je stav částice s momentem a energií popsán její vlnovou funkcí. V QPSO se předpokládá, že každá částice je v kvantovém stavu a je definována její vlnovou funkcí místo pozice a rychlosti, které jsou v PSO. Vlnová funkce udává pravděpodobnost, kde se částice může nacházet.

Další klíčovou vlastností QPSO je využití nejen nejlepšího dosaženého řešení jednotlivých částic, ale také průměrného nejlepšího řešení celého roje. Tato průměrná nejlepší pozice je

pak využita při aktualizaci pozic jednotlivých částic. To umožňuje lépe vyvážit globální a lokální prohledávání prostoru řešení a pomáhá zabránit předčasné konvergenci k suboptimálnímu řešení. [52]

Tyto a další varianty PSO představují významný přínos pro oblast optimalizačních algoritmů a otevírají nové možnosti pro řešení složitých optimalizačních problémů.

4 DIFERENCIÁLNÍ EVOLUCE

Algoritmus Differential Evolution (DE) byl původně vytvořen Rainerem Stornem a Kennethem Pricem v roce 1997. [16] Jedná se o algoritmus inspirovaný biologickými procesy probíhajícími v rámci evoluce v přírodě. Od svého vzniku byl algoritmus DE dále rozvíjen a vylepšován, a to jak samotnými tvůrci, tak i dalšími výzkumníky v oboru.

4.1 Průběh algoritmu

Diferenciální evoluce je evoluční metoda umožňující přímé paralelní prohledávání prostoru řešení. Prohledávání probíhá pomocí parametrů vektorů. Každý vektor obsahuje body v jednotlivých dimenzích prohledávaného prostoru a tvoří tak jedno nalezené řešení. Všechny vektory dohromady tvoří populaci, která je s každou generací obměňována. Například, pokud by byly hledány nejlepší rozměry pro design nábytku, každý vektor by obsahoval různé možné rozměry pro výšku, šířku a hloubku nábytku.

DE začíná vytvořením náhodné populace vektorů, která pokrývá celý prostor možných řešení. Poté začíná evoluční proces, ve kterém se populace vektorů postupně mění a vylepšuje. Při generační změně populace dochází k mutaci, crossoveru a selekci vektorů.

Prvním krokem je mutace. Jako příklad je zde popsána varianta DE/rand/1. U každého vektoru v populaci se vezmou hodnoty dvou náhodně vybraných vektorů (x_{r2} a x_{r3}), vypočítá se jejich rozdíl a vynásobí se určitým koeficientem F. Tento výsledek se pak přičte k dalšímu náhodně vybranému vektoru (x_{r1}), čímž vznikne nový, mutovaný vektor, takto $v_i = x_{r1} + F * (x_{r2} - x_{r3})$. Bez mutace by populaci mohlo chybět dostatečné množství rozmanitosti, což by mohlo vést k uvíznutí v lokálních minimech a neefektivnímu prohledávání prostoru řešení.

Následuje proces zvaný crossover, který je podobný genetickému křížení v přírodě. Parametry mutovaného vektoru se zkombinují s parametry dalšího náhodně vybraného vektoru (cílového vektoru), čímž vznikne nový vektor, tzv. zkoušecí (trial) vektor. Crossover umožňuje sdílení užitečných informací mezi jednotlivými vektory v populaci a pomáhá vytvářet nové a potenciálně lepší kombinace parametrů.

Nakonec přichází selekce. Trial vektor se porovná s původním cílovým vektorem. Pokud je trial vektor lepší (tj. dává lepší výsledek funkce, kterou chceme optimalizovat), nahradí cílový vektor v populaci. Pokud je horší, zůstane původní cílový vektor. Tento proces selekce je to, co řídí evoluci populace směrem k lepším řešením.

Celý tento proces se opakuje pro každý vektor v populaci, a to celé se nazývá jedna generace. Proces se pak opakuje pro další a další generace, až se populace vektorů postupně zlepšuje a konverguje k nejlepšímu možnému řešení.

4.2 Parametry

Diferenciální evoluce (DE) je vysoce účinný optimalizační algoritmus. Jeho efektivita a výkonnost však závisí na správném nastavení tří kritických parametrů: velikosti populace (NP), faktoru diferenciace (F) a konstantě křížení (CR).

Velikost populace (NP): Velikost populace je klíčovým parametrem, protože určuje počet řešení, které algoritmus současně zvažuje. Větší populace může přinést širší spektrum možných řešení, což zvyšuje šance na nalezení globálního minima. Současně však musíme brát v úvahu, že s rostoucí velikostí populace roste také výpočetní náročnost algoritmu.

Faktor diferenciace (F): Tento parametr ovlivňuje míru, jakou se provádějí mutace v algoritmu. Jde o hodnotu v rozmezí od 0 do 2. Pokud je faktor diferenciace vysoký, algoritmus provede větší mutace, což umožňuje prozkoumat širší prostor řešení. To však také zvyšuje riziko, že se algoritmus vzdálí od optimálního řešení.

Konstanta křížení (CR): Tato hodnota určuje pravděpodobnost, s jakou se při křížení použije parametr z mutovaného vektoru. CR je rovněž reálné číslo v rozmezí od 0 do 1. Vyšší hodnoty CR zvyšují diverzitu nových vektorů vytvořených křížením.

Obecně platí, že velká populace vede k lepším výsledkům, ale zároveň zvyšuje výpočetní nároky. Doporučuje se začít s NP o několika desítkách jedincích, pro komplexní a vysokodimeziální problémy je vhodné volit větší populace o stovkách jedinců. Pokud jde o faktor diferenciace F, hodnoty mezi 0,5 a 1 obecně dobře fungují pro většinu problémů. Nicméně, pokud data obsahují hodně šumu, může být užitečné snížit F na hodnoty kolem 0,2. Konstanta křížení CR se doporučuje nastavit na hodnotu blízko 1 pro problémy s malým počtem parametrů ($D < 10$). Pro problémy s větším počtem parametrů může být užitečné nastavit CR na nižší hodnotu.

Při práci s algoritmem diferenciální evoluce je důležité mít na paměti, že neexistuje univerzálně "nejlepší" sada parametrů, která by byla ideální pro všechny problémy. Každý problém je jedinečný a může vyžadovat specifické nastavení parametrů pro dosažení nejlepšího možného výsledku. Proto je důležité provést řádné experimentování a ladění parametrů, aby byl algoritmus co nejefektivnější pro daný problém.

4.3 Varianty algoritmu

Diferenciální evoluce je dostupná v několika variantách, které se liší svými strategiemi mutace a křížení. Tyto varianty jsou označovány jako DE/x/y/z, kde:

"x" specifikuje vektor, který má být mutován. Může to být "rand" nebo "best". "Rand" znamená, že se vybere náhodný vektor z populace pro mutaci. "Best" znamená, že se vybere nejlepší vektor z populace pro mutaci.

"y" je počet párů diferencovaných vektorů použitých v procesu mutace. Toto číslo určuje, kolik párů vektorů se použije k výpočtu rozdílových vektorů, které se přičtou k cílovému vektoru při mutaci.

"z" označuje schéma křížení použité v algoritmu. Může to být "bin" nebo "exp". "Bin" znamená binomické křížení, kde se pro každý parametr rozhoduje nezávisle, zda se převezme z cílového nebo mutovaného vektoru. "Exp" znamená exponenciální křížení, kde se sekvence sousedních parametrů převezme z mutovaného vektoru.

Například, varianta DE/rand/1/bin používá náhodný vektor pro mutaci, jednu dvojici vektorů pro výpočet rozdílového vektoru a binomické křížení. Na druhé straně, varianta DE/best/2/exp používá nejlepší vektor pro mutaci, dvě dvojice vektorů pro výpočet rozdílových vektorů a exponenciální křížení.

Strategie mutace a křížení jsou klíčové pro výkon DE. Použití "best" místo "rand" může zlepšit konvergenci algoritmu tím, že se zaměří na nejlepší řešení. Počet diferencovaných vektorů ("y") ovlivňuje diverzitu nové populace. Použití více vektorů může vést k větší diverzitě, ale také může způsobit, že algoritmus bude více náhodný. Binomické křížení je více náhodné a může vytvářet různorodější řešení, zatímco exponenciální křížení je více konzervativní a může pomoci zachovat dobré řešení.

Je důležité poznamenat, že neexistuje "nejlepší" varianta DE pro všechny problémy. Různé varianty DE se mohou chovat lépe nebo hůře v závislosti na konkrétním problému optimalizace. Výběr nejlepší varianty DE závisí na konkrétním problému a může vyžadovat experimentování a ladění.

4.4 Rozšířené varianty algoritmu

Existují různé varianty a vylepšení DE, které se snaží vylepšit jeho výkonnost a robustnost.

Adaptivní DE (jDE) je jednou z nejpopulárnějších variant DE, která zavádí adaptivní mechanismus pro aktualizaci parametrů mutace a křížení. Tento mechanismus je založen na kvalitě generovaných řešení, což znamená, že parametry jsou aktualizovány tak, aby byly preferovány ty strategie, které generují lepší řešení. Toto vede k tomu, že jDE je obecně robustnější než standardní DE, ale může také vyžadovat více výpočetních zdrojů. [53]

Self-adaptive DE (SaDE) se liší od standardního DE tím, že zavádí mechanismus adaptivní selekce operátorů. To znamená, že pravděpodobnost výběru konkrétního operátoru mutace a křížení se dynamicky mění během procesu optimalizace na základě jeho předchozího výkonu. Co se týče konstant, F je adaptována pro jednotlivce a jednotlivým strategiím jsou přiřazeny střední hodnoty pro generování CR. [54]

Algoritmus Adaptive Differential Evolution with Optional External Archive (JADE) implementuje novou mutační strategie DE/current-to-pbest, externí archiv a využívá aktualizace řídících parametrů adaptivním způsobem. Mutační strategie DE/current-to-pbest je zobecněním klasické strategie DE/current-to-best, kdy je jedinec pro mutaci zvolen náhodně z množiny nejlepších jedinců. Volitelná operace archivu ukládá část selekcí vyřazených jedinců a využívá je pak znova při mutaci. Obě operace diverzifikují populaci a zlepšují konvergenční výkon. [55]

Success-History Based Parameter Adaptation for Differential Evolution (SHADE) je vylepšení DE, založené na JADE, které používá historickou paměť úspěšných nastavení kontrolních parametrů pro výběr budoucích hodnot kontrolních parametrů. [56]

Ensemble of Parameters, Mutation and Strategies DE (EPSDE) vylepšuje standardní DE tím, že každý jedinec disponuje sadou hodnot – strategie, F a CR, které soutěží o to, která z nich je schopna generovat nejlepší potomstvo na základě svých předchozích úspěchů v evolučním procesu. Tento přístup umožňuje dynamickou adaptaci parametrů na základě zpětné vazby získané během procesu hledání. Neefektivní kombinace mutačních strategií a parametrů jsou nahrazovány novými náhodně vybranými. [57]

Další skupinou upravených DE jsou algoritmy zaměřující se na multimodální optimalizaci. Multimodální optimalizace (MO) je typ optimalizačního problému, kde existuje více než jedno optimální řešení. V kontextu evolučních algoritmů je cílem MO identifikovat co nejvíce těchto optimálních řešení v jediném běhu algoritmu.

Pro MO je často využíváno tzv. neighborhood mutation (NM). NM je koncept, kdy se mutace provádí pouze mezi jedinci, kteří jsou si navzájem "blízcí" podle určitého kritéria, například vzdáleností v prostoru řešení, spadají do stejné niky (niche). Tímto způsobem se každý jedinec vyvíjí směrem k nejbližšímu optimu a snižuje se pravděpodobnost vytváření vektorů kombinacemi vektorů mezi jedinci z různých nik.

Příklady algoritmů DE pro MO jsou Neighborhood-based Crowding Differential Evolution (NCDE), Neighborhood-based Speciation Differential Evolution (NSDE) a Neighborhood-based Sharing Differential Evolution (NShDE).

Rozdělení jedinců do nik může být realizováno pomocí některé z následujících metod – Crowding (NCDE), Fitness sharing (NShDE), Clearing, Speciation (NSDE). [58]

5 SOMA

SOMA, což je zkratka pro Self-Organizing Migrating Algorithm, je typ stochastického optimalizačního algoritmu. Tento algoritmus je založen na modelování sociálního chování skupin jedinců a jejich interakce v prostředí. [26]

V rámci algoritmu SOMA, jednotliví jedinci (nazývané také jako "částice" nebo "řešení") procházejí procesem "migrace", během něhož se pohybují v prostoru řešení s cílem najít optimální řešení daného problému. Tento proces je inspirován přirozenými procesy, jako je migrace zvířat.

Jednotlivci v SOMA nejsou vytvářeni noví, ale pouze se mění jejich pozice v prostoru řešení. To znamená, že algoritmus SOMA se zaměřuje na adaptaci a učení stávajících jedinců, místo na generování nových jedinců.

SOMA byl úspěšně aplikována v řadě různých oblastí, včetně učení neuronových sítí či identifikace prediktivních modelů. Díky své flexibilitě a adaptabilitě je SOMA schopen efektivně řešit širokou škálu optimalizačních problémů.

5.1 Průběh algoritmu

Před zahájením algoritmu je nutné definovat parametry SOMA. Tyto parametry, které mohou zahrnovat velikost populace, délku cesty, krok a další, určují, jak se jednotlivci v populaci pohybují v prostoru řešení. Správné nastavení těchto parametrů je klíčové pro účinnost a efektivitu algoritmu.

Po definici parametrů následuje krok vytvoření populace. V tomto kroku je generována populace jedinců s náhodnými pozicemi v prostoru řešení. Každý jedinec je reprezentován vektorem parametrů, který definuje jeho pozici v prostoru řešení.

Následuje migrační smyčka. Během této fáze je každý jedinec vyhodnocen pomocí tzv. fitness funkce a je vybrán jedinec s nejlepší hodnotou, tzv. Leader. Poté všichni ostatní jedinci začnou "migrovat" k Leaderovi nebo jinak dle zvolené varianty algoritmu. Každý jedinec je vyhodnocen po každém kroku a tento proces pokračuje, dokud nedosáhne nové pozice definované délkou cesty. Tento proces je základem mechanismu "samoorganizace" v algoritmu SOMA.

Migrační smyčka se opakuje po určitý počet iterací. Počet iterací je obvykle předem daný jako jeden z parametrů algoritmu. Opakovaná migrace umožňuje jedincům prozkoumat prostor řešení a postupně se přiblížit k optimálnímu řešení.

Po ukončení všech migračních smyček je jako optimální řešení vybrán jedinec s nejlepší hodnotou fitness funkce. Tento jedinec reprezentuje optimální řešení daného problému a jeho vektor parametrů může být použit pro další analýzu.

5.2 Parametry

V algoritmu SOMA se používají následující parametry:

Velikost populace (PopSize): Tento parametr určuje počet jedinců v populaci. Každý jedinec reprezentuje jedno možné řešení problému.

Délka cesty (PathLength): Tento parametr určuje, jak daleko se jedinec může pohybovat od své původní polohy během jedné migrace.

Krok (Step): Tento parametr určuje, jak velký je jednotlivý "skok" jedince během migrace.

Pravděpodobnost Perturbace (PRT): Tento parametr určuje pravděpodobnost, s jakou se jedinec může náhodně pohnout během migrace. Tím se zvyšuje diverzita populace a pomáhá předcházet uvíznutí v lokálních optimech.

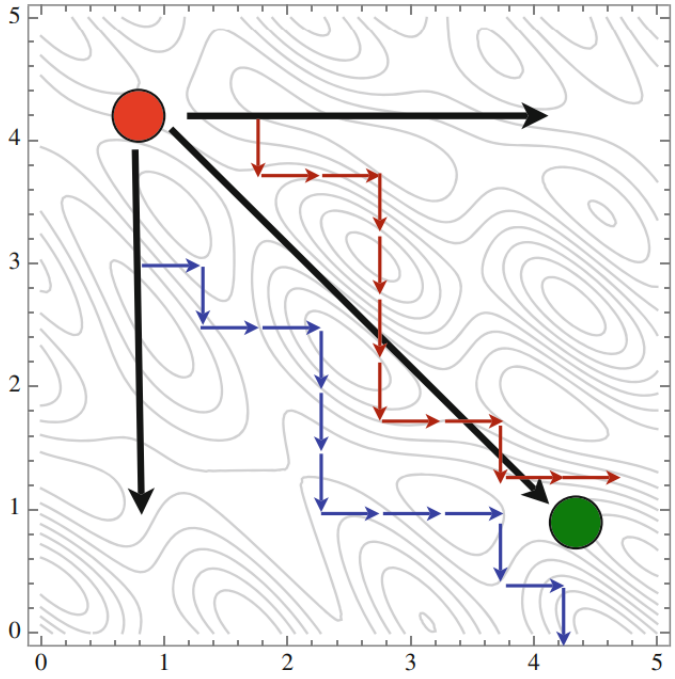
Všechny tyto parametry jsou definovány na začátku algoritmu a ovlivňují jeho chování během celého procesu. Je důležité pečlivě zvolit hodnoty těchto parametrů, aby byl algoritmus efektivní a schopen najít optimální řešení daného problému.

5.3 Proces migrace

Pakliže máme jedince P (na pozici $x_{i,j}^{k}$), kterého chceme přesunout na novou pozici $x_{i,j}^{k+1}$ směrem k Leaderovi L (na pozici $x_{L,j}^{k}$) a prostor řešení je n-rozměrný prostor, takže pozice P a L jsou reprezentovány jako n-rozměrné vektory, pak můžeme proces migrace popsat následovně. Nejprve se vypočítá vektor směru D od P k L. Tento vektor se získá odečtením pozice P od pozice L $\left(x_{L,j}^{k} - x_{i,j}^{k} \right)$. Pak se vektor D vynásobí parametrem cesty (t), kroky od i-tého řešení k Leaderovi. Tím se určí konečný cíl migrace C. Nakonec se jedinec P postupně posouvá k cíli C krok za krokem po trajektorii ovlivněné PRT ($PRTVector$). Velikost každého kroku je dána parametrem kroku (Step).

$$x_{i,j}^{k+1} = x_{i,j}^{k} + \left(x_{L,j}^{k} - x_{i,j}^{k} \right) * t * PRTVector$$

$PRTVector$ je 1, pokud je náhodně vygenerované číslo menší než definovaný parametr PRT, v opačném případě je 0.



Obrázek 3 Vliv PRT vektoru na pohyb částice při přepočtu po každém kroku. Černé šipky znázorňují složky směrového vektoru a červené a modré šipky ukazují jednu z mnoha možných trajektorií, kterou se částice může pohybovat. [26]

V každém kroku se hodnotí fitness funkce pro novou pozici a pokud je nová pozice lepší (má vyšší hodnotu fitness funkce) než původní nejlepší pozice jedince P, pak se tato nová pozice stane novým pbest pro jedince P. Tento proces se opakuje pro každého jedince v populaci (kromě Leadera), což vede k postupnému "přesunu" populace k nejlepšímu dosud nalezenému řešení.

Je důležité poznamenat, že v průběhu migrace by se jedinci nikdy neměli přesunout přímo do polohy Leadera. Místo toho se pohybují v jeho směru, ale jejich konečná poloha závisí na parametrech délky cesty a kroku. Toto umožňuje algoritmu SOMA prozkoumat prostor řešení kolem nejlepšího dosud nalezeného řešení a potenciálně najít ještě lepší řešení.

5.4 Varianty algoritmu SOMA

Existuje několik variant algoritmu SOMA, které se liší v základní strategii pohybu jedinců. Tyto varianty zahrnují:

AllToOne: Toto je základní strategie, která byla popsaná výše. V této strategii všichni jedinci migrují směrem k Leaderovi, kromě Leadera, který zůstává na své pozici.

AllToAll: V této strategii neexistuje Leader. Všichni jedinci se pohybují směrem k ostatním jedincům. Tato strategie je výpočetně náročnější, ale často potřebuje méně hodnocení fitness funkce k dosažení globálního optima než strategie AllToOne.

AllToAllAdaptive: Rozdíl mezi touto a předchozí verzí je, že jedinci nezačínají novou migraci ze stejné staré pozice (jako v AllToAll), ale z poslední nejlepší pozice nalezené během posledního cestování k předchozímu jedinci.

AllToRand: Toto je strategie, kde se všichni jedinci pohybují směrem k náhodně vybranému jedinci během migrační smyčky, bez ohledu na hodnotu fitness tohoto jedince. Existují dvě podstrategie. Buď je počet náhodně vybraných jedinců konstantní během celého procesu SOMA, nebo je pro každou migrační smyčku aktuální počet jedinců určen náhodně.

Clusters: Tato verze SOMA s Clustery může být použita v jakékoliv z výše uvedených strategií. Po výpočtu klastrů jsou vytvořeny klastry se svými Leadery a každý jedinec patří do jednoho klastru. V případě, že všichni jedinci vytvářejí svůj vlastní klastr (1 jedinec = 1 klastr), pak každý jedinec bude skákat směrem ke všem ostatním, což je totožné se strategií AllToAll.

Tyto varianty se liší v způsobu, jakým ovlivňují pohyb jedinců v prostoru řešení, a tím ovlivňují schopnost algoritmu SOMA najít optimální řešení.

5.5 Vylepšení algoritmu

Algoritmus SOMA, který byl původně koncipován pro spojité jednoúčelové problémy s omezenými hranicemi, prošel řadou transformací a vylepšení. Jednou z variant navržených pro řešení problémů s omezeními je Constained Self Organizing Migrating Genetic Algorithm C-SOMGA. [59] Tato varianta implementuje sofistikovanou strategii pro zpracování omezení, která zahrnuje použití speciálně navržených penalizačních funkcí. Tyto funkce penalizují řešení, která porušují omezení, což algoritmu umožňuje efektivně prozkoumávat vyhledávací prostor a současně dodržovat daná omezení. Zároveň se jedná o spojení algoritmu SOMA s binární GA – SOMGA. SOMGA algoritmus využívá selekce, mutace a křížení z GA a vlastností malé velikosti populace, organizace a migrace ze SOMA. [60]

Další z variant SOMA, Multi-Objective Self-Organizing Migrating Algorithm (MOSOMA), se zaměřuje na řešení multiobjektivních problémů. Tato verze algoritmu využívá koncept

Pareto-optimality (Pareto optimality), což je stav, kdy nelze žádnou z cílových funkcí zlepšit bez zhoršení alespoň jedné z ostatních. Tímto způsobem je algoritmus schopen hledat taková řešení, která optimalizují všechny cílové funkce současně. [61]

Další významnou inovací v oblasti algoritmu SOMA je varianta tohoto algoritmu pro řešení diskrétních problémů Discrete Self-Organising Migrating Algorithm (DSOMA). Tato varianta algoritmu byla vyvinuta k řešení kombinatorických optimalizačních problémů založených na permutacích. [62]

Self-Adapting Self-Organizing Migrating Algorithm (SASOMA) je varianta SOMA navržená za účelem snížení omezení vyplývající z nutnosti ladit parametry nastavení původního algoritmu a tím zvýšit schopnost přizpůsobit se optimalizačnímu problému. Migrační strategie v SASOMA se volí na základě počtu vylepšených jedinců. Leader se nevybírá z populace, ale je vytvořen na základě tří různých jedinců z populace, přičemž princip tvorby vůdce závisí na zvolené migrační strategii. Perturbační vektory se vytvářejí na základě skupin interagujících proměnných, anebo je parametr PRT přizpůsoben dle předchozích výsledků. Adaptována je také velikost kroku. Diverzita populace je zajištěna pravidelnou výměnou části jedinců a využitím externího archivu. [63]

Každá z představených variant přináší své vlastní inovace a vylepšení, které rozšiřují možnosti a efektivitu původního algoritmu SOMA. Tyto inovace a vylepšení představují významný přínos pro oblast optimalizačních algoritmů a otevírají nové možnosti pro řešení složitých optimalizačních problémů.

6 ANALYTICKÉ PROGRAMOVÁNÍ

Analytické programování je metoda, která se využívá při aplikaci symbolické regrese.

6.1 Symbolická regrese

Symbolická regrese je proces, při kterém se sestavují jednoduché prvky do složitějších komplexních celků. Tento proces probíhá kombinací elementárních objektů pomocí prvků vybraných z množiny definovaných vztahů, které mohou mezi objekty vznikat. Hledání vhodného řešení za běhu algoritmu probíhá typicky pomocí evolučních výpočetních technik. [64]

Vedle analytického programování jsou dalšími metodami využívanými při symbolické regresi například genetické programování a gramatická evoluce. Při genetickém programování jsou objekty reprezentované pomocí stromových grafů v jazyce LISP. Při aplikaci evolučních technik pak dochází ke kombinování podmnožin grafů. [65] Algoritmy pro gramatickou evoluci mohou být implementovány v libovolném programovacím jazyce a jedinci jsou reprezentováni binárními řetězci s mapováním do takzvané Backus-Naur Formy. [66]

Příkladem využití symbolické regrese je aproximace dat, získání matematického vzorce pro funkci, která data popisuje. Další využití je při syntéze elektrických obvodů, syntéze algoritmů či v teorii řízení.

6.1.1 Příklad aproximace dat

Symbolická regrese je velmi užitečná technika pro aproximaci dat a získání matematického vzorce, který tyto data popisuje. Prvním krokem v tomto procesu je shromažďování dat, která chceme aproximovat. Tato data mohou být získána z různých zdrojů, například z měření v laboratoři, sběru dat v terénu nebo z databází.

Po shromáždění dat následuje fáze přípravy dat pro analýzu. Tato fáze může zahrnovat čištění dat, odstraňování chyb a extrémních hodnot, a normalizaci dat tak, aby byla všechna v podobném měřítku.

Poté je třeba vybrat vhodný model pro aproximaci dat. Model může být jednoduchý, jako je lineární nebo kvadratický model, nebo složitější, jako je polynomiální nebo exponenciální model. Výběr modelu závisí na povaze dat a na tom, jak dobře různé modely data popisují.

Jakmile je model vybrán, je symbolická regrese použita k nalezení nejlepšího možného matematického vzorce, který data popisuje.

Samotný proces využití symbolické regrese pro aproximaci dat zahrnuje kombinaci proměnných, konstant, funkcí a matematických operátorů. Tento proces může být rozdělen do několika klíčových kroků.

V první fázi se definují základní prvky, které budou použity při syntéze vzorce. Tyto prvky mohou zahrnovat proměnné, jako je *x* a *y*, které reprezentují různé aspekty dat, které chceme modelovat.

Konstanty, jako je π, mohou být také součástí základní sady prvků. Tyto konstanty mohou hrát klíčovou roli při modelování určitých typů dat. Například, pokud se snažíme modelovat data, která vykazují periodické oscilace, může být konstanta π užitečná při vytváření modelu, který zahrnuje sinusové nebo kosinusové funkce.

Kromě proměnných a konstant jsou také důležité jednoduché funkce, jako jsou sinus a kosinus. Tyto funkce mohou být užitečné při modelování dat, která vykazují periodické nebo oscilační vzorce. Například, pokud se snažíme modelovat denní teploty v průběhu roku, může být sinusová nebo kosinusová funkce užitečná pro zachycení sezónních oscilací teplot.

Matematické operátory, jako jsou sčítání, odečítání, násobení a dělení, jsou také klíčovou součástí procesu symbolické regrese. Tyto operátory jsou použity ke kombinování základních prvků do složitějších výrazů a vzorců. Například, můžeme vytvořit model, který kombinuje sinusovou funkci a konstantu π pomocí násobení, jako je $sin(\pi * x)$.

Po definování základních prvků je proces symbolické regrese, typicky s pomocí evolučních technik, použit k syntéze vzorce, který nejlépe odpovídá datům. Tento proces je iterativní a zahrnuje vytváření a testování různých kombinací prvků, až je nalezen vzorec, který nejlépe odpovídá datům.

Například, můžeme začít s jednoduchým modelem, jako je $y = x$. Tento model poté můžeme postupně komplikovat přidáváním dalších prvků, jako jsou konstanty, funkce a operátory. Takto můžeme postupně vytvořit složitější modely, jako je $y = sin(\pi * x)$ nebo například $y = x^2 + \pi$. Výsledkem je matematický vzorec, který může poskytnout hluboké pochopení struktury a dynamiky dat, která se snažíme modelovat.

Po vytvoření modelu je důležité ověřit a validovat jeho přesnost. To může zahrnovat porovnání výsledků modelu s reálnými daty, křížovou validaci nebo použití jiných statistických testů. Tento krok je klíčový pro zajištění, že model je přesný a spolehlivý.

Nakonec, jakmile je model ověřen a validován, může být použit k předpovídání nových dat nebo k další analýze existujících dat.

6.2 Princip AP

Principy analytického programování (AP) byly představeny v roce 2001. [67] AP umožňuje analyzovat a manipulovat s jednoduššími prvky a kombinovat je za účelem získání požadovaných složitějších celků, což je základním kamenem pro řešení komplexních problémů.

Kombinace jednodušších prvků může probíhat pomocí různých evolučních výpočetních technik. Implementace algoritmů není omezena na specifický programovací jazyk, což umožňuje využití AP v různých programovacích prostředích a na různých platformách. Tato vlastnost zvyšuje adaptabilitu AP a umožňuje široké uplatnění v různých oblastech. [68]

Množina prvků, ze kterých probíhá syntéza v AP se nazývá general functional set (GFS) a obsahuje funkce, operátory a terminály. Obsah GFS je určen povahou řešeného problému. Například v případě matematických problémů mohou terminály představovat proměnné nebo konstanty. Více informací o obecné syntéze modelu lze dohledat v kapitole 6.1.

Klíčovým aspektem AP je způsob, jakým jsou jedinci reprezentováni a mapováni na prvky z množiny GFS. Každý jedinec v AP je reprezentován množinou celých čísel, které představují indexy prvků z množiny GFS. Například, jedinec může být reprezentován množinou *[1, 2, 3]*, kde *1, 2* a *3* jsou indexy prvků v množině GFS. Následně se jedinci mapují na prvky z množiny GFS. To znamená, že každé celé číslo v reprezentaci jedince je nahrazeno odpovídajícím prvkem z množiny GFS. [68] Například, pokud množina GFS obsahuje prvky *[+, *, x, y, π]*, pak jedinec reprezentovaný množinou *[1, 3, 5]* by byl mapován na vzorec $x + \pi$.

Výsledkem tohoto mapování je složitější struktura, v tomto příkladě matematický vzorec. Tento vzorec může být poté použit k výpočtu hodnoty účelové funkce, která slouží jako měřítko úspěšnosti daného jedince, například jak dobře program předpovídá výsledky na základě daných vstupních dat.

Celý tento proces je iterativní a zahrnuje vytváření a testování různých jedinců, až je nalezen jedinec, který nejlépe řeší daný problém. K vytváření nových jedinců je využito již zmíněných evolučních technik, například těch dále rozebíraných v kapitolách 2, 2, 3 a 4.

7 NEURONOVÉ SÍTĚ

Neuronová síť (NN) je výpočetní model inspirující se chováním neuronů v lidském mozku. Tento model se skládá z algoritmů a matematických modelů, které umožňují počítači učit se a provádět specifické úkoly prostřednictvím zpracování dat. Neuronové sítě jsou základem programovacího paradigmatu, kde místo toho, abychom počítači přesně řekli, co má dělat, ho naprogramujeme, aby se sám učil řešit problém na základě vstupních dat.

Jedním z klíčových typů NN jsou hluboké neuronové sítě, které se učí pomocí technik známých jako hluboké učení (deep learning). Tyto techniky byly vyvinuty v roce 2006 a od té doby dále rozvíjeny. Dnes jsou techniky hlubokého učení schopny dosahovat vynikajících výsledků v mnoha důležitých problémech v oblastech jako je počítačové vidění, rozpoznávání řeči a zpracování přirozeného jazyka. [69] [70]

7.1 Proces vytvoření neuronové sítě

Proces vytvoření neuronové sítě se skládá z několika kroků:

První krok je příprava prostředí. Častou volbou pro implementaci je Python a jeho knihovny, jako je NumPy pro reprezentaci vstupů, a Keras nebo PyTorch pro vývoj a hodnocení NN. [71] [72] [73] Druhý krok je příprava dat, která budou sloužit jako vstup do NN. Data by měla být normalizována a převedena na vhodný formát. Třetí krok je definice modelu NN. To zahrnuje volbu počtu vrstev, počtu neuronů v každé vrstvě, aktivačních funkcí a dalších parametrů. Tak je určena struktura NN. Čtvrtý krok je kompilace modelu, zde je třeba definovat optimalizační a ztrátovou funkci, tedy jakým způsobem se model bude učit a měřit svůj výkon. Pátý krok je trénování modelu. To zahrnuje předání vstupních a výstupních dat modelu a nastavení parametrů jako je počet epoch a velikost dávky (batch). Posledním krokem je použití modelu k předpovědi výsledků na základě nových dat a vyhodnocení výsledků.

7.1.1 Definice modelu NN

Při definici modelu je specifikována struktura NN. Prvním krokem je volba typu modelu. Pak jsou definovány jeho vrstvy. První vrstva je vrstva vstupní a definuje počet vstupních parametrů. Následují další vrstvy, a nakonec vrstva výstupní s definicí počtu výstupních parametrů. V rámci definice vrstev specifikujeme jejich typ, funkci, počet neuronů a jejich

propojení. V následujících podkapitolách jsou představeny možné typy zmíněných nastavení při implementaci NN s pomocí knihovny Keras.

7.1.1.1 Typ modelu

V knihovně Keras existují dva hlavní typy modelů: Sequential a Functional API. Sequential model je lineární řazení vrstev, kde každá vrstva má přesně jeden vstupní vektor a jeden výstupní vektor. Tento model je vhodný pro většinu jednoduchých architektur neuronových sítí, kde data prochází od vstupu k výstupu bez složitých spojení mezi vrstvami. Functional API je způsob, jak vytvořit modely s flexibilnější architekturou než Sequential model. S tímto modelem lze vytvářet modely s více vstupy, více výstupy nebo modely, které sdílejí vrstvy. Také lze vytvářet modely s nelineární topologií, například s reziduálními spojeními.

7.1.1.2 Typ vrstvy

V knihovně Keras existuje řada typů vrstev, které lze použít v závislosti na typu řešeného problému, může se jednat například o některý z následujících:

Dense: Toto je základní vrstva plně propojených neuronů. Každý neuron v této vrstvě je spojen se všemi neurony předchozí vrstvy.

Convolutional (Conv2D): Tyto vrstvy se často používají v konvolučních neuronových sítích (CNN), které jsou široce používány pro zpracování obrazu. Aplikují konvoluční operaci na vstupní data a mohou zachytit prostorové vlastnosti obrazu, jako jsou hrany, rohy atd.

Recurrent (RNN, LSTM, GRU): Tyto vrstvy se používají v rekurentních neuronových sítích, které jsou efektivní pro zpracování sekvencí dat, jako je text nebo časové řady. LSTM (Long Short Term Memory) a GRU (Gated Recurrent Unit) jsou speciální typy rekurentních vrstev, které mohou lépe zachovávat dlouhodobé závislosti v datech.

Dropout: Dropout je technika, která náhodně nastaví určitý podíl vstupních jednotek na nulu během každé aktualizace během trénování. To pomáhá předcházet přeučení.

Pooling (MaxPooling2D, AveragePooling2D): Pooling vrstvy se často používají v konvolučních neuronových sítích jako způsob pro snížení rozměrnosti vstupních dat.

Embedding: Tato vrstva se často používá při zpracování přirozeného jazyka, kde jsou slova převedena na vektory. Embedding vrstva mapuje diskrétní kategorické hodnoty na kontinuální vektorový prostor.

7.1.1.3 Počet neuronů ve vrstvě

Volba počtu neuronů v jedné vrstvě neuronové sítě je důležitým rozhodnutím, které může ovlivnit výkon celého modelu. Neexistuje dané pravidlo pro určení optimálního počtu neuronů, správná volba záleží na mnoha parametrech.

Obecně platí, že čím složitější je problém, tím více neuronů bude pravděpodobně potřeba. Pokud je k dispozici velké množství trénovacích dat, lze mít více neuronů s nižším rizikem přeučení. Jedním z nejlepších způsobů, jak určit optimální počet neuronů, je experimentování s různými konfiguracemi a porovnávání jejich výkonu na validační sadě dat.

Existují také různé heuristické metody, které mohou poskytnout výchozí bod pro zvolení počtu neuronů, například průměr počtu neuronů vstupní a výstupní vrstvy, nebo počet neuronů menší než dvojnásobek počtu vstupů. [74]

7.1.1.4 Aktivační funkce

Aktivační funkce jsou důležitou součástí neuronových sítí. Při analogii s biologickými neurony, aktivační funkce určuje, zda bude neuron přenášen signál a s jakou intenzitou. To umožňuje přidání nelinearity do modelu, a tím možnost učení složitějších vzorců v datech. Existuje několik různých typů aktivačních funkcí, které mohou být použity v závislosti na specifikách modelu.

ReLU (Rectified Linear Unit): ReLU je nejčastěji používaná aktivační funkce v hlubokých neuronových sítích. Vrací 0 pro všechny záporné vstupy a pro kladné vstupy vrací vstupní hodnotu. To umožňuje modelu se rychleji učit a pomáhá řešit problém mizení gradientu.

Sigmoid: Sigmoid funkce vrací hodnotu mezi 0 a 1, což ji činí užitečnou pro binární klasifikaci a pravděpodobnostní predikce. Sigmoid funkce však trpí problémem mizení gradientu, kdy gradienty mohou být velmi malé, což zpomaluje učení.

Tanh (Hyperbolic Tangent): Tanh funkce je podobná sigmoid funkci, ale vrací hodnotu mezi -1 a 1. To znamená, že výstupy jsou centrovány kolem 0, což může usnadnit učení. Ale stejně jako sigmoid funkce, i tanh může trpět problémem mizení gradientu.

Softmax: Softmax funkce se často používá v poslední vrstvě vícevrstvé klasifikace, kde vrací pravděpodobnosti pro každou třídu, které se sečtou do 1. To umožňuje modelu vracet pravděpodobnostní distribuci přes více tříd.

7.1.2 Kompilace modelu NN

Kompilace modelu v Keras je krok, kde je definována ztrátová funkce, optimalizátor a sledované metriky.

7.1.2.1 Ztrátová funkce (loss)

Tato funkce měří, jak dobře se model učí během tréninku. Cílem je minimalizovat hodnotu této funkce. Existuje mnoho různých ztrátových funkcí, a každá z nich je vhodná pro různé typy úloh. Lze použít například některou z následujících:

Mean Squared Error (MSE): Tato funkce je často používána v regresních problémech a je definována jako průměr kvadratických rozdílů mezi skutečnými a předpovězenými hodnotami.

Binary Cross-Entropy: Tato funkce se používá pro binární klasifikační problémy.

Categorical Cross-Entropy: Tato funkce se používá pro více třídní klasifikační problémy. Měří odchylku mezi skutečnými a předpovězenými pravděpodobnostmi pro každou třídu.

Kullback-Leibler Divergence (KLD): Tato funkce měří rozdíl mezi dvěma pravděpodobnostními distribucemi. To je užitečné v některých specifických případech, jako je trénink generativních modelů.

Huber loss: Tato funkce je kombinací MSE a MAE (Mean Absolute Error) a je méně citlivá na odlehlé hodnoty než MSE. To je užitečné pro regresní problémy, kde mohou být přítomny odlehlé hodnoty.

7.1.2.2 Optimalizátor (optimizer)

Optimalizátor je algoritmus, který se používá k aktualizaci vah modelu na základě gradientu ztrátové funkce. Existuje mnoho různých optimalizátorů, jako je SGD (Stochastic Gradient Descent), Adam, RMSprop, atd. Každý z nich má různé vlastnosti, které mohou ovlivnit rychlost a kvalitu trénování vašeho modelu, jedná se například o:

Stochastic Gradient Descent (SGD): Toto je nejjednodušší typ optimalizátoru. Aktualizuje modelové parametry podél směru, který minimalizuje ztrátu. SGD často vyžaduje více času na konvergenci než jiné optimalizátory, ale jeho jednoduchost a efektivita v práci s pamětí ho činí vhodným pro velké datasety.

Momentum: Momentum je varianta SGD, která zahrnuje "momentum" termín. To pomáhá optimalizátoru překonat lokální minima a rychleji konvergovat k optimálnímu řešení.

Adam: Adam (Adaptive Moment Estimation) je populární optimalizátor, který kombinuje výhody jiných pokročilých technik optimalizace, jako je RMSProp a Momentum. Adam se automaticky přizpůsobuje na základě odhadovaných prvních a druhých momentů gradientů, což ho činí efektivním ve většině aplikací hlubokého učení.

AdaGrad: AdaGrad (Adaptive Gradient Algorithm) je optimalizátor, který upravuje učící se rychlost pro každý parametr individuálně na základě historických gradientů. To může být užitečné pro problémy s řídkými daty, ale může také způsobit předčasné zastavení učení v hlubokých neuronových sítích.

RMSprop: RMSprop (Root Mean Square Propagation) je optimalizátor, který se vyvinul z AdaGrad a je velmi efektivní pro nekonvexní optimalizaci. RMSprop se liší od AdaGrad tím, že upravuje svou učící se rychlost na základě průměru nedávných gradientů místo celkového historického gradientu.

7.1.2.3 Metriky (metrics)

Metriky jsou funkce, které se používají k sledování výkonu modelu během tréninku a testování. Nejsou použity během tréninku k aktualizaci modelu, ale jsou vypočítány po každé epoše pro hodnocení. Lze sledovat libovolný počet metrik, které jsou relevantní pro daný problém, například:

Accuracy: Toto je podíl správně klasifikovaných příkladů. Tato metrika je vhodná pro klasifikační problémy s rovnoměrně rozdělenými třídami.

Precision, Recall, F1-score: Tyto metriky jsou užitečné pro klasifikační problémy, kde jsou třídy nerovnoměrně rozděleny, nebo když některé třídy jsou důležitější než jiné. Precision je podíl správně klasifikovaných pozitivních příkladů z celkového počtu příkladů klasifikovaných jako pozitivní. Recall (také známý jako sensitivity nebo true positive rate) je podíl správně klasifikovaných pozitivních příkladů z celkového počtu skutečných pozitivních příkladů.

Mean Squared Error (MSE) a Mean Absolute Error (MAE): Tyto metriky jsou běžně používány pro regresní problémy. MSE je průměr kvadratických rozdílů mezi skutečnými a předpovězenými hodnotami, zatímco MAE je průměr absolutních rozdílů.

7.1.3 Trénování modelu NN

Pro trénování modelu slouží v knihovně metoda *fit()*. Tato metoda přijímá kromě vstupních a výstupních dat také další parametry, které trénování ovlivňují. Jedná se například o:

epochs: Počet epoch je počet průchodů celým trénovacím datasetem. Například při 1000 trénovacích vzorků a batch_size 10, znamená jedna epocha 100 iterací (tj. každá iterace zpracuje 10 vzorků). Větší počet epoch může vést k lepšímu výkonu modelu na trénovacích datech, ale také může způsobit přeučení.

batch_size: Batch size je počet trénovacích vzorků, které jsou zpracovány najednou během jedné iterace gradientního sestupu. Menší batch size může vést k rychlejšímu učení, ale také k nestabilnější konvergenci. Naopak větší batch size může vést ke stabilnější konvergenci, ale pomalejšímu učení.

validation_data: Toto je tuple dvou Numpy polí, které reprezentují validační data a cíle. Tyto data jsou použity k ověření výkonu modelu po každé epoše.

validation_split: Toto je hodnota mezi 0 a 1, která určuje podíl trénovacích dat, která budou použita jako validační data. Data jsou rozdělena na základě tohoto parametru pouze pokud není poskytnut argument validation_data.

callbacks: Callbacks jsou funkce, které mohou být aplikovány na určité stavy tréninkového procesu, jako je konec epochy. Existuje mnoho předdefinovaných callbacků v Keras, například ModelCheckpoint, který pravidelně ukládá model během tréninku, nebo EarlyStopping, který ukončí trénink, když se výkon na validačních datech přestane zlepšovat.

verbose: Tento parametr určuje, jaký typ výstupu bude vidět během tréninku v konzoli.

7.1.4 Vyhodnocení natrénovaného modelu NN

Ověření výkonu natrénovaného modelu neuronové sítě je klíčovou součástí procesu strojového učení. První krokem, který je potřeba provést již před začátkem celého procesu učení je rozdělit data na tréninkovou, validační a testovací sadu. Tréninková sada se používají k učení modelu, validační sada se používá k ladění parametrů a hodnocení výkonu během tréninku, a testovací sada se používá k finálnímu ověření výkonu modelu.

Pomocí nově natrénované NN lze také vytvořit predikce na testovací sadě a poté porovnat tyto predikce se skutečnými hodnotami. Pro klasifikační úlohy mohou být tyto metrikami pro porovnání přesnost (accuracy), recall, precision, F1 skóre atd. Pro regresní úlohy mohou

být tyto metriky mean squared error (MSE), root mean squared error (RMSE) nebo mean absolute error (MAE).

7.2 Výhody a nevýhody NN

Výhody neuronových sítí spočívají především v jejich efektivitě. Díky jejich adaptivním výpočetním mechanismům jsou schopné výrazně zlepšit efektivitu optimalizace. Dále jsou neuronové sítě schopné učit se rozhodovat z různých typů dat. Toto je velmi užitečné v případech, kdy je obtížné navrhnout jiné systémy rozhodování. K dalším výhodám patří také optimalizace výběru funkcí.

Nicméně, použití neuronových sítí pro optimalizaci má také své nevýhody. Jednou z nich je jejich "black box" charakter. Ačkoli neuronové sítě mohou poskytnout vynikající výsledky, jejich interní fungování a procesy může být obtížné pochopit a interpretovat.

V neposlední řadě vyžadují neuronové sítě více výpočetního výkonu než některé jiné metody. To může být nevýhodou, pokud jsou omezené výpočetní zdroje. Navíc jsou také náchylné k přeučení, což znamená, že se mohou "příliš dobře" naučit tréninková data a pak nebudou schopné efektivně generalizovat na nová data.

8 ZPRACOVÁNÍ DAT

8.1 Data Melting

Termín "data melting" odkazuje na proces přeformátování datové tabulky ze "širokého" formátu do "dlouhého" formátu. Tento proces je často používán v data science a statistice pro usnadnění analýzy dat.

V širokém formátu jsou jednotlivé měření pro daný subjekt rozděleny do více sloupců. Například, pokud máme data o teplotě v různých městech v různých měsících, každý měsíc by mohl být vlastní sloupec.

Při "meltingu" těchto dat do dlouhého formátu, by všechny měření teploty byly sloučeny do jednoho sloupce, zatímco měsíc a město by byly dalšími sloupci. Takže místo toho, abychom měli sloupce pro "Teplota v lednu", "Teplota v únoru" atd., budeme mít sloupce pro "Měsíc", "Město" a "Teplota".

8.2 Škálování dat

Škálování dat je proces, kterým se standardizují rozsahy hodnot všech proměnných v datové sadě tak, aby byly srovnatelné. Tento proces je zvláště důležitý v kontextu strojového učení a statistické analýzy, kde může mít významný vliv na výsledky modelů.

Pokud mají proměnné v datasetu velmi odlišné rozsahy, může to způsobit problémy při analýze dat. Škálování dat řeší tento problém tím, že upraví rozsahy hodnot, aby byly srovnatelné. Existují různé metody škálování dat, ale jednou z nejběžnějších je normalizace nebo min-max škálování.

Při normalizaci jsou všechny hodnoty převedeny do rozsahu od 0 do 1. To se provádí tak, že se od každé hodnoty odečte minimální hodnota a výsledek se pak podělí rozdílem mezi maximální a minimální hodnotou. Při min-max škálování je zadáván specifický rozsah.

Škálování nemění tvar distribuce dat. Pokud jsou původní data nesymetrická nebo mají výrazné výkyvy, budou taková i po škálování. [75]

8.3 Shluková analýza

Shluková analýza, neboli clustering je metoda používaná v oblasti machine learningu a data miningu. Tato metoda umožňuje seskupit objekty ze zadaného zdroje na základě vzájemné podobnosti do jednotlivých shluků tzv. clusterů. Tento proces je klíčovým nástrojem při explorativní datové analýze.

Pro ilustraci lze použít příklad z oblasti marketingu. Při zadání databáze zákazníků obsahující informace o jejich nákupech, preferencích a demografických údajích, lze pomocí metody clusteringu identifikovat skupiny zákazníků s podobnými nákupními vzorci a preferencemi. Toto rozdělení je pak možné využít například pro cílení marketingových kampaní. [76]

8.3.1 K-means

Algoritmus K-means je populární metoda shlukové analýzy, která se používá k rozdělení sady dat do *k* počtu shluků.

Prvním krokem K-means je inicializace, při ní je třeba určit počet shluků, do kterých chceme data rozdělit, již zmíněné *k*. Poté je náhodně vybráno *k* bodů z datové sady, které budou sloužit jako počáteční středy shluků neboli centroidy.

Následuje přiřazení bodů k centroidům. Každý bod v datové sadě je přiřazen k nejbližšímu shluku. Vzdálenost mezi bodem a centrem shluku se obvykle měří pomocí euklidovské vzdálenosti, ale mohou být použity i jiné metriky.

Posledním krokem je přepočítání centroidů. Po přiřazení bodů k centroidům se středy shluků přepočítají. Nové středové body se stanoví jako průměr všech bodů přiřazených k danému shluku.

Následuje opakování přiřazování k centoridům a jejich přepočítávání, dokud se poloha středů shluků neustálí (tj. dokud se při dalším kroku už neposunou) nebo dokud nebudou splněna jiná zastavovací kritéria (například maximální počet iterací nebo minimální změna v přiřazení bodů).

Algoritmus K-means je jednoduchý a efektivní pro velké datové sady. Nicméně, výsledky mohou být citlivé na počáteční volbu centroidů a algoritmus může uvíznout v lokálním minimu. Pro příliš velké datasety může být také nevýhodou velká výpočetní náročnost. [77]

8.3.2 MiniBatchKMeans

MiniBatchKMeans je variací na tradiční algoritmus K-means, který je navržen tak, aby byl efektivnější při práci s velkými datovými sadami. Tento algoritmus funguje podobně jako standardní K-means, ale s tím rozdílem, že místo použití celého datasetu pro každou iteraci, používá náhodně vybraný vzorek dat, nazývaný mini-batch.

MiniBatchKMeans je efektivnější než tradiční K-means zejména pro velké datasety, protože pracuje s menšími vzorky dat v každé iteraci. Nicméně, kvůli náhodnému výběru mini-batche může být výsledné rozdělení do shluků méně přesné než u K-means. [78]

8.3.3 DBSCAN

DBSCAN, což je zkratka pro "Density-Based Spatial Clustering of Applications with Noise", je metoda shlukové analýzy, která identifikuje shluky jako husté oblasti v prostoru dat, oddělené oblastmi méně hustými, které jsou považovány za šum. Na rozdíl od algoritmu K-means, DBSCAN nevyžaduje předem určený počet shluků a dokáže pracovat s libovolně tvarovanými shluky.

DBSCAN funguje na základě konceptu vzdálenosti epsilon a minimálního počtu bodů. Vzdálenost epsilon je maximální vzdálenost mezi dvěma body, které ještě mohou být považovány za sousedy. Minimální počet bodů je počet sousedních bodů potřebných k tomu, aby byl bod považován za centrální bod.

Prvním krokem DBSCAN je identifikace jádrových bodů (core points). Jádrový bod je bod, který má určený minimální počet bodů ve svém dosahu. To znamená, že v okolí tohoto bodu je dostatek dalších bodů.

Jakmile je identifikován jádrový bod, vytvoří se shluk. Shluk se poté rozšiřuje tím, že se k němu přidají všechny dosažitelné body, což jsou body, které jsou v dosahu jádrového bodu nebo v dosahu jiného bodu ve stejném shluku.

Tento proces se opakuje, dokud nejsou všechny body buď přiřazeny k nějakému shluku nebo označeny jako šum.

Výhodou DBSCAN je jeho schopnost pracovat s libovolně tvarovanými, i velmi obsáhlými daty a odolnost vůči šumu. Nicméně, může mít potíže při práci s daty různých hustot. [79]

II. PRAKTICKÁ ČÁST

9 POPIS ÚLOHY A DAT

Cílem první poloviny praktické části této práce je implementace AP za účelem analýzy možností syntézy modelu popisujícího matematické vztahy mezi parametry nastavení fyzikálního měření a jeho výstupem. Součástí této části je také natrénování NN, za účelem predikce výstupů měření na základě vstupních parametrů. V rámci druhé poloviny praktické části jsou využity algoritmy DE, PSO a SOMA pro optimalizace nastavení parametrů s využitím nejlepšího syntetizovaného modelu a porovnání výsledku s reálnými výsledky měření.

Prvním krokem práce byl sběr dat z měření. Tato data byla poté předzpracována pro další postup, kterým byla snaha o syntézu modelu. Na základě nejlepšího syntetizovaného modelu byla provedena optimalizace nastavení pomocí EA. Výsledky pak byly porovnány mezi jednotlivými EA a s reálným systémem.

9.1 Data

Data byla získána z fyzikálního měření. Jedná se o dataset ve formátu CSV obsahující 759.376 záznamů. Data popisují pět vstupních parametrů nastavení přístroje, parametr popisující fyzikální charakter dat a jeden výstupní parametr – výsledek měření.

Prvních pět zmíněných parametrů představuje napětí. Jeho hodnoty byly voleny náhodně v rozmezí 0-7 jako typ float pro první tři a 0-100 pro zbylé dva. Rozmezí vyplývá z doménové znalosti a bylo dále zúženo vnitřně definovanými omezeními přístroje, jak jde vidět v tabulce na Obrázku 5. Z doménové znalosti také vyplývá, že první tři ze vstupních napětí mají větší vliv na výsledek měření. Šestý vstupní parametr (m, nebo také x_6) souvisí s fyzikálním charakterem měřených částic a je také typu float. Výstup měření udává naměřenou intenzitu a jedná se o velká čísla typu int.

V původním datasetu získaném z měření jsou v rámci jednotlivých řádků udány hodnoty napětí a jednotlivých naměřených intenzit pro nastavení m. Pro lepší představu je možné vidět část datasetu včetně hlavičky na Obrázku 4, hodnoty následující za x_1-x_5 udávají nastavení m. V další kapitole 10 jsou podrobně popsány změny provedené nad datasetem v rámci předzpracování dat, aby byla ve vhodnějším formátu pro další zpracování.

```
x1,x2,x3,x4,x5,42.0,69.0,195.0,262.0,393.0,622.0,922.0,1221.
4.0,5.14,5.38,92.9,7.14,12035798,0,260987,228057,0,270510,27
6.57,5.57,5.38,57.16,85.7,539235,0,0,0,0,0,0,0,0,0,0,0,0,0,0
3.14,0.43,5.38,0.0,35.72,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
5.72,2.14,5.8,78.56,14.29,24135770,3803821,5471577,29882190,
3.14,2.57,4.14,100.0,64.3,3085233,0,0,0,0,0,0,0,0,0,59144,
3.57,1.29,2.9,7.14,64.3,72358976,13069950,0,0,0,183752,21519
7.0,6.0,5.8,100.0,78.56,122488,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.72,0.43,2.49,100.0,7.14,15488290,179100,5154693,15310061,0
2.72,5.57,3.73,0.0,21.42,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2.28,4.72,4.56,0.0,28.58,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
6.14,0.0,1.66,42.84,28.58,45027096,8041836,4550686,14640592,
4.0,2.14,1.66,85.7,21.42,60732120,3153122,12087020,50597308,
6.14,0.86,4.14,7.14,28.58,45702324,11662349,3007218,7056704,
5.28,2.57,2.9,50.0,64.3,99777808,19473502,10816966,36239228,
2.72,5.14,1.66,14.29,78.56,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Obrázek 4 Náhled na data získaných z fyzikálního měření. Prvních pět sloupců udává hodnoty napětí. Následují sloupce s hodnotou m v hlavičce a naměřenou intenzitou s daným nastaveným napětí v rámci řádku. Sloupců je v tomto původním datasetu mnoho proto jsou ukázková data na obrázku zprava useknuta. Data jsou ve formátů CSV a obsahují přes 700 tisíc řádků.

	průměr	std	min	25%	50%	75%	max	rozptyl
x1	4	1.85134	1	2.28	4	5.72	7	3.42744
x2	3	1.85118	0	1.29	3	4.72	6	3.42687
x3	2.9	1.78977	0	1.24	2.9	4.56	5.8	3.20327
x4	50.0033	30.8642	0	21.42	50	78.56	100	952.60146
x5	50.0033	30.8642	0	21.42	50	78.56	100	952.60146
m (průměr)	6597059	9068528	0	0	396745	1.4E+07	3.9E+07	1.48933E+14

Obrázek 5 Základní statistika vypočtená nad původním datasetem. V prvních pěti řádcích si lze povšimnout, jak byla omezena rozpětí nastavení napětí vnitřně definovanými omezeními v rámci stroje, na kterém bylo prováděno měření. V posledním řádku tabulky lze vidět, jak velkou část dat tvoří měření s naměřenou nulovou intenzitou. Tyto data byla později zredukována, jak je popsáno v kapitole 10.

10 ZPRACOVÁNÍ DAT

Před syntézou modelu byla vstupní data upravena do vhodnějšího formátu. Prvním krokem bylo převedení části tabulky do dlouhého formátu tzv. melting. Tento krok byl nutný, protože jeden ze vstupních parametrů byl v získaných datech rozdělen do samostatných sloupců. Výsledkem je dataset s počtem řádků přes 10 miliónů. Z tohoto datasetu bylo poté odstraněno deset procent záznamů s největší hodnotou intenzity, protože se jednalo o velmi odlehlé hodnoty vzhledem k většině měření, které značně zkreslovaly analýzu.

```
x1,x2,x3,x4,x5,m,intensity
4.0,5.14,5.38,92.9,7.14,42.0,12035798
6.57,5.57,5.38,57.16,85.7,42.0,539235
3.14,0.43,5.38,0.0,35.72,42.0,0
5.72,2.14,5.8,78.56,14.29,42.0,24135770
3.14,2.57,4.14,100.0,64.3,42.0,3085233
3.57,1.29,2.9,7.14,64.3,42.0,72358976
7.0,6.0,5.8,100.0,78.56,42.0,122488
2.72,0.43,2.49,100.0,7.14,42.0,15488290
2.72,5.57,3.73,0.0,21.42,42.0,0
2.28,4.72,4.56,0.0,28.58,42.0,0
```

Obrázek 6 Náhled na dataset pro meltingu dat. Hodnoty m tvoří samostatný sloupec.

Dalším krokem bylo škálování dat za účelem získání podobného rozmezí hodnot pro všechny sloupce. Zvoleno bylo rozmezí 0 až 100.

Takto upravený dataset byl použit pro několik prvních testovacích běhů algoritmu pro syntézu modelu. Výsledky byly velmi zkresleny velkým počtem měření, jejichž výsledkem byla nulová intenzita. Počet takovýchto záznamů tvořil více než polovinu dat. Nastavení, které vedlo k naměření nulové intenzity je z hlediska zkoumané problematiky negativní. Nelze je z datasetu ale úplně vyloučit, protože i informace o negativním nastavení je pro syntézu modelu důležitá. Dalším cílem tedy bylo pouze snížit počet záznamů s nulovou intenzitou za současné co nejmenší ztráty informace o nastavení, které k takovému výsledku měření vedlo.

Pro tento účel byla zvolena metoda shlukování. Počet shluků byl zvolen 100.000. První použitou metodou byl k-means. Překážkou se ale stala dlouhá doba běhu algoritmu. Dalším vyzkoušeným algoritmem byl DBSCAN, který se ale vzhledem k charakteru dat ukázal také jako nevhodný. Posledním použitým algoritmem byl MiniBatchKMeans, pomocí kterého

bylo dosaženo nalezení požadovaných shluků za udržitelný čas. Následně byly hodnoty nastavení v rámci shluků zprůměrovány a nahradily všechny původní záznamy s nulovou intenzitou. Takto vznikl nový dataset obsahující asi 4 milióny řádků, z nichž řádky s nulovou naměřenou intenzitou tvoří zhruba 2 %.

Poslední úpravou před dalším použitím dat bylo zamíchání pořadí řádků v rámci datasetu.

```
x1,x2,x3,x4,x5,m,intensity
71.33333,35.66667,100.0,21.42,21.42,21.64273,0.73542
71.33333,14.33333,21.37931,50.0,50.0,5.71126,5.43
35.66667,0.0,57.24138,14.29,35.72,0.0,37.72124
92.83333,0.0,35.68966,35.72,28.58,100.0,8.19064
85.66667,50.0,85.68966,21.42,35.72,5.71126,3.33124
78.66667,64.33333,100.0,100.0,85.7,32.83663,0.08622
92.83333,78.66667,71.37931,57.16,42.84,32.83663,4.16941
71.33333,57.16667,100.0,92.9,64.3,47.76182,5.07968
92.83333,50.0,71.37931,7.14,57.16,47.76182,10.71721
```

Obrázek 7 Náhled na část datasetu po úpravách v rámci fáze předzpracování dat.

Zpracování dat bylo provedenou pomocí skriptů napsaných v programovacím jazyce Python s využitím knihovny Pandas pro snazší práci s daty, scikit-learn pro úpravy dat, jako například škálování a matplotlib pro zobrazení statistických přehledů. [80] [81] [82]

11 STATISTIKA DAT

Před začátkem procesu syntézy modelu byla provedena statistická analýza předzpracovaných dat za účelem dalšího hlubšího porozumění struktuře dat. Ze statistik zobrazených v tabulce na Obrázku 8 vyplývá, že data jsou poměrně široce rozptýlená pro všechny proměnné. To může znamenat, že malé změny v hodnotách vstupů mohou vést k velkým změnám ve výstupu. Toto může značně komplikovat syntézu modelu, protože data mohou obsahovat mnoho výkyvů, na které může být pro model obtížné reagovat.

	mean	std	min	25%	50%	75%	max	std
x1	67.24453	25.84422	0	50	71.33333	85.66667	100	25.84422
x2	36.80534	27.76442	0	14.33333	35.66667	57.16667	100	27.76442
x3	49.12517	29.99607	0	21.37931	50	71.37931	100	29.99607
x4	55.19261	30.62224	0	28.58	57.16	82.69895	100	30.62224
x5	43.51857	29.87929	0	14.29	42.84	64.3	100	29.87929
m	37.66188	26.98634	0	8.23186	44.03052	55.22442	100	26.98634
intensity	37.70606	28.74258	0	11.18598	35.98765	58.56342	100	28.74258

Obrázek 8 Tabulka obsahující základní údaje o rozdělení dat v datasetu po předzpracování.

Pro další analýzu rozdělení dat byly provedeny dva statistické testy. Na základě výsledků testu normality Shapiro-Wilkova lze konstatovat, že zkoumané proměnné nevykazují charakteristiky normálního rozdělení. Tento závěr je podpořen také výsledky Anderson-Darlingova testu, který ukazuje, že data nejsou konzistentní ani s exponenciálním rozdělením. Tyto statistické analýzy naznačují, že distribuce zkoumaných sloupců dat se odchyluje od standardních rozdělení, což může mít významné důsledky pro další analýzy a interpretaci výsledků.

Dále byla provedena analýza variance (ANOVA), pro posouzení, jak proměnné ovlivňují výsledek. Tato analýza ukázala, že proměnné x_1, x_2, x_3, x_5 a m mají statisticky významný vliv na výsledky, zatímco proměnná x_4 pravděpodobně nemá významný vliv.

	sum_sq	df	F	PR(>F)
C(x1)	189838	65	5.06609	2.59E-33
C(x2)	90632.1	64	2.45643	2.63E-09
C(x3)	72801.8	65	1.94281	1.37E-05
C(x4)	24174.7	66	0.63536	0.99019
C(x5)	92775.2	65	2.47583	1.43E-09
C(m)	261653	55	8.25211	4.23E-56
Residual	1073438	1862		

Obrázek 9 Výsledky analýzy ANOVA

Platnost výsledků ANOVA je závislá na předpokladu nezávislosti chyb, tedy na tom, že dvě nebo více proměnných v datové sadě nejsou silně korelované. Pro zjištění multikolinearity bylo využito faktoru inflace variance (VIF) a analýzy korelační matice.

Hodnota VIF pro proměnnou x_1 je 6,59, což je nad hranicí 5. To naznačuje, že existuje vysoká multikolinearita mezi x_1 a ostatními proměnnými. Tato silná korelace může zkreslit odhady koeficientů a způsobit nestabilitu modelu. Toto je v souladu s doménovou znalostí, která potvrzuje význam proměnné x_1. Všechny ostatní proměnné mají VIF menší než 5 (2–3), což naznačuje, že multikolinearita mezi těmito proměnnými a ostatními proměnnými v modelu je relativně nízká.

Níže na Obrázku 10 lze vidět korelační matici. Vyšší korelace s intenzitou je patrná u proměnné x_1 a m. Korelační matice také odpovídá doménové znalosti, ze které vyplývá, že nastavení proměnné x_1 má vliv na x_2 a x_3. Dle korelační matice má nejmenší vliv na intenzitu proměnná x_4, což odpovídá výsledku testu ANOVA.

Obrázek 10 Korelační matice mezi proměnnými x_1-x_5, parametrem m a intenzitou

Znalosti získané statistickou analýzy napomáhají k lepšímu pochopení dat a jsou dále využity při implementaci syntézy modelu. Analýza byla provedena pomocí skriptů v Pythonu a knihoven Seaborn, Pandas, Statsmodels, Matplotlib a Scipy. [80], [82], [83], [84], [85]

12 SYNTÉZA MODELU

Pro analýzu možností modelování dat získaných z měření byly vybrány dvě metody – analytické programování a trénink neuronové sítě blíže popsán v kapitole 12.2. Přestože obě metody představují významné nástroje pro analýzu dat, primární pozornost byla zaměřena na analytické programování, a to z důvodu lepší interpretovatelnosti potenciálně syntetizovaného modelu. Trénink neuronové sítě byl do praktické části začleněn s experimentálním záměrem, a to konkrétně s cílem porovnat výsledky obou metod a poskytnout tak komplexnější pohled na možnosti jejich využití. Tento přístup umožňuje nejen porovnat efektivitu obou metod, ale také poskytnout hlubší vhled do jejich specifických výhod a omezení.

12.1 Analytické programování

Prvním krokem v rámci syntézy model byla implementace základního algoritmu analytického programování popsaná v kapitole 6 ve spojení se základní variantou algoritmu diferenciální evoluce popsané v kapitole 4.

Pro implementaci byl zvolen Python, vzhledem k dostupnosti mnoha knihoven, které implementaci značně ulehčily. Jedná se například o knihovnu Sympy využitou v reprezentaci matematických výrazů v GFS. Dále byly použity knihovny Math a Random pro matematické výpočty a knihovna Numpy pro práci s jedinci v rámci DE. Knihovna scikit-learn byla využita pro výpočet směrodatné odchylky a knihovna Pandas pro manipulaci se soubory s daty ve formátu CSV. [86] [87] [71] [81] [80]

12.1.1 První nastavení parametrů AP a DE

Do GFS v rámci AP bylo zařazeno sčítání, odečítání, násobení, dělení, funkce sinus a kosinus, operátor odmocniny, šest proměnných x_1-x_5 a m pro vstupní parametry, dále konstanty π, e, 1 a 2. Tyto operátory, funkce a proměnné byly zvoleny pro jejich schopnost reprezentovat širokou škálu matematických výrazů a modelů.

název	sčítání	odčítání	dělení	násobení	odmocnina	sinus	kosinus	x1	x2	x3	x4	x5	m	π	e	1	2
počet operátorů	2	2	2	2	1	1	1	0	0	0	0	0	0	0	0	0	0
terminál?	ne	ne	ne	ne	ne	ne	ne	ano	ano	ano	ano	ano	ano	ano	ano	ano	ano
váha terminálu	-	-	-	-	-	-	-	5	5	5	3	3	5	1	1	3	3

Obrázek 11 Přehled obsahu GFS, význam vah terminálů je popsán později v kapitole 12.1.3

Pro funkce, které implementují odmocninu a dělení, byla zahrnuta implementace omezení, aby se zabránilo výpočtům, které by v matematickém kontextu nebyly možné a mohly by vést k nedefinovaným výsledkům. Toto opatření je klíčové pro udržení integrity a spolehlivosti modelu.

Funkce mapující hodnoty z jedinců DE na indexy z GFS byla pečlivě ošetřena tak, aby byly generovány pouze uzavřené matematické výrazy, což zajišťuje, že všechny vygenerované modely budou matematicky validní.

První nastavení pro diferenciální evoluci bylo zvoleno s ohledem na specifickou povahu úlohy. Velikost populace byla nastavena na poměrně vysokou hodnotu 100, což bylo provedeno za účelem umožnění větší diverzity v rámci populace a tím pádem většího prostoru pro evoluční vývoj. Dimenze jedinců byla stanovena na hodnotu 25, což bylo zvoleno tak, aby bylo možné generovat dostatečně dlouhé a komplexní výrazy schopné modelovat naměřená data.

Faktor diferenciace a konstanta křížení byly obě nastaveny na hodnotu 0,8. Toto nastavení bylo zvoleno opět s cílem podpořit diverzitu v rámci populace a tím zvýšit pravděpodobnost nalezení optimálního řešení. Kontrola hranic byla ošetřena metodou reflection.

Fitness jedinců byla hodnocena pomocí střední absolutní odchylky, dále jen odchylky, která byla vypočtena porovnáním intenzity generované výrazy AP po dosazení hodnot jednotlivých proměnných a skutečnou hodnotou intenzity pro příslušné nastavení.

S touto prvotní konfigurací bylo provedeno několik experimentálních běhů. Během těchto běhů bylo zjištěno množství nedostatků v implementaci. Velká část generovaných výrazů byla na první pohled nevhodná, obsahovala například opakovaně stejnou proměnnou nebo byla příliš krátká, což vedlo k suboptimálním výsledkům.

Po těchto počátečních testech bylo postupně přistoupeno k řadě vylepšení. Jejich detaily jsou popsány v následujících podkapitolách. Tato vylepšení vedla k stabilní verzi algoritmu, která dobře konvergovala pro jednoduché testovací výrazy. Aproximace naměřených dat ale stále nevycházela dobře proto bylo na základě dalšího testování a ladění přistoupeno k dalším úpravám algoritmu, včetně využití specifických vlastností vycházejících z doménové znalosti problému.

Testování algoritmu bylo typicky prováděno v rámci dvou set iterací se sadou 5000 záznamů náhodně vybíraných z datasetu.

12.1.2 Ladění složení výrazů

Jedním z prvních identifikovaných zřetelných problémů v algoritmu, byla generace příliš krátkých výrazů. Jedinci v rámci DE byli generováni náhodně. Je důležité připomenout, že GFS byla navržena tak, aby se skládala ze sedmi funkcí a operátorů společně s devíti terminály. V první verzi algoritmu se proto často stávalo, že například první hodnota odpovídala indexu pro operátor a následující dvě hodnoty odpovídaly terminálům, což vedlo k velmi časnému ukončení generování výrazu.

Prvním krokem k optimalizaci bylo tedy upravit generování jedinců tak, aby první část jejich hodnot tvořily indexy pro funkce a operátory, doplněné o terminály na zbývajících pozicích. Počet míst v konečné verzi algoritmu je pro operátory a funkce náhodně určen v rozmezí mezi 2,5násobkem až 1,5násobkem dimenze. Toto rozmezí bylo určeno experimentálně a ukázalo se jako vhodné pro generování dostatečně dlouhých a komplexních výrazů, zatímco zároveň byla zachována diverzita v rámci populace.

Pro další zajištění dostatečné komplexnosti výrazu byla přidána penalizace při výpočtu fitness pro výrazy, které obsahují méně než pět terminálů.

12.1.3 Ladění práce s terminály

Další úpravou byly změny vedoucí k různým pravděpodobnostem výskytu terminálů dle jejich povahy a vlastností v dané fázi generování výrazu. Na základě definice AP jsou terminály dosazovány do výrazu dle indexů obsažených v rámci jedince. V mnoha případech se ale stane, že algoritmus dorazí na konec množiny indexů jedince, aniž by všechny operátory a funkce byly ukončeny terminály. V této fázi pak dochází k volání metody, která tyto části výrazu terminály ukončí. V první verzi algoritmu byly tyto terminály vybírány náhodně, později ale bylo přistoupeno k sofistikovanějšímu výběru pomocí vah přiřazených jednotlivým terminálům. Iniciální nastavení vah pro terminály je přehledně zobrazeno v tabulce na Obrázku 11. Pro všechny terminály byl implementován proces snižování váhy potom, co se ve výrazu vyskytly, aby pravděpodobnost dalšího výskytu stejného terminálu byla snížena. Váhy byly snižovány o hodnotu 2, případně nastavovány na hodnotu 1, pokud by po odečtení hodnoty 2 vyšla záporná hodnota. Další omezení byla definována na základě toho, zde se jednalo o proměnné nebo konstanty.

12.1.3.1 Proměnné

Pro proměnné x_1-x_5 a m byly nastaveny větší váhy než pro konstanty, což odráží jejich význam v kontextu modelu. Pro proměnné x_1, x_2, x_3 a m byly váhy ještě navýšeny, a to na základě doporučení doménového experta, který zdůraznil jejich klíčový význam. Díky implementaci mechanismu, který snižuje váhy proměnných po jejich zařazení do výrazu, bylo také zajištěno, že se s poměrně velkou pravděpodobností objeví ve vygenerovaném výrazu všechny proměnné najednou.

12.1.3.2 Konstanty

Nejnižší váhy byly přiřazeny konstantám π a e, což odráží jejich relativně menší význam v kontextu modelu oproti proměnným. Pro konstantu π byl navíc implementován adaptivní proces, který zvyšuje její váhu, pokud se v předchozích fázích generování výrazu objevily funkce sinus nebo kosinus. Tento krok byl proveden s ohledem na matematické vztahy těchto funkcí s konstantou π.

Pevně stanovené konstanty 1 a 2 byly nahrazeny proměnnou konstantou c. Její váha byla nastavena vyšší než π a e, ale nižší než váhy proměnných x. Navíc byl přidán mechanismus, který zvyšuje její váhu, pokud byl do výrazu zařazen operátor násobení.

Hodnota konstanty c byla původně nastavena na -2,1 a poté byla v osmi iteracích postupně zvětšována o krok 0,5. Pro každou hodnotu byla vypočtena odchylka vygenerovaného výrazu po dosazení konstanty a otestování na pracovní části datasetu.

Vzhledem ke komplexnosti a časové náročnosti výpočtů v rámci celého algoritmu bylo rozhodnuto neimplementovat sofistikovanější mechanismus, i když by to byla zajímavá možnost pro budoucí výzkum, například s využitím algoritmu gradient descent nebo přístupu navrženého v původní literatuře popisující analytické programování. [68]

12.1.4 Omezení počtu vstupních proměnných

Již dříve v práci byl zdůrazněn význam proměnných x_1, x_2 a x_3 na výsledky měření v porovnání s x_4 a x_5. Tato znalost byla předána doménovým expertem a částečně také potvrzena statistickou analýzou provedenou na naměřených datech. Vzhledem k tomu, že ani po implementaci výše popsaných optimalizací nedošlo k významnému vylepšení výsledků algoritmu bylo několik testovacích běhů provedeno také s datasetem, ze kterého byly proměnné x_4 a x_5 odstraněny. Bohužel ani tyto běhy nevykázaly významné zlepšení odchylky hodnot generovaných syntetizovanými výrazy a naměřenými daty.

12.1.5 Testování dalších EA

V této fázi algoritmus s výše popsaným nastavením a po začlenění všech optimalizací stále generoval výsledky, jejichž odchylka se pohybovala kolem hodnoty 30, což se vzhledem k rozmezí hodnot 0-100 nedá považovat za řešení vhodné k dalšímu použití.

V rámci použití DE bylo dále vyzkoušeno různých kombinací typů mutace a křížení, včetně varianty, kdy bylo použito křížení binomické, ale s každou další iterací se zvětšovala pravděpodobnost použití křížení exponenciálního. Tento postup byl zvolen na základě předpokladu, že by se výrazy ze začátku běhu algoritmu mohly křížit binomicky pro zvýšení diverzity a zvýšení pravděpodobnosti nalezení dostatečně dobrého řešení, v rámci kterého by se pak mohly mezi jedinci zaměňovat pomocí exponenciálního křížení pouze části vhodného výrazu. Vyzkoušena byla také varianta DE s adaptivním nastavováním F a CR. Žádná z těchto optimalizací ale nevedla k významnému posunu v syntéze modelu.

Bylo proto přistoupeno k otestování dalších EA v kombinaci s AP místo DE. Konkrétně se jednalo o GA, PSO, SOMA a sofistikovanější adaptivní DE.

GA bylo implementováno v základní variantě s využitím metody Tournament selection pro selekci a dvoubodového křížení. Vzhledem k tomu, že v kontrastu s ostatními EA dochází k ohodnocení jedinců najednou až po vytvoření celé nové populace, bylo možné snadno implementovat velmi efektivní a rychlé multiparalelní zpracování výpočtů i v rámci EA. Díky tomu byl tento algoritmus jediný, který byl testován na větších částech datasetu o 500 tisíc záznamech. Díky jednoduchosti algoritmu bylo také možné snadno vyzkoušet různé další strategie selekce a křížení.

SOMA byla implementována ve variantě All-to-one s nastavením délky cesty 1,5 a krokem 0,11. Pro PSO byly parametry nastaveny následovně, zpomalovací váha na 0,7298 a c_1 i c_2 na hodnotu 1,49618. Pro další porovnání bylo několik testovací běhů provedeno za využití varianty DE DISH. [88]

U všech implementovaných algoritmů bylo využito výše popsané strategie generování jedinců, ošetření hranic metodou reflection, nastavení dimenze na 25, populace zvolena o velikosti 100 a počet iterací nastaven na 200.

Žádný z algoritmů neposkytl dostatečně dobré řešení s odchylkou alespoň menší než dvacet, které by mohlo být použito pro další ladění. Obecně nejmenší odchylky bylo možné pozorovat u algoritmu PSO.

Nejlepší nalezené řešení PSO s odchylkou 22,6 a část jeho vývoje lze vidět níže. Řešení bylo vygenerováno již při inicializace populace, v tomto konkrétním případě tedy nelze úspěch přímo odvodit od použitého EA. V dalších generacích se odchylka zvětšovala, ale u několika dalších výrazů byla pod hodnotou 23. Toto řešení by mohlo být zajímavé z hlediska dalšího výzkumu.

$i = \pi * \left\{ \sqrt{x3 - (\, x1 * \, \mathrm{e}\,)} + [\, \sin(\, x1 + x2\,)\,] \right\}$ *(inicializace, fitness 22,6)*

$$i = \frac{\frac{x5}{x1} + x1 + x6}{\pi}$$ *(15. iterace, fitness 22,9)*

$i = \pi * \left[\sqrt{x1 * e - x2} + \sin(x1 + x4) \right]$ *(20. iterace, fitness 22,8)*

$i = \pi * [\, \sin(x1 + x4) + \sqrt{x1 * e - x2}\,]$ *(22. iterace, fitness 22,9)*

$i = \pi * [\, \sin(x1 + x3) + \sqrt{x1 * e - x3}\,]$ *(25. iterace, fitness 22,7)*

$i = \pi * [\, \sin(x2 + x3) + \sqrt{x1 * e - x2}\,]$ *(27. iterace, fitness 22,9)*

$$i = \frac{x1 + x6 + \frac{x1 - x4 - x3}{x2}}{\pi}$$ *(33. iterace, fitness 22,8)*

$i = \pi * [\, \sin(\pi + x5) + \sqrt{x1 * e - x3}\,]$ *(47. iterace, fitness 22,7)*

$i = \pi * [\, \sin(\pi + e\,) + \sqrt{x1 * e - x5}\,]$ *(53. iterace, fitness 23)*

12.1.6 Zhodnocení výsledků

Model s dostatečně dobrou odchylkou se v rámci mnou provedených experimentů bohužel nepodařilo nasyntetizovat. Jedná se o velmi komplexní problém, který sice neobsahuje mnoho proměnných, vztahy mezi nimi ale mohou být velmi složité. Syntéza modelu proto teoreticky vůbec nemusí být možná. Důvodů, které syntézu mohou činit obtížnější je mnoho.

Ze statistické analýzy vyplývá, že data jsou velmi rozptýlená a mohou obsahovat mnoho výkyvů, na které může být pro model těžké reagovat. To může dále podpořit nestandardní distribuce dat a naměřená multikolinearita.

V rámci předzpracování dat byly všechny kroky velmi pečlivě zváženy tak, aby další analýzu ovlivnily co nejméně. Teoreticky by ale na syntézu modelu mohlo mít vliv vyloučení části dat s extrémními hodnotami intenzity a sloučení části dat s nulovou intenzitou.

Dalším možným faktorem může být volba algoritmu. Ve spojení s AP, ale bylo vyzkoušeno několik EA. Generované výrazy pomocí AP měly dle mého názoru vhodnou formu i rozsah a v rámci EA byly vhodně modifikovány. V rámci experimentů bylo také vyzkoušeno různé nastavení hyperparametrů.

Posledním zmíněným ale velmi významným faktorem také mohou být chyby vzniklé při měření. Fyzikální měření, pomocí nějž byla data získána se v praxi využívá v mnohem menším rozsahu. Doba měření je typicky několik hodin a je sesbíráno několik tisíc záznamů měření s konstantním nastavením. Pro účely tohoto experimentu bylo provedeno měření, které trvalo tři dny a bylo naměřeno několik stovek tisíc záznamů, a to za neustálých změn nastavení napětí, aby byla pokryta co největší část prostoru popisujícího danou problematiku. Data proto mohou obsahovat šum způsobenými různými okolnostmi, které měření ovlivňují, ale standartně při měřeních menšího rozsahu nemají tak velký vliv. Tento šum potom může zkreslit již tak potenciálně komplexní vztahy, které mohou mezi proměnnými být, a tak velmi zkomplikovat jejich identifikaci modelem.

12.1.7 Návrh dalších možných směrů výzkumu

Rozsah provedených experimentů byl velmi výrazně omezen zpožděním očekávaného dodání výsledků, protože správná konfigurace přístroje použitého k měření, vyžadovala časově náročné experimentování a ladění. Jak již bylo zmíněno výše jednalo o poměrně nestandardní specifické měření, které není běžně prováděno v takto velkém rozsahu.

V rámci mé práce je představeno několik různých směrů syntézy modely, které byly zatím neúspěšně vyzkoušeny, včetně možných důvodů, které to mohly zapříčinit, vysvětlených v předchozí kapitole 12.2. Analýzou těchto nabitých znalostí lze navrhnout několik směrů, kterými by se mohl výzkum dále vyvíjet.

Nejvíce přínosné by dle mého názoru bylo provedení dalšího měření ve stejném rozsahu. Výsledky by mohly být porovnány s těmi z prvního měření a na základě analýzy rozdílů by mohlo být možné určit, jak moc šumu data obsahují. Překážkou v tomto směru je velká náročnost měření na čas a zdroje.

Za předpokladu, že data zkreslena nejsou by mohla být dále použita pro další experimenty. V rámci symbolické regrese by se výzkum mohl vydat dvěma směry. Pozornost by mohla být přenesena na jiné metody, jako je například genetické programování, anebo se zaměřit na zvýšení robustnosti AP. To by mohlo zahrnovat například rozšíření množiny GFS o další

funkce a operátory, anebo také o komplexnější výrazy extrahované z výrazů nalezených v rámci dosavadní analýzy, například těch představených v kapitole 12.1.5.

V rámci implementace optimalizace AP byly přidány váhy k výběrů ukončovacích terminálů pro velmi dlouhé výrazy a ovlivnění poměru terminálů a operátorů ve složení jedinců EA. Zajímavou variantou AP by mohlo být, pokud by jedinci obsahovaly pouze operátor, funkce a případně i složitější výrazy a při mapování byly ve všech případech ukončovány terminály způsobem, který je popsán v kapitole 12.1.3. Takto by byla umožněna větší kontrola nad výskytem jednotlivých terminálů v rámci výrazů a mohl by být více adaptován na charakter jednotlivých částí výrazu. To by umožnilo větší začlenění doménové znalosti do syntézy. Samozřejmě by tak ale byl významně omezen prohledávaný prostor řešení.

V rámci AP by konvergenci mohla také pomoct implementace sofistikovanějšího ladění konstanty c. Do GFS by mohlo být navíc začleněno větší množství různých proměnných pro konstanty, které by pak mohly být laděny jednotlivě. Tento postup by ale mohl výrazně zvýšit časovou náročnost celého výpočtu.

Dále je obecně možné další experimentování s hodnotami hyperparametrů a různých strategií v rámci EA, stejně tak jako použití různých jejich optimalizovaných variant. Na základě mé dosavadní práce na této úloze to ale považuji za spíše doplňující možnost k výše zmíněným směrům.

Za vyzkoušení by jistě stálo také využití jiných technik strojového učení jako je například metoda podpůrných vektorů. Zajímavé by jistě bylo také experimentování s hybridními algoritmy EA a dalšími heuristickými přístupy.

12.2 Neuronová síť

Druhá část praktické části této práce je zaměřena na jednu z nejvýznamnějších oblastí moderního strojového učení – trénink neuronové sítě. Konkrétně se jedná o typ dopředné neuronové sítě (Feed-forward Neural Network). Tato část byla do praktické části zařazena s cílem rozšířit okruh technik využitých pro syntézu modelu a zvýšit tak pravděpodobnost jeho nalezení. To odráží snahu zkoumat různé možnosti a přístupy v rámci tohoto výzkumu.

Pro realizaci byl zvolen programovací jazyk Python. Tento jazyk je široce uznáván v oblasti strojového učení pro svou čitelnost, flexibilitu a bohatou nabídku specializovaných knihoven. Jednou z těchto knihoven je Keras. Jedná so o knihovnu určenou pro hluboké učení, která je postavena na backendu TensorFlow.

Prvním krokem bylo vytvoření modelu neuronové sítě, který je definován jako posloupnost vrstev. Model byl sestaven ze dvou vrstev. První vrstva obsahuje 16 neuronů, druhá 24 a třetí 1. První dvě vrstvy používají ReLU (Rectified Linear Unit) jako aktivační funkci, zatímco poslední vrstva používá lineární aktivační funkci, což je typické pro regresní problémy.

Dále byla definována ztrátová funkce – střední kvadratická chyba a optimalizátor – Adam. Pro trénink byl nastaven počet epoch na 300 a velikost batch na 10. Data byla rozdělena na jednu třetinu testovacích dat a dvě třetiny dat pro trénink. Z tréninkového datasetu bylo dále vybráno dvacet procent dat pro validaci během učení. Navíc byla specifikovaná funkce EarlyStopping s parametrem čekání 30 epoch závislém na odchylce modelu u validačních dat.

Toto je výsledné nastavení, kterým byl nalezen nejúspěšnější model s odchylkou o velikosti 8. V rámci experimentování s tréninkem bylo vyzkoušeno rozšíření modelu NN o další vrstvy, menší i větší počet neuronů ve vrstvách, i zařazení vrstev pro Dropout a BatchNormalization. Nejlepší výsledky ale nakonec byly poskytnuty jednoduchým modelem popsaným výše.

Výsledky tréninku modelu velmi pozitivně ovlivnilo odstranění řádků s extrémními hodnotami intenzity a aplikace shlukování, blíže popsané v kapitole 10.

Odchylka o velikosti 8 je stále poměrně velká a v rámci dalšího výzkumu by bylo vhodné zařadit další ladění parametrů, aby se dostala na co nejnižší hodnotu. Za účelem naplnění zadání práce byl ale i přes velkou odchylku nalezený model použit pro srovnání s reálným systémem.

13 OPTIMALIZACE PARAMETRŮ

Pro optimalizaci parametrů nastavení měření byly zvoleny tři evoluční výpočetní techniky – DE, PSO a SOMA. Zadáním optimalizace je návrh nastavení parametrů x_1-x_5, při zadaném parametru m tak, aby naměřená intenzita byla co největší. Pro reprezentaci sytému byl využit model získaný pomocí tréninku neuronové sítě.

Nastavení parametrů EA bylo zvoleno v porovnání s nastavením použitým v rámci analytického programování méně robustní, vzhledem k menší náročnosti problému. Pro velikost populace byla zvolena hodnota 20 k dimenzi o velikosti 5 odpovídající počtu nastavovaných proměnných. Rozmezí generovaných hodnot řešení bylo nastaveno na 0 až 100, aby odpovídalo škálování dat. Počet iterací byl nastaven na hodnotu 1000. Při bližším zkoumáním bylo možné pozorovat, že všechny tři algoritmy konvergovaly k řešení poměrně rychle v rámci několika stovek iterací a pak už se řešení výrazně neměnilo. Pro další experimenty by proto tato hodnota mohla být teoreticky snížena. Ostatní parametry EA byly nastaveny, jak je popsáno v předchozí kapitole 12.1.

Pro porovnání výsledků byl využit dataset z historického měření na reálném systémů. Tento soubor obsahuje 2000 záznamů. Data byla naškálována stejně jako data použitá po trénování NN. Porovnáno bylo nalezené množství nastavení parametrů, které předpovídá naměření větší intenzity než v rámci původního měření. Zároveň je porovnána odchylka mezi navrženým nastavením a nastavením původním a výsledky jsou interpretovány.

Pro ohodnocení jedinců v rámci EA je využívána hodnota intenzity předpovězená syntetizovaným modelem na základě aktuálního nalezeného řešení nastavení parametrů a zadané hodnoty m. Jako fitness je poté vrácena záporná hodnota intenzity, protože oproti všem dosavadním úlohám se v tomto případě jedná o úlohu, kde je hledáno maximální řešení. Vrácením záporné hodnoty intenzity je možné tohoto docílit bez toho, aby bylo nutné provádět úpravy v rámci jednotlivých EA. Při vyhodnocení výsledků je pak využita absolutní hodnota nalezeného řešení pro správné porovnání.

Výsledky experimentu jsou hodnoceny pomocí dvou parametrů. Prvním z nich je hodnota intenzity, jejíž naměření je předpovězeno modelem na základě nalezeného řešení. Rozhodujícím faktorem pak je, zda se jedná o hodnotu větší nebo menší než v původním měření. Další metrikou je průměr odchylek navrženého nastavení parametrů EA oproti jejich nastavení v původním měření, tedy jak moc se daná nastavení liší.

	% > intenzita	odchylka
DE	95	31.509
PSO	97	185.846
SOMA	96	81.414

Obrázek 12 Přehled výsledků optimalizace nastavení parametrů pomocí EA. Hodnoty v prvním sloupci udávají v procentech, kolik z navržených nastavení předpovídá nalezení větší hodnoty intenzity než v rámci původního měření. Ve druhém sloupci lze vidět průměrnou naměřenou odchylku navržených hodnot nastavení parametrů vůči parametrům nastavených v původním měření.

Výsledky toho experimentu je obecně velmi obtížné zhodnotit. Hlavním důvodem je velká naměřená odchylka modelu použitého pro reprezentaci systému. Nastavení parametrů x_1-x_5 a m je na reálném systému poměrně citlivé i na malé změny. Model s odchylkou o velikosti 8 proto může při rozmezí hodnot 0 až 100 poskytovat velmi nepřesné výsledky. Pro další využití je důležité nalézt takový model jehož odchylka bude v ideálním případě co nejblíže hodnotě 1. Poté by bylo vhodné experiment zopakovat a ověřit jeho výstupy využitím navržených parametrů pro reálné měření. To bohužel v rámci tohoto experimentu nebylo z technických důvodů možné a byla použita pouze data z historického měření pro srovnání.

I přes toto omezení způsobené nepřesností modelu se zde pokusím interpretovat výsledky experimentu. Je ale nutné znovu zdůraznit, že pro potvrzení výsledků by bylo vhodné provést další ověření, jak je popsáno výše. Z tabulky na Obrázku 12 vyplývá, že všechny tři použité EA předpovídají ve většině případů naměření větší hodnoty intenzity než původně naměřené. To by se samo o sobě dalo považovat za velmi dobrý výsledek. Na druhou stranu, ale v sloupci dva této tabulky můžeme pozorovat poměrně velké odchylky od původně nastavených hodnot parametrů. Tyto velké rozdíly mohou ukazovat na nalezení velkého množství řešení, které na reálném systém bude vykazovat jiné výsledky, a to s ohledem na velkou odchylku použitého modelu. Pro správnou interpretaci by bylo nutné otestovat nastavení při reálném měření. Vzhledem k popsaným omezením si netroufám výsledky více hodnotit, ani hodnotit výkon jednotlivých EA.

Hlavním přínosem této kapitoly je implementace a popis nastavení algoritmů pro optimalizaci a způsob hodnocení výsledků experimentu, který může být v budoucnu snadno zreplikován se spolehlivějším modelem. Zahrnutá interpretace výsledků experimentu je poměrně strohá vzhledem k omezením použitého modelu, které mohou výsledky značně zkreslovat.

ZÁVĚR

V rámci této práce byla provedena detailní a multidisciplinární analýza využití analytického programování, evolučních algoritmů a dalších metod umělé inteligence a strojového učení v kontextu optimalizace a modelování v průmyslové praxi. Práce byla založena na pečlivé analýze relevantních zdrojů a jejím primárním cílem bylo provést odbornou rešerši na dané téma v teoretické části a aplikovat popsané metody v části praktické, včetně zhodnocení provedených experimentů.

V rámci teoretické části byly podrobně prozkoumány různé aspekty evolučních výpočetních technik, analytického programování, neuronových sítí a dalších metod využitých při zpracování dat. Tyto metody byly pak aplikovány v praktické části práce, která se skládala z popisu dat získaných z fyzikálního měření a analýzy možností syntézy modelu popisující tato data. Výsledky získané syntézou modelu byly poté využity pro optimalizaci parametrů pomocí evolučních výpočetních technik a byly porovnány s reálnými daty a zhodnoceny.

Využití evolučních algoritmů a analytického programování v kombinaci s dalšími metodami strojového učení může výrazně přispět k optimalizaci v průmyslové praxi. Přestože nebylo v rámci této práce dosaženo plného úspěchu v syntéze modelu, byly identifikovány klíčové faktory, které mohou podobné úlohy ovlivnit, a byly také navrženy možné směry pro další výzkum.

Tato práce představuje významný příspěvek k teoretickému a praktickému pochopení využití evolučních algoritmů, analytického programování a dalších metod umělé inteligence a strojového učení v průmyslové praxi a její výsledky mohou být využity pro další výzkum.

SEZNAM POUŽITÉ LITERATURY

[1] A. Törn a A. Žilinskas, *Global Optimization*. in Lecture Notes in Computer Science, no. 350. Berlin, Heidelberg: Springer-Verlag Springer e-books, 1989.

[2] W. Forst a D. Hoffmann, *Optimization—Theory and Practice*. in Springer Undergraduate Texts in Mathematics and Technology. New York, NY: Springer New York, 2010. doi: 10.1007/978-0-387-78977-4.

[3] J. S. Arora, *Introduction to optimum design*, 2nd ed. Amsterdam: Elsevier/Academic Press, 2004.

[4] S. E. Cox, R. T. Haftka, C. A. Baker, B. Grossman, W. H. Mason, a L. T. Watson, „A comparison of global optimization methods for the design of a high-speed civil transport", *J. Glob. Optim.*, roč. 21, č. 4, s. 415–432, 2001, doi: 10.1023/A:1012782825166.

[5] G. Venter, „Review of Optimization Techniques", in *Encyclopedia of Aerospace Engineering*, 1. vyd., R. Blockley a W. Shyy, Ed., Wiley, 2010. doi: 10.1002/9780470686652.eae495.

[6] A. Neumaier, „Complete search in continuous global optimization and constraint satisfaction", *Acta Numer.*, roč. 13, s. 271–369, kvě. 2004, doi: 10.1017/S0962492904000194.

[7] D. R. Jones, C. D. Perttunen, a B. E. Stuckman, „Lipschitzian optimization without the Lipschitz constant", *J. Optim. Theory Appl.*, roč. 79, č. 1, s. 157–181, říj. 1993, doi: 10.1007/BF00941892.

[8] Axel Thevenot, „Python benchmark test optimization function single objective", *GitHub*. Viděno: 28. duben 2024. [Online]. Dostupné z: https://github.com/AxelThevenot/Python_Benchmark_Test_Optimization_Function_Single_Objective?tab=readme-ov-file

[9] J. Flegr, *Evoluční biologie*, Třetí opravené a Rozšířené vydání. Praha: Academia, 2018.

[10] T. Bartz-Beielstein, J. Branke, J. Mehnen, a O. Mersmann, „Evolutionary Algorithms", *WIREs Data Min. Knowl. Discov.*, roč. 4, č. 3, s. 178–195, kvě. 2014, doi: 10.1002/widm.1124.

[11] P. A. Vikhar, „Evolutionary algorithms: A critical review and its future prospects", in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, Jalgaon, India: IEEE, pro. 2016, s. 261–265. doi: 10.1109/ICGTSPICC.2016.7955308.

[12] J. H. Holland, „Genetic Algorithms", *Sci. Am.*, roč. 267, č. 1, s. 66–73, 1992.

[13] D. E. Goldberg a K. Deb, „A Comparative Analysis of Selection Schemes Used in Genetic Algorithms", in *Foundations of Genetic Algorithms*, roč. 1, Elsevier, 1991, s. 69–93. doi: 10.1016/B978-0-08-050684-5.50008-2.

[14] H. Du, Z. Wang, W. Zhan, a J. Guo, „Elitism and Distance Strategy for Selection of Evolutionary Algorithms", *IEEE Access*, roč. 6, s. 44531–44541, 2018, doi: 10.1109/ACCESS.2018.2861760.

[15] J. H. Holland a J. R. Koza, „Genetic programming", *Sci Am*, roč. 267, s. 66–72, 1992.

[16] R. Storn a K. Price, „Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", *J. Glob. Optim.*, roč. 11, č. 4, s. 341–359, 1997, doi: 10.1023/A:1008202821328.

[17] I. Rechenberg, „Evolutions Strategien", in *Simulationsmethoden in der Medizin und Biologie: Workshop, Hannover, 29. Sept.–1. Okt. 1977*, Springer, 1978, s. 83–114.

[18] H. Schwefel, „A Survey of Evolution Strategies", in *Proceedings of the Fourth International Conference on Genetic Algorithms, RK Belew and LB Booker, eds., Morgan Kaufmann, San Mateo, CA*, 1991, s. 2–9.

[19] H. Schwefel a G. Rudolph, „Contemporary evolution strategies", in *Advances in Artificial Life–Proc. Third European Conf. Artificial Life (ECAL'95)*, 1995, s. 893–907.
[20] H.-G. Beyer a H.-P. Schwefel, „Evolution strategies – a comprehensive introduction", *Nat. Comput.*, roč. 1, s. 3–52, 2002.
[21] D. B. Fogel a L. J. Fogel, „An introduction to evolutionary programming", in *Artificial Evolution*, roč. 1063, J.-M. Alliot, E. Lutton, E. Ronald, M. Schoenauer, a D. Snyers, Ed., in Lecture Notes in Computer Science, vol. 1063. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, s. 21–33. doi: 10.1007/3-540-61108-8_28.
[22] M. A. Potter a K. A. Jong, „A cooperative coevolutionary approach to function optimization", in *Parallel Problem Solving from Nature — PPSN III*, roč. 866, Y. Davidor, H.-P. Schwefel, a R. Männer, Ed., in Lecture Notes in Computer Science, vol. 866. , Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, s. 249–257. doi: 10.1007/3-540-58484-6_269.
[23] Y. Wang, Z. Cai, a Q. Zhang, „Differential evolution with composite trial vector generation strategies and control parameters", *IEEE Trans. Evol. Comput.*, roč. 15, č. 1, s. 55–66, 2011.
[24] S. Baluja, *Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning*. School of Computer Science, Carnegie Mellon University Pittsburgh, PA, 1994.
[25] J. Kennedy a R. Eberhart, „Particle swarm optimization", in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, WA, Australia: IEEE, 1995, s. 1942–1948. doi: 10.1109/ICNN.1995.488968.
[26] I. Zelinka, „SOMA—self-organizing migrating algorithm", *Self-Organ. Migrating Algorithm Methodol. Implement.*, s. 3–49, 2016.
[27] M. F. Tasgetiren, M. Sevkli, Y.-C. Liang, a M. M. Yenisey, „A particle swarm optimization and differential evolution algorithms for job shop scheduling problem", *Int. J. Oper. Res.*, roč. 3, č. 2, s. 120–135, 2006.
[28] P. Preux a E.-G. Talbi, „Towards hybrid evolutionary algorithms", *Int. Trans. Oper. Res.*, roč. 6, č. 6, s. 557–570, 1999.
[29] M. A. M. Shaheen, H. M. Hasanien, a A. Alkuhayli, „A novel hybrid GWO-PSO optimization technique for optimal reactive power dispatch problem solution", *Ain Shams Eng. J.*, roč. 12, č. 1, s. 621–630, bře. 2021, doi: 10.1016/j.asej.2020.07.011.
[30] E. Mirsadeghi a S. Khodayifar, „Hybridizing particle swarm optimization with simulated annealing and differential evolution", *Clust. Comput.*, roč. 24, č. 2, s. 1135–1163, čer. 2021, doi: 10.1007/s10586-020-03179-y.
[31] J. D. Schaffer, D. Whitley, a L. J. Eshelman, „Combinations of genetic algorithms and neural networks: a survey of the state of the art", in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, 1992, s. 1–37. doi: 10.1109/COGANN.1992.273950.
[32] F. Valdez, P. Melin, a O. Castillo, „Evolutionary method combining particle swarm optimization and genetic algorithms using fuzzy logic for decision making", in *2009 IEEE International Conference on Fuzzy Systems*, Jeju Island, South Korea: IEEE, srp. 2009, s. 2114–2119. doi: 10.1109/FUZZY.2009.5277165.
[33] C.-J. Tu, L.-Y. Chuang, J.-Y. Chang, a C.-H. Yang, „Feature Selection using PSO-SVM.", *IAENG Int. J. Comput. Sci.*, roč. 33, č. 1, 2007.
[34] W. Hart, N. Krasnogor, D. Pelta, a J. Smith, „Protein structure prediction with evolutionary algorithms", Sandia National Lab.(SNL-NM), Albuquerque, NM (United States); Sandia …, 1999.
[35] E. Masehian a D. Sedighizadeh, „Multi-objective PSO-and NPSO-based algorithms for robot path planning", *Adv. Electr. Comput. Eng.*, roč. 10, č. 4, s. 69–76, 2010.

[36] J.-Y. Wang, J.-B. Wang, X. Song, M. Chen, a J. Zhang, „Network planning for distributed antenna-based high-speed railway mobile communications", *Trans. Emerg. Telecommun. Technol.*, roč. 25, č. 7, s. 707–722, 2014.

[37] N. Jin a Y. Rahmat-Samii, „Particle Swarm Optimization for Antenna Designs in Engineering Electromagnetics", *J. Artif. Evol. Appl.*, roč. 2008, led. 2008, doi: 10.1155/2008/728929.

[38] M. Andrews a R. Prager, „16 Genetic Programming for the Acquisition of Double Auction Market Strategies", *Adv. Genet. Program.*, roč. 1, s. 355, 1994.

[39] M. R. Hassan, B. Nath, a M. Kirley, „A fusion model of HMM, ANN and GA for stock market forecasting", *Expert Syst. Appl.*, roč. 33, č. 1, s. 171–180, 2007.

[40] N. Noman a H. Iba, „Differential evolution for economic load dispatch problems", *Electr. Power Syst. Res.*, roč. 78, č. 8, s. 1322–1331, 2008.

[41] P. P. Gujar a P. Desai, „A survey of record deduplication techniques", *IJLTET*, roč. 2, č. 4, s. 246–250, 2013.

[42] M. L. Wong, W. Lam, K. S. Leung, P. S. Ngan, a J. C. Cheng, „Discovering knowledge from medical databases using evolutionory algorithms", *IEEE Eng. Med. Biol. Mag.*, roč. 19, č. 4, s. 45–55, 2000.

[43] B. M. Baker a M. Ayechew, „A genetic algorithm for the vehicle routing problem", *Comput. Oper. Res.*, roč. 30, č. 5, s. 787–800, 2003.

[44] Z. Tajbakhsh, P. Fattahi, a J. Behnamian, „Multi-objective assembly permutation flow shop scheduling problem: a mathematical model and a meta-heuristic algorithm", *J. Oper. Res. Soc.*, roč. 65, s. 1580–1592, 2014.

[45] C. W. Reynolds, „Flocks, herds and schools: A distributed behavioral model", *ACM SIGGRAPH Comput. Graph.*, roč. 21, č. 4, s. 25–34, srp. 1987, doi: 10.1145/37402.37406.

[46] F. Heppner a U. Grenander, „A stochastic nonlinear model for coordinated bird flocks", 1990. [Online]. Dostupné z: https://api.semanticscholar.org/CorpusID:53830602

[47] Y. Shi a R. Eberhart, „A modified particle swarm optimizer", in *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, Anchorage, AK, USA: IEEE, 1998, s. 69–73. doi: 10.1109/ICEC.1998.699146.

[48] A. P. Engelbrecht, „Heterogeneous Particle Swarm Optimization", in *Swarm Intelligence*, roč. 6234, M. Dorigo, M. Birattari, G. A. Di Caro, R. Doursat, A. P. Engelbrecht, D. Floreano, L. M. Gambardella, R. Groß, E. Şahin, H. Sayama, a T. Stützle, Ed., in Lecture Notes in Computer Science, vol. 6234. , Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, s. 191–202. doi: 10.1007/978-3-642-15461-4_17.

[49] P. Liu a J. Liu, „Multi-leader PSO (MLPSO): A new PSO variant for solving global optimization problems", *Appl. Soft Comput.*, roč. 61, s. 256–263, pro. 2017, doi: 10.1016/j.asoc.2017.08.022.

[50] Y. Cao, H. Zhang, W. Li, M. Zhou, Y. Zhang, a W. A. Chaovalitwongse, „Comprehensive Learning Particle Swarm Optimization Algorithm With Local Search for Multimodal Functions", *IEEE Trans. Evol. Comput.*, roč. 23, č. 4, s. 718–731, srp. 2019, doi: 10.1109/TEVC.2018.2885075.

[51] N. Lynn a P. N. Suganthan, „Heterogeneous comprehensive learning particle swarm optimization with enhanced exploration and exploitation", *Swarm Evol. Comput.*, roč. 24, s. 11–24, říj. 2015, doi: 10.1016/j.swevo.2015.05.002.

[52] W. Fang, J. Sun, Y. Ding, X. Wu, a W. Xu, „A Review of Quantum-behaved Particle Swarm Optimization", *IETE Tech. Rev.*, roč. 27, č. 4, s. 336, 2010, doi: 10.4103/0256-4602.64601.

[53] J. Brest, S. Greiner, B. Boskovic, M. Mernik, a V. Zumer, „Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems", *IEEE Trans. Evol. Comput.*, roč. 10, č. 6, s. 646–657, pro. 2006, doi: 10.1109/TEVC.2006.872133.

[54] A. K. Qin, V. L. Huang, a P. N. Suganthan, „Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization", *IEEE Trans. Evol. Comput.*, roč. 13, č. 2, s. 398–417, dub. 2009, doi: 10.1109/TEVC.2008.927706.

[55] Jingqiao Zhang a A. C. Sanderson, „JADE: Adaptive Differential Evolution With Optional External Archive", *IEEE Trans. Evol. Comput.*, roč. 13, č. 5, s. 945–958, říj. 2009, doi: 10.1109/TEVC.2009.2014613.

[56] R. Tanabe a A. Fukunaga, „Success-history based parameter adaptation for Differential Evolution", in *2013 IEEE Congress on Evolutionary Computation*, Cancun, Mexico: IEEE, čer. 2013, s. 71–78. doi: 10.1109/CEC.2013.6557555.

[57] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, a M. F. Tasgetiren, „Differential evolution algorithm with ensemble of parameters and mutation strategies", *Appl. Soft Comput.*, roč. 11, č. 2, s. 1679–1696, bře. 2011, doi: 10.1016/j.asoc.2010.04.024.

[58] B. Y. Qu, P. N. Suganthan, a J. J. Liang, „Differential Evolution With Neighborhood Mutation for Multimodal Optimization", *IEEE Trans. Evol. Comput.*, roč. 16, č. 5, s. 601–614, říj. 2012, doi: 10.1109/TEVC.2011.2161873.

[59] K. Deep a Dipti, „A self-organizing migrating genetic algorithm for constrained optimization", *Appl. Math. Comput.*, roč. 198, č. 1, s. 237–250, dub. 2008, doi: 10.1016/j.amc.2007.08.032.

[60] K. Deep a Dipti, „A new hybrid Self Organizing Migrating Genetic Algorithm for function optimization", in *2007 IEEE Congress on Evolutionary Computation*, Singapore: IEEE, zář. 2007, s. 2796–2803. doi: 10.1109/CEC.2007.4424825.

[61] P. Kadlec a Z. Raida, „A Novel Multi-Objective Self-Organizing Migrating Algorithm.", *Radioengineering*, roč. 20, č. 4, 2011.

[62] D. Davendra, I. Zelinka, M. Bialic-Davendra, R. Senkerik, a R. Jasek, „Discrete Self-Organising Migrating Algorithm for flow-shop scheduling with no-wait makespan", *Math. Comput. Model.*, roč. 57, č. 1–2, s. 100–110, led. 2013, doi: 10.1016/j.mcm.2011.05.029.

[63] L. Skanderova, T. Fabian, a I. Zelinka, „Self-adapting self-organizing migrating algorithm", *Swarm Evol. Comput.*, roč. 51, s. 100593, pro. 2019, doi: 10.1016/j.swevo.2019.100593.

[64] Y. Wang, N. Wagner, a J. M. Rondinelli, „Symbolic regression in materials science", *MRS Commun.*, roč. 9, č. 3, s. 793–805, zář. 2019, doi: 10.1557/mrc.2019.85.

[65] S. Sette a L. Boullart, „Genetic programming: principles and applications", *Eng. Appl. Artif. Intell.*, roč. 14, č. 6, s. 727–736, pro. 2001, doi: 10.1016/S0952-1976(02)00013-1.

[66] M. O'Neill a C. Ryan, „Grammatical evolution", *IEEE Trans. Evol. Comput.*, roč. 5, č. 4, s. 349–358, srp. 2001, doi: 10.1109/4235.942529.

[67] I. Zelinka, „Analytic programming by means of new evolutionary algorithms", in *Proceedings of 1st International Conference on New Trends in Physics*, 2001, s. 210–214.

[68] I. Zelinka, D. Davendra, R. Senkerik, R. Jasek, a Z. Oplatkov, „Analytical Programming - a Novel Approach for Evolutionary Synthesis of Symbolic Structures", in *Evolutionary Algorithms*, E. Kita, Ed., InTech, 2011. doi: 10.5772/16166.

[69] G. E. Hinton, S. Osindero, a Y.-W. Teh, „A fast learning algorithm for deep belief nets", *Neural Comput.*, roč. 18, č. 7, s. 1527–1554, 2006.

[70] M. A. Nielsen, *Neural networks and deep learning*, roč. 25. Determination press San Francisco, CA, USA, 2015.

[71] C. R. Harris *et al.*, „Array programming with NumPy", *Nature*, roč. 585, č. 7825, s. 357–362, zář. 2020, doi: 10.1038/s41586-020-2649-2.
[72] F. Chollet a others, „Keras". 2015. [Online]. Dostupné z: https://keras.io
[73] A. Paszke *et al.*, „PyTorch: An Imperative Style, High-Performance Deep Learning Library", 2019, doi: 10.48550/ARXIV.1912.01703.
[74] J. M. Alvarez a M. Salzmann, „Learning the number of neurons in deep networks", *Adv. Neural Inf. Process. Syst.*, roč. 29, 2016.
[75] L. Liu, B. F. Jones, B. Uzzi, a D. Wang, „Data, measurement and empirical methods in the science of science", *Nat. Hum. Behav.*, roč. 7, č. 7, s. 1046–1058, čer. 2023, doi: 10.1038/s41562-023-01562-4.
[76] A. K. Jain, „Data clustering: 50 years beyond K-means", *Pattern Recognit. Lett.*, roč. 31, č. 8, s. 651–666, čer. 2010, doi: 10.1016/j.patrec.2009.09.011.
[77] J. MacQueen a others, „Some methods for classification and analysis of multivariate observations", in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Oakland, CA, USA, 1967, s. 281–297.
[78] D. Sculley, „Web-scale k-means clustering", in *Proceedings of the 19th international conference on World wide web*, 2010, s. 1177–1178.
[79] M. Ester, H.-P. Kriegel, J. Sander, a X. Xu, „A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", in *Knowledge Discovery and Data Mining*, 1996. [Online]. Dostupné z: https://api.semanticscholar.org/CorpusID:355163
[80] W. McKinney, „Data Structures for Statistical Computing in Python", in *Proceedings of the 9th Python in Science Conference*, S. van der Walt a J. Millman, Ed., 2010, s. 56–61. doi: 10.25080/Majora-92bf1922-00a.
[81] F. Pedregosa *et al.*, „Scikit-learn: Machine Learning in Python", *J. Mach. Learn. Res.*, roč. 12, s. 2825–2830, 2011.
[82] J. D. Hunter, „Matplotlib: A 2D graphics environment", *Comput. Sci. Eng.*, roč. 9, č. 3, s. 90–95, 2007, doi: 10.1109/MCSE.2007.55.
[83] M. L. Waskom, „seaborn: statistical data visualization", *J. Open Source Softw.*, roč. 6, č. 60, s. 3021, 2021, doi: 10.21105/joss.03021.
[84] S. Seabold a J. Perktold, „statsmodels: Econometric and statistical modeling with python", in *9th Python in Science Conference*, 2010.
[85] P. Virtanen *et al.*, „SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python", *Nat. Methods*, roč. 17, s. 261–272, 2020, doi: 10.1038/s41592-019-0686-2.
[86] A. Meurer *et al.*, „SymPy: symbolic computing in Python", *PeerJ Comput. Sci.*, roč. 3, s. e103, led. 2017, doi: 10.7717/peerj-cs.103.
[87] G. Van Rossum, *The Python Library Reference, release 3.8.2*. Python Software Foundation, 2020.
[88] A. Viktorin, R. Senkerik, M. Pluhacek, T. Kadavy, a A. Zamuda, „Distance based parameter adaptation for Success-History based Differential Evolution", *Swarm Evol. Comput.*, roč. 50, s. 100462, lis. 2019, doi: 10.1016/j.swevo.2018.10.013.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AP	analytické programování
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DE	diferenciální evoluce
EA	evoluční algoritmy
GFS	general functional set
GA	genetické algoritmy
GP	genetické programování
NN	neuronové sítě
PSO	Particle Swarm Optimization
SOMA	Self-Organizing Migrating Algorithm
UF	účelová funkce

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

CD se zdrojovým kódem a diplomovou prací