Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Optimalizační metody pro vícevrstvý algoritmus genetického programování

Autor: Ing. Jan Merta

Školitel: doc. Ing. Tomáš Brandejský, Dr.

Disertační práce

2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne **27.6.2022**

Jan Merta

PODĚKOVÁNÍ

Chtěl bych touto cestou poděkovat svému školiteli doc. Ing. Tomáši Brandejskému, Dr. Za vedení během celého mého studia, cenné rady a za věcné připomínky při tvorbě této práce. Velký dík patří také kolegovi Ing. Romanu Divišovi, Ph.D nejen za technickou pomoc s experimenty a mé dvorní korektorce Alžbětě za pečlivou analýzu mého psaného projevu. Dále bych rád poděkoval své rodině, blízkým přátelům a dívce z houpačky za podporu a navrácení kreativity při psaní této práce.

ANOTACE

Disertační práce se zaměřuje na vícevrstvý (konkrétně dvouvrstvý) přístup k algoritmu genetického programování a vylepšení trénovacího procesu pomocí moderních optimalizačních metod a přístupů (například sborového učení). Cílem disertační práce bylo navrhnout a implementovat dvouvrstvý algoritmus genetického programování, otestovat jeho chování v rámci symbolické regrese na několika základních testovacích případech a najít vhodná nastavení, která mají potenciál zefektivnit proces učení genetického programování a zvýšit přesnost výsledných modelů. Algoritmus pracuje ve dvou fázích. V první fázi hledá vhodné dílčí modely (stavební bloky) popisující jednotlivé segmenty dat pomocí. Ve druhé fázi hledá výsledný model jako nelineární kombinaci těchto dílčích modelů.

KLÍČOVÁ SLOVA

genetické programování, genetické algoritmy, sborové učení, optimalizace

TITLE

Optimization methods for multilayer genetic programming algorithm

ANNOTATION

This dissertation focuses on a multi-layer (specifically, two-layer) approach to genetic programming algorithm and the improvement of the training process using modern optimization methods and approaches (e.g., ensemble learning). The goal of the dissertation was to design and implement a two-layer genetic programming algorithm, test its behavior in a symbolic regression framework on several basic test cases, and find appropriate settings that have the potential to improve the learning process of genetic programming and increase the accuracy of the resulting models. The algorithm works in two phases. In the first phase, it searches for appropriate submodels (building blocks) describing each segment of the data. In the second phase, it searches for the resulting model as a nonlinear combination of these submodels.

KEYWORDS

genetic programming, genetic algorithms, ensemble learning, optimization

OBSAH

Seznam obrázků

Seznam tabulek

Seznam zkratek

AI	Artificial Intelligence
BSA	Bison Seeker algorithm
EBR	Elite Basis Regression
FFX	Fast Function Extraction
GA	Genetický algoritmus, genetické algoritmy
GP	Genetické programování
GSGP	Geometrické sémantické genetické programování
HGP	Hybridní genetické programování
LCF	Linear Combination of Features
LDA	Lineární diskriminační analýza (Linear Discriminant Analysis)
MBF-GP	Multiple Basis Function Genetic Programming
MEP	Multi Expression Programming
MGGP	Multi-Gene Genetic Programming
MRGP	Multiple Regression Genetic Programming
MSGP	Multistage Genetic Programming
PADO	Parallel Algorithm Discovery and Orchestration
PCA	Principal Component Analysis
PDGP	Parallel Distributed Genetic Programming
PGA	Paralelní genetické algoritmy
PSO	Particle Swarm Optimization
SNGP	Single Node Genetic Programming
SR	Symbolická regrese
SSR	Sekvenční symbolická regrese (Sequential Symbolic Regression)
STROGANOFF	STructured Representation On Genetic Algorithms for NOnlinear Function Fitting
SVM	Support Vector Machines
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem

ÚVOD

S optimalizací se lidé setkávají každý den, když cestují z práce domů, nakupují, putují do kaváren nebo cestují na dovolenou. Mají různé cíle: snaží se ušetřit čas, ujeté kilometry, spotřebu paliva, peníze za jídlo a letenky nebo minimalizovat váhu a rozměry svého zavazadla a zároveň si s sebou vzít všechno potřebné. Firmy se snaží efektivně plánovat výrobu, snižovat náklady a zvyšovat své zisky. Přitom je často omezují fyzikální zákony[1], legislativa nebo konkrétní požadavky zákazníků. Při optimalizaci jde obecně o nalezení nejlepšího (optimálního) řešení daného problému podle zadaného cíle (kritéria).

Tyto problémy se v poslední době dají řešit pomocí moderních optimalizačních metod a umělé inteligence. Některé metody umělé inteligence optimalizační metody využívají a mají je v sobě zabudovány. Metody strojového učení často potřebují pro tvorbu přesných modelů velké množství dat, která často obsahují i velké množství šumu. Vícerozměrné problémy se mohou potýkat také s tím, že i velké množství dat nebude pro trénování stačit (prokletí dimenzionality). Trénování správných modelů pomocí takového objemu dat bývá výpočetně i časově náročné, proto je v poslední době snaha trénovací proces zefektivnit a urychlit. Samotné algoritmy strojového učení a umělé inteligence se tak stávají předmětem optimalizace.

Optimalizovat strojové učení lze různými způsoby. Některé přístupy se snaží o vyčistění zašuměných dat nebo o redukci objemu zpracovávaných dat. Příkladem může být snaha o rozklad větších modelů do souboru menších submodelů pracujících s jinou množinou dat nebo kombinace více algoritmů strojového učení. Jiné přístupy se snaží zefektivnit optimalizační části těchto algoritmů případně nalézt takové architektury modelů, které se budou učit rychleji a s větší přesností. Problémem je i zvolení správného druhu modelu a hyperparametrů algoritmu strojového, které probíhá často ručně pomocí různých heuristik. Proto v poslední době vznikají i přístupy automatického strojového učení. Metody automatického strojové učení využívají principy metaučení a pro modelovaná data hledají nejvhodnější druh modelu (nebo jejich kombinaci) a optimální nastavení hyperparametrů použitých algoritmů strojového učení.

V první části disertační práce je zpracován nezbytný teoretický podklad. V rámci praktické části disertační práce je navržen a implementován dvouvrstvý algoritmus genetického programování, který využívá techniky sborového učení. Tato kombinace skrývá potenciální šanci zefektivnit algoritmus genetického programování a zpřesnit jeho výsledky. Implementovaný dvouvrstvý algoritmus je poté otestován na vybraných testovacích případech.

[1] Ve fyzice ctí pohyb všech těles od bodu A do bodu B princip nejmenší akce, který minimalizuje rozdíl mezi kinetickou a potenciální energií. Tento princip vysvětluje trajektorii vrženého tělesa, lom světla na rozhraní dvou prostředí, pohyb tělesa v gravitačním poli apod. Vesmír se tak díky tomuto matematickému modelu reality jeví jako přirozený optimalizátor.

1 MOTIVACE A CÍLE PRÁCE

Disertační práce se zaměřuje na vícevrstvý (konkrétně dvouvrstvý) přístup k algoritmu genetického programování a vylepšení trénovacího procesu pomocí moderních optimalizačních metod a přístupů. Svět strojového učení poslední dobou ovládají umělé neuronové sítě, které však produkují modely dat typu „black-box“, které se složitě interpretují. Genetické programování dokáže produkovat modely dat typu „white-box“, protože jejich výsledkem může být (ve spojení se symbolickou regresí) algebraický vztah mezi vstupními proměnnými. Takové modely mohou mít například ve fyzice a ekonomice lepší uplatnění než „black-box“ modely produkované neuronovými sítěmi. Například ve fyzice vzniká poslední dobou reálná poptávka po automatizovaném generování fyzikálních zákonů z naměřených dat[2].

Genetické programování je oproti genetickým algoritmům relativně mladou metodou, která se zvyšujícím se výpočetním výkonem a zefektivňováním postupně nalézá praktické využití. Každé zlepšení efektivity genetického programování přibližuje metodu reálnému využití. V inspiraci výkonnostního skoku hlubokých neuronových sítí v posledních letech vznikla idea vícevrstvého přístupu ke genetickému programování, pro začátek dvouvrstvého genetického programování. Samotný dvouvrstvý přístup vyvolává mnoho otázek, například ohledně vhodného nastavení parametrů na obou vrstvách nebo ohledně toho, jak se tyto parametry navzájem ovlivňují. Možnosti nastavení se přidáním další vrstvy a zvětšením počtu parametrů exponenciálně rozšiřují.

Další inspirací byl úspěch sborových metod (ensemble learning) při optimalizaci algoritmů strojového učení. Kombinování více vhodných dílčích modelů do přesnějšího výsledného modelu také odpovídá principům hlubokého učení, které se snaží složitější funkce učit kombinací funkcí jednodušších. Sborové učení inspirovalo i některé další moderní přístupy ke genetickému programování, které jsou popsány v následujících kapitolách.

Cílem disertační práce bylo navrhnout a implementovat dvouvrstvý algoritmus genetického programování, otestovat jeho chování v rámci symbolické regrese na několika základních testovacích případech a najít vhodná nastavení, která mají potenciál zefektivnit proces učení genetického programování a zvýšit přesnost výsledných modelů.

Disertační práce si neklade za cíl obecně popsat dynamiku dvouvrstvého (případně vícevrstvého) genetického programování do nejmenších detailů a nehledá obecně platné vztahy mezi nastavením jednotlivých parametrů. V rámci rozsahu práce by šlo o velmi složitý úkol.

[2] Například: BRUNTON, Steven L., Joshua L. PROCTOR a J. Nathan KUTZ. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences [online]. 2016, 113(15), 3932-3937 [cit. 2022-06-25]. ISSN 0027-8424. Dostupné z: doi:10.1073/pnas.1517384113

2 STROJOVÉ UČENÍ A UMĚLÁ INTELIGENCE

Strojové učení je podoborem umělé inteligence a aplikované statistiky. Zabývá se metodami adaptivních, učících, sebevylepšujících se algoritmů. Metody strojového učení hledají v datech užitečné vzory a pomáhají odhalovat složité funkce, které data popisují [1, 2].

1.1 Výběr vhodného algoritmu a modelu

Samuel Karlin tvrdí, že *„The purpose of models is not to fit the data but to sharpen the questions.“* [3]. Pro úspěch algoritmu strojového učení je nutné vybrat správnou reprezentaci řešeného problému. Výběr reprezentace problému zahrnuje zejména výběr sady vstupních proměnných (features) a případně transformaci jejich hodnot. K tvorbě modelu by měly být vybrány pouze vstupní proměnné, které mají vliv na predikci, na rozdíl od vstupních proměnných, které tvorbu přesného modelu ztěžují. Výběrem správné reprezentace se může hledání správného modelu dramaticky zjednodušit.

Cílem strojového učení je vytvářet tzv. důvěryhodné modely [4]. Pro jednodušší tvorbu důvěryhodných modelů je nutné dodržovat několik pravidel:

- Dat pro trénování modelu by mělo být dostatečně velké množství, aby v nich byl dostatek příkladů k natrénování modelu a zároveň dost příkladů k otestování generalizace (validaci) modelu.
- Používaná data by měla dostatečně popisovat modelovaný systém a jeho dynamiku.
- Data by měla být vyvážená, tak aby nepopisovala některé situace mnohem podrobněji než jiné.

Výběr vhodného algoritmu strojového učení a modelu by měl respektovat charakteristiku modelovaného systému a trénovacích dat (linearita systému, nezávislost vstupních proměnných, významnost proměnných, šum apod.). Pro daný problém je nutné zvolit model s přiměřeně velkou kapacitou. Výběr výsledného modelu by se také měl řídit principem Occamovy břitvy, který v případě, že matematické modely mají srovnatelné výsledky, radí vybrat ten méně složitý.

1.2 Principy hlubokého učení

K výběru správné reprezentace je často potřeba expertní znalost. U některých problémů je výběr správné reprezentace velmi složitou úlohou. Algoritmy hlubokého učení tuto situaci řeší tím, že zpracovávají surová vstupní data a správnou reprezentaci se snaží naučit samy. Správně naučené reprezentace mají zpravidla větší úspěch než ručně vybírané a také umožňují algoritmu se lépe přizpůsobovat změnám a novým úlohám [1].

Hlavní ideou hlubokého učení je kompoziční přístup k tvorbě reprezentací, kde se složitější abstraktnější reprezentace vyšší úrovně řešeného problému tvoří z jeho jednodušších reprezentací nižší úrovně. Složitější koncepty jsou budovány z jednodušších konceptů nižší úrovně. Matematicky lze tento princip zajistit pomocí na sebe navazujících funkcí. Každá

funkce má své vstupy a výstup. Funkce první úrovně mají za vstup vstupní proměnné. Funkce vyšších úrovní se tvoří skládáním jednodušších funkcí tak, že výstup funkcí z nižších úrovní je vstupem do funkcí vyšší úrovně. Každá funkce vždy vyjadřuje novou reprezentaci původního surového vstupu. Struktura dat lze prozkoumat pomocí autoenkodérů, které mohou sloužit například k tvorbě různých filtrů a ke kompresi dat [1].

Hluboké učení je typicky implementováno pomocí hlubokých neuronových sítí. Základní princip je však obecný a neomezuje se pouze na neuronové sítě. Podobnou ideu splňují i modely vytvořené pomocí metod sborového učení (ensemble learning) popsané v kapitole 4.2.

3 GENETICKÉ PROGRAMOVÁNÍ

Genetické programování [5] pomocí evoluce vyvíjí počítačové programy [6] jako rozšíření genetických algoritmů. Genetický algoritmus (GA) představil John Holland v roce 1975 [7]. Jde o stochastickou optimalizační metodu založenou na strategii populace řešení. GA jako optimalizační metodu je vhodné použít pro složité problémy [7, 8, 9], u kterých je cílem nalézt alespoň přijatelné řešení v přijatelném čase (ne nutně globální optimum). Genetické algoritmy jsou postaveny na mechanismu přírodního výběru a genetiky. GA nefunguje jako pouhá náhodná procházka, náhodný výběr používá sofistikovaně pouze jako nástroj pro efektivní a systematický průchod neznámých částí stavového prostoru [10].

Základní cyklus GA obsahuje následující kroky:

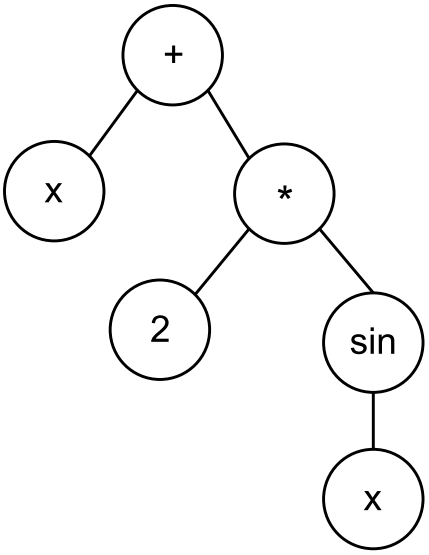
1. vygenerování náhodné počáteční populace jedinců,
2. ohodnocení jednotlivých jedinců pomocí fitness funkce (závislá na prostředí a genech jedince),
3. výběr rodičovských jedinců pro tvorbu potomků v závislosti na jejich hodnotě fitness funkce,
4. náhodná mutace některých jedinců,
5. případná eliminace méně zdatných jedinců,
6. kontrola podmínky ukončení, v případě splnění předání výsledků,
7. přechod na krok 2.

Genetické programování umožňuje vysokoúrovňové automatické řešení zadaných problémů bez explicitního zápisu programu programátorem [6], kdy stačí definovat očekávané chování a základní programové prvky, ze kterých by se výsledné řešení mělo skládat. Očekávané chování výsledného programu je definováno pomocí fitness funkce. Algoritmus genetického programování má k dispozici základní programové prvky, ze kterých postupně sestavuje složitější programy, jejichž chování se blíží chování definovanému ve fitness funkci. Prohledávaný prostor programů je enormně velký (ne-li nekonečný) a složitý. Jednomu zadání obecně odpovídá mnoho různých programů, množina programů je velmi rozsáhlá [11].

3.1 Jedinec

Jedinci v genetickém programování představují jednotlivé kandidátní programy a jsou reprezentovány pomocí nelineárních hiearchických chromozomů s proměnlivou délkou ve formě syntaktických stromů (původně ve formě S výrazů jazyka Lisp). Jedinec představuje spustitelný počítačový program, který je pro ohodnocení potřeba spustit například pomocí interpretu.

Syntaktické stromy se skládají ze dvou typů genů: z terminálů (proměnných a konstant) a neterminálů (funkcí). Terminály a neterminály (souhrnně primitiva) tvoří vrcholy syntaktického stromu (k-cestný strom) a je nutné je pro každý problém předem definovat, protože jsou závislé na konkrétním problému.



Obrázek 1 – Stromová reprezentace funkce $y = x + 2\sin x$

Obrázek 1 ukazuje stromovou reprezentaci rovnice $y = x + 2\sin x$. Prvky stromu proměnná x a číselná konstanta 2 jsou terminály, operace +, * a funkce sin (sinus) patří mezi neterminály.

3.1.1 Neterminály (funkce)

Pro konkrétní problém je nutné definovat množinu funkcí (neterminálů) $F = \{f_1, \ldots, f_N\}$, které může evoluční algoritmus použít. Neterminály jsou vrcholy, které se nenachází v listech stromu. Očekávají další vstupní hodnoty (parametry). Parametry jsou v syntaktickém stromu navázány na vrchol s neterminálem jako jeho synové.

Vstupem neterminálů může být terminál nebo výstup jiného neterminálu. Jednotlivé funkce mohou mít různou aritu (počet parametrů). Funkce zároveň vrací výsledek operace pomocí výstupní hodnoty, která se může stát vstupem další funkce.

Mezi neterminály patří:

- aritmetické operace (+, –, *, /, chráněné dělení %, …),
- matematické funkce (sin, cos, exp, …),
- logické operace (and, or, not, …),
- podmínky (if-then-else),
- cykly (while, for),

funkce specifické pro daný problém (například robotický vysavač může mít funkce otoč_se(úhel), vysávej(sací_výkon) apod., některé specifické funkce jako tyto mají vedlejší efekty, protože posouvají robotem v prostoru či mění jeho stav).

3.1.2 Terminály

Pro konkrétní problém je nutné definovat také množinu terminálů $\tau = \{t_1, \ldots, t_N\}$, které může evoluční algoritmus pro tvorbu programů použít. Jde o listy stromu (vrcholy na konci stromu). Neterminály na rozdíl od funkcí nemají žádné parametry.

Mezi terminály patří:

- externí programové vstupy (například proměnné *x*, *y*, *rychlost* apod.),
- konstanty (specifikované napevno předem nebo náhodně vygenerované během vytváření stromu nebo při mutaci, například 4, 0,81, π, …),
- funkce bez parametrů (funkce představující senzory, například funkce vzdálenost_ke_zdi() pro program robota, funkce výška_hladiny() pro regulaci vodní nádrže apod., případně funkce generující náhodná čísla rand()).

3.1.3 Požadavky na terminály a neterminály

Aby bylo možné složit syntaktické stromy reprezentující programy, je nutné při volbě množiny terminálů τ a množiny neterminálů *F* splnit několik omezujících podmínek. Mezi nejdůležitější podmínky patří požadavek uzavřenosti a požadavek postačitelnosti [9].

Požadavek uzavřenosti (closure)

Požadavek uzavřenosti (closure) požaduje, aby každá z funkcí z množiny *F*, byla schopná přijmout jako vstup hodnotu jakéhokoli terminálu z množiny τ. Zároveň každá funkce musí být schopná přijmout jako svůj vstup hodnotu, která je výsledkem jakékoli funkce z množiny *F*. Aby mohl evoluční proces syntaktické stromy libovolně sestavovat, křížit a mutovat, musí všechny funkce v množině *F* vracet na výstupu stejný datový typ a zároveň musí mít všechny vstupní parametry stejného typu. Terminály musí být také stejného typu. U problematických funkcí je nutné definovat jejich chráněné verze, které mají ošetřené chování pro speciální stavy (například dělení nulou). Složitější problémy lze řešit například pomocí typovaného GP [9, 12].

Požadavek postačitelnosti (sufficiency)

Požadavek postačitelnosti se zabývá vhodným zvolením množin *F* (funkcí) a τ (terminálů) tak, aby bylo pomocí prvků těchto množin vůbec možné daný problém vyřešit. Pro daný problém je nutné zvolit vhodné konstanty, proměnné a funkce. Chybějící terminály mohou zkomplikovat hledání řešení nebo ho dokonce znemožnit. Nedostatečné množiny terminálů a neterminálů vedou k pouhé aproximaci řešení [12]. Nadbytečné funkce a terminály neúměrně[3] zvětšují prostor možných řešení [9]. Nevhodně zkombinované funkce se mohou při hledání řešení vzájemně vyrušit a sabotovat. V takovém případě hrozí i riziko bezúčelné evoluce, kdy je funkce během genetických operací nahrazována svým ekvivalentem, např. $sin(x)$ a $cos(x - \frac{\pi}{2})$).

[3] Neúměrně k potenciálnímu zlepšení programu.

3.1.4 Ohodnocení jedince (fitness funkce)

Pro ohodnocení jedince je nutné spustit kandidátní program, který jedinec představuje. Program je možné spustit pomocí interpretu. Druhou méně častou možností je zkompilování programu do strojového kódu a spuštění. Interpretace programu probíhá jako procházení příslušného syntaktického stromu. Pro ohodnocení jedince je často nutné program spustit víckrát, pro všechny testovací situace a vstupy.

Fitness funkce může být například:

- rozdíl opravdového výstupu s očekávaným, chyba či součet chyb (například u regrese),
- přesnost klasifikace (u klasifikačních problémů),
- čas potřebný k rozvozu zásilek logistické společnosti, případně ujeté kilometry nebo množství spotřebovaného paliva [8] apod.

V genetickém programování je možné optimalizovat i vícekriteriálně.

3.2 Generování počáteční populace

Při vytváření počáteční populace generuje genetické programování několik náhodných syntaktických stromů. Aby první generace neobsahovala příliš rozsáhlé stromy představující překomplikovaná řešení, omezuje se hloubka (vzdálenost od kořene k nejvzdálenějšímu listu) stromů v počáteční generaci na určitou maximální počáteční hloubku. Existují dvě základní metody s odlišnou strategií: úplná metoda (full method) a růstová metoda (grow method). Úplná metoda generuje úplné stromy [12]. Všechny větve úplných stromů jsou stejně dlouhé a jejich délka odpovídá zadané maximální počáteční hloubce. Růstová metoda vytváří pro počáteční populaci stromy s rozmanitějším tvarem a různou hloubkou, protože vygenerovaný strom nemusí mít všechny větve stejně dlouhé. Kombinací růstové a plné metody je metoda ramped half-and-half, která generuje půlku počáteční populace pomocí úplné metody a druhou půlku počáteční populace pomocí růstové metody [12].

3.3 Selekce

Selekce jedinců k reprodukci v genetickém programování je podobně jako u genetických algoritmů závislá na ohodnocení jedinců. Nejčastěji se v genetickém programování používá turnajové schéma (méně často pak ruletové kolo). Výhoda turnajového schématu oproti ruletovému kolu je, že algoritmus nemusí vědět, o kolik lepší je jeden program než druhý. Díky tomu lze ohodnocování a celý algoritmus genetického programování urychlit.

3.4 Genetické operátory

Mezi základní genetické operátory genetického programování patří (podobně jako u genetických algoritmů) křížení a mutace. Oproti reprezentacím s pevnou délkou chromozomu musí být genetické operátory v genetickém programování upraveny pro chromozomy v podobě syntaktických stromů s proměnlivou délkou.

Nejpoužívanějším křížením v genetickém programování je podstromové křížení (subtree crossover) [12]. Zatímco podstromové křížení vyměňuje mezi dvěma stromy rodičů jejich náhodné podstromy, jednobodové křížení provádí výměnu pouze jednoho vrcholu ze společné části dvou rodičů. V analogii s jednobodovým křížením zavedl Poli s Langdonem rovnoměrné (uniformní) křížení stromů ve společné části rodičů podobné klasickému rovnoměrnému (uniformnímu) křížení z genetických algoritmů. Dalšími druhy jsou kontext zachovávajícího křížení (context preserving crossover), size-fair crossover, selective crossover apod.

Mutace v genetickém programování probíhá jako náhodná změna syntaktického stromu (kódu programu). Mezi dva základní mutační operátory patří podstromová mutace a bodová (uzlová) mutace. V podstromové mutaci (subtree mutation) je vybrán náhodný vrchol syntaktického stromu, podstrom začínající v tomto vrcholu je odebrán a nahrazen novým náhodně vygenerovaným stromem. V bodové mutaci se iteruje stromem po jednotlivých vrcholech a s jistou pravděpodobností dochází k výměně funkce náhodně vybraného [12]. Dále existuje operátor Smooth [13], který kombinuje několik standardních aritmetických operací dohromady a umožňuje postupnou (hladší) změnu jednoho operátoru v druhý pomocí parametrů.

3.4.1 Neustálý růst programů a bloat

Bloat je jev, při kterém střední délka programů během evoluce roste. Podle [12] nejde o problém křížení ale selekce. Zatímco křížení způsobí prodloužení některých programů, jiné programy zase zkracuje. Delší programy mají oproti krátkým lepší ohodnocení, což dává delším programům selekční výhodu. Dále během evoluce často neúměrně roste množství kódu, který nemá v programu žádnou praktickou funkci a nemá ani žádný znatelný pozitivní vliv na výkon svého programu (například výsledek jedné větve je hodnota 0 a vstupuje jako parametr do funkce sčítání apod.), což je v rozporu s principem Occamovy břitvy.

Proti bloatu bylo vyvinuto několik různých strategií. Nejjednodušší strategií je definování a kontrola maximální velikosti nebo maximální hloubky programů (syntaktických stromů). Účinnou strategií se jeví prořezávání syntaktických stromů (pruning) [13]. Další možností je penalizace delších, složitých a nadměrně parametrizovaných programů a změnit tak směr selekčního tlaku na šetrnost (parsimony pressure) [6]. Pro omezení bloatu je také možné zavést speciální genetické operátory jako jsou size fair křížení, size fair mutace, hoist mutace a shrink mutace apod. [9]

3.5 Další druhy genetického programování

Lineární genetické programování [14] na rozdíl od syntaktických stromů používá lineární chromozomy s pevnou délkou [6] a pracují s pamětí a registry. Naopak grafové genetické programování využívá grafovou reprezentaci programů (PDGP, PADO [12], Kartézské genetické programování [15], MEP [16] a SNGP [17]).

3.6 Genetické algoritmy, genetické programování a strojové učení

Genetické algoritmy se jako optimalizační metoda se dají používat ve strojovém učení. Mohou být využity pro ladění hyperparametrů neuronové sítě a jiných algoritmů strojového

učení, nalezení vhodné architektury neuronové sítě, nejvhodnějšího prořezávání rozhodovacích stromů apod. Genetické algoritmy řeší složité kombinatorické a dopravní problémy jako jsou TSP, VRP apod. Lze je využít i pro metody snížení dimenzionality dat. Některé aplikace genetických algoritmů a genetického programování jsou supervizovaným strojovým učením a konkurují například neuronovým sítím nebo metodě SVM. Symbolická regrese slouží pro hledání modelů dat v podobě matematických rovnic a predikci. Hledání programů pomocí GP nebo automatický design elektrických obvodů, antén, křídel a dalšího hardwaru zase připomínají metody zpětnovazebního učení (reinforcement learning).

3.6.1 Symbolická regrese

Symbolická regrese je supervizovaná metoda strojového učení pro hledání symbolických popisů matematických modelů, které aproximují konečnou množinu datových bodů [12]. Cílem je nalezení vztahu závislé proměnné na množině nezávislých proměnných. Řešením symbolické regrese je interpretovatelná matematická rovnice složená z množiny povolených aritmetických výrazů, funkcí, proměnných a konstant.

Zatímco obyčejná regrese (například lineární regrese ale i regrese pomocí neuronových sítí) hledá pouze správné hodnoty parametrů pro předem danou strukturu modelu, symbolická regrese kromě parametrů matematického modelu hledá zároveň i jeho optimální strukturu (nejlepší kombinaci operací, funkcí a parametrů). Celý proces učení je založen na minimalizaci chyby předpovědi modelu vůči skutečné hodnotě závislé proměnné. Prohledávaný prostor je plný velkého množství duplicitních řešení.

Symbolická regrese se dá řešit i jinými metodami než pomocí GP. Lze použít například metodu neevoluční metodu FFX. V nedávné době byl pro symbolickou regresi uveden algoritmus AI Feynman, pro hledání fyzikálních zákonů [18].

4 OPTIMALIZACE STROJOVÉHO UČENÍ

Proces strojového učení obecně lze zrychlit paralelizací výpočtů a použitím specializovaného hardwaru. Příkladem mohou být neuronové sítě, jejichž trénování a inference může probíhat na grafických kartách či speciálních neuronových čipech. Pro opakované výpočty lze implementovat softwarové paměti cache. Strojové učení na reálných množinách trénovacích dat se potýká s mnoha druhy problémů. Množiny trénovacích dat bývají plné duplicitních, poškozených a nepřesných datových bodů. Dalším problémem jsou množiny trénovacích dat s nevyváženým zastoupením jednotlivých klasifikačních tříd, na které je nutné použít techniky strojového učení na nevyvážených datech. Nedokonalostí algoritmů strojového učení bývá i tendence k přeučení a z toho plynoucí špatná schopnost generalizace. Pro zlepšení této situace lze použít předzpracování (preprocessing) dat pomocí různých filtrů nebo autoenkodérů.

4.1 Redukce objemu dat

Množiny trénovacích dat s obrovským množství záznamů jsou náročné na zpracování, protože samotné vyhodnocení celkové chyby na velkém množství dat zabírá hodně času. Reálné problémy s datovými sadami[4] s velkou datovou dimenzionalitou (obsahující velký počet proměnných) jsou výzvou jak pro optimalizaci tak pro klasifikační a regresní analýzu.

4.1.1 Snížení dimenzionality dat

U mnohorozměrných problémů velikost prohledávaného prostoru roste exponenciálně s počtem proměnných. Pokud má hledaný model obsahovat všechny proměnné, bude složitější než model obsahující malý počet proměnných. Možnosti vzájemných vztahů mezi proměnnými exponenciálně rostou s jejich počtem. Klasifikátory pracující s větším počtem proměnných (vlastností) jsou náročnější na velikost trénovací množiny a jsou náchylnější k přeučení. Se zvyšujícím se počtem uvažovaných vlastností je složitější identifikovat, které vlastnosti jsou pro predikci modelu důležité. Řešením těchto problémů může být snížení počtu vstupních proměnných (snížení dimenzionality dat) pomocí metod jako jsou:

- feature selection (selekce vlastností),
- feature extraction (extrakce vlastností),
- feature construction (konstrukce vlastností).

Cílem těchto technik je snížení množství dat, zaměření se na relevantní data a zlepšení jejich kvality a tím i výkonu dataminingových algoritmů, které se z těchto dat učí [19]. Mezi výhody feature selection patří zjednodušení vizualizace a lepší pochopení dat, snížení nároků na měření hodnot, ušetření místa na úložišti, zkrácení času potřebného k naučení modelu a překonání prokletí dimenzionality následované zvýšením přesnosti a schopnosti generalizace výsledného modelu [20].

[4] Pojem datová sada (z anglického dataset) se zde používá jako synonymum pro množinu trénovacích dat.

Feature selection

Technika feature selection vybírá podmnožinu m vlastností z původní množiny n vlastností, přičemž platí, že $m \leq n$ [19]. Cílem metody je vybrat podmnožinu n vlastností tak, aby byl prostor vlastností optimálně zredukován. Někdy se může stát, že skupina vlastností, které jsou samy o sobě neužitečné (málo informativní) mohou být dohromady užitečné (informativní). Nejde tedy jen o hledání užitečných vlastností ale skupiny vlastností s dobrou synergií [21].

Feature extraction

Technika feature extraction extrahuje nové vlastnosti z původních vlastností pomocí transformace n-prvkovou množinu vlastností $\{a_1, a_2, \ldots, a_n\}$ na m-prvkovou množinu vlastností $\{b_1, b_2, \ldots, b_m\}$ přičemž $m < n$ [19]. Cílem techniky feature extraction je najít takovou transformaci, která minimalizuje množinu vlastností s ohledem na vybranou metriku a najít nejlepší reprezentaci dat, která zvýší přesnost návazného algoritmu strojového učení [22]. Správnou transformaci lze nalézt například pomocí metody shlukování (k-means), metody hlavních komponent (PCA), lineární diskriminační analýzou (LDA) nebo pomocí neuronové sítě [19] či genetického programování [21].

Feature construction

Příbuznou technikou k feature extraction je technika feature construction. Cílem techniky feature construction je najít správnou reprezentaci dat, zvýšení vyjadřovací schopnosti původních vlastností a objevení skrytých vztahů mezi jednotlivými vlastnostmi (vstupními proměnnými) [21] pomocí konstruování nových dodatečných vlastností, například pomocí nelineární expanze (například $x_1 * x_2$). Pomocí skládání původních vlastností jsou skládány nové vlastnosti. Pro skládání vlastností se používají různé operátory. Mezi operátory pracující s logickými výrazy patří například konjunkce, disjunkce, negace, operátor m-of-n (vrací pravdu, pokud alespoň m z n vlastností je pravda) [21]. Pro numerické vlastnosti se používají různé algebraické operátory jako jsou sčítání, násobení, odčítání, dělení, maximum, minimum, průměr, rovnost, nerovnost apod.

4.2 Sborové učení (ensemble learning)

Hlavní idea sborového učení (ensemble learning) je inspirována lidským rozhodováním založeným na názorech více expertů či hlasování (členové týmu, poslanci, porota u soudu). Sborové učení je skupinou statistických metod a metod strojového učení, které finální model kombinují pomocí více dílčích modelů za účelem zpřesnění predikce modelů, zlepšení schopnosti generalizace a snížení míry přeučení [23]. Výsledkem kombinace dílčích modelů je tzv. sbor modelů. Je experimentálně ověřeno, že sborové učení je schopné zefektivnit hledání regresního či klasifikačního modelu na jednom počítači [23]. Metody sborového učení jsou zároveň jednoduše paralelizovatelné [24].

Hledání optimálního modelu bývá v praxi složitým problémem. Učící algoritmy spouštěné nad malou množinou datových bodů mají tendenci k přeučení a špatné schopnosti generalizace. Kombinováním či zprůměrováním více vhodně vybraných modelů naučených

nad různými podmnožinami původních datových bodů je možné se k optimálnímu modelu dostatečně přiblížit, vyhnout se přeučení a zlepšit jeho generalizaci [23, 24].

Problémem je i omezená vyjadřovací kapacita různých algoritmů strojového učení, která omezuje prostor možných modelů. Je možné, že optimální model díky vlastnostem modelů produkovaných algoritmem není sestavitelný, protože se optimální model nenachází ve stavovém prostoru, ve kterém učící algoritmus provádí vyhledávání. Správnou kombinací více dílčích modelů z omezených stavových prostorů se však vyjadřovací schopnosti algoritmů zvyšují [23]. Rozdělení na dílčí modely také zefektivňuje prohledávání složitých stavových prostorů možných modelů. Zkombinováním více různých suboptimálních modelů nalezených učícími algoritmy s různě nastavenými počátečními podmínkami lze dosáhnout o mnoho lepší aproximace optimálního modelu [23, 24]. Predikce dílčích modelů nesmí být korelované, protože by nepřidávaly do rozhodovacího procesu novou informaci. Celková přesnost zkombinovaných modelů je tedy závislá na přesnosti dílčích klasifikátorů a zároveň na diverzitě jejich výstupů [23].

4.2.1 Architektura sboru modelů

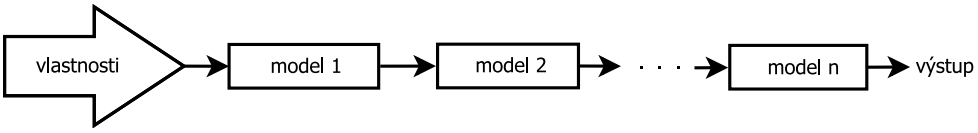
Architektura sboru modelů se zabývá tvorbou a výběrem dílčích modelů, topologií jejich propojení a návrhem fúzního modulu, která kombinuje výstupy dílčích modelů [24].

Topologie propojení dílčích modelů

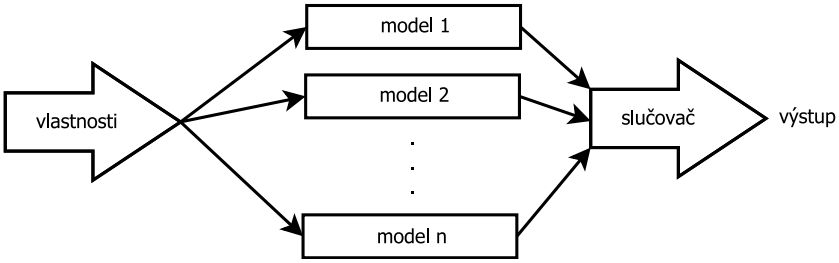
Existují dva základní způsoby, jak mezi sebou dílčí modely propojit:

- sériová topologie (Obrázek 2),
- paralelní topologie (Obrázek 3).

V paralelní topologii dílčí modely zpracovávají vstupní data nezávisle a finální výsledek je kombinací jejich výstupů [24]. V sériové topologii jsou dílčí modely seřazeny v sekvenci za sebou. První model zpracovává vstupní data a každý další dílčí model upravuje výsledky modelu, který se v sekvenci nachází před ním. Složitější topologie zahrnují různé kombinace sériové a paralelní topologie nebo vícevrstvou hiearchickou paralelní topologii, ve které dochází ke kombinování dílčích sborů modelů.



Obrázek 2 – Sériová topologie sborového modelu, zdroj: autor



Obrázek 3 – Paralelní topologie sborového modelu, zdroj: autor

Tvorba a výběr dílčích modelů

Dílčí modely by do sboru modelů měly být vybírány tak, aby se navzájem dobře doplňovaly (byly vůči sobě komplementární), disponovaly vysokou přesností a diverzitou předpovědí. Diverzita modelů se dá měřit různými způsoby. Jednou z metrik může být odchylka předpovědí. Diverzity lze dosáhnout tím, že každý dílčí model bude trénován na jiné množině datových bodů. Každý model může být vytvořen pomocí jiné podmnožiny vstupních proměnných (vlastností), stejným algoritmem s jinými počátečními podmínkami či jiným nastavením hyperparametrů nebo dokonce jiným učícím algoritmem [23].

Diverzifikace vstupních dat je založena na předpokladu, že modely naučené na rozdílných vstupních datech budou vůči sobě komplementární. Diverzifikovat data lze pomocí dvou strategií [24]: rozdělení množiny vstupních dat na různé podmnožiny nebo rozdělení množiny vstupních proměnných na různé podmnožiny. První strategie nechá jednotlivé modely trénovat na různých podmnožinách vstupních datových bodů. Druhá strategie používá pro dílčí modely rozdílné podmnožiny množiny (podprostory prostoru) vstupních proměnných a každý z modelů zkoumá odlišné závislosti.

Návrh fúzního modulu

Návrh fúzního modulu tzv. slučovače (fuser) se zabývá hledáním optimální kombinace jednotlivých dílčích modelů do finálního výstupu sborového modelu. Existují dvě základní varianty slučovače. V první variantě slučovač slučuje pouze výstupy dílčích modelů. V druhé variantě slučovač slučuje výstupy dílčích modelů s hodnotami vstupních proměnných [24].

Pro fúzi dílčích klasifikačních modelů se používá metoda většinového hlasování (majority voting) [24]. Metoda většinového hlasování jako výsledek fúze vybírá předpověď, na které se shodne nejvíce dílčích klasifikátorů. Rozšířením většinového hlasování je metoda váženého hlasování, ve kterých má každý klasifikátor svou hlasovací váhu (různý počet hlasů). Váha klasifikátoru může být odvozena od jeho přesnosti klasifikace. Váhy mohou být konstantní nebo mohou být závislé na vstupních proměnných [24].

Nejjednodušším přístupem pro fúzi dílčích regresních modelů je použití jednoduchých operátorů, mezi které patří například operátor minimum, maximum, průměr, uspořádaný vážený průměr nebo fuzzy agregační operátory (fuzzy integrál) [23].

Fúzní moduly mohou být i trénovatelné [23] ve smyslu, že chování slučovače může být natrénováno na konkrétní data. V metodě stacking [24] je trénovací fúzní model reprezentován metamodelem (modelem nad dílčími modely). Metamodel přebírá výstupy jednotlivých dílčích modelů z předchozí vrstvy a učí se je správně zkombinovat do finálního výsledku sborového modelu. Metamodel nebývá trénován na úplně stejných datech jako dílčí modely, protože může dojít k přeučení sborového modelu.

4.2.2 Negenerativní sborové metody

Negenerativní sborové metody se věnují pouze výběru nebo kombinování již hotových dílčích modelů ale dílčí modely nevytváří. Patří sem již zmíněné metody pro fúzi dílčích modelů a metody pro selekci dílčích modelů.

Metody pro selekce dílčích modelů

Metody pro selekci dílčích modelů se snaží nalézt nejlepší dílčí model sborového modelu (jeho výstup bude výstupem sborového modelu) nebo několik nejlepších dílčích modelů. V tomto případě je nutné zkombinovat výstupy skupiny nejlepších modelů do jednoho výstupu pomocí nějaké fúzní metody. Pro tvorbu sborových modelů z hotových dílčích modelů lze použít i metody z oblasti feature selection. Dopředná selekce (forward selection) a zpětná eliminace (backward elimination) sestavuje sborový model přidáváním a odebíráním dílčích modelů a snaží se maximalizovat přesnost a schopnost generalizace sborového modelu za pomoci různých optimalizačních metod [23].

4.2.3 Generativní sborové metody

Generativní sborové metody vytvářejí dílčí modely pomocí učícího algoritmu a diverzifikují množinu trénovacích dat za účelem vytvoření diverzních dílčích modelů a zvýšení přesnosti a generalizace sborového modelu.

Metody založené na resamplingu trénovací množiny dat

Metody založené na resamplingu metody využívají k diverzifikaci trénovacích dat techniku bootstrapping [23]. Technika boostraping pomocí náhodného výběru generuje z trénovací množiny dat několik různých nedisjunktních podmnožin trénovacích dat. Poté jsou aplikováním učícího algoritmu na jednotlivých vygenerovaných podmnožinách vytvořeny dílčí modely. Metoda bagging (zkratka pro bootstrap aggregating) vytváří pomocí techniky boostraping dílčí modely (každý datový bod má stejnou pravděpodobnost výběru) a jejich výsledky kombinuje například zprůměrování výstupů v regresních problémech nebo pomocí většinového či váženého hlasovaní v klasifikačních problémech [23]. Metoda wagging (weighted bagging) je speciální variantou metody bagging založená na nerovnoměrné pravděpodobnosti výběru datových bodů z trénovací množiny [23].

Metoda boosting je iterační metoda, která v jednotlivých krocích mění rozdělení zastoupení datových bodů v trénovací množině (nebo jejich váhy) vzhledem k chybě dílčích modelů [23]. Konkrétní variantou metody boosting je například metoda AdaBoost (Adaptive Boosting). AdaBoost při tvorbě každého dalšího dílčího modelu dává důraz na datové body trénovací množiny, které se předchozí dílčí modely špatně naučily.

Metody založené na diverzifikaci množiny vstupních proměnných

Pro řešení problémů s daty s vysokou dimenzionalitou a řídkými daty lze použít metody feature selection a feature extraction původně vytvořených ke snížení dimenzionality dat. Metody feature selection a feature extraction se dají použít k vytvoření diverzních dílčích modelů naučených na různých podmnožinách vstupních proměnných (vlastností). Dále lze použít deterministickou metodu Input Decimation nebo metodu náhodných podprostorů (random subspace methods) [23]. V metodě rotačních stromů (rotation forrests) [23] jsou za účelem zvýšení diverzity dílčích modelů náhodné podmnožiny vstupních proměnných transformovány náhodnou rotací.

4.3 Optimalizace genetických algoritmů

Optimalizovat GA lze z hlediska jejich rychlosti nebo kvality produkovaných řešení.

4.3.1 Paralelní genetické algoritmy

Paralelní genetické algoritmy (PGA) používají více oddělených souběžně vyvíjejících se subpopulací, které spolu komunikují a mezi kterými dochází k průběžné migraci jednotlivců [9]. Pro migraci mezi subpopulacemi existují různé topologie s odlišnými vlastnostmi: kruhová topologie, kde jedinci migrují pouze mezi sousedními subpopulacemi, topologie ve tvaru hvězdy, různé dvourozměrné a vícerozměrné topologie v podobě matic nebo čistě náhodná migrace. Výhodou PGA je možnost rozvíjet jednotlivé subpopulace na různých procesorech a tím urychlit běh.

4.3.2 Náhradní modely

V případě výpočetně náročného, zdlouhavého či drahého získávání hodnoty fitness funkce se nabízí využít techniky náhradních modelů (surrogate models) neboli metamodelů. Náhradní modely aproximují chování reálných komplexních systémů a detailních simulací a snižují tak čas či cenu potřebné k získání hodnoty fitness funkce. Náhradní modely se vytváří pomocí metod strojového učení s učitelem (supervizované algoritmy) [25, 26]. Použitím náhradních modelů v evolučních algoritmech se zabývají například publikace [27] a [28].

4.3.3 Hybridní genetické algoritmy

Genetické algoritmy jsou ve své univerzální podstatě slepé prohledávací algoritmy, které o optimalizovaném problému prakticky nic nepředpokládá (black-box optimalizace). GA by bylo možné zrychlit použitím užitečných informací o problému [10] či kombinací s tradičním algoritmem, který se pro daný problém používá. Další možností je použít tradiční algoritmus pro tvorbu počáteční populace pro rychlejší start GA. Podle [29] je vhodné kombinovat GA s nějakou jinou optimalizační metodou (například přidat jedincům paměť jako u tabu search, koncept teploty a postupného ochlazování ze simulovaného žíhání apod.). Další možností je využít jedince GA pro nalezení vhodných počátečních podmínek pro problémově specifickou metodu lokálního prohledávání [30].

Memetické algoritmy

Memetické algoritmy vznikají spojením GA (obecně evolučních algoritmů) s nějakou metodou lokálního prohledávání. Kombinací těchto dvou přístupů může dojít ke zlepšení prohledávací schopnosti (exploration) okolí jedinců GA. V tomto případě hybridní GA kombinuje globální prohledávání evolučního přístupu na úrovni celé populace s lokálním prohledáváním na úrovni jednotlivých jedinců. Lokální učení představuje učení během života jedince a posouvá ho směrem blíže k (lokálnímu) optimu [31, 32, 33]. Učení během života jedince vyhlazuje tvar geometrie fitness funkce a zjednodušuje tak evoluci [34]. Pro učení jedince existují dva hlavní přístupy: Lamarckismus a Baldwinismus. Jde o pojmy známé z biologie.

V Lamarckovské evoluci dochází k přenosu naučených vlastností přímo na potomky. Jelikož dochází ke změně genotypu, Lamarckismus potřebuje inverzní mapování z fenotypu (a prostředí) na genotyp [34, 35]. Whitley v článku [31] tvrdí, že tento přístup deformuje populaci. Lamarckovská evoluce není biologicky přesná, nicméně odpovídá naopak kulturní evoluci, při které si jedinci předávají zkušenosti mezi generacemi. Baldwinismus (Baldwinův efekt) je čistě darwinovský a vlastnosti naučené během života ovlivňují pouze jedincovu hodnotu fitness funkce. Baldwinův efekt ovlivňuje jedince nepřímým způsobem. Jedinci, kteří se během života dobře učí a jsou blíže optima mají větší hodnotu fitness funkce a a díky tomu mají průměrně více potomků. Baldwinův efekt dává šanci dobrým (ale ne skvělým) genům odolat náhodnému neúspěchu během selekce a zůstat tak v populaci [36]. Lokální adaptace většinou představuje asociální učení jedince (každý jedinec se učí sám). V poslední době však probíhá i výzkum sociálního učení jedinců pomocí imitace ostatních jedinců [37, 32, 38].

Je však nutné myslet na fakt, že příliš silné lokální prohledávání může narušit evoluční charakter prohledávání a vést tak ke konvergenci do lokálních minim. Lokální učení může za jistých podmínek urychlit evoluci [33, 34]. Může zlepšit diverzitu populace [39], čímž může proces evoluce urychlit. Mezi zápory lokálního učení patří zvýšená režie a větší zátěž procesoru. V některých případech může být silné lokální učení kontraproduktivní a zpomalovat evoluci.

4.4 Optimalizace genetického programování a symbolické regrese

Mezi hlavní možnosti optimalizace GP patří: zkrácení vyhodnocování fitness funkce a zlepšení konvergence GP.

4.4.1 Zkrácení vyhodnocování fitness funkce

Zkrátit vyhodnocování fitness funkce lze pomocí zmenšení velikosti stromů, zmenšením trénovací množiny, zavedením paměti cache nebo v případě složitých účelových funkcí pomocí zavedení náhradních modelů (surrogate models).

Zmenšení velikosti stromů

Velké stromy (programy) se pomalu vyhodnocují, proto je vhodné pro rychlejší běh GP omezovat jejich velikost. Většinou se používá omezení maximální hloubky stromu

nebo maximálního počtu vrcholů stromu. Další možností je zmenšit velikost stromů nepřímo zavedením faktoru šetrnosti (parsimony factor), který penalizuje větší programy. V případě podobného výkonu evoluce se preferují kompaktnější programy. Podobně lze penalizovat stromy jejichž běh trvá příliš dlouho [12]. Při hybridizaci GP s lokálním učením pro optimalizaci konstant může ke zmenšení velikosti stromů může dojít nepřímo, protože větší stromy s více konstantami se optimalizují složitěji a výhodu tak dostanou stromy se s obdobně dobrou strukturou ale s menším počtem konstant.

Zmenšení trénovací množiny

Při používání menší trénovací množiny lze urychlit vyhodnocení fitness funkce jedince. Tento přístup může zabránit přeučení modelu a zlepšit jeho generalizaci. Rizikem je, že dojde k naučení pouze části hledané funkce [12].

Dynamická fitness funkce

Fitness funkce může být statická (pro všechny vyhodnocení stejná podmnožina trénovacích dat) nebo dynamická (pro každé vyhodnocení může být jiná). Problémem statické funkce může být situace, kdy dobré programy fungující dobře na většině částí trénovací množiny, mají nižší ohodnocení než ty, které fungují skvěle jen v několika malých regionech. Dynamická fitness díky změnám trénovací množiny lépe zabraňuje přeučení vzniku jedinců specializovaných na konkrétní části hledané funkce [12].

Zavedení paměti cache

Z principu fungování GP mají jedinci (programy) v populaci mnoho společného kódu (stejné podstromy). Při vyhodnocování se tak často duplicitně provádění stejné výpočty pro všechny vstupní hodnoty. Při použití paměti cache, do které se uloží výsledky jednotlivých podstromů, lze výrazně zkrátit vyhodnocování podobných jedinců se společnými částmi [12].

Zvolení vhodného selekčního mechanismu

Další možností je využití turnajové selekce s malými turnaji, přičemž se na některé programy ani nedostane, takže se ušetří ohodnocování některých programů [12]. Rizikem může být potenciální vyloučení výkonného programu v souboji s ještě výkonnějším nebo v případě, že se do turnajů vůbec nedostane.

4.4.2 Hybridní GP a ladění konstant

Cílem hybridního přístupu je zvýšení výkonu algoritmu genetického programování pomocí vybraných metod lokálního učení a heuristik. Většina metod je založena na myšlence, že genetické programování může během evoluce vygenerovat strom se správnou strukturou terminálů jehož výkon bude negativně ovlivněn pouze špatnou kombinací číselných konstant (parametrů) v terminálech. Jedním ze základních přístupů je ladění konstant (parametrů modelu), při kterém se pro vybraný strom s danou strukturou neterminálů (model) hledá optimální kombinace číselných konstant (optimální parametry modelu) pomocí dalších optimalizačních metod. Evett a Fernandez [40] navrhli k ladění konstant jednoduchou mutaci, čímž snížili výpočetní nároky. Raidl a Gunther v článku [41] představili HGP (hybridní

genetické programování), přidali váhy k jednotlivým členům modelu na nejvyšší úrovni (tak aby stačila lineární regrese) a ladili je pomocí robustní metody nejmenších čtverců. Pro optimalizaci konstant lze použít například několik iterací gradientního sestupu (gradient descent) [42], simulované žíhání v kombinaci se Simplexovou metodou [43], optimalizaci hejnem částic (PSO) [44], vícenásobnou regresi v metodě STROGANOFF [45, 46], evoluční strategie [47], algoritmus Bison Seeker [48] apod. Konstanty ve stromě se ovlivňují často nelineárně, takže bylo navrženo použít algoritmus Levenberg-Marquardt pro nelineární regresi [49, 50]. Publikace [51] vylepšuje výkon GP pomocí aplikace metod bagging a boosting.

4.4.3 Moderní metody optimalizace GP a symbolické regrese

Intervalová aritmetika a lineární škálování

Keijzer v [52] navrhl vylepšení symbolické regrese pomocí intervalové aritmetiky a lineárního škálování. Intervalová aritmetika se stará, aby GP nepracovalo s matematickými funkcemi v nedefinovaných hodnotách pomocí výpočtů hranic použitelných hodnot pro jednotlivé funkce. Lineární škálování se zabývá minimalizací chyby pomocí posunutí funkce a její roztažení, což má za následek, že funkce se správným tvarem ale špatným posunutím nebo roztažením nezmizí během selekce a je nalezena rychleji[5]. Hledá optimální posun a a koeficient roztažení funkce b pro funkci $f(x)$ tak, aby se minimalizovala chyba modelu ve tvaru $y = a + bf(x)$ za pomoci jednoduché lineární regrese. Hledání těchto dvou konstant (posunu a a roztažení b) již není problémem GP, které se může soustředit na hledání správného druhu funkce (správné struktury modelu).

Multistage Genetic Programming (MSGP)

MSGP [53] postupně vytváří pomocí GP modely pro jednotlivé vstupní proměnné (vlastnosti) zvlášť a následně sestavuje model vzájemné interakce jednotlivých proměnných mezi sebou. Vytvořené modely poté dávají dohromady jeden velký model $f(X)$ v rovnici (1).

$$f(X) = f_1(x_1) + f_2(x_2) + \cdots + f_n(x_n) + f_{int}(X) = \sum_{i=1}^{n} f_i(x_i) + f_{int}(X) \qquad \textbf{(1)}$$

Kde n je počet vstupních proměnných, x_n je konkrétní vstupní proměnná, $f_n(x_n)$ je model pracující s proměnnou x_n a $f_{int}(X)$ je model interakce mezi jednotlivými proměnnými x_n.

Multiple Basis Function Genetic Programming (MBF-GP)

Multiple Basis Function Genetic Programming [54] je velmi podobné klasickému GP. Jediným rozdílem je, že každý jedinec se skládá z více stromů, přesněji řečeno je lineární sumou několika vybraných nelineárních bázových funkcí. Na makroskopické úrovni mají jednotlivé bázové funkce (dílčí stromy) své váhové koeficienty, které tvoří parametry

[5] Podobnou ideu v kvalitativním porovnávání (tvarů) funkcí lze nalézt i v Brandejský, T.: Genetic Algorithm of Numerical Concretization of Qualitative Model. In: Mendel 2003. Brno: Brno University, 2003, p. 65-69. ISBN 80-214-2411-7

makroskopického modelu. Váhy se počítají pomocí lineární regrese. Konstanty uvnitř bázových funkcí lze optimalizovat pomocí Levenberg-Marquardt algoritmu.

Genetické operátory používá MBF-GP stejné jako GP a to rovnou na dvou úrovních [49]. Na mikroskopické úrovni probíhá křížení klasicky mezi stromy vybraných bázových funkcí dvou náhodně vybraných makroskopických jedinců. Makroskopické křížení naopak vyměňuje celé stromy bázových funkcí mezi dvěma makroskopickými jedinci. Jelikož jde o chromozomy s proměnlivou délkou, může se stát, že nově vzniklí jedinci budou mít po křížení odlišné počty stromů bázových funkcí.

Multi-Gene GP (MGGP)

Metoda MGGP rozšiřuje klasické GP a zvyšuje jeho přesnost [55]. Podobně jako v MBF-GP se chromozom skládá z více submodelů v podobě klasických GP stromů krátké délky. Každý takový strom představuje jeden gen. Narozdíl od MBF-GP se stromy netvoří sekvenčně, nýbrž souběžně. Výsledný model je lineární kombinací jednotlivých submodelů. Jednotlivé výstupy jednotlivých stromů mají svou váhu a tyto váhy se hledají pomocí lineární regrese (Moore-Penroseova pseudoinverze). V tomto ohledu je MGGP sborovou metodou. MGGP používá dva druhy křížení: makroskopické (vysokoúrovňové, high level) a mikroskopické (nízkoúrovňové). V makroskopickém křížení dochází k výměně celých genů (stromů) mezi dvěma náhodnými rodiči. Jelikož může množství genů v chromozomu neustále růst, je vhodně zvolit konstantu g_{max} a pokud počet genů v chromozomu překročí tuto hranici, je nutné některé geny (stromy) smazat. Mikroskopické křížení je podobné jako v klasickém GP. V obou rodičích se náhodně vybere jeden gen (strom) a tyto stromy si mezi sebou vymění náhodné podstromy.

Článek [56] popisuje MGGP v kombinaci Linear combination of Features (LCF). MGGP s LCF zavádí v rámci stromů nový druh listů, které nereprezentují proměnnou nebo vlastnost (feature), ale lineární kombinaci všech vlastností (features) řešeného problému. Matematicky jde o funkci $lcf(x) = a + bx$, kde x je vektor hodnot jednotlivých vlastností (feature values), a a b jsou konstanty (a je aditivní váha a b je vektor multiplikativních vah). LCF efektivně provádí afinní transformace prostoru vlastností (feature space) [57], čímž zlepšuje možnosti algoritmu GP.

Další metody

Mezi další metody patří sekvenční symbolické regrese (SSR) (Sequential Symbolic Regression) obsahující dekompoziční fázi [58]. Multiple Regression Genetic Programming (MRGP) [59] rozkládá stromy na potenciálně cenné komponenty a vytváří jejich lineární kombinace ve spolupráci s vícenásobnou regresí. Evolutionary Feature Synthesis (EFS) [60] je evoluční metoda podobná FFX, ve které populace neobsahuje jednotlivé stromy, místo toho se skládá z jednotlivých vlastností (features), které dohromady tvoří jeden velký model. Geometrické sémantické genetické programování (GSGP) [61] rozlišuje mezi syntaxí (stromy) a sémantikou (výstupní hodnoty jednotlivých kandidátních funkcí) a pro prohledávání stavového prostoru používá jednoduché lineární operátory. Stavový prostor je tak unimodální a jednoduchý na prohledávání. Vylepšením GSGP je GSGP-LS,

které je kombinací GSGP s lokálním prohledáváním [62]. Fast function extraction (rychlá extrakce funkcí), neboli FFX, je deterministický algoritmus pro symbolickou regresi [63, 64], který používá metodu feature construction (konstrukce vlastností), deterministicky generuje nové vlastnosti (bázové funkce). Výsledný model poté vytváří jako lineární kombinace vytvořených bázových funkcí. Podobnou metodou jako FFX je metoda Elite Basis Regression (EBR) [65], která na rozdíl od FFX většinu vygenerovaných bázových funkcí vyřadí a zachovává a aktualizuje pouze ty bázové funkce, které dobře korelují s výstupními daty. Článek Temporal Feature Selection with Symbolic Regression [66] popisuje nový druh terminálu, tzv. terminál rozsahu (range terminal), který zjednodušuje symbolickou regresi časových řad. Terminál rozsahu má tvar $\psi(f, a, b)$, kde f je agregační funkce (sum, min, max, mean, skew), a a b je rozsah příslušné proměnné (od – do).

5 POROVNÁNÍ LAMARCKISTICKÉHO A BALDWINISTICKÉHO PŘÍSTUPU V GP

V publikaci [67] bylo cílem vyzkoušet implementovanou knihovnu pro symbolickou regresi a porovnat lamarckistického a baldwinistického přístupu k hybridnímu algoritmu genetického programování při optimalizaci konstant. Pro porovnání byla náhodně vygenerována datová sada odpovídající jednoduché polynomické funkci v rovnici (2) s celočíselnými koeficienty o 20 prvcích z intervalu [-10; 10]:

$$y = 2x^3 + 3x^2 + 9 \quad \textbf{(2)}$$

Velikost populace byla nastavena na 20 jedinců, elitismus na hodnotu 1, pravděpodobnost mutace na 10 %. Množina terminálů obsahovala konstanty 1.0, 2.0, 3.0, proměnnou *x* a množina neterminálů obsahovala z matematických operací násobení (*) a sčítání (+). Maximální hloubka stromů v 1 generaci byla 2. Jako genetické operátory byly zvoleny podstromové křížení a podstromová mutace. Pro selekci bylo zvoleno deterministické turnajové schéma o 2 účastnících. Pro generování náhodných čísel byla využita implementace generátoru Mersenne Twister v jazyce Java. Jako fitness funkce byl zvolen kvadrát chyby modelu přepočtený na jeden trénovací příklad. Pro optimalizaci konstant byl použit stochastický horolezecký algoritmus, který vytvářel sousední stavy tak, že zvyšoval nebo zmenšoval náhodný celočíselný koeficient stromu o hodnotu 1. Pro úspěšné ukončení algoritmu muselo GP nalézt matematický model popisující data a minimalizující chybu na hodnotu 0.0 (bez jakékoli odchylky).

Pro účely experimentů byl pro oba přístupy algoritmus genetického programování spuštěn 10000krát a vnořený stochastický horolezecký algoritmus generoval vždy 4 sousední stavy ve 3 iteracích. V obou přístupech byl porovnáván medián a průměr generace, při které došlo k nalezení modelu s chybou 0.0. Výsledky zobrazuje Tabulka 1.

Tabulka 1 – Porovnání regrese bez adaptace a s adaptacemi

Druh regrese	Medián úspěšné generace	Průměrná úspěšná generace
Bez adaptace	287	465
Lamarckismus	186	299
Baldwinismus	274	426

Pro účely druhého experimentu byl počet kroků stochastického horolezeckého algoritmu zvýšen pro oba přístupy na hodnotu 8. Výsledky zobrazuje Tabulka 2.

Tabulka 2 – Porovnání regrese bez adaptace s adaptacemi se zvýšeným počtem kroků

Druh regrese	Medián úspěšné generace	Průměrná úspěšná generace
Bez adaptace	297	472
Lamarckismus	182	298
Baldwinismus	264	415

Dále byly porovnány oba přístupy s jinými ukončovacími podmínkami. Algoritmus byl vždy 10000krát spuštěn, aby se ukázalo, kolikrát dokáže GP nalézt model s omezením v podobě různých hodnot pro maximální počet generací. Počet kroků stochastického horolezeckého algoritmu byl nastaven na hodnotu 3 jako v prvním experimentu. Výsledky zobrazuje Tabulka 3.

Tabulka 3 – Porovnání počtu úspěšných řešení jednotlivých regresí v generačních limitech

Limit	**Bez adaptace**	**Lamarckismus**	**Baldwinismus**
5	5	6	9
25	223	325	194
50	875	1204	756
250	4457	6020	4819
500	6719	8141	7152
2500	9916	9992	9949

Ve všech třech experimentech se pro zvolené nastavení projevil lamarckistický přístup jako úspěšnější. Dokázal nalézt bezchybný model v průměru dřív než baldwinistický přístup a měl většinou větší úspěšnost při jednotlivých omezeních v podobě maximálního počtu generací. Zvýšení počtu kroků stochastického horolezeckého algoritmu přineslo téměř neznatelné zlepšení v obou přístupech.

Na základě těchto pokusů byl pro další etapu disertační práce vybrán lamarckistický přístup pro optimalizaci konstant při pokusech se symbolickou regresí.

6 MEMETICKÉ GP S ALGORITMEM BISON SEEKER

V rámci počátečního výzkumu během přípravy disertační práce byl vytvořen nový memetický algoritmus GP využívající optimalizační metodu pro ladění konstant u symbolické regrese. Memetické GP s algoritmem Bison Seeker (BSA-GP) je memetická varianta genetického programování, ve které dochází k učení jedinců pomocí hejnového algoritmu Bison Seeker (BSA) [68].

Algoritmus bison seeker je vícerozměrná spojitá optimalizační metoda, která prohledává stavový prostor napodobováním vzorců chování stáda bizonů. Při této metodě je populace bizonů rozdělena do dvou skupin:

- rojová skupina (swarming group),
- běžící skupina (running group).

Jedinci z rojové skupiny se pohybují ve směru váženého středu počítaného z pozice nejlepších rojových jedinců (elitní skupina). Délka kroku pohybu směrem k váženému středu je ovlivněna parametrem překročení (overstep). Jedinci z běžící skupiny se pohybují kolem nejlepších rojových bizonů a testují neprozkoumané body stavového prostoru. Pokud běžec najde lepší řešení než jedinec z rojové skupiny, je povýšen (zkopírován) do rojové skupiny a nejhorší jedinec z rojové je zcela odstraněn z populace.

Parametry algoritmu zahrnují:

- velikost celé populace bizonů,
- velikost rojové skupiny,
- velikost elitní rojové skupiny (obsahuje špičkové/nejlepší jedince rojové skupiny),
- parametr překročení (ovlivňuje pohyb roje směrem k váženému středu),
- hranice vyhledávacího prostoru v každé dimenzi (minimální a maximální souřadnice).

Bison Seeker Algorithm (BSA) je rozšířením Bison Algorithm a používá stejnou sadu parametrů. Autoři v něm k původnímu algoritmu přidali fázi hledání. Při kopírování úspěšných běžců se v dalších iteracích z běžců stanou hledači a prozkoumají slibné řešení. Během této fáze se chovají jako rojoví jedinci. Jejich vážený střed se vypočítá z pozic každého úspěšně povýšeného bizona. Délka fáze hledajícího je přímo úměrná počtu povýšených běžců. Pro tuto fázi autoři doporučují dočasné snížení parametru překročení (overstep). Po fázi hledání se běžci vrátí do své původní formace a algoritmus pokračuje s původním chováním skupiny rojové a běžící skupiny, dokud nejsou povýšeni další úspěšní běžci.

V publikaci [48] bylo cílem vyzkoušet BSA k optimalizaci celočíselných konstant pomocí lamarckistické přístupu. Pro experimenty byly vygenerovány datové sady pomocí dvou funkcí y_1 a y_2 v rovnicích (3) a (4):

$$y_1 = 2x^2 + 3x + 5 \quad \textbf{(3)}$$

$$y_2 = \sin(x) \quad \textbf{(4)}$$

Jako fitness byla zvolena hodnota kvadrátu chyb přepočtena na jeden příklad z datové sady. Pro generování náhodných čísel byl využit algoritmus Mersenne Twister. Příliš hluboké stromy byly za běhu algoritmu GP prořezávány pomocí techniky pruning. V každém experimentu bylo provedeno vždy 1000 pokusů (běhů GP).

Pro první funkci y_1 bylo ekvidistantně vygenerováno do datové sady 15 bodů z intervalu od -5 do 5 včetně. Nastavení dalších parametrů zobrazují Tabulka 4 a Tabulka 5.

Tabulka 4 – Parametry pro GP pracující s první funkcí y_1

Parametr	**Hodnota**
Velikost populace	100
Elitismus	1
Pravděpodobnost mutace	0,01
Inicializační metoda	růstová metoda
Inicializační hloubka stromu	3
Cílová fitness	0,0
Generační limit	500
Délka stromu pro pruning	30
Selekční metoda	deterministická turnajová selekce
Velikost turnaje	2
Množina terminálů	x; 1; 2; 3; náhodné celé číslo od 1 do 10
Množina neterminálů	*; +; –; % (chráněné dělení)

Tabulka 5 – Parametry pro vnitřní algoritmus BSA pracující s první funkcí

Parametr	**Hodnota**
Velikost bizoního stáda	5 a 15
Velikost rojové skupiny	4 a 12
Velikost elitní rojové skupiny	2 a 6
Překročení	2,0
Velikost sousedství	2
Minimální souřadnice (všechny dimenze)	-5
Maximální souřadnice (všechny dimenze)	5
Počet iterací	3

Pro druhou funkci bylo do datové sady ekvidistantně vygenerováno 20 bodů z intervalu od 1 do 15 včetně. Zbytek parametrů zobrazuje Tabulka 6 a Tabulka 7.

Tabulka 6 – Parametry pro GP pracující s první funkcí y_1

Parametr	**Hodnota**
Velikost populace	100
Elitismus	1
Pravděpodobnost mutace	0,01
Inicializační metoda	růstová metoda
Inicializační hloubka stromu	6
Cílová fitness	0,1
Generační limit	1000
Délka stromu pro pruning	65
Selekční metoda	deterministická turnajová selekce
Velikost turnaje	2
Množina terminálů	x; 1,0; 2,0; 3,0, náhodné celé číslo od 1 do 10
Množina neterminálů	*; +; –; % (chráněné dělení)

Tabulka 7 – Parametry pro vnitřní algoritmus BSA pracující s první funkcí

Parametr	**Hodnota**
Velikost bizoního stáda	5
Velikost rojové skupiny	4
Velikost elitní rojové skupiny	2
Překročení	2,0
Velikost sousedství	2
Minimální souřadnice (všechny dimenze)	-5
Maximální souřadnice (všechny dimenze)	5
Počet iterací	1 a 3

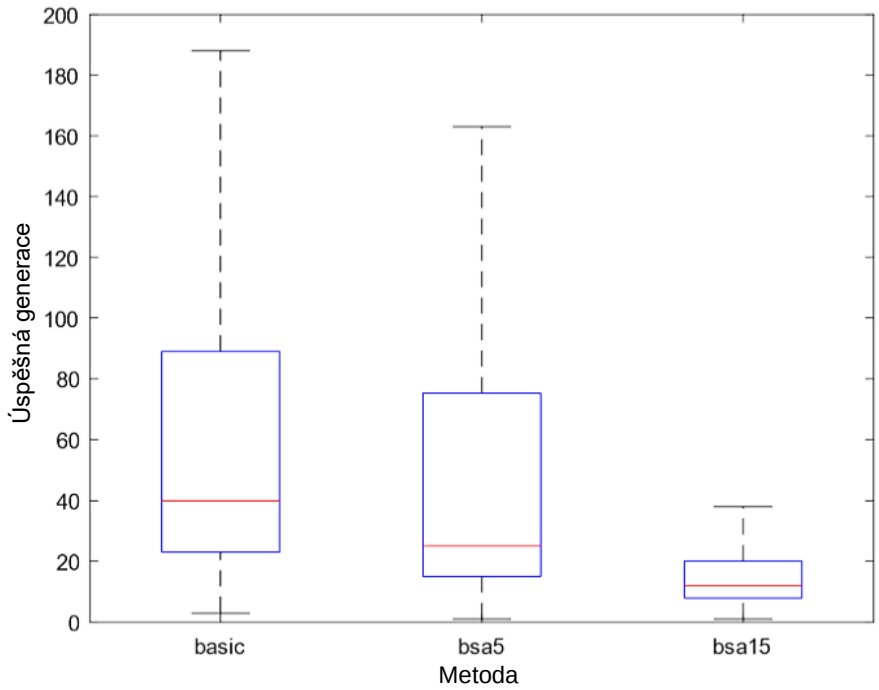
V první sérii experimentů se porovnávalo několik verzí symbolické regrese:

- základní verze SR,
- hybridní symbolická regrese 3 iteracemi BSA s 5 bizony,
- hybridní verze SR s 15 kroky BSA s 5 bizony.

Hlavním kritériem byla úspěšnost konvergence a jak brzy (v jaké generaci) k ní došlo. Výsledky experimentů s první funkcí zobrazují Tabulka 8 a Obrázek 4:

Tabulka 8 – Úspěšnost experimentů s první funkcí y_1

Experiment	**Úspěšnost v procentech**
Základní verze symbolické regrese (basic)	96,5 %
Hybridní verze s BSA (5 bizonů)	98,2 %
Hybridní verze s BSA (15 bizonů)	100 %

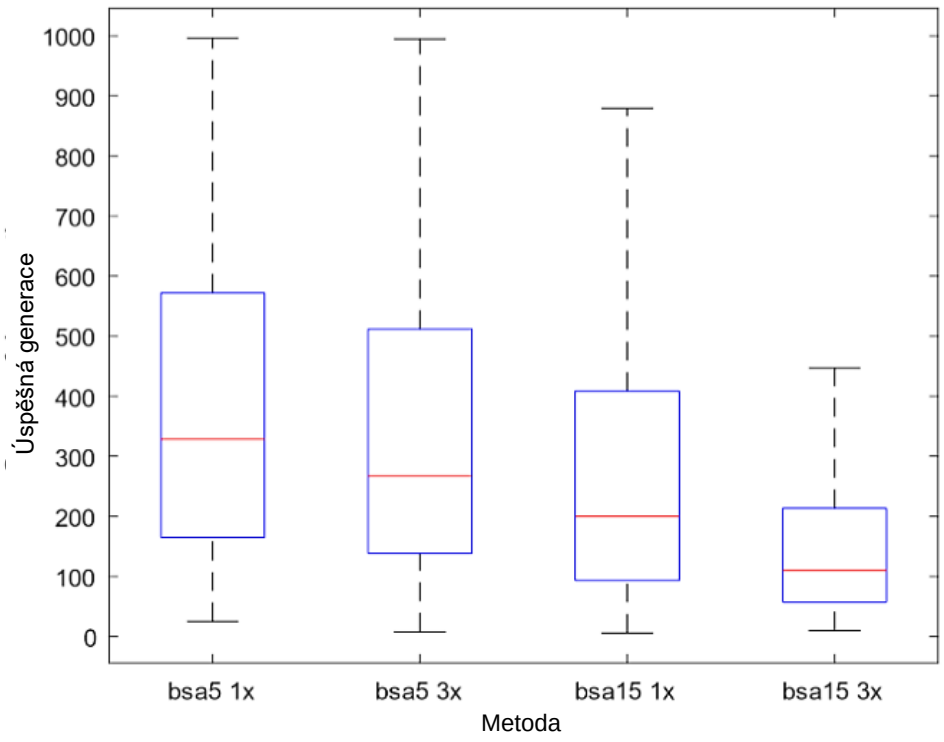


Obrázek 4 – Porovnání úspěšnosti pro 1. funkci pomocí jednotlivých metod, zdroj: autor

V druhé sérii experimentů se porovnávala základní verze SR a hybridní verze SR pomocí BSA s 1 a 3 iteracemi a s 3 a 5 bizony. Výsledky experimentů s první funkcí zobrazují Tabulka 9 a Obrázek 5.

Tabulka 9 – Úspěšnost experimentů s druhou funkcí y_2

Experiment	**Úspěšnost v procentech**
Základní verze symbolické regrese (basic)	0 %
Hybridní verze s BSA (5 bizonů, 1 iterace)	25,2 %
Hybridní verze s BSA (5 bizonů, 3 iterace)	69,1 %
Hybridní verze s BSA (15 bizonů, 1 iterace)	87 %
Hybridní verze s BSA (15 bizonů, 3 iterace)	95,7 %



Obrázek 5 – Porovnání úspěšnosti pro 2. funkci pomocí jednotlivých metod, zdroj: autor

Výsledky experimentů podporují tezi, že hybridizace GP může za určitých podmínek zvýšit úspěšnost evoluce modelu a urychlit ji. Pro takto nastavené experimenty se úspěšnost nalezení modelu pro datové body vygenerované první funkcí zvýšila. BSA se podařilo snížit počet potřebných generací k nalezení úspěšného modelu. Zvýšení počtu bizonů z 5 na 15 mělo pozitivní vliv na rychlost evoluce.

Hledání modelu pro datové body vygenerované funkcí sinus pomocí základní verze SR nebylo úspěšné v ani jednom z 1000 pokusů. Použití BSA pro ladění konstant pomohlo nalézt řešení a to dokonce pouze při jedné iteraci. Větší počet iterací a bizonů zvýšil rychlost evoluce, kvalitu modelů a konzistenci algoritmu GP.

Obrázek 6 – Umělecké ztvárnění BSA-GP

7 DVOUVRSTVÝ ALGORITMUS GENETICKÉHO PROGRAMOVÁNÍ

Navržený dvouvrstvý algoritmus genetického programování je rozdělený do dvou fází. Lze si jej představit jako dva algoritmy genetického programování, které na sebe bezprostředně navazují:

1. V první fázi se pomocí genetického programování vytváří dílčí modely (submodely) ze základní množiny terminálů a funkcí.
2. Ve druhé fázi již dochází ke skládání výsledného modelu (metamodelu) z dílčích modelů vytvořených v první fázi.

Celý dvouvrstvý algoritmus GP si lze představit jako tvorbu základních stavebních bloků a následné hledání jejich vhodné kombinace. V rámci biologie to může připomínat nejdříve evoluce jednodušších specializovaných organismů a následně o vývoj složitějších organismů těžících ze vzájemné symbiózy.

Celý dvouvrstvý algoritmus GP lze popsat v několika krocích:

1. Vytvoření dílčích modelů pomocí samostatných běhů klasického GP.
2. Přidání dílčích modelů do množiny terminálů[6] pro druhou vrstvu dvouvrstvého GP.
3. Jeden běh druhé vrstvy pomocí klasického GP s dílčími modely.
4. Navrácení výsledku z druhé vrstvy.

Počet běhů první vrstvy je závislý na počtu generovaných dílčích modelů. Druhá vrstva má samostatné nastavení parametrů. Výsledný model je produktem běhu druhé vrstvy.

7.1 Motivace

Návrh dvouvrstvého algoritmu GP vychází z několika různých motivací:

1. Snaha zabránit bloatu a tvorbě stále zvětšujících se stromů bez zlepšování fitness funkce (snižování chyby modelu) pomocí restartu GP, ve kterém aktuálně vytvořené stromy budou použity jako základní stavební bloky pro nový běh GP, kde bude kód vytvořených stromů využit novým konstruktivním způsobem.
2. Možnost změnit nastavení GP během evoluce, jelikož různé složitostí stromů (modelů) mohou vyžadovat jiné operace a nastavení.
3. Vytvářet robustnější výsledné modely za pomocí technik sborového učení.

Ve chvíli, kdy nastane bloat, přidávaný kód na konec stromu nemá žádný produktivní význam. Start navazujícího GP, může využít stávající kód novým způsobem a začít dělat změny viditelné změny generovaných programů, kde bude kód vytvořených stromů využit novým konstruktivním způsobem.

[6] Vytvořené dílčí modely nepřijímají žádné parametry, proto jsou v druhé vrstvě vedeny jako terminály.

Sborový charakter navrženého dvouvrstvého algoritmu GP je založen na paralelní topologii dílčích modelů z první vrstvy s trénovatelným slučovačem na druhé vrstvě. Jednotlivé dílčí modely jsou vytvářeny pomocí samostatných běhů první vrstvy GP.

7.2 Rozdělení vrstev

Obě vrstvy (fáze) mohou mít odlišné nastavení parametrů a hyperparametrů GP, včetně počtu generací, velikosti populace, množiny terminálů a množiny funkcí. Obě vrstvy je také možné nezávisle memeticky optimalizovat pomocí BSA.

7.2.1 První vrstva algoritmu

Pro tvorbu dílčích modelů na první vrstvě byly zvoleny postupy, které diverzifikují množinu trénovacích dat:

- sampling,
- bootstrapping,
- metoda náhodného pohyblivého okna,

které celou množinu dat, rozdělily na různé podmnožiny pro jednotlivé dílčí modely.

Pro generování dílčích modelů byla implementována i možnost nediverzifikovat množinu trénovacích dat a trénovat všechny dílčí modely na stejné úplné datové sadě.

Sampling

V metodě sampling byly podmnožiny trénovacích dat pro dílčí modely tvořeny tak, že u každého datového bodu z celé množiny trénovacích dat byla pravděpodobnost 50 %, že se dostane do konkrétní trénovací podmnožiny pro dílčí model. Trénovací množiny dílčích modelů tak obsahovaly průměrně polovinu trénovacích dat.

Bootstrapping

V metodě bootstraping se volila velikost trénovací množiny pro dílčí modely (velikost bootstrapu) a následně se přidávaly náhodné body z celé množiny trénovacích dat. Vlastní implementace této metody přijímá jako parametr velikost trénovací množiny jako procento velikosti původní celé trénovací množiny, například velikost množiny bude 30 % celé množiny trénovacích dat. V použité implementaci metody bootstrapping bylo možné, že se jeden bod vyskytne v trénovací množině pro dílčí model víckrát.

Metoda náhodného pohyblivého okna

V této metodě se na rozdíl od předchozích dvou generují podmnožiny z celé množiny trénovacích dat souvisle s náhodným prvním (začátkem okna) a pevným počtem bodů (délkou okna). V případě, že je začátek okna vygenerován tak, že už nezbývá příliš mnoho bodů do konce trénovací množiny, je tvořená podmnožina menší a body se nijak nedoplňují.

Algoritmus lze spustit i bez vytváření diverzifikovaných podmnožin trénovacích dat, jelikož samotná koncepce by mohla mít potenciál pomoci s bloatem. Dále pokud bude množina

terminálů a funkcí v první vrstvě dostatečně velká a různorodá, může dojít samovolně k diverzifikaci dílčích modelů podle použitých funkcí[7].

7.2.2 Druhá vrstva

Podle výběru funkcí v druhé fázi může jít o:

- jednoduché lineární skládání
- nebo nelineární skládání dílčích modelů.

Pokud se dílčí modely sčítají (případně odčítají), násobí konstantou nebo skládají aritmetickým nebo váženým průměrem, půjde o lineární skládání dílčích modelů. Výsledný model je lineární kombinací dílčích modelů z první vrstvy.

Dílčí modely se mohou ve druhé fázi skládat i nelineárně, pokud budou použity funkce, které umožní například násobení nebo dělení submodelů mezi sebou. V takovém případě je výsledný model nelineární kombinací dílčích modelů a nalezení správné kombinace je úlohou nelineární optimalizace.

7.2.3 Implementace

Algoritmus dvouvrstvého genetického programování byl implementován v jazyce Java nad frameworkem Watchmaker[8]. Do frameworku bylo nutné přidat nejprve vlastní implementaci genetického programování a poté jeho dvouvrstvou variantu. Pro generování náhodných čísel byla využita standardní třída Random z balíčku java.util.

[7] Do budoucna by bylo vhodné zkusit přidat cílenou diverzifikaci dílčích modelů používáním různých podmnožin množiny funkcí a terminálů.
[8] Dostupný na adrese: https://watchmaker.uncommons.org/

8 POROVNÁNÍ JEDNOVRSTVÉHO GP A DVOUVRSTVÉHO GP

Pro otestování chování navrženého dvouvrstvého algoritmu byla provedena sada experimentů na několika testovacích funkcí. Všechny experimenty probíhaly pouze s celočíselnými konstantami. Pro každou funkci se měnila nastavení zejména počtu generací, velikosti populace, množiny terminálů a funkcí, výběr metody tvorby dílčích modelů na první vrstvě a počet generovaných dílčích modelů.

Následující parametry byly pro všechny experimenty na všech datových sadách stejné, jsou tedy uvedeny zde na tomto místě, aby se informace o jejich nastavení nemusela uvádět opakovaně. Pro jednovrstvý algoritmus jsou neměnné hodnoty zobrazuje Tabulka 10.

Tabulka 10 – Neměnné nastavení jednovrstvého GP

Parametr	**Hodnota**
Elitismus	1
Pravděpodobnost mutace	0,01
Inicializační metoda	růstová metoda
Inicializační hloubka stromu	3
Cílová fitness jednovrstvé GP	0,0005
Délka stromu pro pruning	65
Selekční metoda	deterministická turnajová selekce
Velikost turnaje	2

Neměnné hodnoty pro dvouvrstvý algoritmus zobrazuje Tabulka 11. Parametry platí pro první i druhou vrstvu, pokud nejsou uvedeny explicitně zvlášť.

Tabulka 11 – Neměnné nastavení dvouvrstvého GP

Parametr	**Hodnota**
Elitismus	1
Pravděpodobnost mutace 1. vrstvy	0,01
Pravděpodobnost mutace 2. vrstvy	0,1
Inicializační metoda	růstová metoda
Inicializační hloubka stromu	3
Cílová fitness 1. vrstvy	0,005
Cílová fitness 2. vrstvy	0,0
Délka stromu pro pruning	65
Selekční metoda	deterministická turnajová selekce
Velikost turnaje	2

Zbývající parametry budou popsány u konkrétních experimentů. Všechny experimenty provedené v této kapitole byly prováděny ve 100 opakováních. Pro výpočet funkce fitness pro ohodnocení kvality výsledných matematických modelů byla používána zpravidla střední kvadratická chyba modelu oproti trénovací množině dat.

8.1 Datové sady

Pro začátek byly zvoleny experimenty, které budou porovnávat základní jednovrstvé GP a dvouvrstvé GP na úlohách symbolické regrese. Pro tento účel bylo zvoleno několik testovacích funkcí a datových sad, konkrétně:

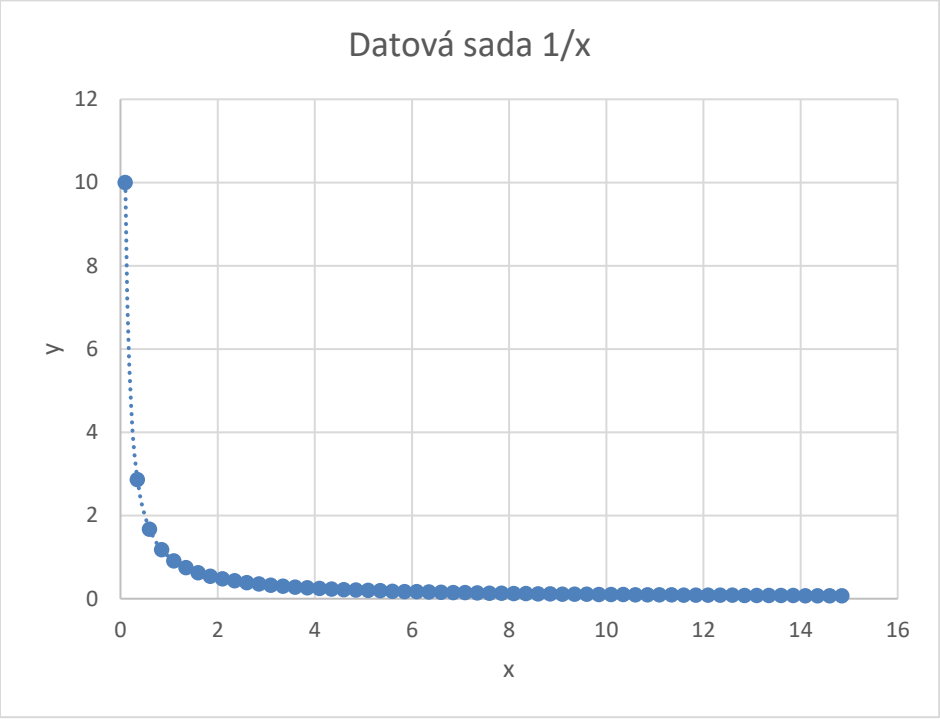
- hyperbola podle rovnice (5), 60 hodnot s $x \in \langle 0{,}1; 14{,}85 \rangle$,
- jednoduchá funkce sinus podle rovnice (6), 188 hodnot s $x \in \langle 0{,}03; 5{,}64 \rangle$,
- a funkce složená ze tří různých funkcí sinus (7), 120 hodnot s $x \in \langle -4; 2 \rangle$,
- a zkrácená datová sada (prvních 399 datových bodů) naměřených teplot z Indického města Dillí[9].

$$y_1 = \frac{1}{x} \tag{5}$$

$$y_2 = \sin(x) \tag{6}$$

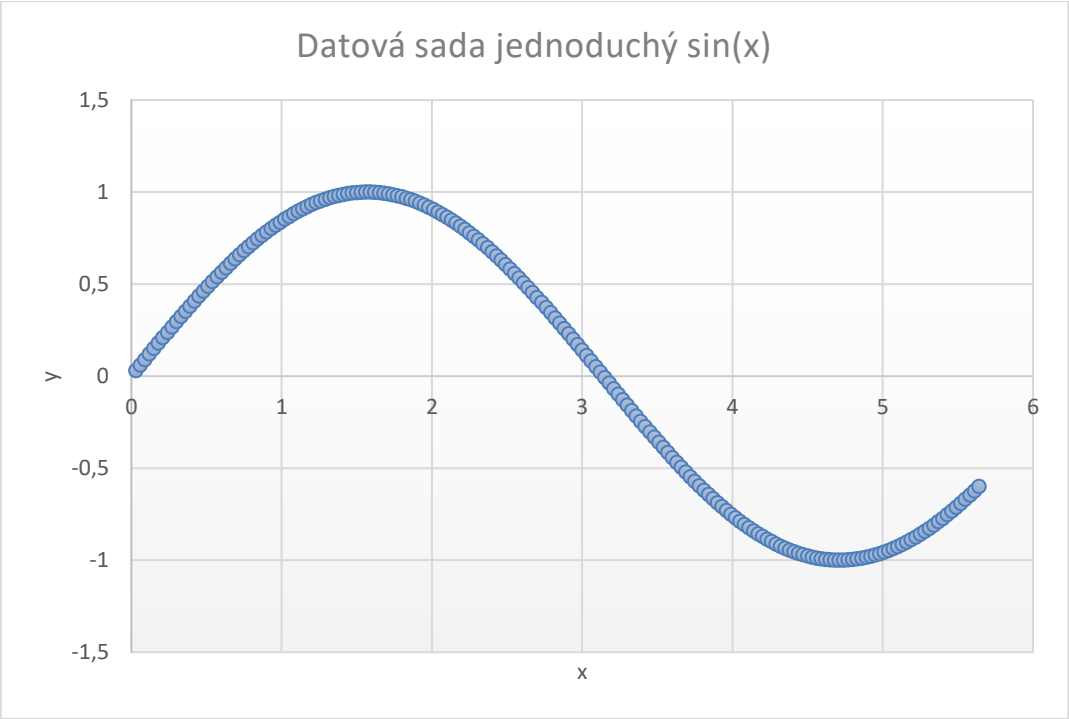
$$y_3 = \sin(x) + \sin(2x) + \sin(3x + 5) \tag{7}$$

Uměle vytvořené hodnoty testovaných funkcí byly generovány ekvidistantně po celém rozsahu příslušných intervalů. Datové sady zobrazují následující grafy: hyperbola (Obrázek 7), funkce sinus (Obrázek 8), funkce složený sinus (Obrázek 9) a teplota ve městě Dillí (Obrázek 10).

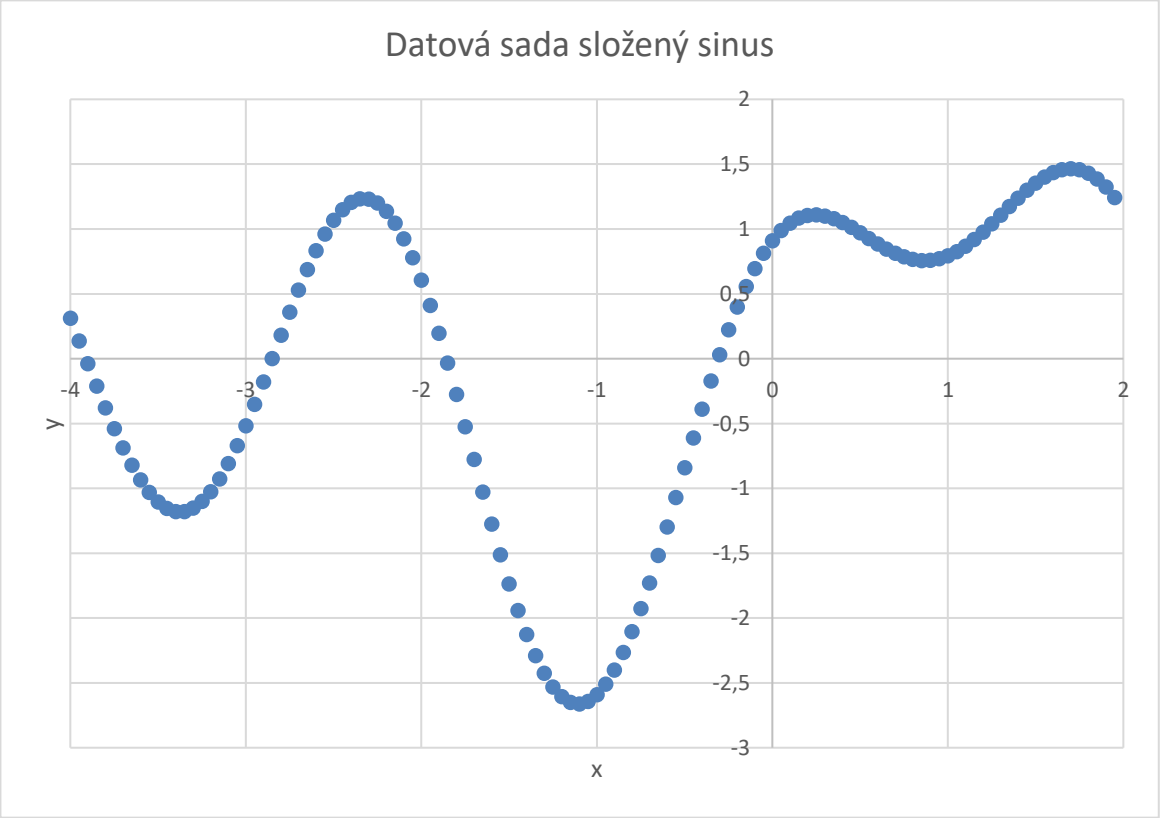


Obrázek 7 – Datová sada funkce hyperbola

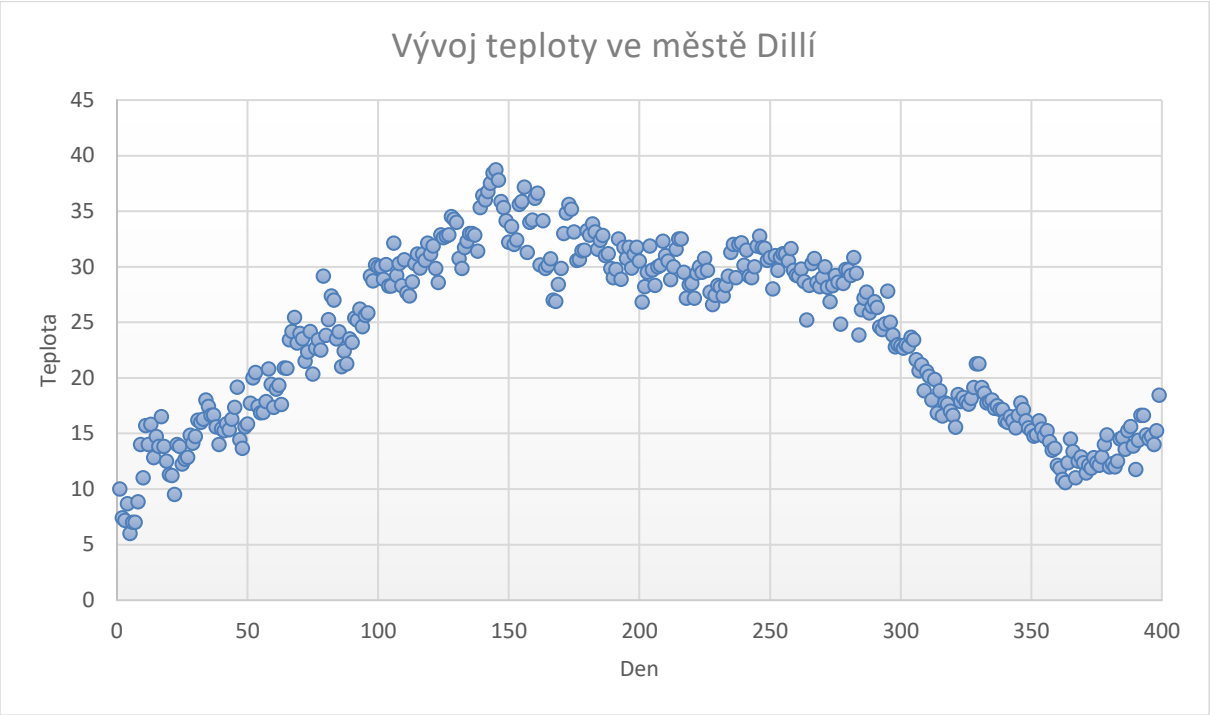
[9] Celá datová sada je dostupná na adrese https://www.kaggle.com/datasets/sumanthvrao/daily-climate-time-series-data?resource=download&select=DailyDelhiClimateTest.csv



Obrázek 8 – Datová sada funkce sinus



Obrázek 9 – Datová sada funkce složený sinus



Obrázek 10 – Použitá datová sada s vývojem teploty v Indickém městě Dillí

8.2 Hyperbola

Následující experimenty byly prováděny na datové sadě vygenerované funkcí z rovnice (5). Základní nastavení parametrů jednovrstvého a dvouvrstvého GP zobrazují Tabulka 12 a Tabulka 13.

Tabulka 12 – Parametry pro jednovrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Velikost populace	50
Generační limit	230
Množina terminálů	*x*; –1,0; 1,0; 2,0; 3,0
Množina neterminálů	*; +; –;

Tabulka 13 – Parametry pro dvouvrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Počet dílčích modelů	30
Velikost populace 1. vrstva	50
Generační limit 1. vrstva	200
Množina terminálů 1. vrstva	*x*; –1,0; 1,0; 2,0; 3,0
Množina neterminálů 1. vrstva	*; +; –;
Velikost populace 2. vrstva	50
Generační limit 2. vrstva	30
Základní množina terminálů 2. vrstva	x; –1,0; 1,0; 2,0; 3,0
Množina neterminálů 2. vrstva	*; +; –; avg

Neterminál *avg* je funkce průměr, která přebírá jako vstup dva parametry a vypočítá z nich aritmetický průměr.

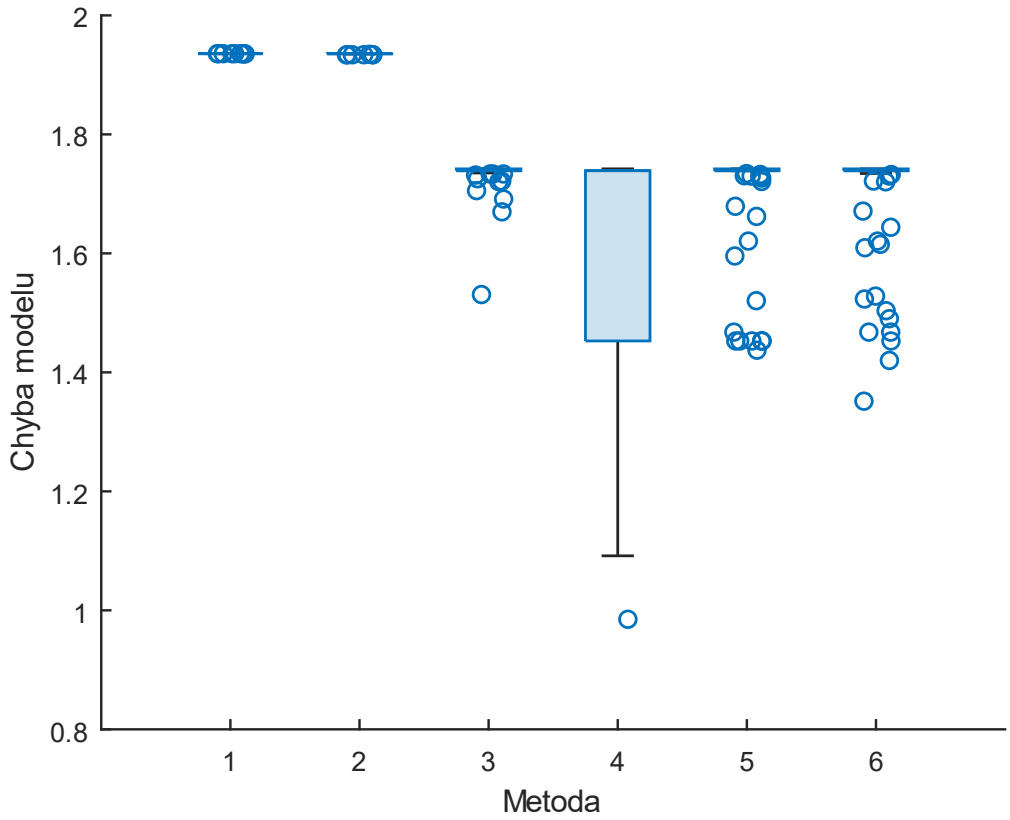
8.2.1 Experimenty s omezenou sadou funkcí

Nejprve byly prováděny experimenty se základní množinou funkcí, které popisují Tabulka 12 a Tabulka 13. Cílem bylo porovnat jednovrstvé GP s dvouvrstvou variantou. Pro lepší srovnání byl přidán experiment s jednovrstvým GP s 1500 jedinci (více se počtem jedinců blíží dvouvrstvému GP, které vytváří 30 dílčích modelů, každý pomocí 50 jedinců).

Konkrétně šlo o očíslované experimenty:

1. jednovrstvé GP,
2. jednovrstvé GP s 1500 jedinci,
3. dvouvrstvé GP bez tvorby podmnožin trénovacích dat,
4. dvouvrstvé GP s metodou bootstrapping (10 %),
5. dvouvrstvé GP s metodou bootstrapping (30 %),
6. dvouvrstvé GP s metodou bootstrapping (60 %).

Výsledky experimentů zobrazuje následující graf (Obrázek 11).

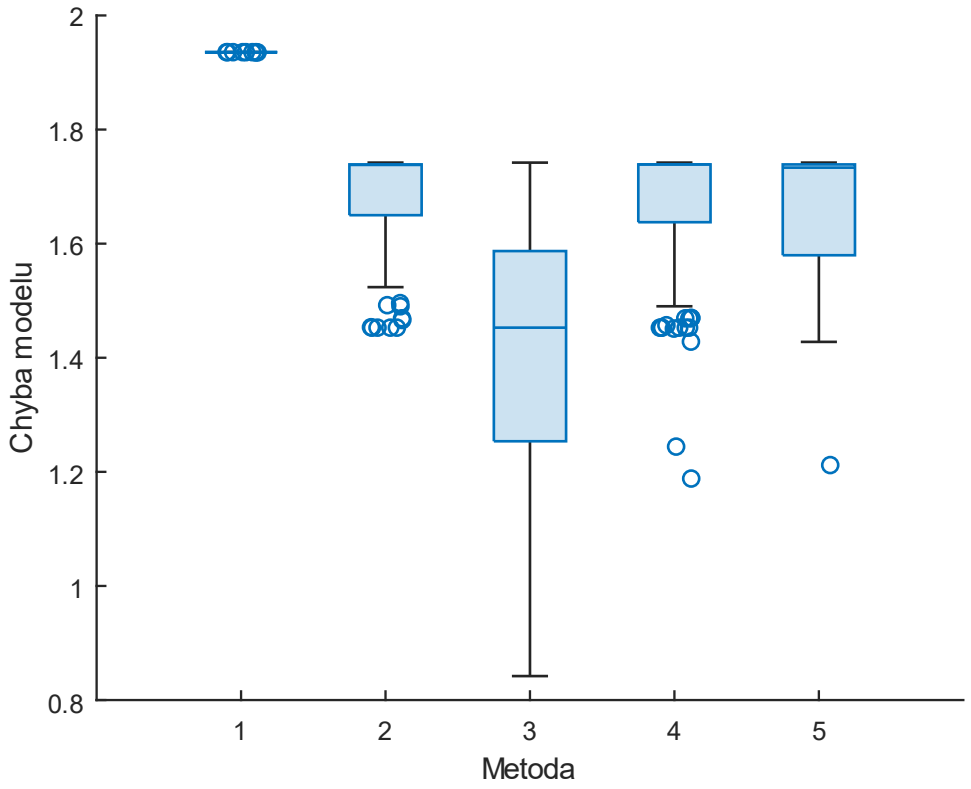


Obrázek 11 – Porovnání jednovrstvého GP s dvouvrstvým GP na funkci hyperbola s omezenou sadou funkcí

První vrstva vygenerovala málo rozmanité modely a to vytvořilo i málo diverzifikované modely rozmanité modely na druhé vrstvě. Z tohoto důvodu byly experimenty zopakovány (kromě experimentu jednovrstvého GP s 1500 jedinci) s jedinou změnou, která spočívala v nastavení počtu jedinců na druhé vrstvě na 150 jedinců. Výsledky jsou vidět na následujícím grafu (Obrázek 12).

Konkrétně šlo o očíslované experimenty:

1. jednovrstvé GP, 150 jedinců na druhé vrstvě,
2. dvouvrstvé GP bez tvorby podmnožin trénovacích dat, 150 jedinců na druhé vrstvě,
3. dvouvrstvé GP s metodou bootstrapping (10 %), 150 jedinců na druhé vrstvě,
4. dvouvrstvé GP s metodou bootstrapping (30 %), 150 jedinců na druhé vrstvě,
5. dvouvrstvé GP s metodou bootstrapping (60 %), 150 jedinců na druhé vrstvě.



Obrázek 12 – Porovnání jednovrstvého GP s dvouvrstvým GP se 150 jedinci na druhé vrstvě na funkci hyperbola s omezenou sadou funkcí

Jednovrstvé GP pro toto nastavení vygenerovalo velmi podobné modely s téměř totožnou chybou. Toto chování je dáno zejména tím, že bylo k dispozici málo funkcí a tak nemělo GP mnoho možností, jak vytvořit výsledný model.

Chování dvouvrstvého GP s metodou bootstrapping trochu odporuje intuici. Bootstrapping s nejmenší velikostí podmnožiny trénovacích dat vykazuje nejnižší průměrnou chybu. Malá (prakticky žádná) rozmanitost dílčích modelů generovaných první vrstvou nedává druhé vrstvě příliš mnoho možností ke zlepšení. Paradoxně má tak lepší výsledky bootstrapping

s nejmenší velikostí, protože dokáže na první vrstvě generovat více diverzifikované modely a tak dojde k většímu zlepšení.

Pro zlepšení situace byly přidány další funkce do množiny funkcí s ideou, že více funkcí by mohlo zlepšit rozmanitost modelů generovaných první vrstvou nehledě na velikost podmnožiny trénovacích dat u metody bootstrapping. Z hlediska nekonstruktivního chování druhé vrstvy s malým počtem funkcí nebyly pro toto nastavení zkoumány další metody.

8.2.2 Přidání funkcí sqrt a sin

V dalších experimentech došlo k přidání funkcí *sqrt* (odmocnina) a *sin* (sinus) do množiny funkcí (u dvouvrstvého GP byly tyto funkce přidány na první i druhou vrstvu). Nastavení experimentů popisují Tabulka 14 a Tabulka 15.

Tabulka 14 – Parametry pro jednovrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Velikost populace	50
Generační limit	230
Množina terminálů	x; –1,0; 1,0; 2,0; 3,0
Množina neterminálů	*; +; –; sqrt, sin

Tabulka 15 – Parametry pro dvouvrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Počet dílčích modelů	30
Velikost populace 1. vrstva	50
Generační limit 1. vrstva	200
Množina terminálů 1. vrstva	x; –1,0; 1,0; 2,0; 3,0
Množina neterminálů 1. vrstva	*; +; –;
Velikost populace 2. vrstva	50
Generační limit 2. vrstva	30
Základní množina terminálů 2. vrstva	x; –1,0; 1,0; 2,0; 3,0
Množina neterminálů 2. vrstva	*; +; –; sqrt, sin, avg

U dvouvrstvého GP byly vyzkoušeny různé přístupy:

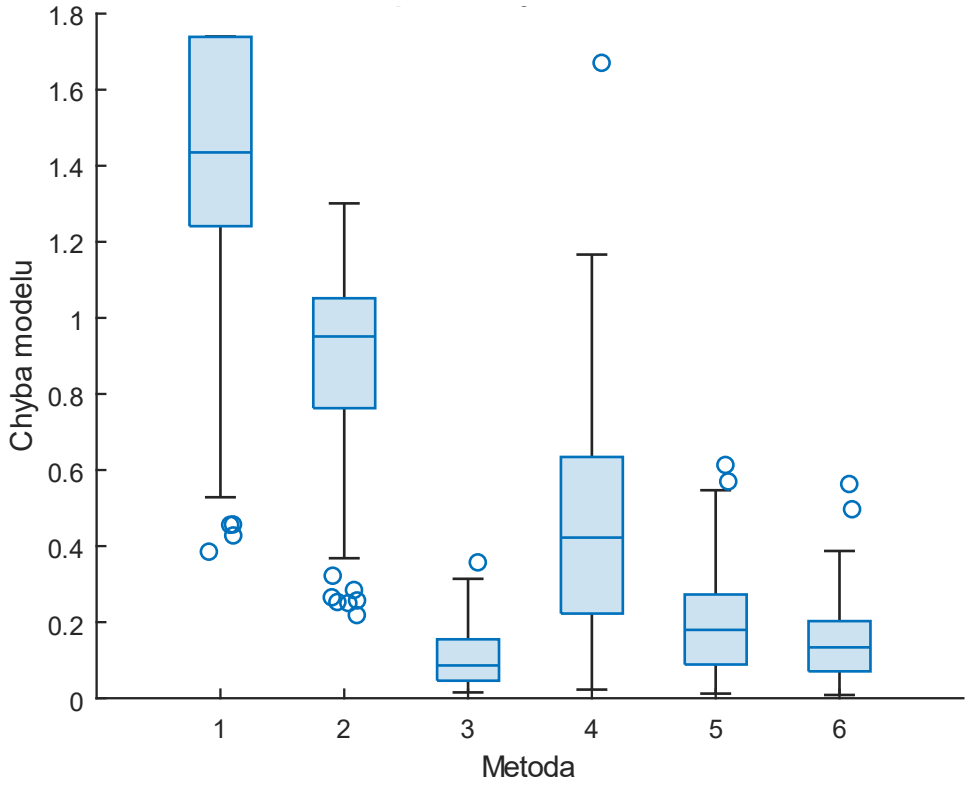
- prosté dvouvrstvé GP bez diverzifikace podmnožin trénovacích dat,
- dvouvrstvé GP s metodou bootstrapping,
- dvouvrstvé GP s metodou sampling,
- dvouvrstvé GP s metodou náhodného pohyblivého okna.

Metoda bootstrapping

S metodou bootstraping byly provedeny následující experimenty:

1. jednovrstvé GP + sqrt, sin,
2. jednovrstvé GP s 1500 jedinci + sqrt, sin,
3. dvouvrstvé GP bez tvorby podmnožin trénovacích dat + sqrt, sin,
4. dvouvrstvé GP s metodou bootstrapping (10 %) + sqrt, sin,
5. dvouvrstvé GP s metodou bootstrapping (30 %) + sqrt, sin,
6. dvouvrstvé GP s metodou bootstrapping (60 %) + sqrt, sin,

U všech experimentů s dvouvrstvým GP se na první vrstvě generovalo 30 dílčích modelů.

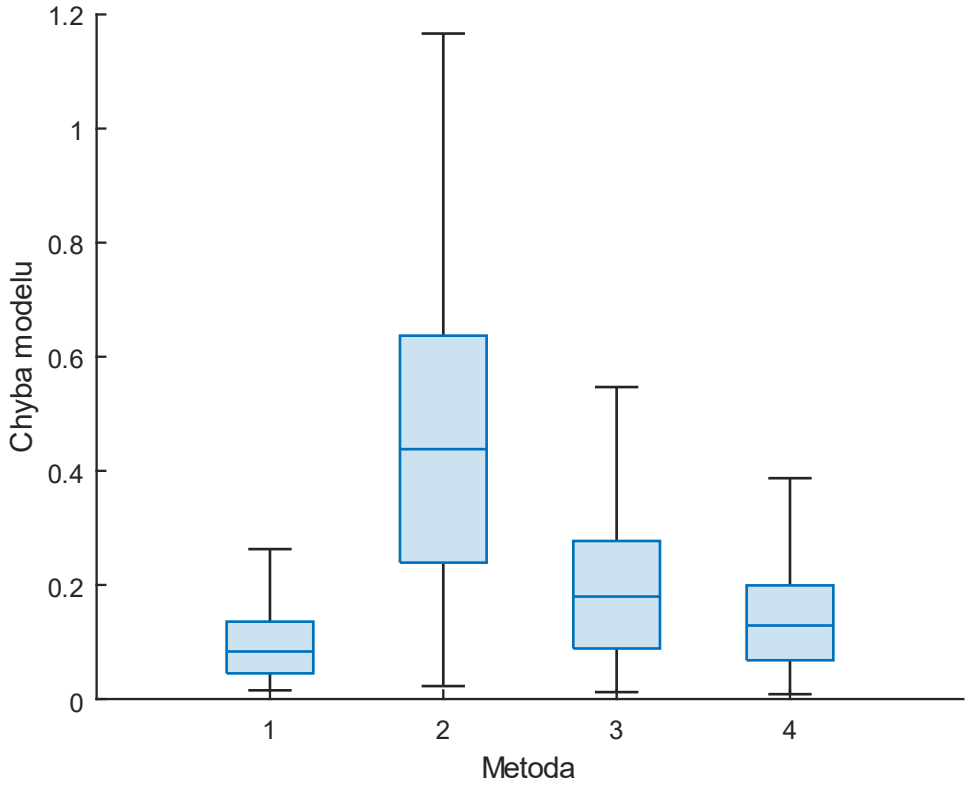


Obrázek 13 – Porovnání jednovrstvého GP s dvouvrstvým GP po přidání funkcí sqrt a sin

Z grafu (Obrázek 13) je patrné, že po přidání funkce odmocnina a funkce sinus generuje pomocí jednovrstvého GP rozmanitější diverzifikované dílčí modely. I průměrná chyba po přidání funkcí se snížila. U jednovrstvého GP s 1500 jedinci došlo ke znatelnému zlepšení (díky dodatečným funkcím má počet jedinců konečně nějaký vliv). Přidání funkcí se projevuje pozitivně i u dvouvrstvého GP. V tomto případě se již přesnost se zvětšováním podmnožiny trénovacích dat u metody bootstrapping projevuje snižující se chybou a zvyšující se přesností výsledných modelů. V tomto případě je rozdíl mezi jednovrstvým GP s 1500 jedinci a dvouvrstvým GP znatelný ve prospěch dvouvrstvého GP.

Následuje graf srovnání pouze prostého dvouvrstvého GP a různých nastavení bootstrappingu bez zobrazení odlehlých hodnot pro detailnější srovnání jednotlivých metod. Konkrétně jde o experimenty:

1. dvouvrstvé GP bez tvorby podmnožin trénovacích dat,
2. dvouvrstvé GP s metodou bootstrapping (10 %) + sqrt, sin,
3. dvouvrstvé GP s metodou bootstrapping (30 %) + sqrt, sin,
4. dvouvrstvé GP s metodou bootstrapping (60 %) + sqrt, sin.



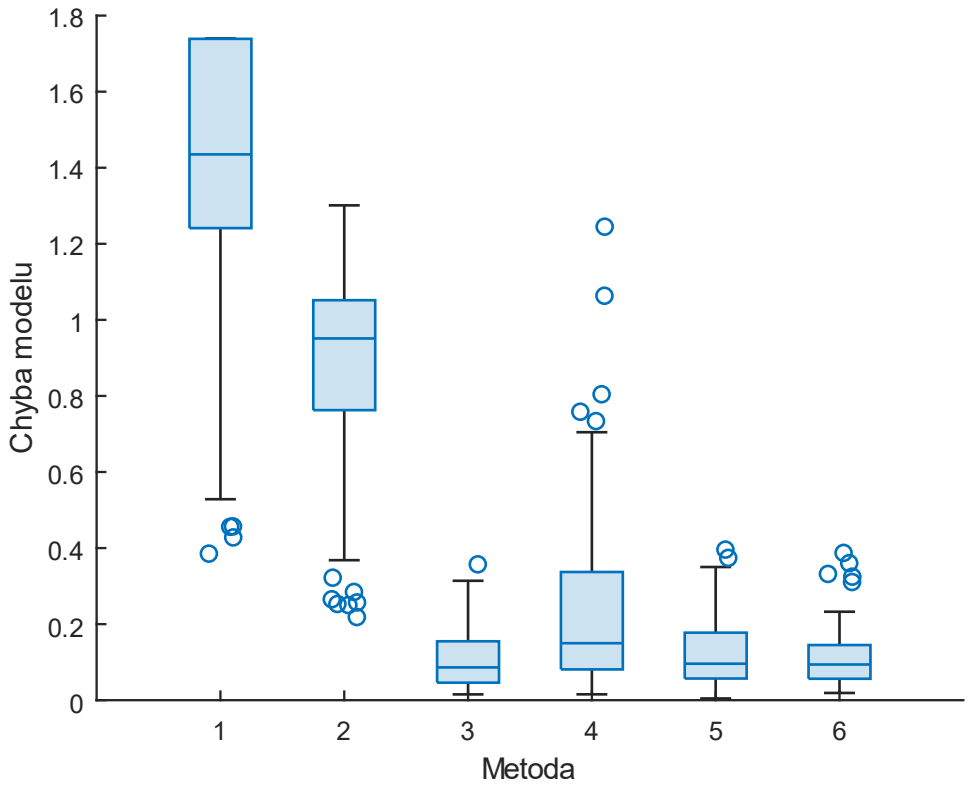
Obrázek 14 – Porovnání prostého dvouvrstvého GP a dvouvrstvého GP s metodou bootstrapping po přidání funkcí sqrt a sin

Z grafu (Obrázek 14) lze vyčíst, že zvětšování velikosti bootstrapu se projevilo zvětšenou přesností výsledných modelů. Zajímavé výsledky má i prostá dvouvrstvá metoda bez tvorby podmnožin trénovacích dat. Její výsledky v tomto konkrétním případě překonávají dvouvrstvé GP s metodou bootstrapping.

Metoda sampling

S metodou sampling byly provedeny experimenty se stejnými základními parametry jako s metodou bootstrapping. Jelikož nedocházelo ke změnám pravděpodobnosti výskytu datového bodu v podmnožině trénovacích dat pro dílčí model, byly vyzkoušeny různé počty vytvářených dílčích modelů na první vrstvě. Konkrétně šlo o experimenty:

1. jednovrstvé GP + sqrt, sin,
2. jednovrstvé GP s 1500 jedinci + sqrt, sin,
3. dvouvrstvé GP bez tvorby podmnožin trénovacích dat + sqrt, sin,
4. dvouvrstvé GP sampling + sqrt, sin, 10 dílčích modelů,
5. dvouvrstvé GP sampling + sqrt, sin, 30 dílčích modelů,
6. dvouvrstvé GP sampling + sqrt, sin, 60 dílčích modelů.

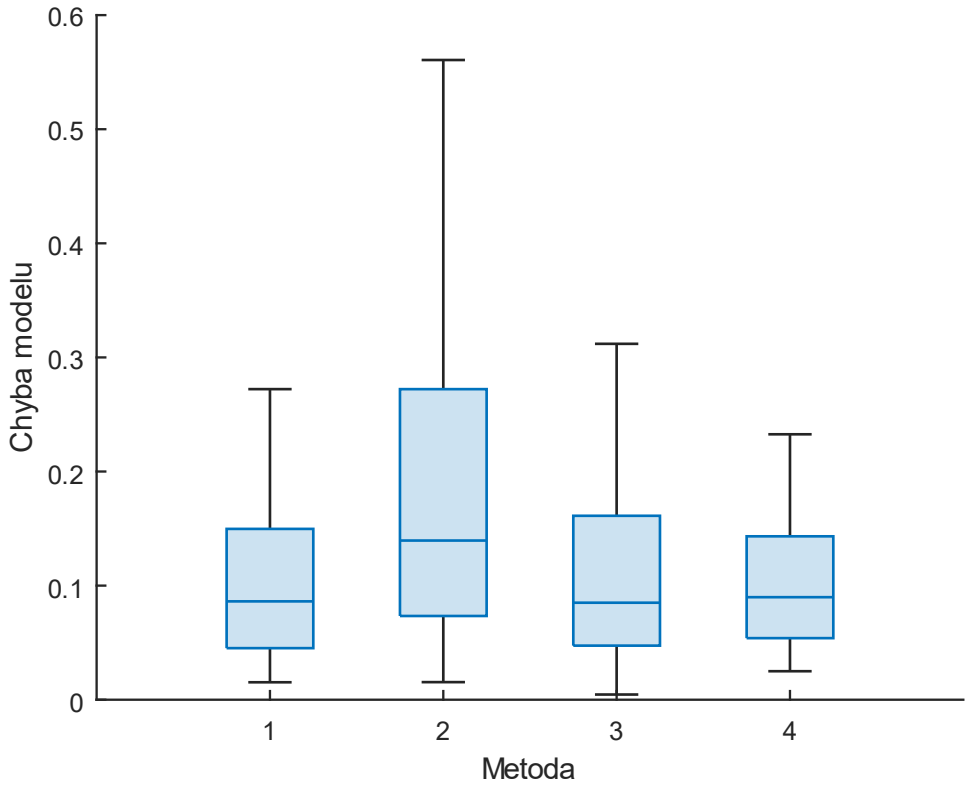


Obrázek 15 – Porovnání jednovrstvého GP s dvouvrstvým GP s metodou sampling po přidání funkcí sqrt a sin

Z grafu (Obrázek 15) lze vyčíst, že metoda sampling v tomto případě (podobně jako metoda bootstrapping) vytvořila průměrně znatelně lepší modely než jednovrstvé GP a jednovrstvé GP s 1500 jedinci. Sampling s 30 a 60 dílčími modely se více blížil prostému dvouvrstvému GP.

Následuje graf (Obrázek 16) základní dvouvrstvé GP a různá nastavení metody sampling bez odlehlých hodnot, pro detailnější srovnání. Jde o experimenty:

1. dvouvrstvé GP bez tvorby podmnožin trénovacích dat + sqrt, sin
2. dvouvrstvé GP sampling, 10 dílčích modelů + sqrt, sin,
3. dvouvrstvé GP sampling, 30 dílčích modelů + sqrt, sin,
4. dvouvrstvé GP sampling, 60 dílčích modelů + sqrt, sin.



Obrázek 16 – Porovnání prostého dvouvrstvého GP s dvouvrstvým GP s metodou sampling po přidání funkcí sqrt a sin

Z grafu (Obrázek 16) lze vyčíst, že se zvyšováním počtu dílčích modelů došlo na této datové sadě ke zvětšení přesnosti generovaných výsledných modelů. Nejen, že se snižil medián chyby, ale také se zmenšilo mezikvartilové rozpětí. To znamená, že se zvyšujícím se počtem dílčích modelů se dvouvrstvá metoda stávala sama o sobě stále robustnější a přesnější. Přesto se ji však v přesnosti nedařilo znatelně překonávat prosté dvouvrstvé GP.

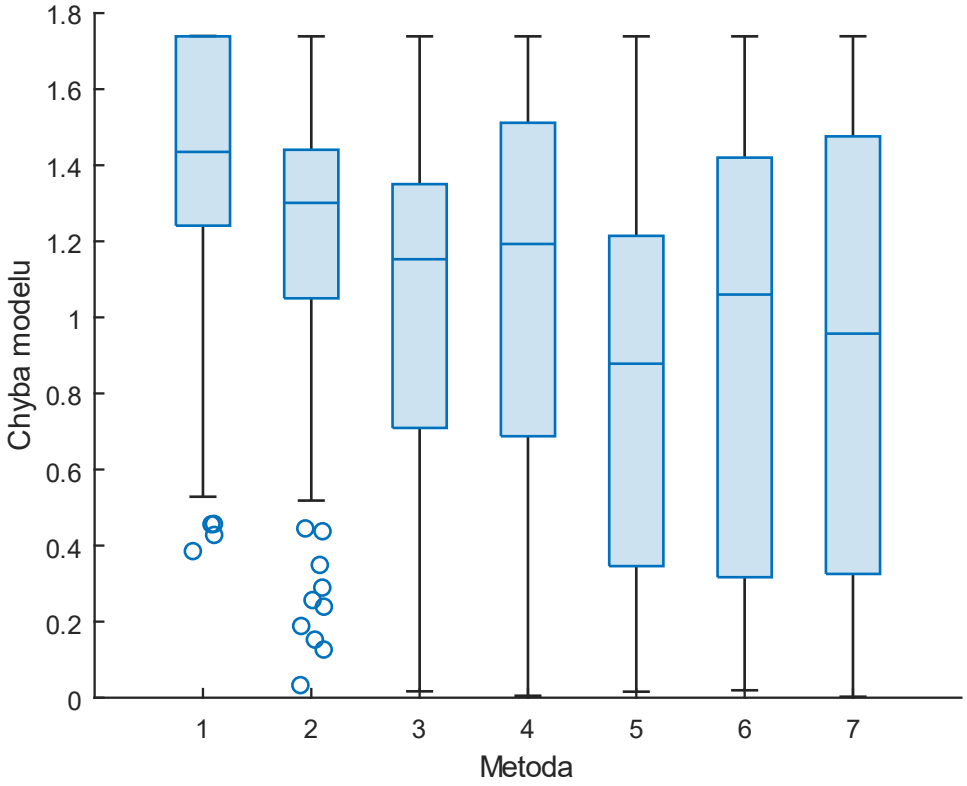
Metoda pohyblivého okna

Následující graf (Obrázek 17) zobrazuje výsledky experimentů s přidanými funkcemi (sqrt, sin) u dvouvrstvého GP s metodou pohyblivého okna a porovnání s jednovrstvým GP.

Konkrétně jde o experimenty:

1. jednovrstvé GP + sqrt, sin,
2. dvouvrstvé GP pohyblivé okno s 10 datovými body + sqrt, sin,

3. dvouvrstvé GP pohyblivé okno s 15 datovými body + sqrt, sin,
4. dvouvrstvé GP pohyblivé okno s 20 datovými body + sqrt, sin,
5. dvouvrstvé GP pohyblivé okno s 30 datovými body + sqrt, sin,
6. dvouvrstvé GP pohyblivé okno se 40 datovými body + sqrt, sin,
7. dvouvrstvé GP pohyblivé okno s 50 datovými body + sqrt, sin.



Obrázek 17 – Porovnání jednovrstvého GP s dvouvrstvým GP s metodou náhodného pohyblivého okna po přidání funkcí sqrt a sin

Z grafu (Obrázek 17) je dále patrné, že metoda pohyblivého okna v průměru generuje pro toto nastavení ve srovnání s jednovrstvým GP přesnější modely. Nicméně výsledky jsou znatelně horší než při použití prostého dvouvrstvého GP a dokonce i horší než u dvouvrstvého GP s metodou bootstrapping.

Při pohledu na širší mezikvartilová rozpětí generovala metoda pohyblivého okna v tomto případě celkem rozmanité výsledné modely různých kvalit. Přestože byla schopná někdy najít celkem kvalitní řešení, nebyla schopna si v tomto smyslu udržet nějakou konzistenci.

8.2.3 Přidání dalších funkcí pow2 a pow3

U dalších experimentů došlo k přidání dalších funkcí do množiny funkcí, konkrétně druhé mocniny *pow2* a třetí mocniny *pow3*.

Tabulka 16 – Parametry pro jednovrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Velikost populace	50
Generační limit	230
Množina terminálů	x; -1,0; 1,0; 2,0; 3,0
Množina neterminálů	*; +; –; sqrt, sin, pow2, pow3

Tabulka 17 – Parametry pro dvouvrstvé GP na funkci hyperbola

Parametr	**Hodnota**
Počet dílčích modelů	30
Velikost populace 1. vrstva	50
Generační limit 1. vrstva	200
Množina terminálů 1. vrstva	x; -1,0; 1,0; 2,0; 3,0
Množina neterminálů 1. vrstva	*; +; –;
Velikost populace 2. vrstva	50
Generační limit 2. vrstva	30
Základní množina terminálů 2. vrstva	x; -1,0; 1,0; 2,0; 3,0
Množina neterminálů 2. vrstva	*; +; –; sqrt, sin, pow2, pow3, avg

Nastavení jednovrstvého GP a dvouvrstvého GP s přidáním dalších funkcí zachycují Tabulka 16 a Tabulka 17.

Metoda bootstrapping

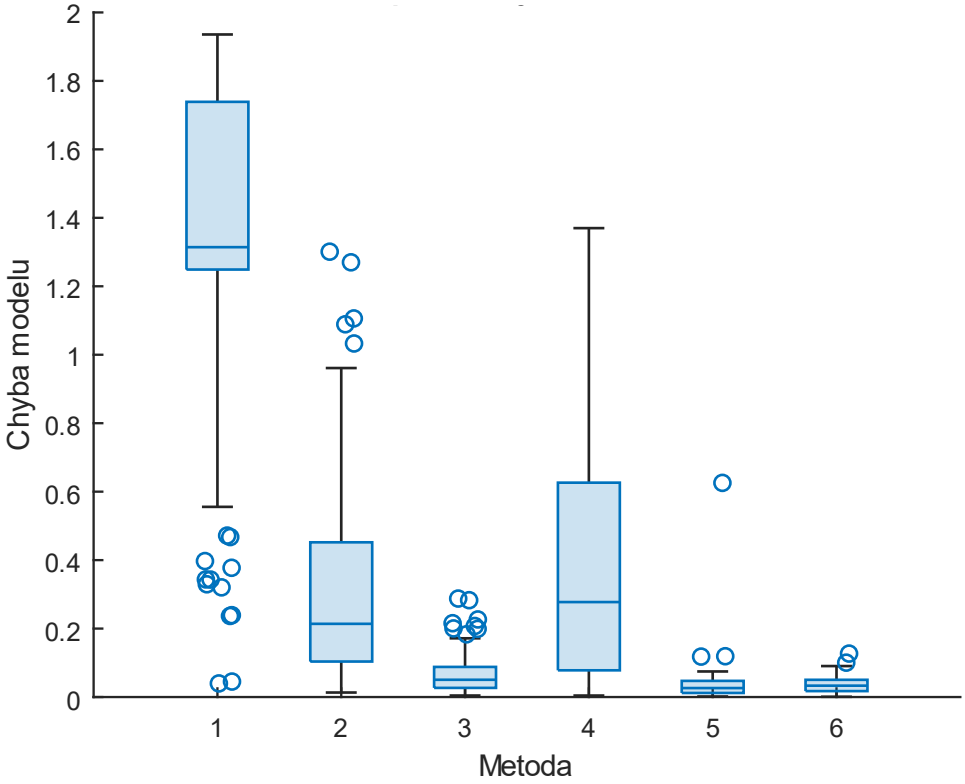
Pro metodu bootstrapping byly vyzkoušeny tři různé velikosti bootstrapu:

- 10 %,
- 20 %,
- 30 %.

Šlo konkrétně o experimenty:

1. jednovrstvé GP s dalšími přidanými funkcemi pow2 a pow3,
2. jednovrstvé GP s dalšími přidanými funkcemi pow2 a pow3, 1500 jedinců,
3. prosté dvouvrstvé GP bez diverzifikace s dalšími přidanými funkcemi pow2 a pow3,
4. dvouvrstvé GP s bootstrapping (10 %) s dalšími přidanými funkcemi pow2 a pow3,
5. dvouvrstvé GP s bootstrapping (30 %) s dalšími přidanými funkcemi pow2 a pow3,
6. dvouvrstvé GP s bootstrapping (60 %) s dalšími přidanými funkcemi pow2 a pow3.

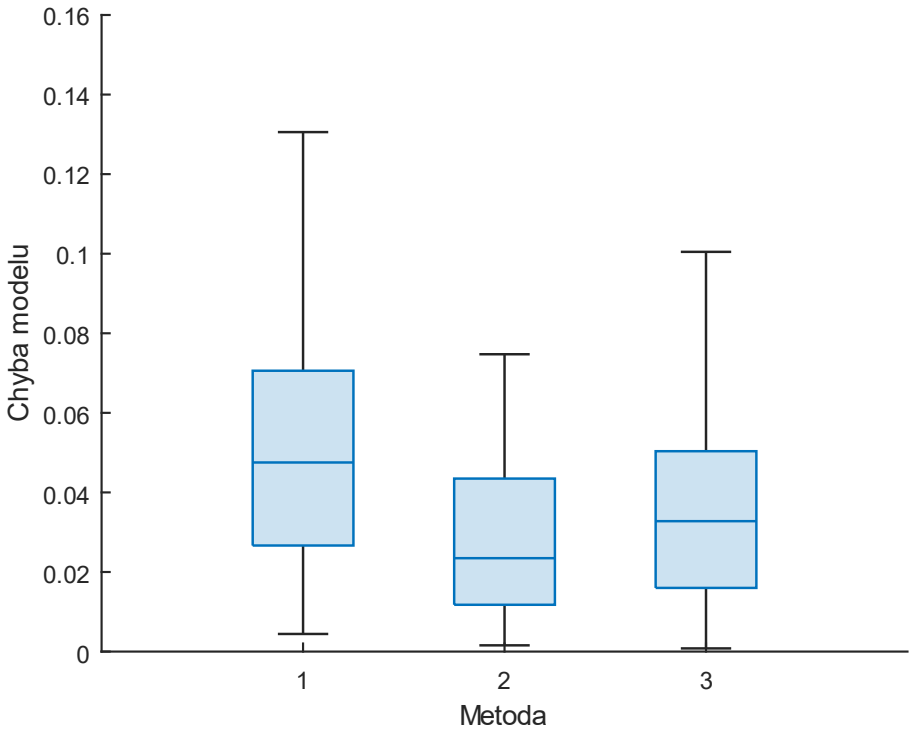
Výsledky experimentů jsou vidět v následujícím grafu (Obrázek 18).



Obrázek 18 – Porovnání jednovrstvého GP s dvouvrstvým GP s metodou bootstrapping s dalšími přidanými funkcemi pow2 a pow3

Následuje graf (Obrázek 19) pro detailnější porovnání tří úspěšnějších metod s vynecháním odlehlých hodnot:

1. prosté dvouvrstvé GP bez diverzifikace s dalšími přidanými funkcemi pow2 a pow3,
2. dvouvrstvé GP s bootstrapping (30 %) s dalšími přidanými funkcemi pow2 a pow3,
3. dvouvrstvé GP s bootstrapping (60 %) s dalšími přidanými funkcemi pow2 a pow3.



Obrázek 19 – Porovnání prostého dvouvrstvého GP s dvouvrstvým GP s metodou bootstrapping s dalšími přidanými funkcemi pow2 a pow3

Po přidání funkcí *pow2* a *pow3* se kvalita výsledných modelů generovaných pomocí jednovrstvého GP s malým počtem jedinců příliš nezměnila. Zato se zlepšila kvalita modelů jednovrstvého GP s 1500 jedinci.

Dále kvalita výsledných modelů generovaných pomocí dvouvrstvého GP s metodou bootstrapping znovu výrazně vzrostla. V tomto případě dokonce došlo při použití metody bootstrapping pro vytvoření 30 a 60 dílčích modelů k částečnému zlepšení oproti prostému GP bez diverzifikace trénovacích dat.

Při vytváření 10 dílčích modelů došlo k zlepšení, ale výsledky byly méně konzistentní a obecně nepřekonávaly jednovrstvé GP s 1500 jedinci.

Metoda sampling

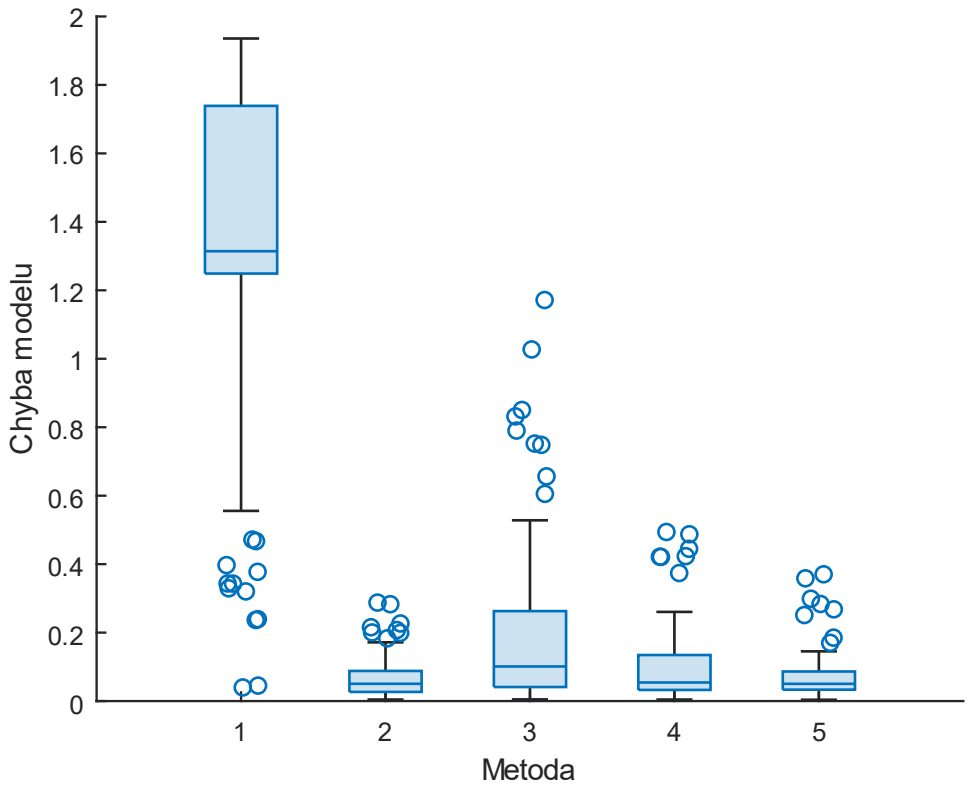
Pro metodu bootstrapping byly znovu vyzkoušeny tři různé počty dílčích modelů:

- 10 dílčích modelů,
- 20 dílčích modelů,
- 30 dílčích modelů.

Pro metodu sampling s dalšími přidanými funkcemi byly provedeny následující experimenty:

1. jednovrstvé GP s přidanými funkcemi pow2 a pow3
2. prosté dvouvrstvé GP bez diverzifikace s přidanými funkcemi pow2 a pow3, 30 dílčích modelů,
3. dvouvrstvé GP sampling, 10 dílčích modelů, s přidanými funkcemi pow2 a pow3,
4. dvouvrstvé GP sampling, 30 dílčích modelů, s přidanými funkcemi pow2 a pow3,
5. dvouvrstvé GP sampling, 60 dílčích modelů, s přidanými funkcemi pow2 a pow3.

Výsledky experimentů zobrazuje následující graf (Obrázek 20).

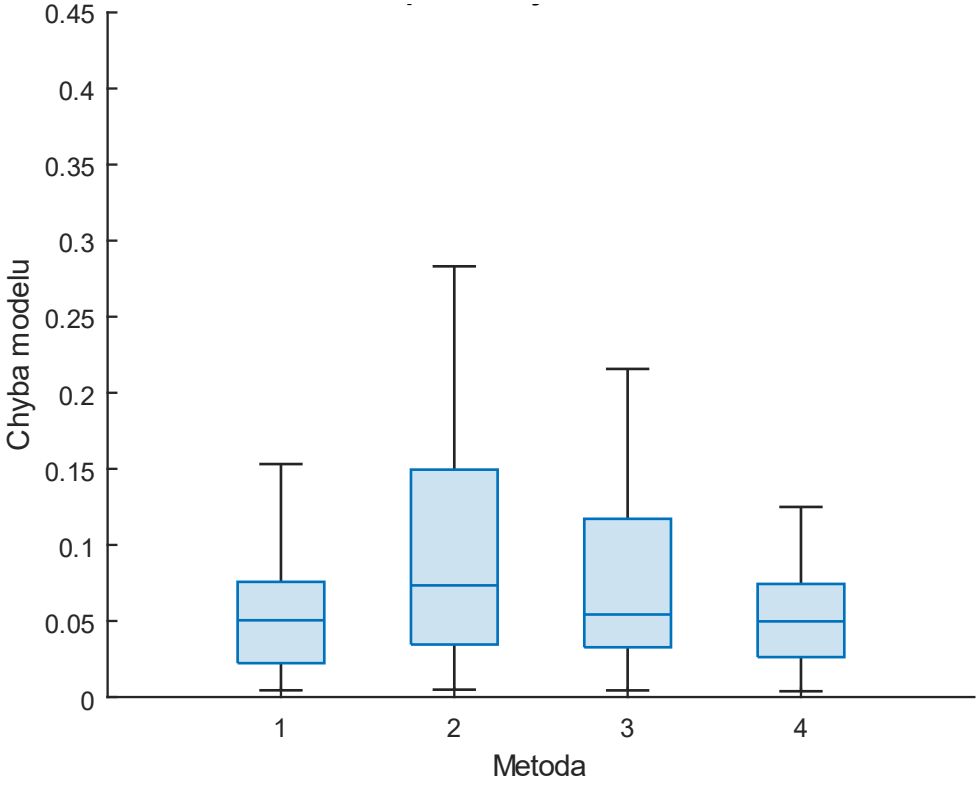


Obrázek 20 – Porovnání jednovrstvého GP s dvouvrstvým GP s metodou sampling s dalšími přidanými funkcemi pow2 a pow3

Následuje graf zobrazující výsledky dvouvrstvých verzí GP bez odlehlých hodnot pro detailnější porovnaní.

Konkrétně jde o experimenty:

1. prosté dvouvrstvé GP bez diverzifikace s přidanými funkcemi pow2 a pow3, 30 dílčích modelů,
2. dvouvrstvé GP sampling, 10 dílčích modelů, s přidanými funkcemi pow2 a pow3,
3. dvouvrstvé GP sampling, 30 dílčích modelů, s přidanými funkcemi pow2 a pow3,
4. dvouvrstvé GP sampling, 60 dílčích modelů, s přidanými funkcemi pow2 a pow3.



Obrázek 21 – Porovnání dvouvrstvého GP s dvouvrstvým GP s metodou sampling po přidání dalších funkcí pow2 a pow3

Graf (Obrázek 21) ukazuje, že se v tomto případě metoda sampling ukázala jako nápomocná k tvorbě přesnějších modelů než jednovrstvé GP. Prosté dvouvrstvé GP znovu v přesnosti porazilo metodu sampling s 10 a 30 dílčími modely. Metoda sampling s 60 dílčími modely měla v tomto případě výsledky srovnatelné s prostým dvouvrstvým GP s 30 dílčími modely.

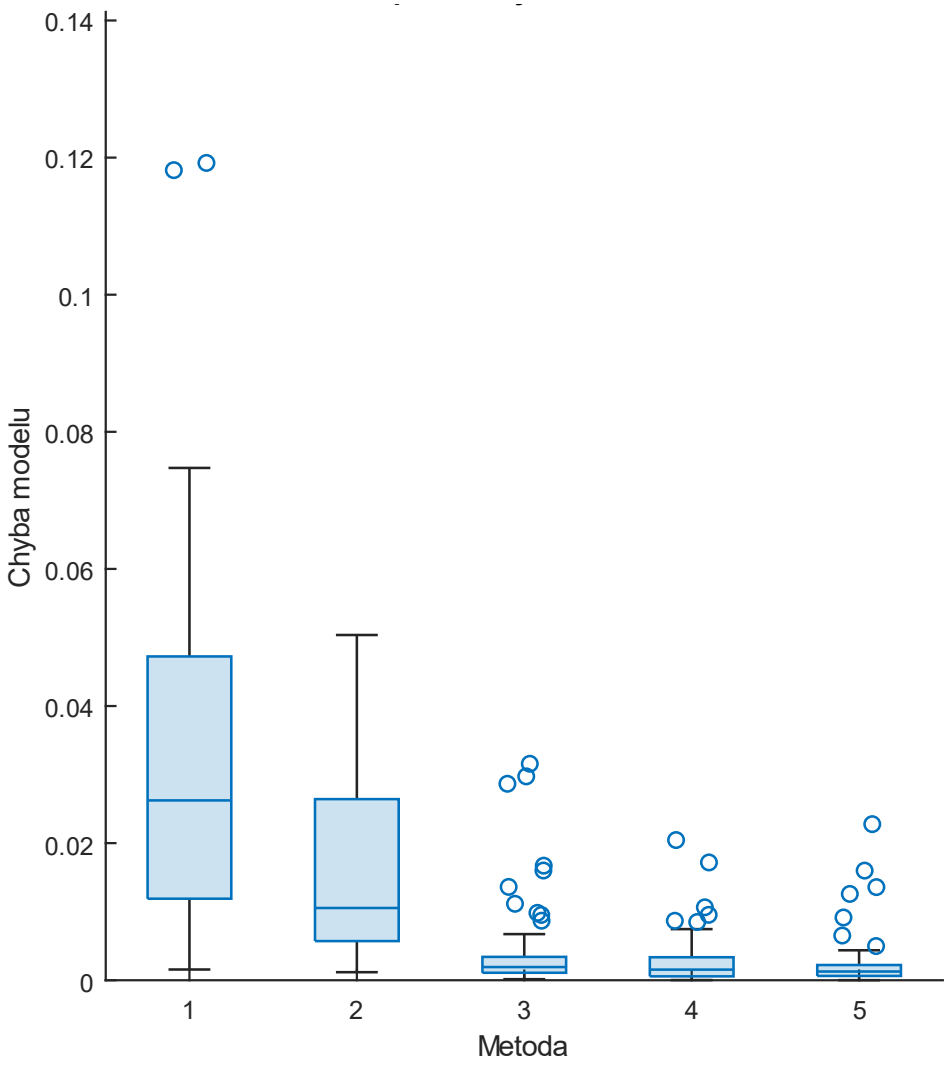
Víc generací a jedinců na druhé vrstvě

Dosavadní experimenty ukázaly informace o povaze chování dvouvrstvých GP, nicméně přesnost výsledných modelů nebyla dostatečná. Proto byly provedeny experimenty, ve kterých došlo ke zvýšení počtu jedinců a generací na druhé vrstvě. Množina funkcí obsahovala původní a všechny doposud přidané funkce (sqrt, sin, pow2, pow3).

První experimenty byly provedeny s různými nastaveními dvouvrstvého GP s metodou bootstrapping a velikostí podmnožin nastavenou na hodnotu 30 % celé množiny trénovacích dat.

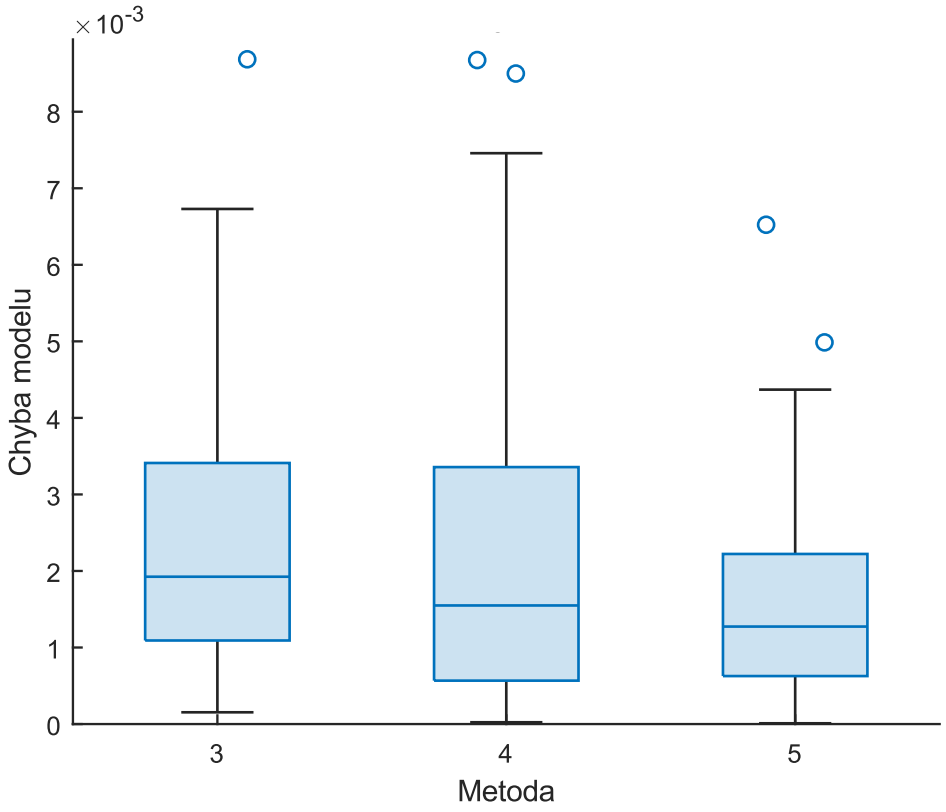
Následující graf (Obrázek 22) zobrazuje výsledky experimentů s dvouvrstvým GP s tímto nastavením:

1. dvouvrstvé GP s metodou bootstrapping (30 %), 30 dílčích modelů, 50 jedinců a 30 generací na druhé vrstvě,
2. dvouvrstvé GP s metodou bootstrapping (30 %), 30 dílčích modelů, 150 jedinců a 30 generací na druhé vrstvě,
3. dvouvrstvé GP s metodou bootstrapping (30 %), 30 dílčích modelů, 250 jedinců a 120 generací na druhé vrstvě,
4. dvouvrstvé GP s metodou bootstrapping (30 %), 30 dílčích modelů, 350 jedinců a 120 generací na druhé vrstvě,
5. dvouvrstvé GP s metodou bootstrapping (30 %), 30 dílčích modelů, 500 jedinců a 120 generací na druhé vrstvě.



Obrázek 22 – Dvouvrstvé GP s metodou bootstrapping (30 %) s různým počtem jedinců a generací na druhé vrstvě

Přiblížení na tři poslední nastavení pro detailnější srovnání zobrazuje Obrázek 23:

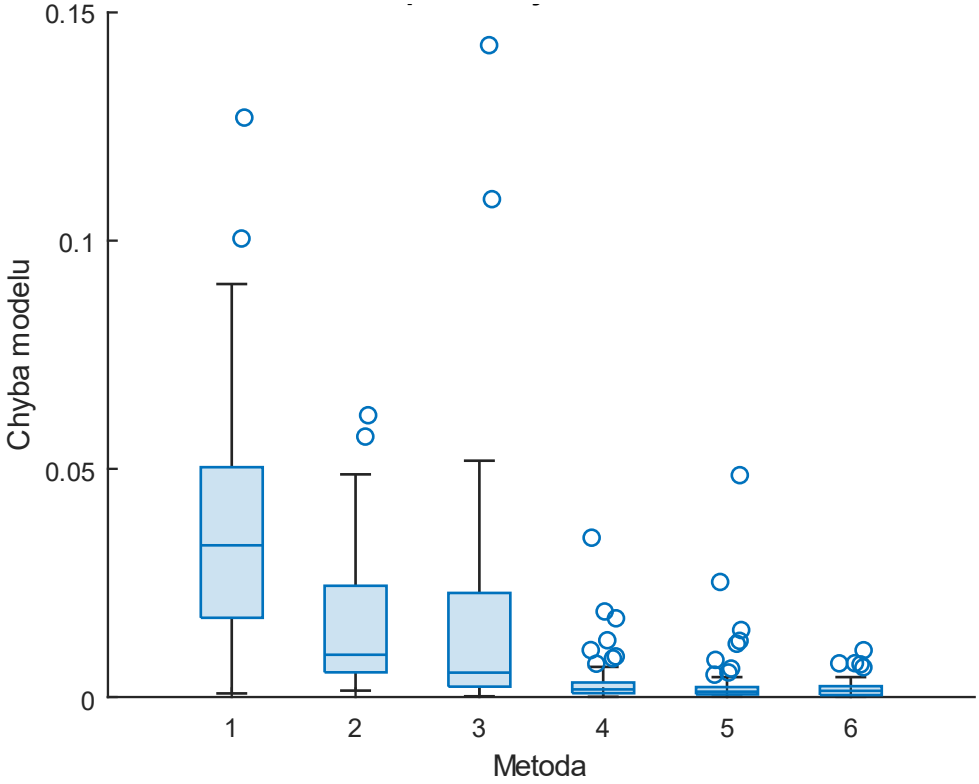


Obrázek 23 – Porovnání tří nejpřesnějších nastavení metody bootstrapping u funkce hyperbola

Zvýšením počtu generací a jedinců na druhé vrstvě došlo k výraznému zvýšení přesnosti generovaných modelů. Poslední tři nastavení vytvářejí výsledné modely s chybou řádově 10^{-3} a to i s velikostí bootstrapu 30 %.

Následující graf (Obrázek 24) zobrazuje výsledky experimentů s dvouvrstvým GP s tímto nastavením:

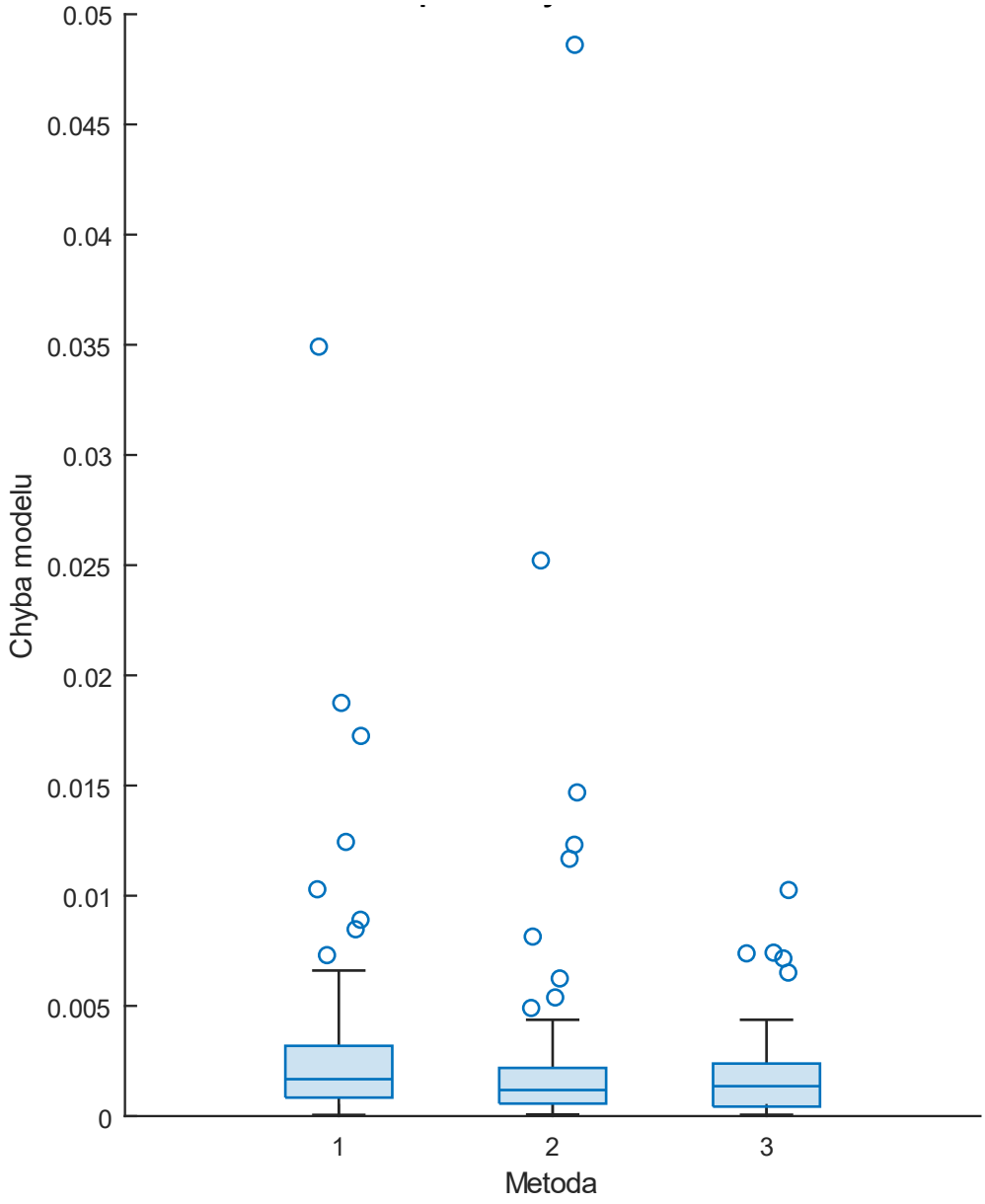
1. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 50 jedinců a 30 generací na druhé vrstvě,
2. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 150 jedinců a 30 generací na druhé vrstvě,
3. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 50 jedinců a 120 generací na druhé vrstvě,
4. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 250 jedinců a 120 generací na druhé vrstvě,
5. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 350 jedinců a 120 generací na druhé vrstvě,
6. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 500 jedinců a 120 generací na druhé vrstvě.



Obrázek 24 – Dvouvrstvé GP s metodou bootstrapping (60 %) s různým počtem jedinců a generací na druhé vrstvě

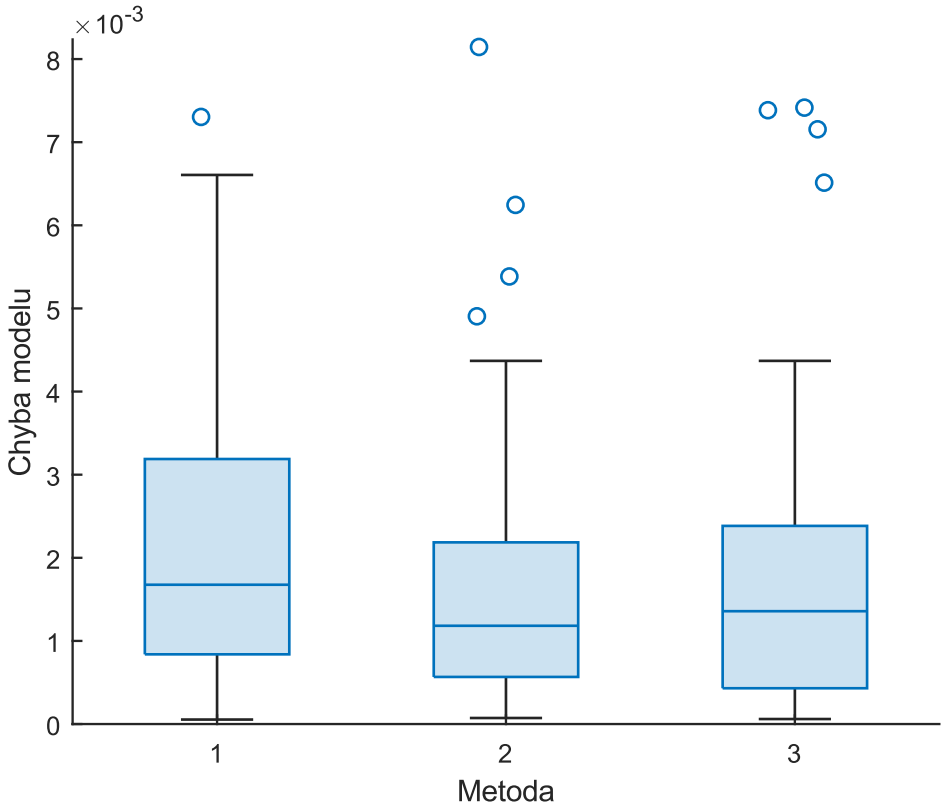
Pro lepší přehlednost a srovnání nastavení s přesnějšími výsledky jsou zobrazeny poslední tři nastavení v samostatném grafu:

1. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 250 jedinců a 120 generací na druhé vrstvě,
2. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 350 jedinců a 120 generací na druhé vrstvě,
3. dvouvrstvé GP s metodou bootstrapping (60 %), 30 dílčích modelů, 500 jedinců a 120 generací na druhé vrstvě.



Obrázek 25 – Nejúspěšnější dvouvrstvé GP s metodou bootstrapping (60 %) s různým počtem jedinců a generací na druhé vrstvě

Pro detailnější srovnání Obrázek 26 zobrazuje přiblíženou variantu předchozího grafu (Obrázek 25).



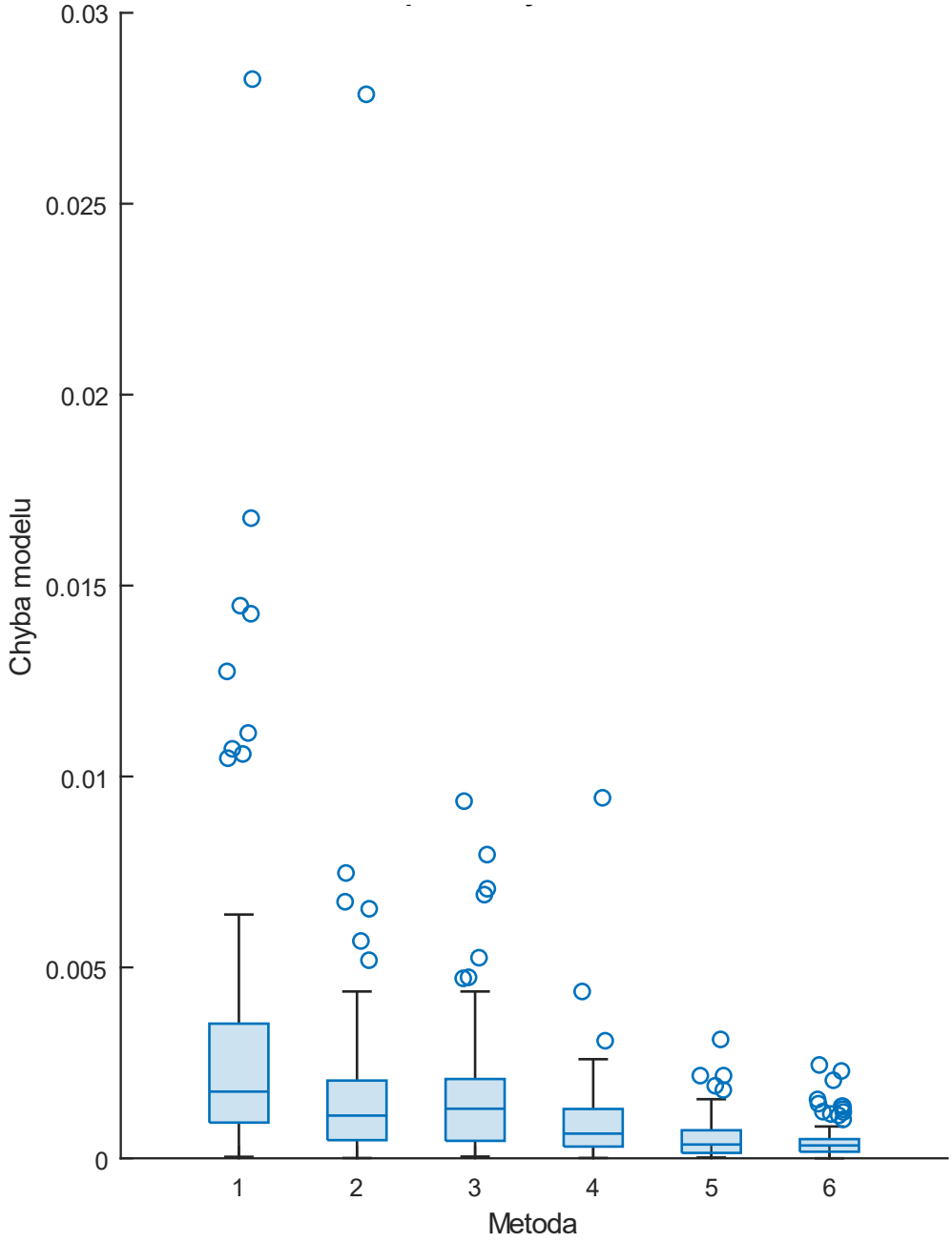
Obrázek 26 – Přiblížený pohled na nejúspěšnější dvouvrstvé GP s metodou bootstrapping (60 %) s různým počtem jedinců a generací na druhé vrstvě

Z grafů (Obrázek 24, Obrázek 25 a Obrázek 26) lze vyčíst, že zvyšování počtu jedinců a generací na druhé vrstvě mělo v tomto případě výrazný pozitivní vliv na chybu generovaných modelů a většina vygenerovaných modelů pomocí těchto tří nastavení má chybu v řádu 10^{-3}, medián chyby se pohybuje pod hranicí 0,005 a dolní kvartil chyby je pod hranicí 10^{-4}. Zvětšená velikost bootstrapu měla v tomto případě pozitivní vliv na kvalitu modelů, ale nebyla nijak výrazná.

Se samplingem:

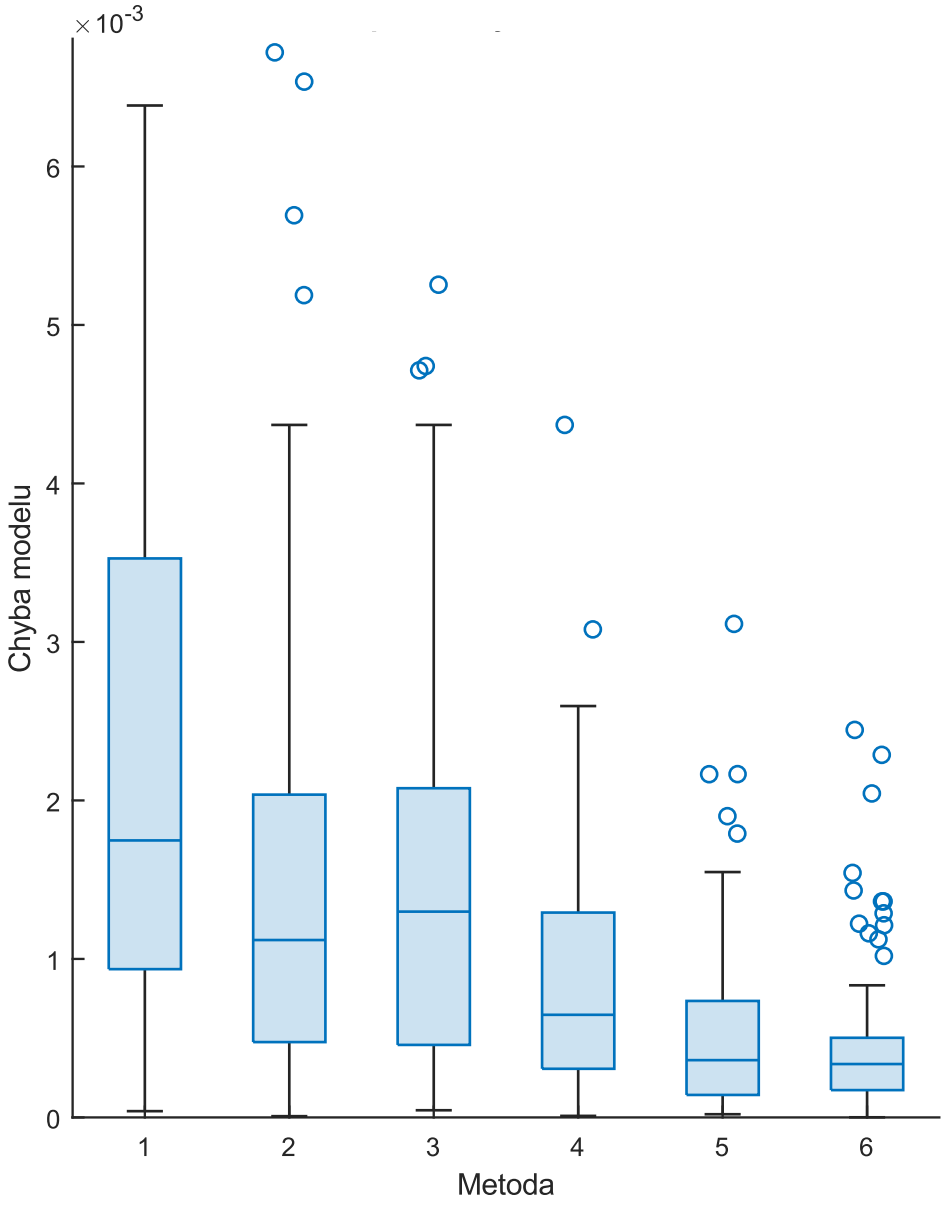
1. dvouvrstvé GP s metodou sampling, 250 jedinců 120 generací na druhé vrstvě,
2. dvouvrstvé GP s metodou sampling, 350 jedinců 120 generací na druhé vrstvě,
3. dvouvrstvé GP s metodou sampling 500 jedinců 120 generací na druhé vrstvě,
4. dvouvrstvé GP s metodou sampling 250 jedinců 240 generací na druhé vrstvě,
5. dvouvrstvé GP s metodou sampling 500 jedinců 240 generací na druhé vrstvě,
6. dvouvrstvé GP s metodou sampling 500 jedinců 350 generací na druhé vrstvě.

Výsledky jsou zobrazeny v následujícím grafu (Obrázek 27).



Obrázek 27 – Dvouvrstvé GP s metodou sampling se zvýšenými hodnotami parametrů

Následuje Obrázek 28, který zachycuje přiblížení předcházejícího grafu pro detailnější pohled na dosažené výsledky.



Obrázek 28 – Dvouvrstvé GP s metodou sampling se zvýšenými hodnotami parametrů (detail)

Z grafu je patrné, že dvouvrstvé GP s metodou sampling v tomto případě generovalo výsledné modely obdobné kvality jako dvouvrstvé GP ve spojení s metodou bootstrapping (30 % a 60 %). Zvyšování jedinců a generací na druhé vrstvě způsobilo znatelnému snížení chyby výsledných modelů. Při nastavení 500 jedinců v kombinaci s 350 generacemi na druhé vrstvě došlo ke snížení chyby takovým způsobem, že se nejen dolní ale i horní kvartil chyby dostal pod hranici 10^{-4}.

8.3 Jednoduchý sinus

8.3.1 Základní experimenty

Pro funkci sinus z rovnice (6) byly provedeny pokusy s jednovrstvým a dvouvrstvým GP. Následující Tabulka 18 zachycuje nastavení parametrů jednovrstvého GP pro hledání modelu funkce sinus. Generační limit na jednovrstvého GP s hodnotou 230 kompenzuje celkový počet generací dvouvrstvého GP (200 generací na 1. vrstvě plus 30 generací na 2. vrstvě). Nastavení dvouvrstvého GP jsou shrnuty postupně v Tabulka 19 a Tabulka 20.

Tabulka 18 – Nastavení parametrů jednovrstvého GP pro hledání funkce sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	230
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 19 – Nastavení parametrů 1. vrstvy GP pro hledání funkce sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	200
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 20 – Nastavení parametrů 2. vrstvy pro hledání funkce sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	30
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv, avg

Následuje vysvětlení zkratek použitých neterminálů:

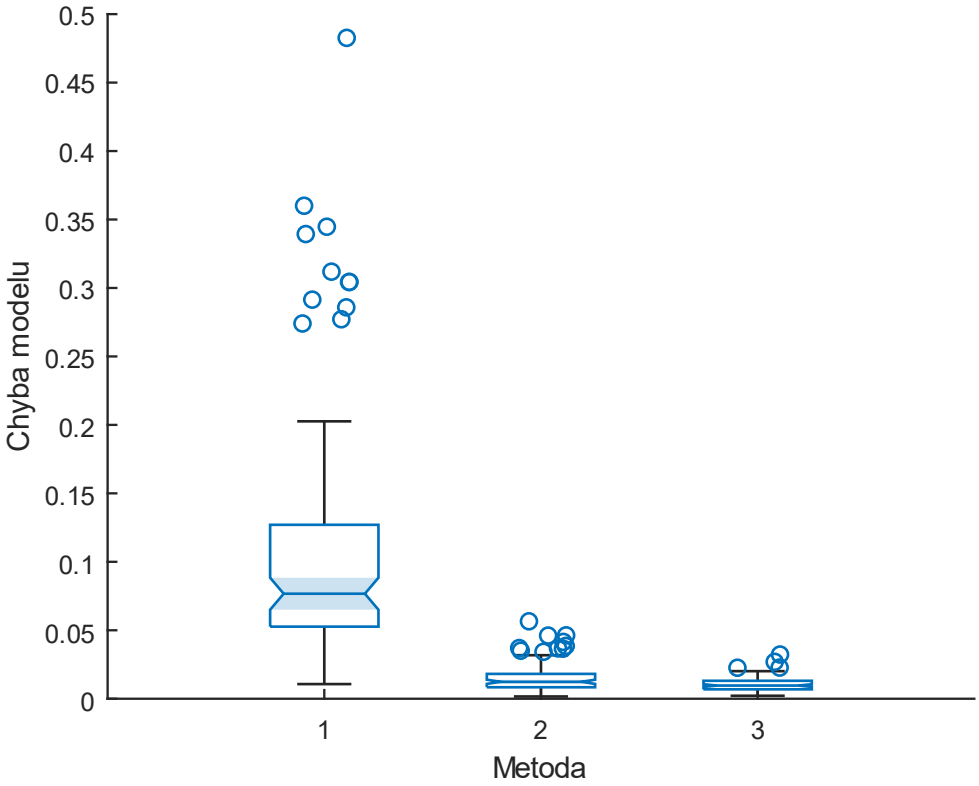
- Neterminál % je operace chráněné dělení, které v případě dělení nulou vrací hodnotu 0.
- Neterminál *inv* je funkce inverze s jedním parametrem a vytváří převrácenou hodnotu předaného parametru.
- Neterminál *avg* je funkce průměr, která přebírá jako vstup dva parametry a spočítá z nich aritmetický průměr.

V první sérii experimentů bylo u dvouvrstvého GP generováno na první vrstvě 30 dílčích modelů. Nejprve došlo k porovnání jednovrstvého GP s dvouvrstvým GP bez diverzifikace podmnožin trénovacích dat. Jelikož dvouvrstvé GP vytváří na první vrstvě 30 dílčích model pomocí 50 jedinců, byl ještě pro srovnání proveden pokus na jednovrstvém GP, u kterého

byl počet jedinců nastaven na 1500. Výsledky zachycují dva krabicové grafy, Obrázek 29 a Obrázek 30.

Na prvním grafu (Obrázek 29) jsou zobrazeny tyto experimenty:

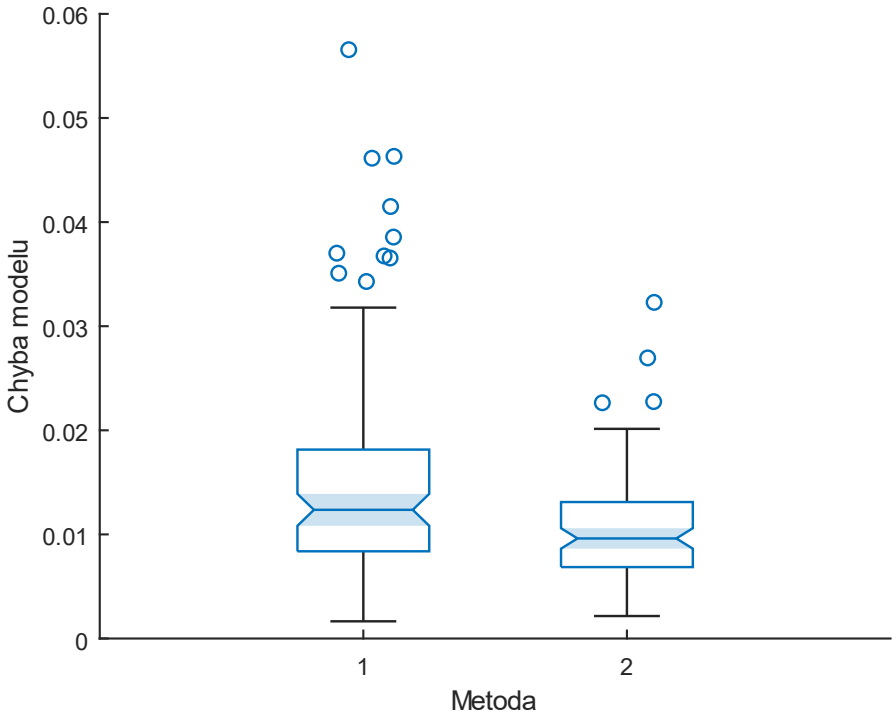
1. jednovrstvé GP,
2. jednovrstvé GP s 1500 jedinci,
3. dvouvrstvé GP s 30 dílčími modely.



Obrázek 29 – Srovnání jednovrstvého GP s dvouvrstvým GP

Pro lepší pohled na rozdíl mezi jednovrstvým GP s velkým počtem jedinců a dvouvrstvým GP s 50 jedinci druhý graf (Obrázek 30) zobrazuje experimenty:

1. jednovrstvé GP s 1500 jedinci,
2. dvouvrstvé GP s 30 dílčími modely.

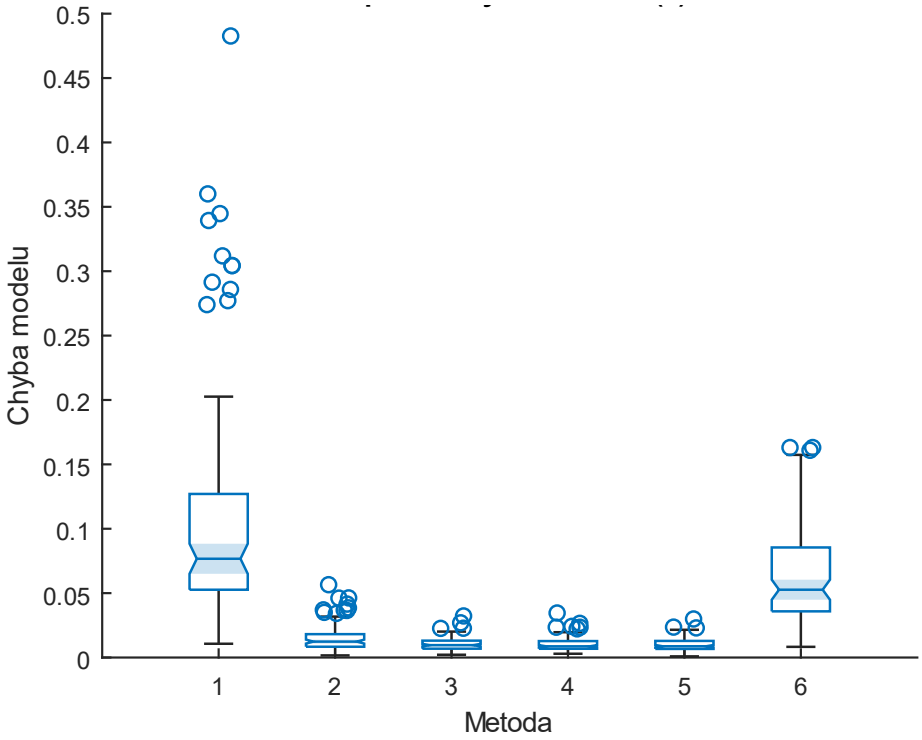


Obrázek 30 – Srovnání jednovrstvého GP s velkým počtem jedinců a dvouvrstvého GP.

V dalších pokusech se předchozí výsledky porovnávaly s jednotlivými přístupy k diverzifikaci podmnožin trénovacích dat. Velikost podmnožiny u metody bootstraping byla nastavena na 60 % velikosti celé trénovací množiny. Velikost okna u metody náhodného pohyblivého okna byla nastavena na 100, náhodný začátek okna se pohybuje mezi hodnotami 0 a 0,6 velikosti celé podmnožiny dat. Výsledky zobrazují Obrázek 31 a Obrázek 32.

První graf (Obrázek 31) zachycuje následující pokusy:

1. jednovrstvé GP,
2. jednovrstvé GP s 1500 jedinci,
3. dvouvrstvé GP s 30 dílčími modely,
4. dvouvrstvé GP s 30 dílčími modely + sampling,
5. dvouvrstvé GP s 30 dílčími modely + bootstraping (60 %),
6. metoda náhodného pohyblivého okna (velikost okna 100).

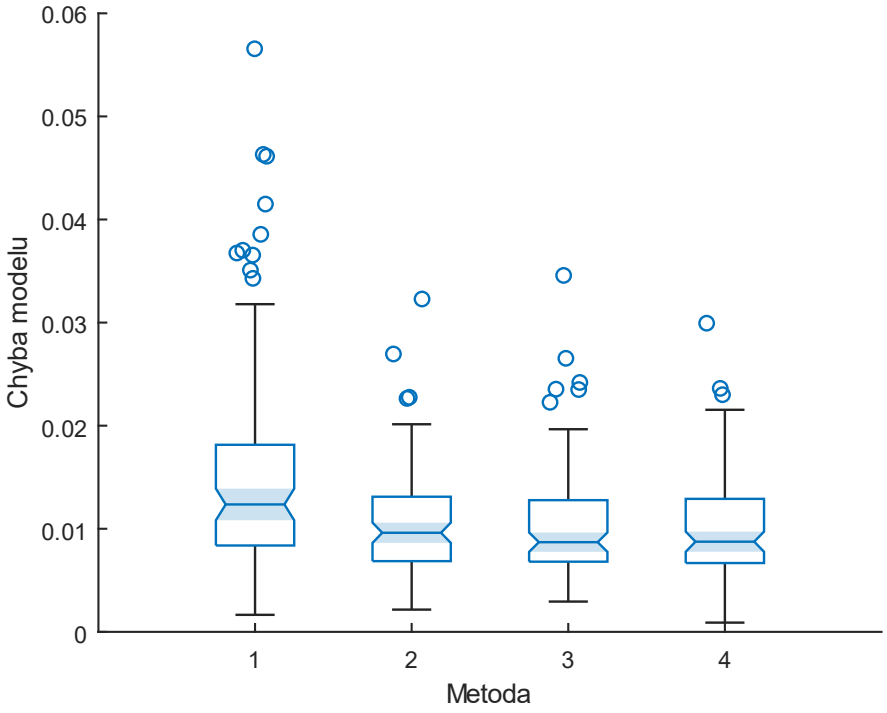


Obrázek 31 – Srovnání se samplingem, bootstrappingem a metodou pohyblivého okna

Druhý graf (Obrázek 32) vynechává jednovrstvé GP s 50 jedinci a dvouvrstvé GP s náhodným pohyblivým oknem pro detailnější pohled na rozdíl mezi jednovrstvou metodou s velkou populací a různými variantami dvouvrstvého GP s generováním 30 dílčích modelů.

Jmenovitě jde o experimenty:

1. jednovrstvé GP s 1500 jedinci,
2. dvouvrstvé GP s 30 dílčími modely,
3. dvouvrstvé GP s 30 dílčími modely + sampling,
4. dvouvrstvé GP s 30 dílčími modely + bootstraping (60 %).



Obrázek 32 – Srovnání základní verze dvouvrstvého GP se dvouvrstvým GP samplingem a bootstrapingem

Mezi jednovrstvým GP s 50 jedincí a dvouvrstvým GP s 50 jedinci je při tomto nastavení na funkci sinus velmi znatelný rozdíl v přesnosti. Jednovrstvé GP s velkou populací také dosahuje velmi dobrých výsledků.

Přestože rozdíl mezi jednovrstvým GP s velkou populací a dvouvrstvým GP s 50 jedinci není tak velký jako mezi jednovrstvým GP s 50 jedinci a dvouvrstvým GP se 1500 jedinci, stále pro tato nastavení vytváří dvouvrstvé GP přesnější výsledné modely. Srovnání není dokonalé ani v tomto případě. Sice je počet jedinců stejný, ale v případě jednovrstvého GP jde o jednolitou populaci, kde se může křížit každý s každým. V případě více dílčích modelů je více menších populací. Tyto populace se navzájem neovlivňují a tak dvouvrstvé GP v tomto ohledu připomíná spíše genetické programování založené na ostrovních modelech bez migrace mezi ostrovy.

Rozdíl mezi jednotlivými dvouvrstvými variantami není v tomto případě příliš patrný.

8.3.2 Dvouvrstvé GP s metodou sampling

Pro funkci sinus z rovnice (6) byly provedeny pokusy dvouvrstvým GP používající metodu bootstrapping na první vrstvě pro tvorbu diverzifikovaných dílčích modelů. Nastavení experimentů shrnují Tabulka 21 a Tabulka 22.

Tabulka 21 – Nastavení parametrů 1. vrstvy GP s metodou sampling pro funkci sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	200
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

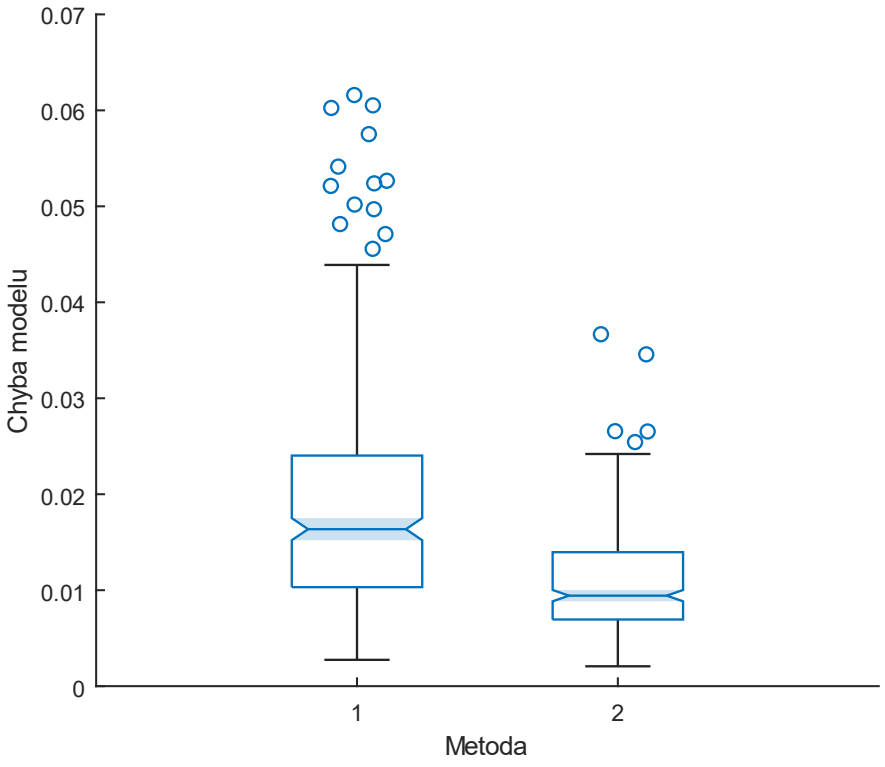
Tabulka 22 – Nastavení parametrů 2. GP s metodou sampling pro funkci sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	30
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv, avg

Pro otestování metody sampling byly provedeny experimenty se dvěma různými počty generovaných dílčích modelů:

1. 10 dílčích modelů,
2. 30 dílčích modelů.

Výsledky zachycují dva krabicové grafy. První graf (Obrázek 33) porovnává čistě dvě nastavení dvouvrstvého GP s metodou sampling.



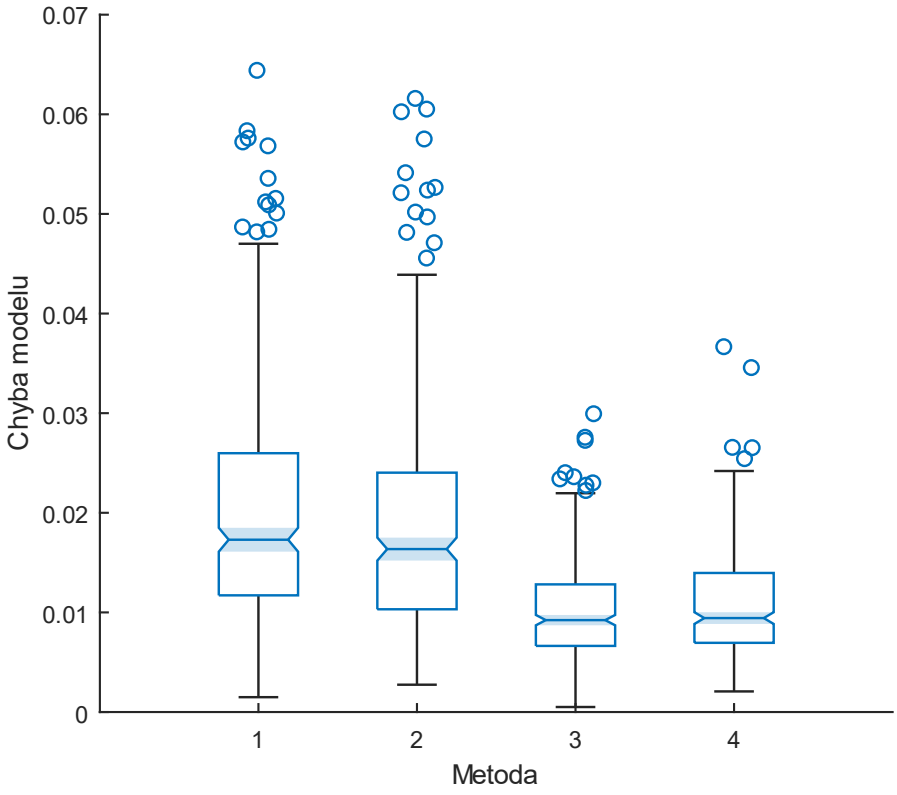
Obrázek 33 – Porovnání metody sampling s 10 a 30 dílčími modely

Z prvního grafu (Obrázek 33) si lze vyvodit, že pro tato nastavení na funkci sinus metoda sampling s 30 dílčími modely byla přesnější než s 10 dílčími modely.

Druhý graf (Obrázek 34) porovnává metodu sampling s metodou bootstrapping s velikostí podmnožiny trénovacích dat nastavenou na 60 % velikosti celé množiny trénovacích dat a odpovídajícími počty generovaných dílčích modelů.

Jmenovitě jde o tyto experimenty:

1. bootstraping (60 %), 10 dílčích modelů,
2. sampling, 10 dílčích modelů,
3. bootstraping (60 %), 30 dílčích modelů,
4. sampling, 30 dílčích modelů.



Obrázek 34 – Porovnání metody sampling s 10 a 30 dílčími modely s metodou bootstrapping

V tomto grafu je vidět, že metoda sampling byla na funkci sinus téměř srovnatelná s metodou bootstraping s velikostí podmnožiny nastavenou na 60 % celé množiny trénovacích dat. U 10 dílčích modelů byl sampling průměrně opticky o trochu lepší, v případě 30 dílčích modelů je tomu naopak. Medián a jeho okolí si však byly u obou přístupů tak blízké, že byl rozdíl zanedbatelný.

8.3.3 Dvouvrstvé GP s metodou bootstrapping

Pro funkci sinus z rovnice (6) byly provedeny pokusy s dvouvrstvým GP používající metodu bootstrapping. Nastavení experimentů shrnují Tabulka 23 a Tabulka 24.

Tabulka 23 – Nastavení parametrů 1. vrstvy GP s metodou bootstrapping pro funkci sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	200
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 24 – Nastavení parametrů 2. GP s metodou bootstrapping pro funkci sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	30
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv, avg

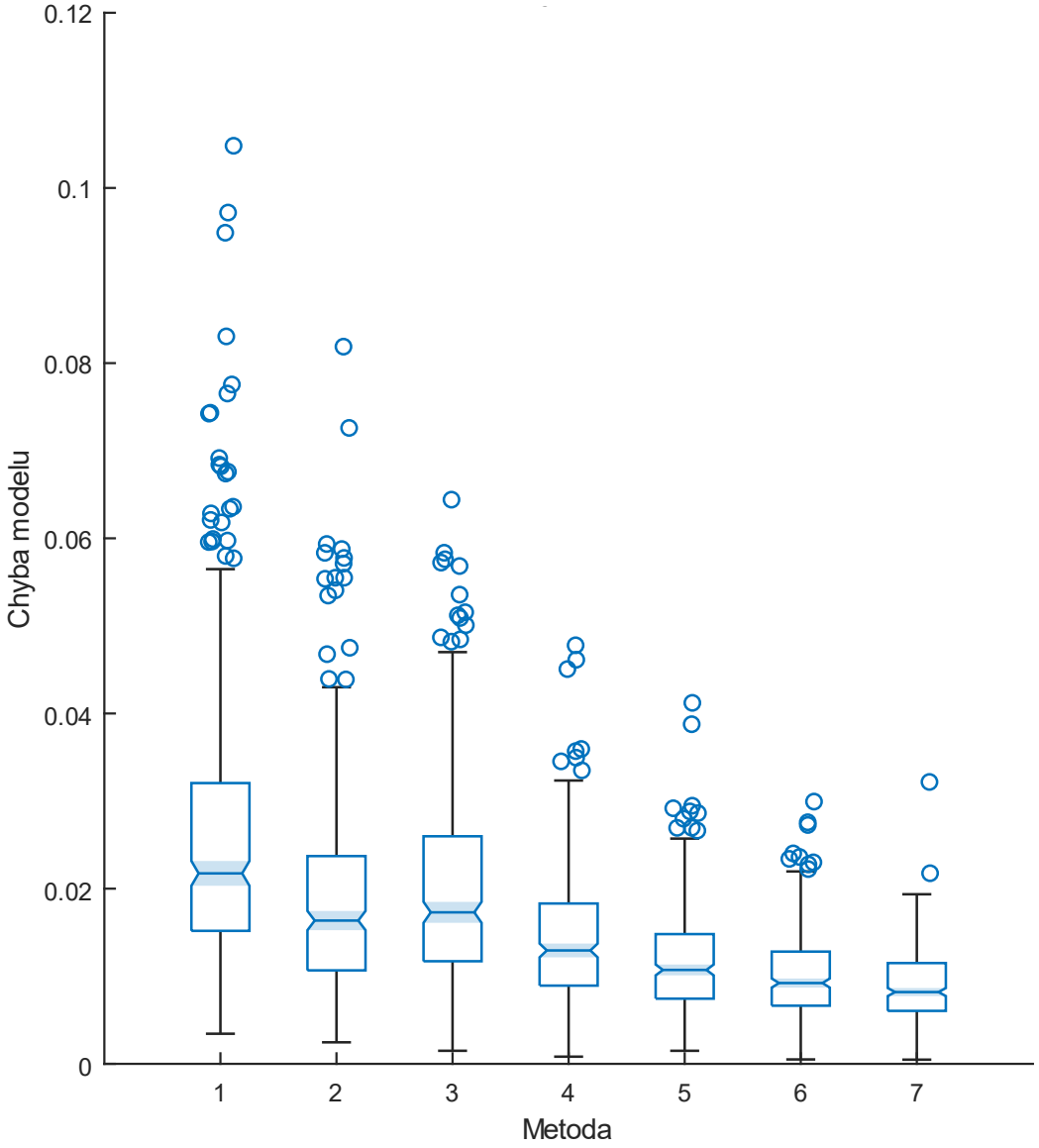
Postupně byly testovány následující konfigurace metody bootstrapping, které zachycuje Tabulka 25:

Tabulka 25 – Nastavení experimentů metody bootstrapping pro funkci sinus

Číslo experimentu	**Velikost podmnožiny[10]**	**Počet dílčích modelů**
1	10 %	10
2	30 %	10
3	60 %	10
4	10 %	30
5	30 %	30
6	60 %	30
7	60 %	60

Výsledky jsou zachycuje krabicový graf (Obrázek 35), který zobrazuje informace o chybě jednotlivých modelů vygenerovaných různými nastaveními metody bootstrapping.

[10] Procentuální část celé původní množiny trénovacích dat.



Obrázek 35 – Porovnání modelů vytvořených pomocí dvouvrstvého GP s metodou bootstrapping

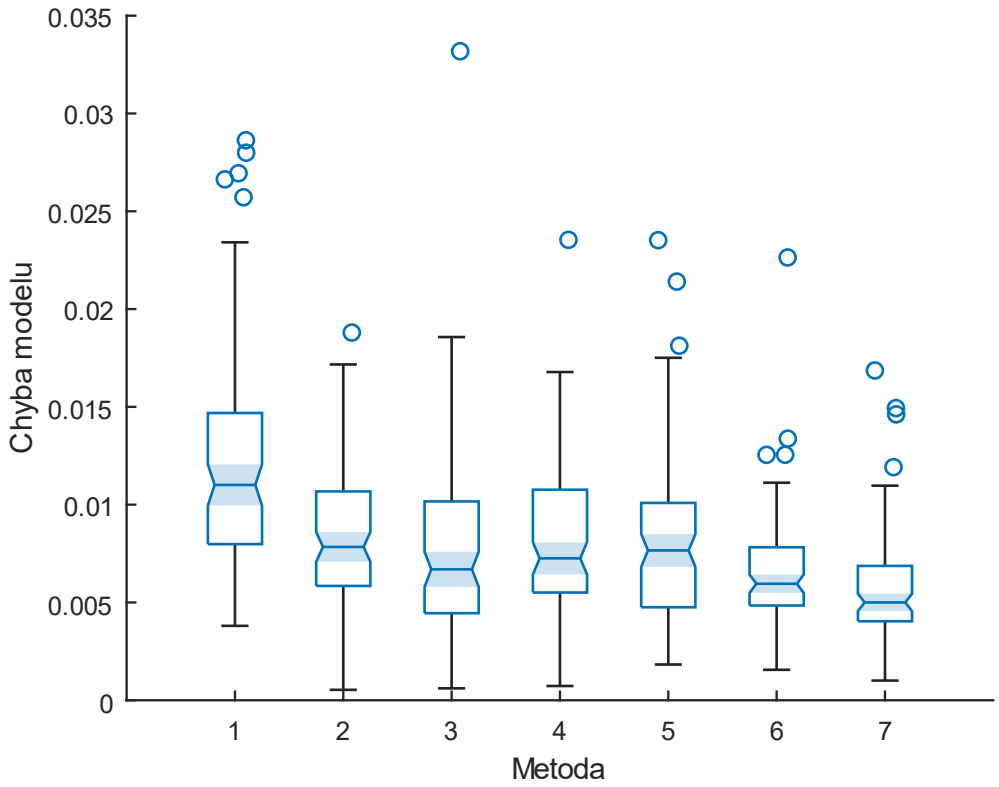
Pro tato nastavení je z grafu patrné, že se zvyšujícím se počtem dílčích modelů došlo k průměrnému zlepšení přesnosti výsledných modelů (snížení chyby). Většinou zvětšení velikosti podmnožiny trénovacích dat vedlo ke snížení chyby výsledných modelů. V případě 10 dílčích modelů to vypadá, že zvětšení podmnožiny trénovacích dat z 30 % na 60 % bylo spíše kontraproduktivní.

8.3.4 Zvýšení výkonu

Pro zvýšení výklonu došlo ke zvětšení počtu jedinců a generací a přidání funkcí:

1. dvouvrstvé GP s metodou bootstrapping (30 %),
2. dvouvrstvé GP s metodou bootstrapping (30 %), 250 jedinců a 120 generací na druhé vrstvě,
3. dvouvrstvé GP s metodou bootstrapping (30 %), 350 jedinců a 120 generací na druhé vrstvě,
4. dvouvrstvé GP s metodou bootstrapping (30 %), 500 jedinců a 120 generací na druhé vrstvě,
5. dvouvrstvé GP s metodou bootstrapping (30 %), 500 jedinců a 240 generací na druhé vrstvě,
6. dvouvrstvé GP s metodou bootstrapping (30 %), 150 jedinců a 300 generací na první vrstvě + pow2, pow3,
7. dvouvrstvé GP s metodou bootstrapping (30 %), 150 jedinců a 300 generací na první vrstvě, 250 jedinců a 120 generací na druhé vrstvě + pow2, pow3.

Výsledky jsou zobrazeny v následujícím grafu (Obrázek 36).



Obrázek 36 – Zvýšené počty jedinců a generací u funkce sinus

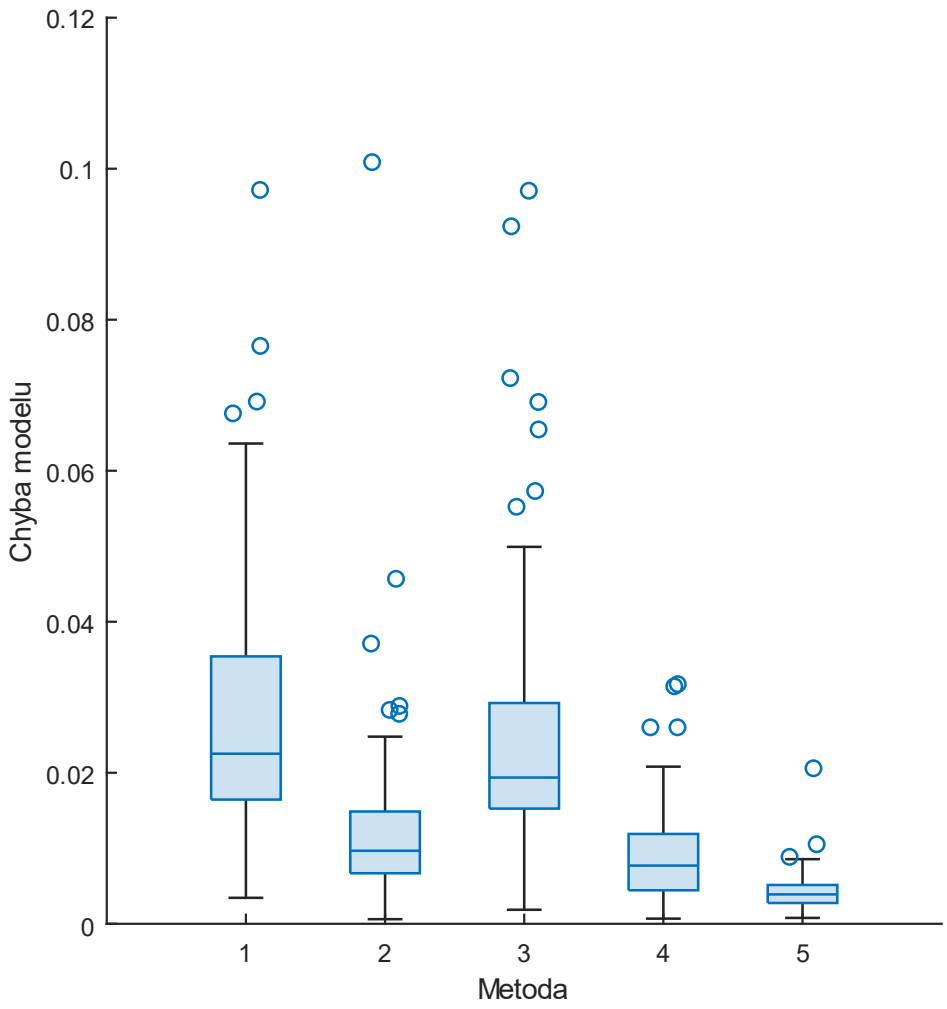
Po zvýšení počtu jedinců a generací na obou vrstvách a přidání funkcí (mocnina a třetí mocnina) došlo k zlepšení kvality výsledných modelů a snížení horního kvartilu chyby pod hranici 0,01. Pomohlo i zvýšení počtu jedinců a generací na první vrstvě. Poslední dvě nastavení vytvářejí výsledné modely konzistentnějším způsobem než předchozí nastavení.

Metoda bootstraping hybridizovaná s BSA

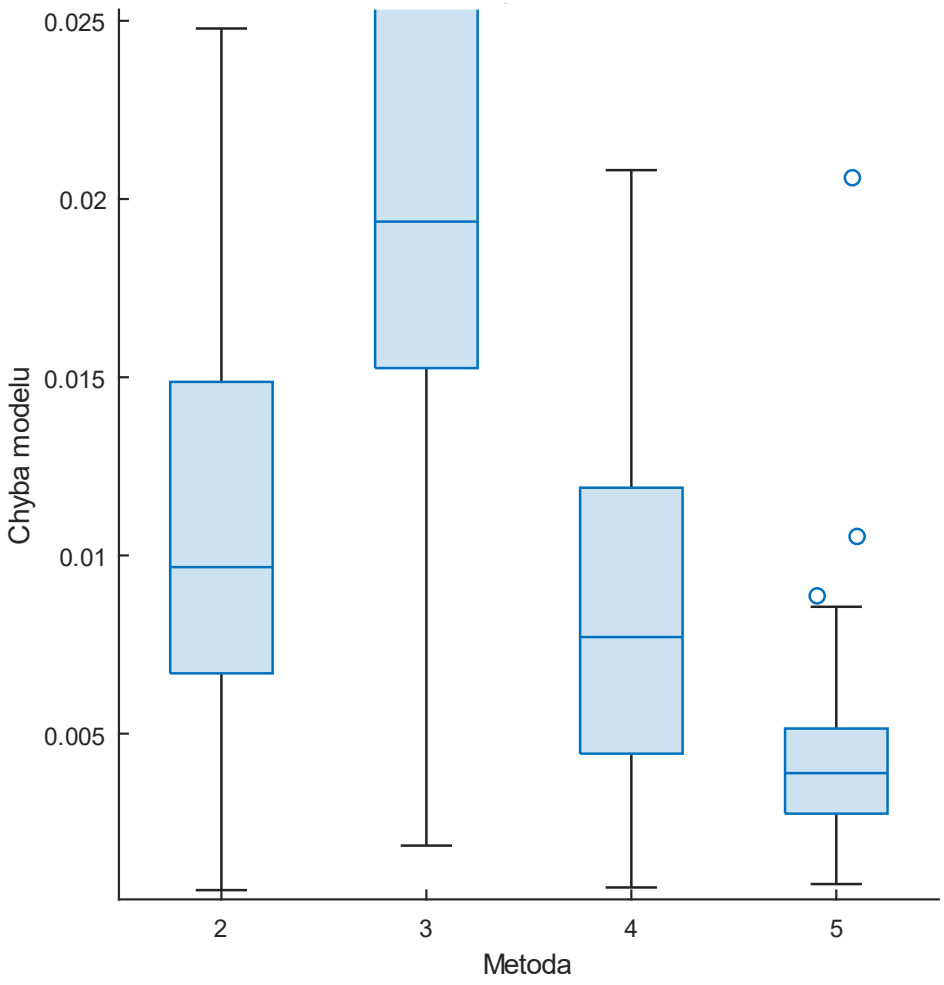
Pro otestování chování dvouvrstvého GP s metodou bootstrapping (10 %) v kombinaci s algoritmem Bison Seeker (BSA) byly porovnány následující konfigurace:

1. dvouvrstvé GP s metodou bootstraping (10 %), 10 dílčích stromů,
2. dvouvrstvé GP s metodou bootstraping (10 %), 10 dílčích stromů, BSA s 15 bizony (12 roj, 6 elitních) a jednou iterací na první vrstvě
3. dvouvrstvé GP s metodou bootstraping (10 %), 10 dílčích stromů, BSA s 15 bizony (12 roj, 6 elitních) a jednou iterací na druhé vrstvě
4. dvouvrstvé GP s metodou bootstraping (10 %), 10 dílčích stromů, BSA s 30 bizony (24 roj, 12 elitních) a jednou iterací na první vrstvě
5. dvouvrstvé GP s metodou bootstraping (10 %), 30 dílčích stromů, s 30 bizony (24 roj, 12 elitních) a jednou iterací na první vrstvě

Výsledky jsou zobrazeny v následujících grafech (Obrázek 37 a Obrázek 38).



Obrázek 37 – Porovnání dvouvrstvého GP s metodou bootstrapping s dvouvrstvým GP hybridizovaným s BSA



Obrázek 38 – Porovnání dvouvrstvého GP s metodou bootstrapping a dvouvrstvým GP hybridizovaným s BSA (přiblíženo)

Z grafů (Obrázek 37 a Obrázek 38) lze vyčíst, že hybridizace dvouvrstvého GP s BSA v těchto konfiguracích pomohla s kvalitou výsledných modelů. Nejvíce pomohl algoritmus BSA na první vrstvě, na druhé v tomto případě tak výrazný nebyl. BSA velmi pomohl konfiguraci s metodou boostraping s malým bootstrapem (10 %) a 10 dílčími stromy. Tato konfigurace se v přesnosti přiblížila konfiguracím s větším bootstrapem a počtem dílčích modelů. Poslední varianta s 30 dílčími modely dokonce překonala konfiguraci se zvýšenými počty generací a jedinců.

8.4 Složený sinus

Funkce z rovnice (7) složená ze třech sinusových složek představuje bez zahrnutí funkce sinus do množiny funkcí pro celočíselné GP složitý problém a může ukázat další aspekty chování dvouvrstvého GP.

8.4.1 Pokusy s 50 jedinci

Základní nastavení parametrů pro hledání složené funkce sinus je podobné jako nastavení pro hledání jednoduchého sinu. Tabulka 26 zobrazuje nastavení parametrů pro jednovrstvé GP a Tabulka 27 a Tabulka 28 zobrazují nastavení parametrů pro dvouvrstvé GP.

Tabulka 26 – Nastavení parametrů jednovrstvého GP pro hledání funkce složenou sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	230
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 27 – Nastavení parametrů 1. vrstvy GP s metodou bootstrapping pro složenou funkci sinus

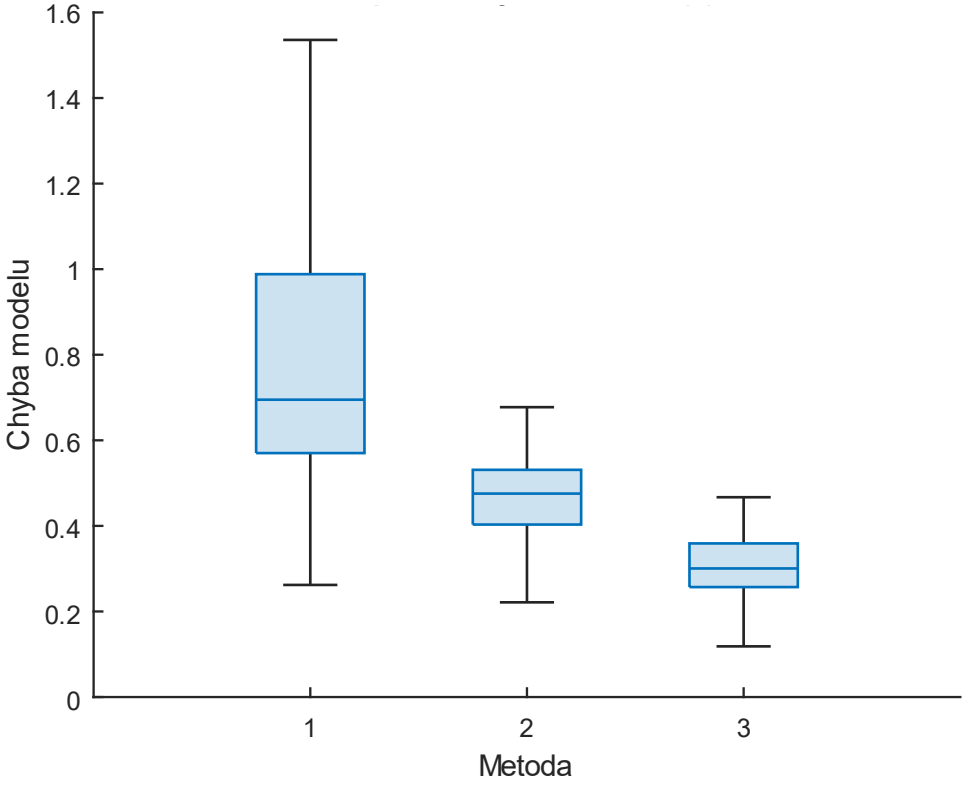
Parametr	**Hodnota**
Velikost populace	50
Generační limit	200
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 28 – Nastavení parametrů 2. GP s metodou bootstrapping pro složenou funkci sinus

Parametr	**Hodnota**
Velikost populace	50
Generační limit	30
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv, avg

První sada experimentů porovnávala jednovrstvé GP s dvouvrstvým GP a se skládala z těchto položek:

1. jednovrstvé GP,
2. jednovrstvé GP 1500 jedinců,
3. dvouvrstvé GP bez diverzifikace trénovacích dat.



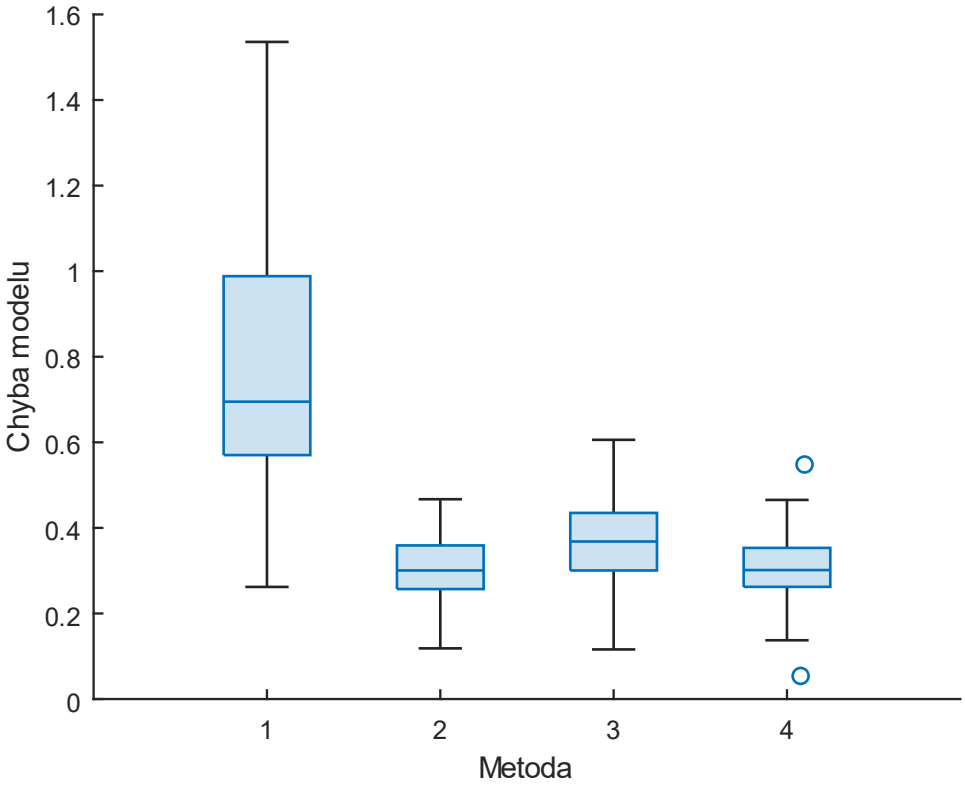
Obrázek 39 – Porovnání jednovrstvého GP s prostým dvouvrstvým GP na složené funkci sinus

Z grafu (Obrázek 39) je patrné, že dvouvrstvé GP mělo v tomto případě znovu pozitivní efekt na kvalitu výsledných modelů a v průměrné přesnosti překonalo jednovrstvé GP i s 1500 jedinci. Jednovrstvé GP s 50 jedinci generuje velmi rozmanité modely. Naopak jednovrstvé GP a prosté dvouvrstvé GP vytvořily výsledné modely konzistentnější způsobem.

Metoda bootstrapping

Pro otestování metody bootstrapping na složené funkci sinus byly provedeny následující experimenty:

1. jednovrstvé GP,
2. dvouvrstvé GP bez diverzifikace trénovacích dat,
3. dvouvrstvé GP bootstrapping 30 %,
4. dvouvrstvé GP bootstrapping 60 %.



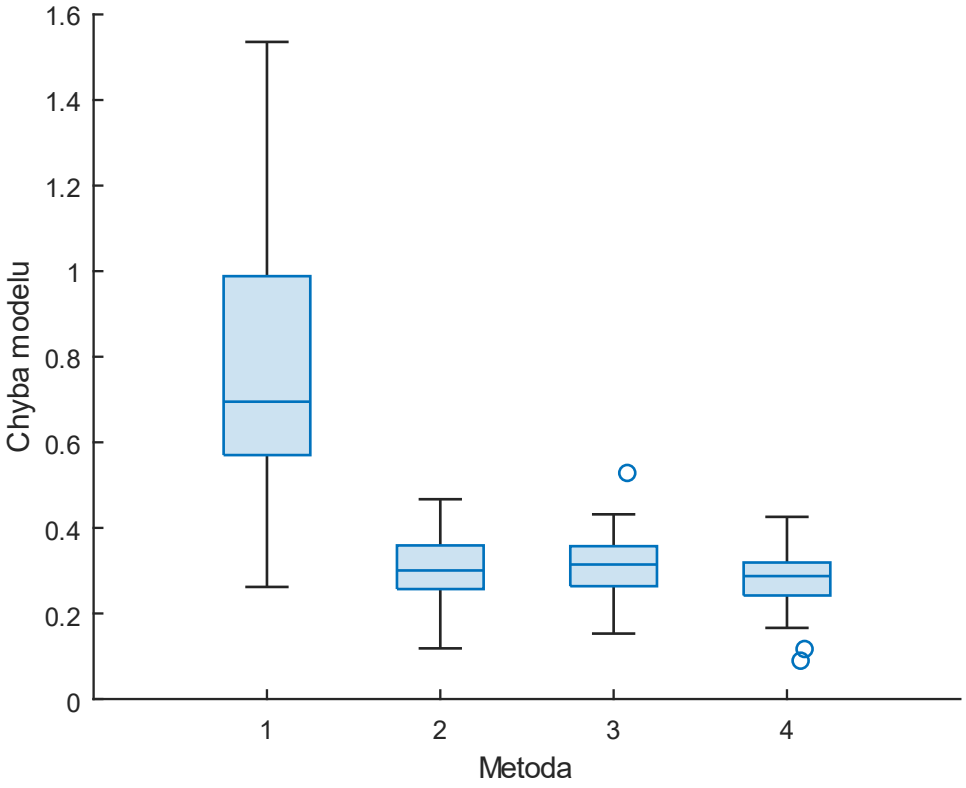
Obrázek 40 – Porovnání metody boostrapping s prostým dvouvrstvým GP na složené funkci sinus

Z grafu (Obrázek 40) lze vyčíst, že v tomto případě bootstrapping zlepšil kvalitu modelů oproti jednovrstvému GP. Bootstrapping s velikostí podmnožiny 60 % původní množiny trénovacích dat vygeneroval lepší modely než bootstrapping s poloviční velikostí a blížil se přesnosti prostého dvouvrstvého GP.

Metoda sampling

Pro otestování metody sampling na složené funkci sinus byly provedeny následující experimenty:

1. jednovrstvé GP,
2. dvouvrstvé GP bez diverzifikace trénovacích dat,
3. dvouvrstvé GP sampling, 30 dílčích modelů,
4. dvouvrstvé GP sampling, 60 dílčích modelů.



Obrázek 41 – Porovnání metody boostrapping s prostým dvouvrstvým GP na složené funkci sinus

Z grafu (Obrázek 41) je patrné, že i použití metody sampling ve dvouvrstvém GP přineslo v tomto případě znatelné zlepšení kvality výsledných modelů. Generované modely jsou srovnatelné prostým dvouvrstvým GP. Zvětšení počtu dílčích modelů na 60 přineslo pozitivní nicméně téměř zanedbatelnou změnu.

8.4.2 Pokusy se 150 jedinci

Pro potenciální zvýšení přesnosti výsledných modelů na složené funkci sinus došlo ke zvýšení počtu jedinců z 50 na 150. Tabulka 29 zobrazuje nastavení parametrů pro jednovrstvé GP a Tabulka 30 a Tabulka 31 zobrazují nastavení parametrů pro dvouvrstvé GP.

Tabulka 29 – Nastavení parametrů jednovrstvého GP složenou funkci sinus

Parametr	**Hodnota**
Velikost populace	150
Generační limit	230
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 30 – Nastavení parametrů 1. vrstvy GP pro složenou funkci sinus

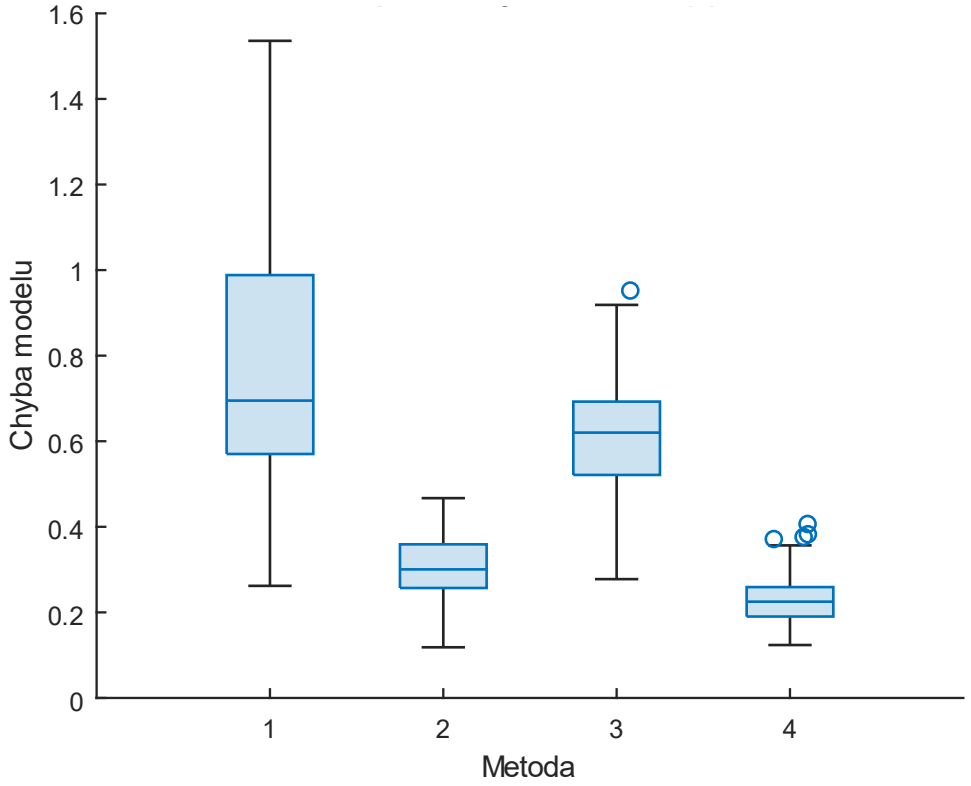
Parametr	**Hodnota**
Velikost populace	150
Generační limit	200
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv

Tabulka 31 – Nastavení parametrů 2. GP pro složenou funkci sinus

Parametr	**Hodnota**
Velikost populace	150
Generační limit	30
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*; +; –; %, inv, avg

Následující graf (Obrázek 42) zobrazuje výsledky těchto experimentů:

1. jednovrstvé GP 50 jedinců,
2. dvouvrstvé GP bez diverzifikace trénovacích dat 50 jedinců,
3. jednovrstvé GP 150 jedinců,
4. dvouvrstvé GP bez diverzifikace trénovacích dat 150 jedinců.



Obrázek 42 – Porovnání obou verzí GP s 50 a 150 jedinci na složené funkci sinus

Zvětšení počtu jedinců na 150 (u dvouvrstvého na obou vrstvách) zlepšilo kvalitu modelů generovaných jednovrstvým i prostým dvouvrstvým GP. Jednovrstvé GP se 150 jedinci

vytvářelo výsledné modely konzistentnějším způsobem než varianta s 50 jedinci. Prosté dvouvrstvé GP se 150 jedinci v tomto případě vygenerovalo přesnější modely než dvouvrstvé se 150 jedinci (a přesnější než dvouvrstvé s 50 jedinci) a také se snížil rozptyl kvality generovaných modelů.

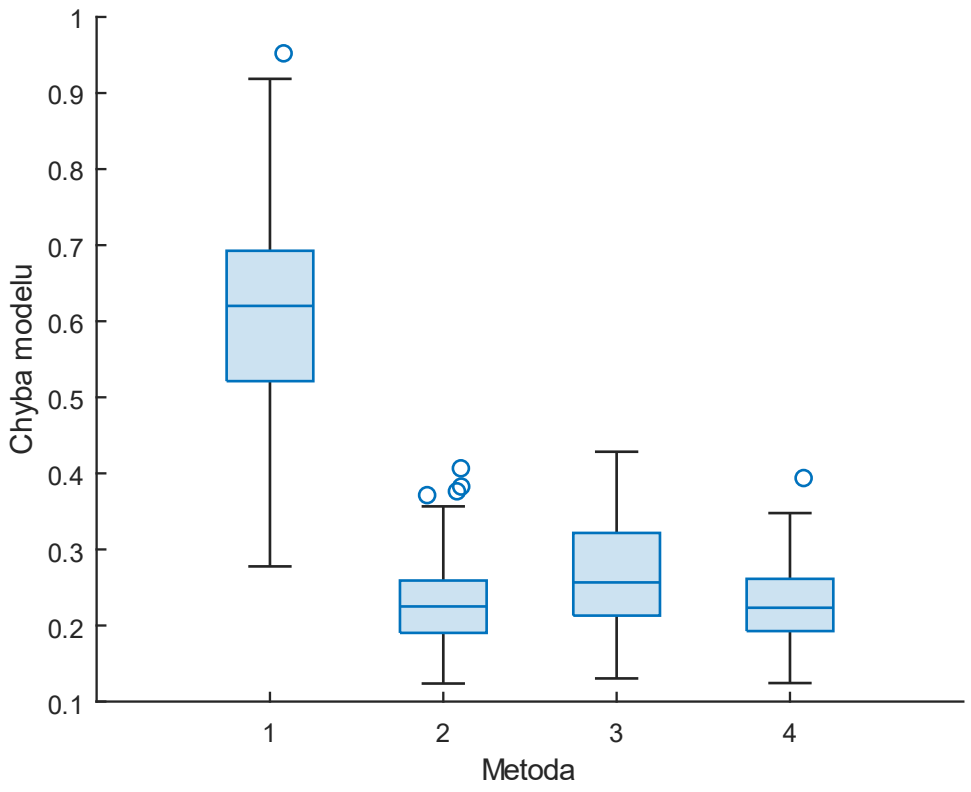
Metoda bootstrapping

Metoda boostrapping byla srovnávána s jednovrstvým GP a dvouvrstvým GP.

Konkrétně šlo o experimenty:

1. jednovrstvé GP 150 jedinců,
2. dvouvrstvé GP bez diverzifikace trénovacích dat 150 jedinců,
3. dvouvrstvé GP bootstrapping 30 % 150 jedinců,
4. dvouvrstvé GP bootstrapping 60 % 150 jedinců,

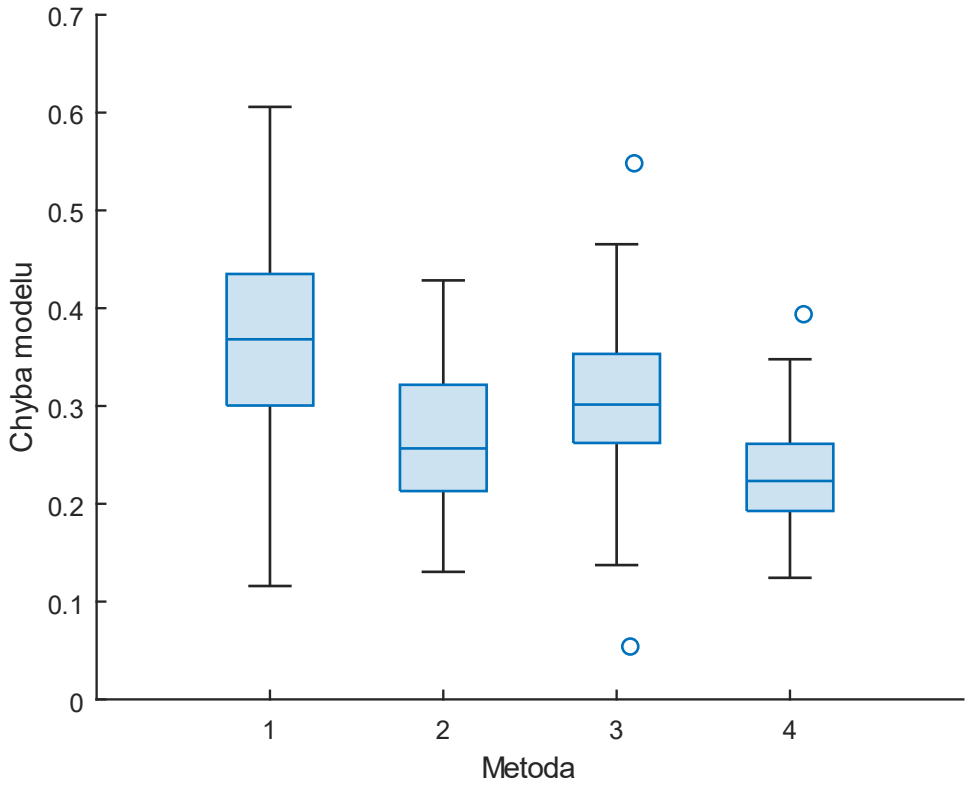
jejichž výsledky jsou vidět na následujícím grafu (Obrázek 43).



Obrázek 43 – Porovnání obou jednovrstvého GP a dvouvrstvého GP s metodou bootstrapping se 150 jedinci na složené funkci sinus

Na dalším grafu (Obrázek 44) je vidět porovnání dvouvrstvých s metodou boostrapping s 50 a 150 jedinci a dvěma různými velikostmi bootstrapu (30 % a 60 %). Konkrétně jde o experimenty:

1. dvouvrstvé GP bootstrapping 30 %, 50 jedinců,
2. dvouvrstvé GP bootstrapping 30 %, 150 jedinců,
3. dvouvrstvé GP bootstrapping 60 %, 50 jedinců,
4. dvouvrstvé GP bootstrapping 60 %, 150 jedinců.



Obrázek 44 – Porovnání dvouvrstvých GP s metodou bootstrapping s 50 a 150 jedinci na složené funkci sinus

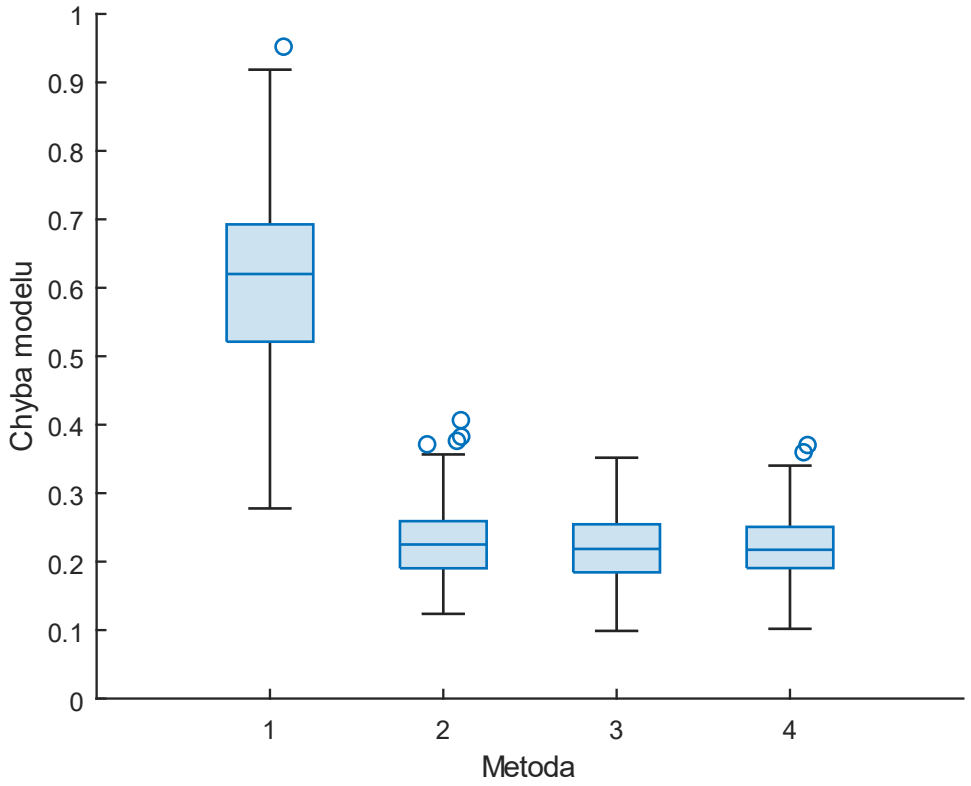
Z obou grafů (Obrázek 43 a Obrázek 44) je patrné, že zvětšení počtu jedinců (u dvouvrstvého na obou vrstvách) přineslo u všech nastavení pozitivní změny v kvalitě generovaných modelů. Dvouvrstvé GP znovu vytvořilo znatelně přesnější modely. Prosté dvouvrstvé GP vytvořilo modely kvalitativně srovnatelné s modely vytvořené dvouvrstvým GP s metodou bootstrapping s velikostí bootstrapu s hodnotou 60 %.

Metoda sampling

Metoda boostrapping byla srovnávána s jednovrstvým GP a dvouvrstvým GP.

Nejprve byly porovnány tyto konfigurace:

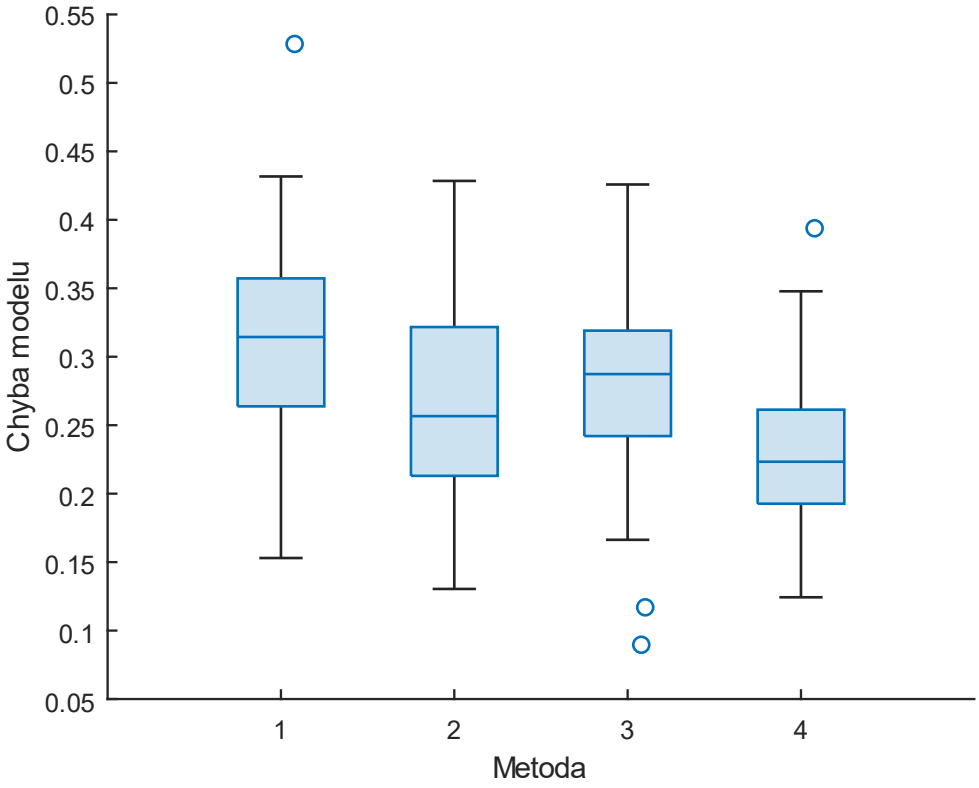
1. jednovrstvé GP 150 jedinců,
2. dvouvrstvé GP bez diverzifikace trénovacích dat, 150 jedinců,
3. dvouvrstvé GP sampling, 30 dílčích modelů, 150 jedinců,
4. dvouvrstvé GP sampling, 60 dílčích modelů, 150 jedinců.



Obrázek 45 – Porovnání obou jednovrstvého GP a dvouvrstvého GP s metodou sampling se 150 jedinci na složené funkci sinus

V následujícím grafu (Obrázek 46) lze vidět porovnání metody sampling s 150 jedinci s předchozí konfigurací s 50 jedinci, jde o tyto konfigurace:

1. dvouvrstvé GP sampling, 30 dílčích modelů, 50 jedinců,
2. dvouvrstvé GP sampling, 60 dílčích modelů, 50 jedinců,
3. dvouvrstvé GP sampling, 30 dílčích modelů, 150 jedinců,
4. dvouvrstvé GP sampling, 60 dílčích modelů, 150 jedinců.



Obrázek 46 – Porovnání dvouvrstvých GP s metodou sampling s 50 a 150 jedinci na složené funkci sinus

Dvouvrstvé GP s metodou sampling vytvořil znatelně přesnější modely než jednovrstvé GP i ve variantě se 150 jedinci. Metoda sampling se 150 jedinci znovu vygenerovala srovnatelné modely jako prosté dvouvrstvé GP se 150 jedinci. Zlepšení pomocí přidání jedinců na obě vrstvy výsledných modelů mezi odpovídajícími pokusy s metodou sampling není tak znatelné jako u metody bootstrapping.

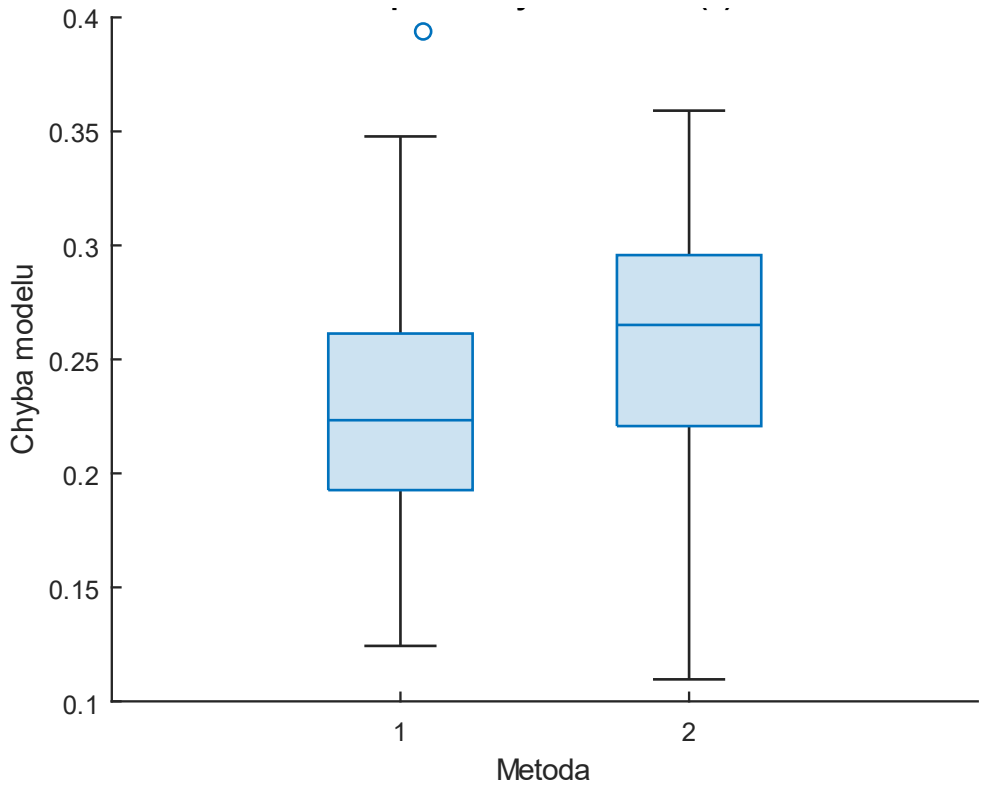
Pokus o zvětšení přesnosti modelů

Jelikož přesnost vytvářených modelů nebyla dostatečná, byly navrženy experimenty s rozšířenou množinou funkcí a s navýšenými populacemi a generacemi.

Šlo o tyto konfigurace:

1. dvouvrstvé GP bootstrapping 60 %, 150 jedinců,
2. dvouvrstvé GP bootstrapping 60 %, 150 jedinců + sqrt a pow2.

Výsledky jsou vidět v následujícím grafu (Obrázek 47).

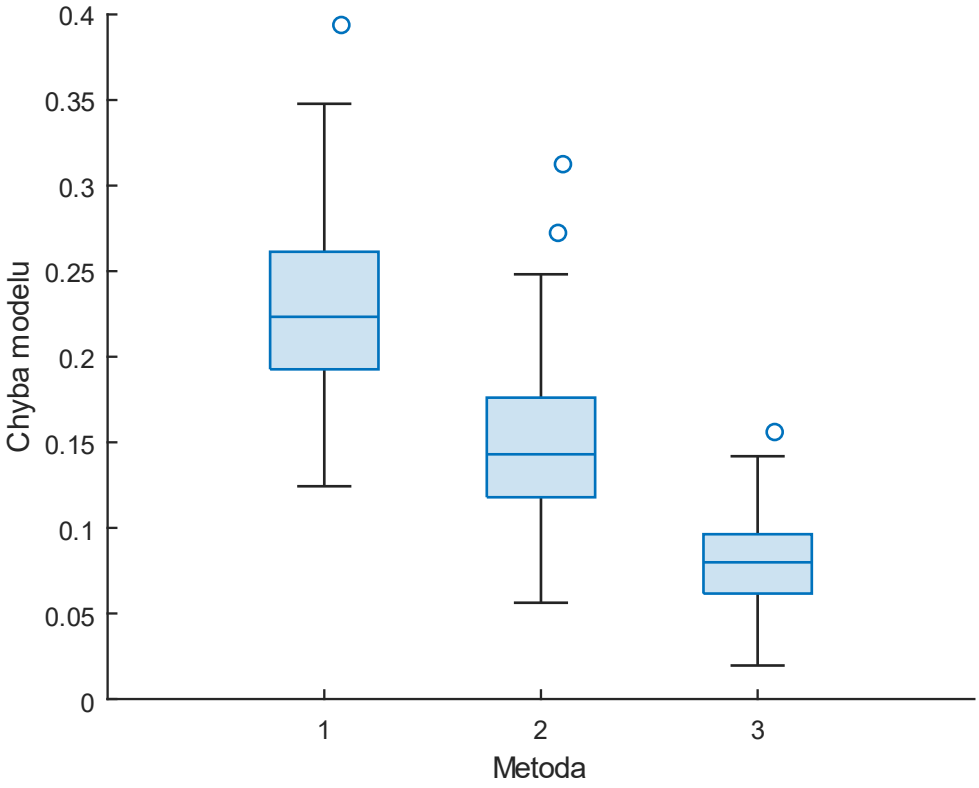


Obrázek 47 – Přidání funkcí sqrt a pow2 k metodě bootstrapping na složené funkci sinus

V tomto experimentu po přidání funkcí sqrt a pow2 došlo dokonce ke zhoršení kvality generovaných modelů. Dále šlo o změny počtu generací a jedinců. Pro zlepšení přesnosti byly navrženy experimenty s větším počtem generací a jedinců. V poslední konfiguraci došlo i k navýšení počtu dílčích modelů na 90. Výsledky těchto experimentů jsou vidět v následujícím grafu (Obrázek 48).

Experimenty:

1. dvouvrstvé GP bootstrapping 60 %, 150 jedinců,
2. dvouvrstvé GP bootstrapping 60 %, 150 jedinců a 200 generací na první vrstvě, 350 jedinců a 240 generací na druhé vrstvě,
3. dvouvrstvé GP bootstrapping 60 %, 150 jedinců a 230 generací na první vrstvě, 500 jedinců a 350 generací na druhé vrstvě + sqrt, pow2 funkce na první vrstvě, 90 dílčích modelů.



Obrázek 48 – Porovnání různých konfigurací metody bootstrapping (60 %) na složené funkci sinus

Při navýšení parametrů dvouvrstvého GP s metodou bootstrapping se chyba výsledných modelů viditelně snížila a v poslední konfiguraci se dostala k hranici 0,1. Tato přesnost ještě není dostatečná. Složená funkce sinus se jeví jako složitý problém, pokud samotná funkce sinus není zahrnuta v množině funkcí.

8.5 Teploty Dillí

V poslední sérii experimentů byla vyzkoušena reálná datová sada zachycující teploty v Indickém městě Dillí.

Tabulka 32 obsahuje nastavení parametrů pro jednovrstvé GP a Tabulka 33 a Tabulka 34 obsahuje nastavení parametrů pro dvouvrstvé GP. Změnou je nestejné nastavení množiny funkcí na první a druhé vrstvě u dvouvrstvé varianty.

Tabulka 32 – Nastavení parametrů jednovrstvého GP pro datovou sadu s teplotou v Dillí

Parametr	**Hodnota**
Velikost populace	50 a 250
Generační limit	530
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*, +, -, %, inv, sqrt, pow2

Tabulka 33 – Nastavení parametrů 1. vrstvy GP pro datovou sadu s teplotou v Dillí

Parametr	**Hodnota**
Počet dílčích modelů	30
Velikost populace	50 a 250
Generační limit	500
Množina terminálů	x; –1; 1; 2; 3;
Množina neterminálů	*, +, -, %, inv, sqrt, pow2

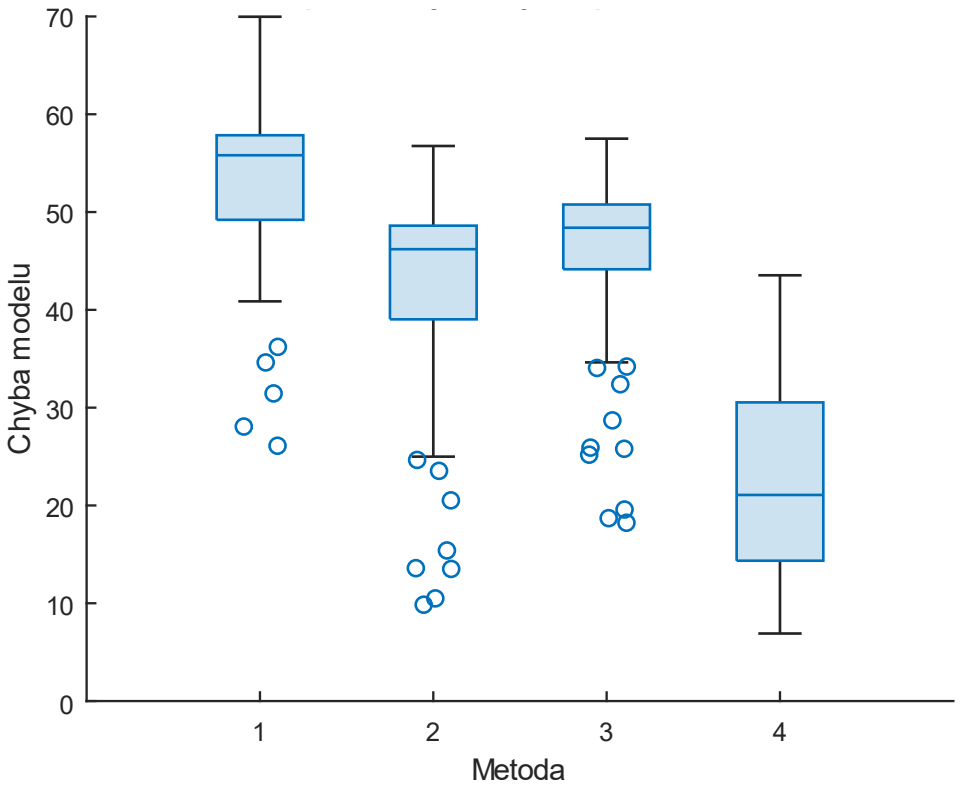
Tabulka 34 – Nastavení parametrů 2. GP pro datovou sadu s teplotou v Dillí

Parametr	**Hodnota**
Velikost populace	50 a 250
Generační limit	30
Množina terminálů	x, –1; 1; 2; 3;
Množina neterminálů	*, +, -, avg

První sada experimentů s touto datovou sadou vypadala následovně:

1. jednovrstvé GP, 50 jedinců,
2. dvouvrstvé GP, bootstrapping 60 %, 50 jedinců na obou vrstvách,
3. jednovrstvé GP, 250 jedinců,
4. dvouvrstvé GP, bootstrapping 60 %, 250 jedinců na obou vrstvách.

Výsledky těchto experimentů jsou zobrazeny na následujícím grafu (Obrázek 49).



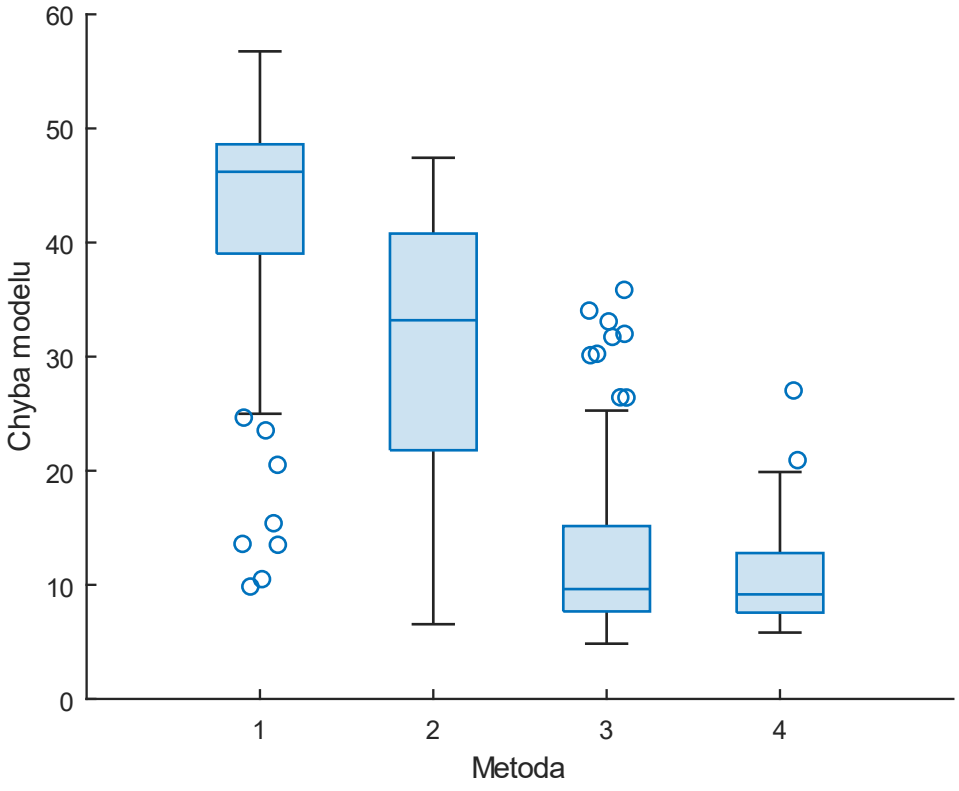
Obrázek 49 – Porovnání jednovrstvého GP s dvouvrstvým GP s metodou bootstrapping (60 %) na datové sadě s teplotami v Dillí

Z něj lze vyčíst, že na reálných datech s teplotami se dvouvrstvému GP s metodou bootstrapping podařilo zlepšit přesnost výsledných modelů oproti jednovrstvé variantě. Větší zlepšení nastalo o pokusů s 250 jedinci (u dvouvrstvého na první vrstvě). Zatímco u prvních třech kofiguracích byly modely s chybou pod hodnotou 30 pouze výjimečné (odlehlé hodnoty), v případě dvouvrstvého GP s metodou bootstrapping (60 %) byly modely s chybou pod hodnotou 30 časté (samotný horní kvartil se pohyboval okolo hodnoty 30). Zároveň u této konfigurace výrazně ubylo odlehlých hodnot.

8.5.1 Zvýšení výkonu

Pro zvýšení výkonu dvouvrstvého GP na této datového sadě byly navrženy tyto experimenty:

1. dvouvrstvé GP, bootstrapping 60 %, 50 jedinců,
2. dvouvrstvé GP, bootstrapping 60 %, 50 jedinců na první vrstvě, 250 jedinců a 120 generací na druhé vrstvě,
3. dvouvrstvé GP, bootstrapping 60 %, 150 jedinců na první vrstvě, 250 jedinců a 120 generací na druhé vrstvě, množina funkcí na první vrstvě = { *, +, -, %, inv, pow2, pow3, sqrt, sin, abs },
4. dvouvrstvé GP, bootstrapping 60 %, 250 jedinců obě vrstvy, generace 500 a 30, množina funkcí na první vrstvě = { *, +, -, %, inv, pow2, pow3, sqrt, sin, abs}, 60 dílčích modelů.



Obrázek 50 – Porovnání dvouvrstvých GP s metodou bootstrapping s rozšířenými parametry

Postupným zvyšování hodnot parametrů se chyba snížila na relativně nízkou hodnotu (vzhledem k charakteru dat, použitým funkcím a použité implementace GP s celočíselnými konstantami). Chyba odpovídá spíše naučení se trendu bez přeučení. Poslední nastavení vygenerovalo nejkonzistentnější modely ze všech testovaných nastavení.

9 ZÁVĚR

Cílem nebylo najít přesné modely, ale prozkoumat dynamiku a výhody dvouvrstvého algoritmu genetického programování na těžkých problémech. Z tohoto důvodu byly vybírány pokusy tak, aby hledání matematických modelů a prohledávání stavového prostoru bylo pro genetické programování složité.

V případě hledání matematického modelu pro hyperbolu nebyl v množině funkcí přítomen žádný operátor nebo funkce pro dělení (ani % či inverze). Hledání jednoduché funkce sinus používalo relativně jednoduché funkce. Při hledání funkce složeného sinu také byla funkce sinus z množiny funkcí vynechána. A celý algoritmus genetického programování používal pouze celočíselné konstanty. Reálné konstanty tak genetické programování muselo vyvíjet evolucí za pomocí vhodné kombinace celočíselných konstant a funkcí dělení, inverze nebo funkce odmocniny.

Dvouvrstvé genetické programování spojené s technikami sborového učení na provedených experimentech ukázalo potenciál pro zlepšení výkonu genetického programování. Prosté dvouvrstvé genetické programování ve zvolených nastaveních vytvářelo v průměru přesnější a konzistentnější matematické modely zkoumaných datových sad. Některé přesnosti vygenerovaných stromů nedosahovaly použitelných výsledků.

Také dvouvrstvé genetické programování spojené s metodami sampling, bootstrapping a metodou náhodného pohyblivého okna dosahovaly v průměru lepších výsledků než jednovrstvé alternativy. Zatímco zkoumaná nastavení metody sampling a metody bootstrapping měly podobný výkon, použitá nastavení náhodného pohyblivého okna tak přesné modely nevytvářela. Přesto při použití metody náhodného pohyblivého okna docházelo k relativnímu zlepšení.

Se zvětšující se velikostí bootstrapu (podmnožin trénovacích dat) u metody bootstrapping zvyšovalo průměrnou přesnost vytvářených modelů. Větší velikost bootstrapu však také znamenala větší počet vyhodnocení funkce fitness, tudíž i delší výpočetní čas, než metoda bootstrapping s menší velikostí bootstrapu.

Na přesnost výsledných modelů vytvářených pomocí dvouvrstvého genetického programování mělo v prováděných experimentech pozitivní vliv i zvyšování počtu dílčích modelů. Druhá vrstva pozitivně reagovala nejen na počet ale i na větší rozmanitost modelů vytvořených první vrstvou. Velký vliv na kvalitu výsledných modelů měl i výběr množiny funkcí a to na obou vrstvách.

9.1 Perspektivy dalšího vývoje

V budoucím výzkumu by bylo dobré prozkoumat detailněji dynamiku nastavení jednotlivých parametrů a hyperparametrů, zejména počet dílčích modelů, počet generací a jedinců na obou vrstvách. Dalším předmětem zkoumání by mohla být vhodnost jednotlivých funkcí na druhé vrstvě. Dále by bylo zajímavé zkusit dvouvrstvé programování na datech obsahujících šum,

na datových sadách s větším počtem proměnných a problémech z kategorie velkých dat (big data).

Dalším zajímavým námětem pro výzkum by bylo použití různých metod pro tvorbu dílčích modelů na první vrstvě současně, čímž by se zajistila větší diverzita dílčích modelů. Dalším předmětem zkoumání by mohla být cílená diverzifikace podmnožin funkcí pro tvorbu dílčích modelů. Při hledání modelů vícerozměrných dat by bylo potřeba přidat i metody pro diverzifikaci vstupních proměnných.

Vytváření dílčích modelů může být díky většímu počtu vyhodnocení časově náročnější než přímý jednovrstvý přístup. Dvouvrstvé genetické programování je však z principu jednoduše paralelizovatelné. Zejména vytváření dílčích modelů by bylo možné implementovat na více procesorech nebo počítačích souběžně. Tím by mohlo dojít i k úspoře výpočetního času.

Další zkoumání by zasloužila možnost přidání dalších vrstev a vliv na přesnost výsledných modelů. Přidáním dalších vrstev by se genetické programování přiblížilo principům hlubokého učení a vznikl by tak termín vícevrstvé, respektive hluboké genetické programování.

V průběhu testování dvouvrstvého algoritmu genetického programování a návrhu metodiky byly využíván i výpočetní cluster: gridová infrastruktura MetaCentrum [69]. Výpočty byly úspěšně aplikovány i v tomto výpočetním prostředí a jeho využití je plánováno i v rámci budoucího vývoje.

10 BIBLIOGRAFIE

[1] GOODFELLOW, Ian, Yoshua BENGIO a Aaron COURVILLE. Deep learning. Cambridge, MA: MIT press, [2016]. Adaptive computation and machine learning series. ISBN 9780262035613.

[2] RUSSELL, Stuart J. a Peter NORVIG. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River: Prentice Hall, 2010. Prentice Hall series in artificial intelligence. ISBN 978-0136042594.

[3] KOTANCHEK, Mark E., Ekaterina Y. VLADISLAVLEVA a Guido F. SMITS. Symbolic Regression Via Genetic Programming as a Discovery Engine: Insights on Outliers and Prototypes. RIOLO, Rick, Una-May O'REILLY a Trent MCCONAGHY, ed. Genetic Programming Theory and Pr.

[4] BHOWAN, Urvesh, Mark JOHNSTON, MENGJIE ZHANG a XIN YAO. Reusing Genetic Programming for Ensemble Selection in Classification of Unbalanced Data. IEEE Transactions on Evolutionary Computation [online]. 2014, 18(6), 893-908 [cit. 2021-03-25]. ISSN 1089-778X.

[5] KOZA, John R. Genetic programming: On the programming of computers by means of natural selection. Cambridge: Bradford Book, 1992. ISBN 978-0262111706.

[6] AFFENZELLER, Michael. Genetic algorithms and genetic programming: modern concepts and practical applications. Boca Raton: CRC Press, c2009. ISBN 978-1584886297.

[7] HOLLAND, John. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Reprint edition. Cambridge, MA, United States: MIT Press, 1992. ISBN 9780262275552..

[8] ZELINKA, Ivan. Evoluční výpočetní techniky: principy a aplikace. Praha: BEN - technická literatura, 2009. ISBN 9788073002183.

[9] HYNEK, Josef. Genetické algoritmy a genetické programování. Praha: Grada, 2008. Průvodce (Grada). ISBN 978-80-247-2695-3.

[10] GOLDBERG, David Edward. Genetic algorithms in search, optimization, and machine learning. Boston: Addison-Wesley, 1989. ISBN 978-0201157673..

[11] LANGDON, William B. a Riccardo POLI. Foundations of genetic programming. Berlin: Springer-Verlag, 2002. ISBN 978-3540424512..

[12] POLI, Riccardo, W. B. LANGDON, Nicholas F. MCPHEE a John R. KOZA. A field guide to genetic programming. [S.l.: Lulu Press], 2008. ISBN 978-1-4092-0073-4..

[13] BOHANEC, Marko a Ivan BRATKO. Machine Learning [online]. 15(3), 223-250 [cit. 2021-03-25]. ISSN 08856125. Dostupné z: doi:10.1023/A:1022685808937.

[14] BRAMEIER, Marcus a Wolfgang BANZHAF. Linear genetic programming. New York: Springer, 2007. ISBN 978-0-387-31029-9..

[15] MILLER, Julian F., ed. Cartesian Genetic Programming [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011 [cit. 2021-03-25]. Natural Computing Series. ISBN 978-3-642-17309-7. Dostupné z: doi:10.1007/978-3-642-17310-3.

[16] OLTEAN, M. a D. DUMITRESCU, Multi Expression Programming. Technical Report; Babeş-Bolyai University: Cluj-Napoca, Romania, 2002..

[17] KUBALÍK, Jiří a Robert BABUŠKA. An Improved Single Node Genetic Programming for Symbolic Regression. In: Proceedings of the 7th International Joint Conference on Computational Intelligence [online]. SCITEPRESS - Science and and Technology Publications,, 2015, 2015, s. 244-251 [cit. 2021-03-25]. ISBN 978-989-758-157-1. Dostupné z: doi:10.5220/0005598902440251.

[18] UDRESCU, Silviu-Marian a Max TEGMARK. AI Feynman: A physics-inspired method for symbolic regression. Science Advances [online]. 2020, 6(16) [cit. 2021-03-25]. ISSN 2375-2548. Dostupné z: doi:10.1126/sciadv.aay2631.

[19] LIU, Huan a Hiroshi MOTODA, ed. Feature Extraction, Construction and Selection [online]. Boston, MA: Springer US, 1998 [cit. 2021-03-25]. ISBN 978-1-4613-7622-4. Dostupné z: doi:10.1007/978-1-4615-5725-8.

[20] GUYON, I. a A. ELISSEEFF. An introduction to variable and feature selection. J. Mach. Learn. Res. 2003,3, 1157–1182..

[21] KRAWIEC, Krzysztof. Genetic Programming-based Construction of Features for Machine Learning and Knowledge DiscoveryGenetic Programming and Evolvable Machines [online]. 3(4), 329-343 [cit. 2021-03-25]. ISSN 13892576. Dostupné z: doi:10.1023/A:1020984725014.

[22] ICKE, Ilknur a Joshua C. BONGARD. Improving genetic programming based symbolic regression using deterministic machine learning. In: 2013 IEEE Congress on Evolutionary Computation [online]. IEEE, 2013, 2013, s. 1763-1770 [cit. 2021-03-25]. ISBN 978-1-4799-.

[23] MATTEO R. a V. GIOGIO, Ensemble methods: a review, no. January 2012. 2001..

[24] WOŹNIAK, Michał, Manuel GRAÑA a Emilio CORCHADO. A survey of multiple classifier systems as hybrid systems. Information Fusion [online]. 2014, 16, 3-17 [cit. 2021-03-25]. ISSN 15662535. Dostupné z: doi:10.1016/j.inffus.2013.04.006.

[25] ONG, Yew S., Prasanth B. NAIR a Andrew J. KEANE. Evolutionary Optimization of Computationally Expensive Problems via Surrogate Modeling. AIAA Journal [online]. 2003, 41(4), 687-696 [cit. 2021-03-25]. ISSN 0001-1452. Dostupné z: doi:10.2514/2.1999.

[26] JIN, Yaochu. Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation [online]. 2011, 1(2), 61-70 [cit. 2021-03-25]. ISSN 22106502. Dostupné z: doi:10.1016/j.swevo.2011.05.001.

[27] PERIĆ, Vladimir. Symbolic regression as a surrogate model in evolutionary algorithms. Praha, 2016. Diplomová práce. ČVUT. Vedoucí práce Ing. Petr Pošík, Ph.D..

[28] HEJL, Vojtěch. Selection of surrogate models for evolutionary black-box optimization in noisy environment. Praha, 2018. Diplomová práce. ČVUT. Vedoucí práce Doc. Ing. RNDr. Martin Holeňa, CSc..

[29] MICHALEWICZ, Zbigniew. Genetic Algorithms + Data Structures = Evolution Programs [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996 [cit. 2021-03-25]. ISBN 978-3-642-08233-7. Dostupné z: doi:10.1007/978-3-662-03315-9.

[30] DE JONG, Kenneth A. Evolutionary computation: a unified approach. Cambridge: MIT Press, 2006. ISBN 0-262-04194-4..

[31] WHITLEY, Darrell, V. Scott GORDON a Keith MATHIAS. Lamarckian evolution, the Baldwin effect and function optimization. DAVIDOR, Yuval, Hans-Paul SCHWEFEL a Reinhard MÄNNER, ed. Parallel Problem Solving from Nature — PPSN III [online]. Berlin, Heidelberg:.

[32] LE, Nam, Anthony BRABAZON a Michael O'NEILL. How the "Baldwin Effect" Can Guide Evolution in Dynamic Environments. FAGAN, David, Carlos MARTÍN-VIDE, Michael O'NEILL a Miguel A. VEGA-RODRÍGUEZ, ed. Theory and Practice of Natural Computing [online]. Cham: S.

[33] RED'KO, Vladimir G., Oleg P. MOSALOV a Danil V. PROKHOROV. A model of evolution and learning. Neural Networks [online]. 2005, 18(5-6), 738-745 [cit. 2021-03-25]. ISSN 08936080. Dostupné z: doi:10.1016/j.neunet.2005.06.005.

[34] Turney, P. D. 2002. Myths and legends of the Baldwin effect. arXiv: cs/0212036. Retrieved from https://arxiv.org/abs/cs/0212036.

[35] HINTON, G. E. a NOWLAN, S. J. 1987. How learning can guide evolution. Complex

Systems 1, s. 495–502.

[36] FRENCH, R. a MESSINGER, A. 1994. Genes, Phenes and the Baldwin Effect: Learning and Evolution in a Simulated Population. In Artificial Life IV. MIT Press, Cambridge, MA, USA.

[37] LE, Nam, Michael O'NEILL a Anthony BRABAZON. The Baldwin Effect Reconsidered Through the Prism of Social Learning. In: 2018 IEEE Congress on Evolutionary Computation (CEC) [online]. IEEE, 2018, 2018, s. 1-8 [cit. 2021-03-25]. ISBN 978-1-5090-6017-7., Dostupné z: doi:10.1109/CEC.2018.8477654.

[38] LE, Nam, Michael O'NEILL a Anthony BRABAZON. Adaptive Advantage of Learning Strategies: A Study Through Dynamic Landscape. AUGER, Anne, Carlos M. FONSECA, Nuno LOURENÇO, Penousal MACHADO, Luís PAQUETE a Darrell WHITLEY,, ed. Parallel Problem Solving from Nature – PPSN XV [online]. Cham: Springer International Publishing, 2018, 2018-08-21, s. 387-398 [cit. 2021-03-25]. Lecture Notes in Computer Science. ISBN 978-3-319-99258-7. Dostupné z: doi:10.1007/978-3-319-99259-4_31.

[39] ANDERSON, Russell W. Learning and evolution: A quantitative genetics approach. Journal of Theoretical Biology [online]. 1995, 175(1), 89-101 [cit. 2021-03-25]. ISSN 00225193. Dostupné z: doi:10.1006/jtbi.1995.0123.

[40] FERNANDEZ, T. a M. EVETT: Numeric Mutation as an Improvement to Symbolic Regression in Genetic Programming. In: Porto, V.W., Waagen, D. (eds.) EP 1998. LNCS, vol. 1447, s. 251–260. Springer, Heidelberg (1998).

[41] RAIDL, G. A Hybrid GP Approach for Numerically Robust Symbolic Regression. In Proc. of the 1998 Genetic Programming Conference. Madison, Wisconsin, 1998, s. 323–328.

[42] SCHOENAUER M., LAMY, B. a F. JOUVE: Identification of Mechanical Behaviour by Genetic Programming Part II: Energy formulation. Technical report, Ecole Polytechnique, France, 1995.

[43] GRAY, G.J., Y. LI, D.J. MURRAY-SMITH a K.C. SHARMAN. Structural system identification using genetic programming and a block diagram oriented simulation tool. Electronics Letters [online]. 1996, 32(15) [cit. 2021-03-25]. ISSN 00135194., Dostupné z: doi:10.1049/el:19960888.

[44] HASHIMOTO, Nozomi, Nobuhiko KONDO, Toshiharu HATANAKA a Katsuji UOSAKI. Nonlinear System Modeling by Hybrid Genetic Programming. IFAC Proceedings Volumes [online]. 2008, 41(2), 4606-4611 [cit. 2021-03-25]. ISSN 14746670. Dostupné z: doi:10.3182/20080706-5.

[45] IBA, H., T. SATO a H. DEGARIS. Recombination guidance for numerical genetic programming. In: Proceedings of 1995 IEEE International Conference on Evolutionary Computation [online]. IEEE, 1995, s. 97- [cit. 2021-03-25]. ISBN 0-7803-2759-4., Dostupné z: doi:10.1109/ICEC.1995.489292.

[46] NIKOLAEV, N.Y. a H. IBA. Adaptive Learning of Polynomial Networks [online]. Boston: Kluwer Academic Publishers, 2006 [cit. 2021-03-25]. Genetic and Evolutionary Computation. ISBN 0-387-31239-0. Dostupné z: doi:10.1007/0-387-31240-4.

[47] BRANDEJSKY, Tomas. Dependency of GPA-ES Algorithm Efficiency on ES Parameters Optimization Strength. ZELINKA, Ivan, Pavel BRANDSTETTER, Tran TRONG DAO, Vo HOANG DUY a Sang Bong KIM, ed. AETA 2018 - Recent Advances in Electrical Engineering, and Related Sciences: Theory and Application [online]. Cham: Springer International Publishing, 2020, 2020-04-13, s. 294-302 [cit. 2021-03-25]. Lecture Notes in Electrical Engineering. ISBN 978-3-030-14906-2. Dostupné z: doi:10.1007/978-3-030-14907-9_29.

[48] MERTA, Jan. Hybrid Symbolic Regression with the Bison Seeker Algorithm. MENDEL [online]. 2019, 25(1), 79-86 [cit. 2021-03-25]. ISSN 2571-3701. Dostupné z: doi:10.13164/mendel.2019.1.079.

[49] HINCHLIFFE, Mark. Dynamic Modelling Using Genetic Programming. Newcastle, 2001. Disertační práce. University of Newcastle upon Tyne.

[50] BETTENHAUSEN, K.D. Self-organizing structured modelling of a biotechnological fed-batch fermentation by means of genetic programming. In: 1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA), [online]. IEE, 1995, 1995, s. 481-486 [cit. 2021-03-25]. Dostupné z: doi:10.1049/cp:19951095.

[51] IBA, H.: Bagging, boosting, and bloating in genetic programming, 1999. Dostupné z: https://dl.acm.org/doi/pdf/10.5555/2934046.2934063.

[52] KEIJZER, Maarten. Improving Symbolic Regression with Interval Arithmetic and Linear Scaling. RYAN, Conor, Terence SOULE, Maarten KEIJZER, Edward TSANG, Riccardo POLI a Ernesto COSTA, ed. Genetic Programming [online]. Berlin, Heidelberg: Springer Berlin, Heidelberg, 2003, 2003-5-13, s. 70-82 [cit. 2021-03-26]. Lecture Notes in Computer Science. ISBN 978-3-540-00971-9. Dostupné z: doi:10.1007/3-540-36599-0_7.

[53] GANDOMI, Amir Hossein a Amir Hossein ALAVI. Multi-stage genetic programming: A new strategy to nonlinear system modeling. Information Sciences [online]. 2011, 181(23), 5227-5239 [cit. 2021-03-26]. ISSN 00200255. Dostupné z: doi:10.1016/j.ins.2011.07.026.

[54] WILLIS, Mark, Hugo HIDEN, Mark HINCHLIFFE, Ben MCKAY a Geoffrey W. BARTON. Systems modelling using genetic programming. Computers & Chemical Engineering [online]. 1997, 21, S1161-S1166 [cit. 2021-03-26]. ISSN 00981354., Dostupné z: doi:10.1016/S0098-1354(97)87659-4.

[55] DANANDEH MEHR, Ali a Vahid NOURANI. Season Algorithm-Multigene Genetic Programming: A New Approach for Rainfall-Runoff Modelling. Water Resources Management [online]. 2018, 32(8), 2665-2679 [cit. 2021-03-26]. ISSN 0920-4741, Dostupné z: doi:10.1007/s11269-018-1951-3.

[56] ŽEGKLITZ, Jan a Petr POŠÍK. Learning Linear Feature Space Transformations in Symbolic Regression. 2017. Dostupné také z: https://arxiv.org/pdf/1704.05134.pdf.

[57] ŽEGKLITZ, Jan a Petr POŠÍK. Sequential Model Building in Symbolic Regression. ITAT 2019: Conference Information Technologies - Applications and Theory. 2019. Dostupné také z: http://ceur-ws.org/Vol-2473/paper5.pdf.

[58] MOUSAVI ASTARABADI, Samaneh Sadat a Mohammad Mehdi EBADZADEH. A decomposition method for symbolic regression problems. Applied Soft Computing [online]. 2018, 62, 514-523 [cit. 2021-03-26]. ISSN 15684946. Dostupné z: doi:10.1016/j.asoc.2017.10.041.

[59] ARNALDO, Ignacio, Krzysztof KRAWIEC a Una-May O'REILLY. Multiple regression genetic programming. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation [online]. New York, NY, USA: ACM, 2014, 2014-07-12, s. 879-886, [cit. 2021-03-26]. ISBN 9781450334723. Dostupné z: doi:10.1145/2739480.2754693.

[60] ARNALDO, Ignacio, Una-May O'REILLY a Kalyan VEERAMACHANENI. Building Predictive Models via Feature Synthesis. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation [online]. New York, NY, USA: ACM, 2015, 2015-07-11,, s. 983-990 [cit. 2021-03-26]. ISBN 9781450334723. Dostupné z: doi:10.1145/2739480.2754693.

[61] MORAGLIO, Alberto, Krzysztof KRAWIEC a Colin G. JOHNSON. Geometric Semantic Genetic Programming. COELLO, Carlos A. Coello, Vincenzo CUTELLO, Kalyanmoy DEB, Stephanie FORREST, Giuseppe NICOSIA a Mario PAVONE,, ed. Parallel Problem Solving from Nature - PPSN XII [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 2012, s. 21-31 [cit. 2022-06-25]. Lecture Notes in Computer Science. ISBN 978-3-642-32936-4. Dostupné z: doi:10.1007/978-3-642-32937-1_3.

[62] CASTELLI, Mauro, Leonardo TRUJILLO, Leonardo VANNESCHI, Sara SILVA, Emigdio Z-FLORES a Pierrick LEGRAND. Geometric Semantic Genetic Programming with Local Search. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, [online]. New York, NY, USA: ACM, 2015, 2015-07-11, s.

999-1006 [cit. 2021-03-26]. ISBN 9781450334723. Dostupné z: doi:10.1145/2739480.2754795.

[63] ŽEGKLITZ, Jan a Petr POŠÍK. Sequential Model Building in Symbolic Regression. ITAT 2019: Conference Information Technologies - Applications and Theory. 2019. Dostupné také z: http://ceur-ws.org/Vol-2473/paper5.pdf.

[64] ICKE, Ilknur a Joshua C. BONGARD. Improving genetic programming based symbolic regression using deterministic machine learning. In: 2013 IEEE Congress on Evolutionary Computation [online]. IEEE, 2013, 2013, s. 1763-1770 [cit. 2021-03-26]., ISBN 978-1-4799-0454-9. Dostupné z: doi:10.1109/CEC.2013.6557774.

[65] CHEN, Chen, Changtong LUO a Zonglin JIANG. Elite bases regression: A real-time algorithm for symbolic regression. In: 2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD) [online]. IEEE, 2017, 2017,, s. 529-535 [cit. 2021-03-26]. ISBN 978-1-5386-2165-3. Dostupné z: doi:10.1109/FSKD.2017.8393325.

[66] FUSTING, Christopher Winter. Temporal Feature Selection with Symbolic Regression. Vermont, 2017. Disertační práce. University of Vermont.

[67] MERTA Jan a Tomáš BRANDEJSKÝ, Computational Statistics and Mathematical Modeling Methods in Intelligent Systems [online]. Cham: Springer International Publishing, 2019 [cit. 2021-03-26]. Advances in Intelligent Systems and Computing. ISBN 978-3-030-31361-6. Dostupné z: doi:10.1007/978.

[68] KAZIKOVA, Anezka, Michal PLUHACEK a Roman SENKERIK. Regarding the Behavior of Bison Runners Within the Bison Algorithm. MENDEL [online]. 2018, 24(1), 63-70 [cit. 2021-03-31]. ISSN 2571-3701. Dostupné z: doi:10.13164/mendel.2018.1.063.

[69] „MetaCentrum VO,“ b. r.. [Online]. Available: https://metavo.metacentrum.cz/cs/. [Přístup získán 2020-08-23].

SEZNAM PUBLIKACÍ AUTORA

Článek v odborném časopisu:

DOLEŽEL, Petr, Filip HOLÍK, Jan MERTA a Dominik ŠTURSA. Optimization of a Depiction Procedure for an Artificial Intelligence-Based Network Protection System Using a Genetic Algorithm. Applied Sciences [online]. 2021, 11(5). ISSN 2076-3417. Dostupné z: doi:10.3390/app11052012

Články ve sborníku na téma GA, GP a AI:

HRBEK, Václav a Jan MERTA. Searching the Hyper-heuristic for the Traveling Salesman Problem with Time Windows by Genetic Programming. SILHAVY, Radek, Petr SILHAVY a Zdenka PROKOPOVA, ed. Software Engineering Perspectives in Intelligent Systems [online]. Cham: Springer International Publishing, 2020, 2020-12-16, s. 939-946. Advances in Intelligent Systems and Computing. ISBN 978-3-030-63321-9. Dostupné z: doi:10.1007/978-3-030-63322-6_81

MERTA, Jan. Hybrid Symbolic Regression with the Bison Seeker Algorithm. MENDEL [online]. 2019, 25(1), s. 79-86. ISSN 2571-3701. Dostupné z: doi:10.13164/mendel.2019.1.079.

MERTA Jan a Tomáš BRANDEJSKÝ, Computational Statistics and Mathematical Modeling Methods in Intelligent Systems [online]. Cham: Springer International Publishing, 2019. Advances in Intelligent Systems and Computing. ISBN 978-3-030-31361-6. Dostupné z: doi:10.1007/978.

MERTA, Jan a Tomas BRANDEJSKY. Three-Dimensional Genetic Algorithm for the Kirkman's Schoolgirl Problem. MENDEL [online]. 2018, 24(1), 25-30. ISSN 2571-3701. Dostupné z: doi:10.13164/mendel.2018.1.025

HOLIK, Filip, Petr DOLEZEL, Jan MERTA a Dominik STURSA. Development of Artificial Intelligence Based Module to Industrial Network Protection System. ARAI, Kohei, Supriya KAPOOR a Rahul BHATIA, ed. Intelligent Systems and Applications [online]. Cham: Springer International Publishing, 2021, 2021-08-25, s. 229-240. Advances in Intelligent Systems and Computing. ISBN 978-3-030-55189-6. Dostupné z: doi:10.1007/978-3-030-55190-2_18

DOLEZEL, Petr, Dominik STURSA, Daniel HONC, Jan MERTA, Veronika ROZSIVALOVA, Ladislav BERAN a Ivo HORA. Counting Livestock with Image Segmentation Neural Network. HERRERO, Álvaro, Carlos CAMBRA, Daniel URDA, Javier SEDANO, Héctor QUINTIÁN a Emilio CORCHADO, ed. 15th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2020) [online]. Cham: Springer International Publishing, 2021, 2021-08-29, s. 237-244. Advances in Intelligent Systems and Computing. ISBN 978-3-030-57801-5. Dostupné z: doi:10.1007/978-3-030-57802-2_23

MERTA, Jan a Jan FIKEJZ. Utilization of Machine Learning to Detect Sudden Water Leakage for Smart Water Meter. In: 2019 29th International Conference Radioelektronika (RADIOELEKTRONIKA) [online]. IEEE, 2019, 2019, s. 1-5. ISBN 978-1-5386-9322-3. Dostupné z: doi:10.1109/RADIOELEK.2019.8733447

HONC, Daniel, Petr DOLEŽEL a Jan MERTA. Thermal Process Control Using Neural Model and Genetic Algorithm. SILHAVY, Radek, Petr SILHAVY a Zdenka PROKOPOVA, ed. Intelligent Systems Applications in Software Engineering [online]. Cham: Springer International Publishing, 2019, 2019-09-20, s. 393-403. Advances in Intelligent Systems and Computing. ISBN 978-3-030-30328-0. Dostupné z: doi:10.1007/978-3-030-30329-7_35

Ostatní články ve sborníku

FIKEJZ Jan a Jan MERTA. Utilization of computer simulation for detection non-standard situations within single-track lines. 31st European Modeling and Simulation Symposium, EMSS 2019, Lisabon, Portugalsko. Janov: University of Genoa, Itálie, 2019. s. 352-358. ISBN: 978-88-85741-26-3

FIKEJZ Jan a Jan MERTA. Use reduced track profile and discrete simulation to calculate train travel time. 30th European Modeling and Simulation Symposium, EMSS 2018, Budapešť, Maďarsko. Janov: University of Genoa, Itálie, 2018. s. 165-171. ISBN: 978-88-85741-06-5

FIKEJZ Jan a Jan MERTA. Utilization of railway network model for dynamic calculation of train delays. 29th European Modeling and Simulation Symposium, EMSS 2017, Barcelona, Šapnělsko. Janov: University of Genoa, Itálie, 2017. s. 248-254. ISBN: 978-1-5108-4765-1

FIKEJZ Jan, MERTA Jan a Jakub OBORNÍK. Use of web services for the localization of rolling stock with utilization technology oracle spatial. 28th European Modeling and Simulation Symposium, EMSS 2016. Janov: University of Genoa, Itálie, 2016, s. 128-133. ISBN: 978-889799968-3.

FIKEJZ Jan a Jan MERTA. 27th European Modeling and Simulation Symposium, EMSS 2015. Bergeggi, Itálie. Janov: University of Genoa, Itálie, 2015, s. 191-196. ISBN: 978-889799948-5.

HONC, Daniel a Jan MERTA. Smart, Precision or Digital Agriculture and Farming - Current State of Technology. HERRERO, Álvaro, Carlos CAMBRA, Daniel URDA, Javier SEDANO, Héctor QUINTIÁN a Emilio CORCHADO, ed. 15th International Conference on Soft Computing Models in Industrial and Environmental Applications (SOCO 2020) [online]. Cham: Springer International Publishing, 2021, 2021-08-29, s. 245-254 [cit. 2021-04-06]. Advances in Intelligent Systems and Computing. ISBN 978-3-030-57801-5. Dostupné z: doi:10.1007/978-3-030-57802-2_24

HONC, Daniel, František DUŠEK a Jan MERTA. Static Compensator for Nonsquare Systems – Application Example. SILHAVY, Radek, ed. Applied Informatics and Cybernetics in Intelligent Systems [online]. Cham: Springer International Publishing, 2020, 2020-08-08,

s. 267-273. Advances in Intelligent Systems and Computing. ISBN 978-3-030-51973-5. Dostupné z: doi:10.1007/978-3-030-51974-2_25

DUŠEK, František, Daniel HONC a Jan MERTA. Static Compensator for Decentralized Control of Nonsquare Systems. SILHAVY, Radek, Petr SILHAVY a Zdenka PROKOPOVA, ed. Computational Statistics and Mathematical Modeling Methods in Intelligent Systems [online]. Cham: Springer International Publishing, 2019, 2019-09-20, s. 1-6. Advances in Intelligent Systems and Computing. ISBN 978-3-030-31361-6. Dostupné z: doi:10.1007/978-3-030-31362-3_1

DUSEK, Frantisek, Daniel HONC, Libor HAVLICEK a Jan MERTA. Software for Thermogravimeter. In: 2019 22nd International Conference on Process Control (PC19) [online]. IEEE, 2019, 2019, s. 245-249. ISBN 978-1-7281-3758-2. Dostupné z: doi:10.1109/PC.2019.8815040

Software:

MERTA Jan a Jan FIKEJZ. Demonstrační softwarové řešení lokalizace kolejových vozidel v rámci třívrstvého modelu železniční sítě (2015)

MERTA Jan a Jan FIKEJZ. Optimalizace vybraných vyhledávacích operací v rámci multidimenzionálních dat nad databází Oracle Spatial (2013)