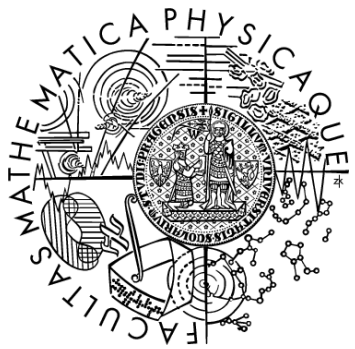
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Lukáš Bajer

Urychlení evolučních algoritmů pomocí směsí rozdělení pravděpodobnosti

Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Martin Holeňa, CSc.
Ústav Informatiky Akademie věd ČR

Studijní program: Informatika, obor Teoretická informatika

2009

Největší dík za vedení práce, podnětné připomínky, trpělivost a především neobyčejně lidský přístup patří RNDr. Martinu Holeňovi, CSc. Poděkovat bych chtěl také novému vedení našeho skautského oddílu Ichthys, které ochotně převzalo mé zodpovědnosti na letošním táboře v době dokončování této práce, a rodičům, kteří mi v téže době doma poskytli výborné zázemí.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze dne 7. srpna 2009 Lukáš Bajer

Obsah

1 Úvod **6**
1.1 Cíle práce 7
1.2 Struktura práce 8

2 Použité metody **9**
2.1 Evoluční algoritmy 9
2.2 Náhradní modely cílové funkce 14
2.3 Sítě s radiálními bázovými funkcemi 15
2.4 Vyhodnocení a výběr modelu 18

3 Použité postupy a algoritmy **21**
3.1 Systém evoluční optimalizace s náhradním modelem 21
3.2 Genetický algoritmus využívající náhradního modelu 24
3.3 Příprava a trénování modelu 26
3.4 Ohodnocování populace modelem 30

4 Implementace **32**
4.1 Prostředí 32
4.2 Struktura a běh programu 33
4.3 Regresní model cílové funkce 34
4.4 Genetický algoritmus a evoluční řízení 37

5 Výsledky testů **38**
5.1 Testovací problém 1: Umělá testovací fitness 38
5.2 Testovací problém 2: Katalyzátor výroby kyseliny HCN 48

6 Závěr **55**

6.1 Další vývoj 55

6.2 Shrnutí 56

Literatura **58**

Příloha: Obsah CD **61**

Název práce: Urychlení evolučních algoritmů pomocí směsí rozdělení pravděpodobnosti
Autor: Lukáš Bajer
Katedra (ústav): Katedra teoretické informatiky
Vedoucí diplomové práce: RNDr. Martin Holeňa, CSc.
e-mail vedoucího: martin(zavináč)cs.cas.cz

Abstrakt: Evoluční, a zvláště genetické algoritmy se staly jednou z nejúspěšnějších metod optimalizace empirické cílové funkce. V mnoha reálných aplikacích je však hlavní nevýhodou to, že ohodnocení empirickou funkcí stojí nemalé finanční prostředky nebo trvá značnou dobu. V naší práci je použit náhradní model původní cílové funkce sloužící jako její rychlý odhad. Původně bylo zamýšleno použití směsí rozdělení pravděpodobnosti, což však nebylo možné realizovat. V naší práci jsou tedy konkrétně použity RBF sítě, které se směsmi velmi úzce souvisí. Díky modelu je možné najednou vyvíjet podstatně větší populace, nebo je možné nechat ohodnocovat několik generací pouze modelem. Výsledkem je znatelně rychlejší konvergence ve smyslu počtu ohodnocení původní empirickou funkcí.

Klíčová slova: genetické algoritmy, náhradní modely cílové funkce, RBF sítě

Title: Improving evolutionary algorithms using probability mixture models
Author: Lukáš Bajer
Department: Department of Theoretical Computer Science
Supervisor: RNDr. Martin Holeňa, CSc.
Supervisor's e-mail address: martin(zavináč)cs.cas.cz

Abstract: Evolutionary, and especially genetic algorithms have become one of the most successful methods for the optimization of empirical objective functions. However, in many engineering applications, evaluation of the empirical fitness function can be very time consuming or cost a considerable amount of money. In this article, we employ a surrogate model of the original fitness function which serves as a fast approximation whenever needed. First, we intended to use finite mixture models, but radial basis function networks was finally used as a particular surrogate model because of implementability. With this method, much larger populations or several generations can be simulated without waiting for expensive objective function evaluation. As a result, faster convergence in terms of the number of the original empirical fitness evaluations is achieved.

Keywords: genetic algorithms, surrogate modelling, RBF networks

Kapitola 1

Úvod

V nejrůznějších oblastech našeho všedního, profesního i vědeckého života se dnes setkáváme s problémy, na něž potřebujeme nalézt nejlepší možné řešení, nebo se takovému řešení alespoň dostatečně přiblížit. Problémy tohoto druhu se nazývají problémy *optimalizační*.

Evoluční techniky, jejichž podstata souvisí s biologickým vývojem a genetikou, jsou jednou z úspěšných optimalizačních metod, které tyto problémy řeší. Své uplatnění nachází ve vědě i průmyslu, a to zejména v případech, kdy jiné standardní možnosti selhávají. Jedná se zejména o problémy charakterizované vysokou dimenzí, výskytem diskrétních proměnných nebo vykazující nespojitost. Příklady takových úloh můžeme najít například při optimalizaci designu konstrukcí [16] nebo ve vývoji materiálů [11].

Velkou výhodou evoluční optimalizace jsou slabé požadavky na tzv. cílovou (také fitness nebo ohodnocovací) funkci, která číselně vyjadřuje kvalitu navrhovaného řešení a jejíž optimum se tak snažíme nalézt. Je potřeba znát pouze hodnoty funkce pro každé navržené řešení; žádné další požadavky na její tvar či hladkost již nejsou. Právě tento fakt umožňuje využít evoluční metody v situacích, kdy je ohodnocovací funkce empirická – ohodnocení kvality řešení je získáno měřením, pokusem nebo náročnou simulací.

Ohodnotit řešení empirickou funkcí však bývá nákladné. Uskutečnit pokus a zpracovat jeho výsledky často stojí spoustu času i nemalé finanční prostředky. Oba tyto zdroje bývají omezené, a tak je zřejmé, že počet ohodnocení potřebujeme co možná nejvíce snížit, přitom však stále nalézt optimální řešení. V kontrastu s tímto zde stojí fakt, že evoluční techniky ke své konvergenci potřebují funkci vyhodnocovat často.

Uvedený rozpor zmírňuje metoda náhradních modelů. Na základě omezeného počtu dat, sestávajících z definic řešení a jejich přesného ohodnocení empirickou cílovou funkcí, je pro tuto funkci zkonstruován regresní model. Ten dokáže přibližně, zato však velmi rychle ohodnocovat každé řešení, které je navržené evoluční metodou. Praxe ukazuje, že pokud se ve vhodném poměru používají přesná a modelovaná fitness, evoluční optimalizace dokáže nalézt nejlepší řešení s podstatně nižšími náklady vynaloženými na vyhodnocování původní empirické funkce než je tomu bez použití modelu.

1.1 Cíle práce

Koncept, který používá náhradní modely k odhadům fitness v evoluční optimalizaci, se rozvíjí v posledních deseti, výrazněji pak pěti letech. Konkrétních implementací však ještě není příliš mnoho, a proto si naše práce klade za cíl následující:

Náhradní model spojitých i diskrétních hodnot. Všechny námi doposud známé publikace používají náhradní modely ohodnocovacích funkcí jen při evoluci řešení složených ze spojitých proměnných. Prvním cílem této práce je detailní návrh a prototypová implementace náhradního modelu fitness funkce do evoluční optimalizace. Model využívající sítě s radiálními bázovými funkcemi (RBF sítě) bude správně reagovat nejen na proměnné spojité, ale i na kombinaci s diskrétními a celočíselnými hodnotami. Model bude otestován na umělé testovací cílové funkci a na reálných datech z vývoje chemických katalyzátorů, na kterých byl v minulosti použit genetický algoritmus bez náhradního modelu.

Genetický algoritmus využívající navržený model. Aby bylo možné náhradní model otestovat, cílem práce je také podrobný návrh a velmi jednoduchá prototypová implementace rozšířeného genetického algoritmu, který navrženého modelu bude využívat. Součástí práce je i otestování na syntetické testovací fitness funkci.

Algoritmy budou zpracovány co nejobecněji. Jejich implementace pak bude používat jako vstup definici úlohy ve formátu používaném v oboru syntézy chemických katalyzátorů a proběhne v systému Matlab.

1.2 Struktura práce

V kapitole 1 jsou po obecném úvodu do problematiky evoluční optimalizace s empirickou cílovou funkcí vytyčeny cíle této práce. Kapitola 2 poskytuje základní přehled o metodách, které tato práce využívá. Jde zejména o evoluční algoritmy, používání náhradních modelů v optimalizaci, sítě s radiálními bázovými funkcemi a jejich vztah ke směsím rozdělení pravděpodobnosti. Závěr kapitoly je věnován metodám vyhodnocování kvality modelu.

Kapitola 3 podrobně popisuje hlavní přínos této práce. Jak již bylo uvedeno výše, jde o konkrétní genetický algoritmus využívající náhradního modelu, o trénování modelu a jeho použití. Genetický algoritmus i algoritmus trénování modelu jsou popsány podrobným pseudokódem, další parametry a vysvětlení jsou v textu. V kapitole 4 se zabýváme implementačními detaily algoritmů, popsány jsou také další funkce, které byly do algoritmů přidány na základě zkušeností s reálnými daty.

Konkrétní ověření našich výsledků na provedených testech lze nalézt v kapitole 5. Naše algoritmy byly otestovány na dvou různých úlohách: umělé testovací funkci a datech z reálné aplikace z oblasti chemických katalyzátorů. Kapitola 6 celou práci uzavírá: nastiňuje další možný vývoj a shrnuje dosažené výsledky.

Kapitola 2

Použité metody

2.1 Evoluční algoritmy

Algoritmy, které nacházejí inspiraci v evoluční biologii a genetice, se vyvíjejí od začátku druhé poloviny dvacátého století. První pokusy byly orientovány spíše jako počítačové simulace biologické evoluce [7], následovaly však přístupy řešící obecné problémy.

Postupně vzniklo mnoho variant algoritmů, které jsou evolucí přímo ovlivněny. Do nejpodstatnější skupiny *evolučních algoritmů* spadají kromě nejrozšířenějších *genetických algoritmů* také například *evoluční strategie* a *genetické*, resp. *evoluční programování*. Poslední dva jmenovaní zástupci nechávají vyvíjet celé programy, resp. jejich parametry, a dále se jimi zabývat nebudeme. Genetické algoritmy a evoluční strategie však jsou pro nás zajímavé, neboť jejich účelem je hledat optimální řešení obecných problémů.

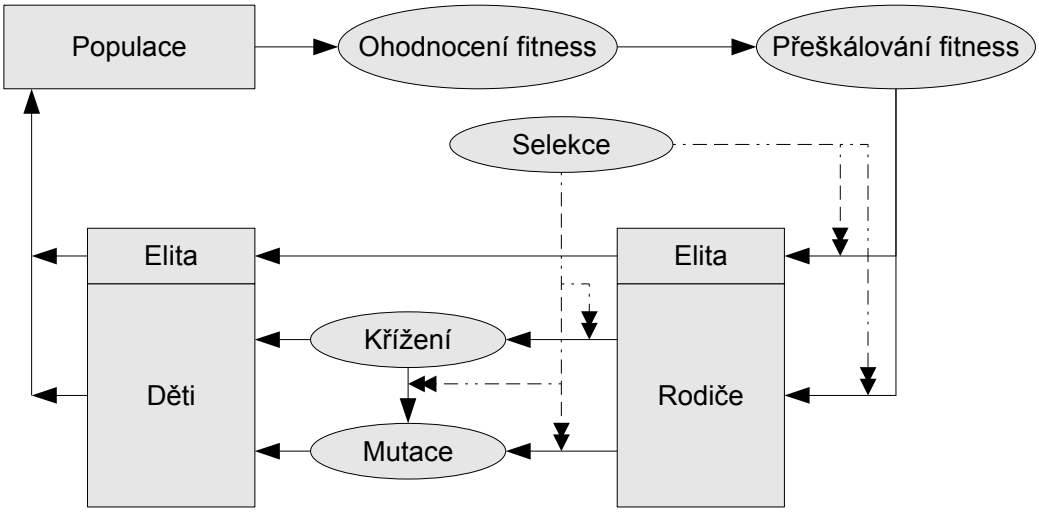
Evoluční algoritmy patří do širší rodiny stochastických optimalizačních metod. Po dobu svého běhu udržují množinu přípustných řešení, tzv. *populaci*, kterou v každém kroku ohodnotí cílovou funkcí. Podle výsledků tohoto ohodnocení upraví pravděpodobnostní rozložení, podle kterého generují populaci novou [21]. Těmto krokům se říká generace.

Pro evoluční strategie [20] je charakteristická tzv. *fenotypická* reprezentace jedinců vektorem hodnot přímo z domény daného problému – dimenze jedinců odpovídají dimenzím řešení. Populace se mění na základě náhodné *mutace* a deterministické *selekce*, která vybírá jedince do další generace podle pořadí daného ohodnocením cílovou funkcí. Také tyto algoritmy v dalším textu nebudeme probírat, některé postupy se však přenesly do genetických algoritmů.

Genetické algoritmy

Způsob, jakým genetické algoritmy upravují distribuci, a také reprezentace řešení jsou inspirovány v biologii. Jedinci se reprezentují genetickým kódem. Ten je tradičně *binární* a řešení se do nebo z tohoto kódu musí transformovat. Kódovat však můžeme také fenotypicky, stejně jako v případě evolučních strategií. Tento způsob budeme využívat v našem algoritmu.

Populace se upravuje v několika podkrocích (viz obrázek 2.1). Po již zmíněném *ohodnocení* populace se mohou hodnoty fitness funkce *přeškálovat*. Často se místo výsledků funkce vezme pro každý prvek pouze pořadí, v jakém by se umístil při seřazení podle původních ohodnocení. Dále se *selekcí* vyberou rodiče, ze kterých se noví jedinci budou generovat. Větší šanci na výběr mají řešení s lepší hodnotou (přeškálované) fitness.



Obrázek 2.1: Schéma jedné generace GA

Z rodičů se pak noví jedinci vytváří *křížením* anebo *mutací*. Při křížení se vyměňují části genetického kódu rodičů a vzniká jeden nebo dva potomci. Mutace odpovídá náhodné, většinou poměrně malé změně rodiče a potomek vzniká vždy jeden.

Uvedenými operacemi se může stát, že všichni potomci budou mít horší fitness ohodnocení než nejlepší jedinci z generace předešlé. Aby se tomuto zhoršování zabránilo, často se zavádí tzv. *elitismus*. Při jeho využití několik málo nejlépe

ohodnocených jedinců vynechá selekci, křížení a mutaci a do další generace se dostane přímo.

Genetické operace

Věnujme nyní pár řádků různým druhům genetických operací. Zřejmě nejjednodušší selekcí je selekce *ruletová*, někdy též *proporcionální*. Pokud máme populaci o velikosti $|p|$ ohodnocenou (přeškálovanými) fitness hodnotami $F_1 < F_2 < \ldots < F_{|p|}$ (indexy odpovídají očíslování jedinců), pak k-tý jedinec má pravděpodobnost výběru selekcí rovnou

$$p_k = \frac{F_k}{\sum_{j=1}^{|p|} F_j}, \quad k = 1, \ldots, |p|. \tag{2.1}$$

Vlastní výběr je pak proveden tolikrát, kolik potřebujeme vybrat jedinců.

Používaná je také selekce *turnajová* s parametrem t, obvykle je zvoleno $t = 2$. Probíhá v N kolech, kde N je potřebný počet vybraných jedinců. V každém kole je (zpravidla s rovnoměrným rozložením) vybráno t jedinců, a z této skupiny je pak zvolen vítěz. Nejčastěji se jednoduše vezme jedinec nejlépe ohodnocený, ale možností je také vybrat vítěze na základě rozdělení pravděpodobnosti vychýleného k jedincům s vyšším ohodnocením fitness, například podle (2.1).

Stochastická uniformní selekce představuje poslední námi vybraný typ. S využitím použitého značení zaveďme kumulativní pravděpodobnost k-tého jedince jako

$$P_k = \sum_{j=1}^{k} p_j, \quad k = 1, \ldots, |p|; \quad P_0 = 0. \tag{2.2}$$

Selekce pak zvolí pouze jedno náhodné číslo $r \sim U(0, \frac{1}{N})$, vytvoří N-prvkový vektor $v = (r, r + \frac{1}{N}, r + \frac{2}{N}, \ldots, r + \frac{N-1}{N})$ a pro každý jeho element v_i vybere takového jedince k, pro kterého platí $v_i \in (P_{k-1}, P_k]$.

Princip této selekce se dá jednoduše vysvětlit také geometrickým obrazem (viz obrázek 2.2). Vezměme úsečku jednotkové délky a rozdělme ji na úseky délky $p_k, k = 1, \ldots, |p|$, které odpovídají jedincům v populaci. Algoritmus pak určí na úsečce postupně body v rovnoměrné vzdálenosti $\frac{1}{N}$. Selekcí jsou vybíráni jedinci v okamžiku, kdy se na jim odpovídajícím úseku určí bod. Vzdálenost r prvního bodu od počátku je určena náhodně z intervalu $(0, \frac{1}{N})$.

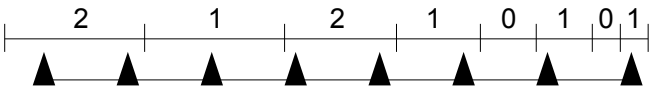
Tradiční *diskrétní* křížení je věrnou obdobou křížení buněčných chromozomů: mezi rodiči se přímo vymění několik částí binárního kódu. Existuje v různých

variantách, nejčastější je *jednobodové*, *dvoubodové* nebo *uniformní*. Pro fenotypické kódování však pouze odpovídá zaměňování hodnot z odpovídajících dimenzí rodičů, což v případě spojitých domén dostatečně nezvyšuje diversitu populace. Vznikly proto různé spojité varianty křížení, například *simulované binární*, křížení *s míšením* (simulated binary resp. blended crossover [6]), nebo *aritmetické* křížení. Poslední jmenované vytváří potomka, jehož náhodně zvolené dimenze jsou průměrem odpovídajících dimenzí rodičů. Výhoda tohoto křížení je, že lze snadno u potomků kontrolovat splnění případných lineárních omezení (viz dále).

Mutace binárních genomů se zpravidla děje překlopením hodnoty jednoho nebo několika náhodných bitů. V případě kódování fenotypem se u náhodné proměnné vybere jiná hodnota z její dimenze. Reálné proměnné se nejčastěji upravují přičtením náhodného čísla z normálního $\mathcal{N}(0, \sigma^2)$ nebo rovnoměrného rozdělení $U(-\Delta, +\Delta)$. V Matlabu se setkáme ještě s jedním zajímavým druhem adaptivní mutace, která umísťuje potomka v náhodně zvoleném směru a vzdálenosti od rodiče. Směrový vektor se bere z rovnoměrného rozložení, rozložení vzdáleností se upravuje podle vývoje genetického algoritmu v několika posledních generacích. U každého vygenerovaného jedince se kontroluje splnění případných omezení: pokud by byl potomek mimo rodičův mnohostěn, mutace zmenší vzdálenost a umístí potomka na smyšlené polopřímce tak, aby omezení byla splněna.

Vygenerováním a ohodnocením nových jedinců se generace uzavírá. Genetické algoritmy se ukončují několika možnými způsoby – zpravidla se po ohodnocení nové populace testuje několik ukončovacích kritérií. Nejjednodušším je omezení počtu generací nebo počtu ohodnocení cílovou funkcí. Často se také algoritmus ukončuje v okamžiku, kdy se již nemění nejlepší fitness v několika generacích za sebou, případně kdy se generují všichni jedinci příliš blízko u sebe.

Mezi výhody genetických algoritmů patří, že ve srovnání s jinými technikami často dokáží lépe nacházet globální optima a neulpívat v optimech lokálních. Je to



Obrázek 2.2: Stochastická uniformní selekce

především tím, že udržují celou populaci možných řešení, a do jisté míry tak mají schopnost prohledávat na více místech najednou. Stále však platí, že zpravidla najdou pouze řešení, která jsou optimu blízká. Naopak, často právě v závěrečných fázích, kdy už se nejlepšímu řešení blíží, bývají genetické algoritmy značně neefektivní. Další negativní vlastností pak bývá dlouhý celkový čas jejich běhu a vysoký počet ohodnocení cílovou funkcí. Pokud lze použít techniku, která dokáže využívat tvar ohodnocovací funkce, pak bude tato metoda nejspíš podstatně rychlejší.

Evoluční optimalizace s omezeními

Poměrně často se lze setkat s optimalizačními problémy, jejichž řešení podléhají *omezením*; v naší práci se budeme zabývat pouze lineárními. Bývají definovány soustavou lineárních rovnic a nerovnic zadaných pro jednotlivé dimenze uvažovaných řešení problému, například pomocí matic $\mathbf{A}_e$, $\mathbf{A}_i$ a jim odpovídajícím pravým stranám $\mathbf{b}_e$, $\mathbf{b}_i$. Výsledné řešení $\mathbf{x} = (x_1, x_2, \ldots, x_m)^T$ optimalizačního problému pak musí splňovat rovnost

$$\mathbf{A}_e \, \mathbf{x} = \mathbf{b}_e$$

a nerovnost

$$\mathbf{A}_i \, \mathbf{x} \leq \mathbf{b}_i.$$

Existuje několik způsobů, jakým lze zajistit splňování omezení za běhu genetického algoritmu. Patrně nejjednodušším je upravit fitness funkci tak, aby jedincům, kteří omezení nesplňují, ohodnocení snižovala. Tím naopak podpoří evoluční výběr těch, kteří omezení splňují. Algoritmus se musí rozšířit o kontrolu splnění omezení konečného řešení, které algoritmus vrací.

Další možností je změnit křížení a mutaci, aby generovaly pouze validní řešení. Tím je zaručeno splnění omezení po celý běh algoritmu a žádné další úpravy potřeba nejsou. Tento způsob je použit v této práci.

Třetí variantou je kontrolovat splnění omezení po vygenerování jedinců genetickými operacemi. Pokud by byla porušena, místo jednice se vezme jeho průmět na nejbližší mnohostěn daný omezeními.

Kombinace diskrétních a spojitých hodnot

V mnoha aplikacích, ve kterých se používají genetické algoritmy, jsou součástí optimalizovaného řešení spojité i diskrétní proměnné. Obvyklá je při tom nestabilita vzhledem ke změnám diskrétních proměnných: změna jedné takové hodnoty

často znamená velké změny v cílové funkci, změnu oborů hodnot, počtu použitých spojitých proměnných nebo změnu omezení.

Je proto potřeba vycházet z vlastností řešeného problému a určit, které genetické operace nad jakými proměnnými mají smysl. Snadno se například dostaneme do situace, kdy mutace jedné diskrétní proměnné znamená kompletní nové vygenerování spojitých hodnot, nebo kdy lze smysluplně křížit pouze jedince se stejnou kombinací hodnot diskrétních.

2.2 Náhradní modely cílové funkce

Důvod použití náhradních modelů jsme naznačili již v úvodu: je jím ušetření času nebo jiných zdrojů při optimalizaci s empirickou cílovou funkcí. Základním principem je ve většině případů použít dopředu natrénovaný model a jen občas využít původní fitness, aby se algoritmus příliš neodchýlil díky nepřesnosti modelu [3], [17], [19].

Pokud jsou k dipozici trénovací data před startem algoritmu, může se model použít již od první generace. V opačném případě je během několika počátečních generací nutné nejdříve nasbírat potřebné množství dat. V obou případech je běžnou praxí, že se s dalšími generacemi rozšiřuje trénovací množina, a tak se i zpřesňuje model.

K vytvoření modelu se používají různé regresní metody, většinou nelineární. Nejrozšířenější jsou neuronové sítě [10], [25], support vector regression models [22], [23], regresní stromy [4] nebo neparametrická regrese [1], [9]. V naší práci jsme použili RBF sítě (viz oddíl 2.3), které používá k regresi spojitých proměnných například Zhou a kol. [26].

Evoluční řízení

Způsob, kterým se určuje, jestli se použije model, nebo původní fitness, budeme nazývat *evoluční řízení*. V literatuře se můžeme setkat se dvěma obecnými přístupy. Prvním z nich je tzv. *individuální* řízení. Vychází z předpokladu, že v každé generaci je možné původní fitness funkcí ohodnotit určitý omezený počet jedinců. Důležitý je ale fakt, že genetické operace z těchto jedinců na konci generace vytvoří populaci podstatně větší. Originální cílová funkce tak vždy ohodnotí pouze část. Jin [12] navrhuje čtyři způsoby výběru jedinců k přehodnocení:

- *Shlukování.* Jedinci jsou rozděleni do shluků podle jejich přibližného ohodnocení náhradním modelem a původní cílovou funkcí se ohodnotí ti, kteří jsou nejblíže středům shluků.

- *Náhodný výběr.* Je určitě nejjednodušším a nejrychlejším způsobem, sám o sobě však vzhledem ke snižování celkového počtu drahých ohodnocení příliš efektivní není.

- *Nejlepší jedinci.* Výběr nejlepších jedinců je dobrý způsob, jak se zaměřit na již nalezená slibná řešení a v kombinaci s elitismem je neztratit díky nepřesnosti modelované fitness. Pokud je však jediným způsobem výběru, může vést k malé diversitě populace, čímž může algoritmus snadněji uvíznout v lokálním minimu.

- *Jedinci s největší chybou predikce.* Použitím této metody se podporuje prohledávání dosud neprozkoumaných oblastí. Potřebný je odhad chyby predikce, který lze získat například křížovou validací (viz oddíl 2.4).

Druhým typem evolučního řízení je *generační*. Tento způsob používá takovou velikost populace, aby jí bylo možné najednou ohodnotit původní cílovou funkcí. Generace jsou seskupeny do cyklů pevné délky λ, z nichž pouze η je ohodnoceno originální fitness. Zbylých $(\lambda - \eta)$ generací v každém cyklu použije k ohodnocení model. Parametr η se mění v závislosti na odhadu chyby modelu: čím větší je chyba, tím větší je počet generací η ohodnocených původní cílovou funkcí.

2.3 Sítě s radiálními bázovými funkcemi

Pro náš náhradní model jsme použili RBF sítě. Jsou speciálním případem neuronových sítí; mají podobnou stavbu (viz obrázek 2.3) jako častěji používané vícevrstvé perceptronové sítě. Jsou složeny ze vstupní a jedné skryté vrstvy a výstupních neuronů.

Hlavní rozdíl je v typu aktivačních funkcí f_i. Vícevrstvé perceptrony typicky používají sigmoidální funkce, například arcus tangens nebo logistickou sigmoidální funkci, které jako svůj argument berou přímo hodnoty na vstupních neuronech nebo výstupy z předešlé vrstvy. V případě RBF sítí je ke každé radiální funkci f_i přiřazeno centrum $\mathbf{c}_i$, a jako argument funkce je pak vzata vzdálenost vstupního vektoru od tohoto centra $|\mathbf{c}_i - \mathbf{x}|$. Výsledky funkcí jsou vynásobeny váhami π_i a nakonec sečteny.

RBF sítě mapují vstupní podprostor $\mathbb{R}^n$ do $\mathbb{R}^m$. Můžeme je vyjádřit sadou rovnic

$$f_j(\mathbf{x}) = \sum_{i=1}^{g_j} \pi_{ij} f_{ij}(||\mathbf{x} - \mathbf{c}_{ij}||), \quad j = 1, \ldots, m, \tag{2.3}$$

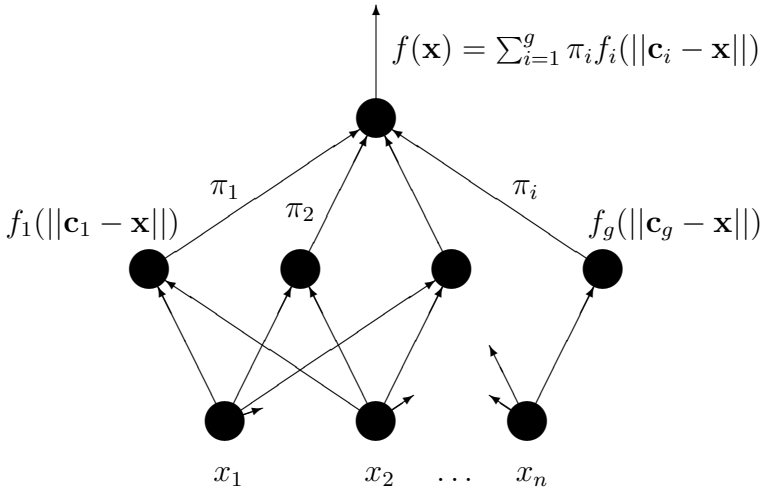
kde $\mathbf{x}$ je vstupní vektor, g_j počet radiálních funkcí, které také budeme označovat jako komponenty, f_{ij} jsou radiální bázové funkce, $\mathbf{c}_{ij}$ značí jejich odpovídající centra a $||.||$ je norma. Z výrazu (2.3) můžeme nahlédnout, že $\{f_{ij}\}_{i=1}^{g_j}$ tvoří bázi vektorového prostoru všech funkcí, které j-tá síť počítá.

Jako norma je typicky použita eukleidovská, ale ostatní normy

$$\sqrt{(\mathbf{x} - \mathbf{c}_i)^T \mathbf{A} (\mathbf{x} - \mathbf{c}_i)} \tag{2.4}$$

s libovolnou pozitivně definitní maticí $\mathbf{A}$ mohou být použity také. Pozitivní definitnost zaručuje, že

$$(\mathbf{x} - \mathbf{c}_i)^T \mathbf{A} (\mathbf{x} - \mathbf{c}_i) > 0$$



Obrázek 2.3: Síť s radiálními bázovými funkcemi s jedním výstupním neuronem

pro všechny reálné vektory $\mathbf{x} \neq \mathbf{c}_i$. Jako funkce f_i jsou nejčastěji použity Gaussovy funkce

$$f_i(\mathbf{x}) = g_i(\mathbf{x}; \mathbf{c}_i, \beta) = \mathrm{e}^{-\beta||\mathbf{x}-\mathbf{c}_i||^2}. \tag{2.5}$$

Znaky za středníkem označují parametry funkce a v dalším textu je budeme vynechávat, pokud je nebude potřeba explicitně zdůraznit. Gaussovy funkce jsou používané především díky tomu, že jsou tzv. *lokalizované*

$$\lim_{|x|\to\infty} f_i(x) = 0$$

a nekonečně derivovatelné. Pokud použijeme normu (2.4), pro matici $\mathbf{A} = \boldsymbol{\Sigma}^{-1}$ dostaneme funkci úměrnou hustotě n-rozměrného normálního rozdělení $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

$$f_n(\mathbf{x}) = \frac{1}{(2\pi)^{n/2}\,|\boldsymbol{\Sigma}|^{1/2}} \mathrm{e}^{\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)} \tag{2.6}$$

Je dokázáno, že RBF sítě mají na kompaktní podmnožině $\mathbb{R}^n$ univerzální aproximační schopnost [18]. Tím je zaručeno, že v daném prostoru funkcí mohou být (s ohledem na metriku prostoru) libovolně blízko jakékoliv funkci. Další podrobnosti o RBF sítích lze nalézt v literatuře [2], [5].

Směsi pravděpodobnostních rozdělení

Se sítěmi s radiálními bázovými funkcemi po formální stránce velmi úzce souvisejí směsi pravděpodobnostních rozdělení (dále jen směsi). Jejich použití bylo součástí první verze zadání této práce, a proto jim věnujme několik odstavců.

Hlavní úkoly směsí jsou modelovat složité hustoty pravděpodobnosti u rozdělení, která neodpovídají žádnému obecně známému, a dále rozdělování dat do shluků na základě změřené složené hustoty pravděpodobnosti výskytu jevu podmíněného těmito daty. V obou případech se vyjadřuje celková hustota jako součet několika hustot základních.

Značí-li $\mathbf{x}$ pozorované hodnoty nějakého náhodného vektoru, pak celkovou hustotu pravděpodobnosti vyjádřenou směsí označíme jako

$$f(\mathbf{x}) = \sum_{i=1}^{g} \pi_i f_i(\mathbf{x}; \theta_i). \tag{2.7}$$

V rovnici najdeme mnoho analogií s RBF sítěmi. Parametr g určuje počet komponent, π_i jsou nezáporné váhy v součtu rovny jedné ($0 \leq \pi_i \leq 1, i = 1, \ldots, g \wedge \sum_{i=1}^{g} \pi_i = 1$)), a f_i tzv. komponenty – základní hustoty pravděpodobnosti. Podobně jako u RBF sítí se pro f_i nejčastěji používá hustot Gaussova rozložení. Symbol θ_i označuje další parametry hustot, například střední hodnoty a rozptyly nebo varianční matice v případě normálního rozdělení. Další podrobnosti najdeme v literatuře [15].

Směsi by bylo zajímavé použít v kombinaci s evolučními algoritmy, které pracují s pravděpodobnostními distribucemi, například s *algoritmy odhadujícími rozdělení* (Estimation of Distribution Algorithms) [13]. V sytému Matlab však pro ně neexistuje žádná podpora, a tak jsme zvolili použití RBF sítí s algoritmy genetickými. Díky podobnosti RBF sítí a směsí jsme se však od původní verze zadání příliš neodchýlili.

2.4 Vyhodnocení a výběr modelu

Je-li k dispozici dostatek času, je možné natrénovat několik různých variant modelů, a pak vybrat tu, která je pro nás nejlepší. Zpravidla volíme model s nejmenší chybou, v úvahu se však bere také složitost modelu. Příliš složitý model totiž sice dokáže dobře popsat data, na kterých byl trénován, může však dojít k tzv. *přeučení*. Přeučenému modelu chybí schopnost generalizace a regrese v oblastech, kde bylo málo trénovacích dat.

Měření chyby modelu

Chybu regresních modelů můžeme vyjádřit například pomocí *reziduálního součtu čtverců* (angl. residual sum of squares, RSS). Rezidua e_i vyjadřují, jakou část naměřené hodnoty závislé proměnné y_i se nepodařilo vysvětlit modelem

$$e_i = y_i - f(\mathbf{x}_i). \tag{2.8}$$

V rovnici f označuje regresní model a $\mathbf{x_i}$ nezávislé proměnné, které model přijímá jako argument a pro které byla naměřena hodnota y_i. Reziduální součet čtverců je pak pro n dat součet druhých mocnin těchto reziduí

$$\mathrm{RSS} = \sum_{i=1}^{n} e_i^2. \tag{2.9}$$

Pokud reziduální součet čtverců vydělíme počtem dat, získáme *střední kvadratickou chybu* (angl. mean squarred error, MSE), která se používá patrně nejčastěji

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(\mathbf{x}_i)\right)^2. \tag{2.10}$$

Průměr absolutních hodnot reziduí pak označujeme jako *střední absolutní chybu* (angl. mean absolute error, MAE)

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - f(\mathbf{x}_i)|. \tag{2.11}$$

Nejjednodušší způsob měření chyby je počítání reziduí z trénovacích dat, tedy stejných dat, na kterých se model trénoval. Výhodou je kromě nejmenší náročnosti na výpočetní výkon také fakt, že se k natrénování mohla použít veškerá dostupná data. Vypovídací schopnost takto spočtené chyby je však velmi omezená. Určuje, na kolik jsme se přiblížili trénovacím datům, ale neříká mnoho o tom, jak se model bude chovat na datech nových, pro které je vlastně konstruován.

Chybu je proto vhodné měřit na datech odlišných od trénovacích. Pokud se chyba používá k výběru parametrů modelu, označují se tato data jako *validační*. Data se nazývají *testovací*, pokud se jimi ohodnocuje výsledný natrénovaný model. Abychom nepřišli o možnost trénovat model na co největším množství dat, možností je nejdříve model natrénovat na jedné části dat a na druhé změřit chybu. Trénování výsledného modelu určeného k regresi pak proběhne ještě jednou na datech trénovacích i validačních, ovšem už bez měření chyby.

Dobrý odhad chyby modelu poskytuje *křížová validace*. Dostupná data se náhodně rozdělí do k skupin. Počet skupin je nejčastěji deset, nejpřesnější možností je pak vzít k rovno počtu dat n. Poté se postupně každá skupina použije jako validační pro měření chyby modelu natrénovaného na $(k-1)$ skupinách zbylých. Jako střední kvadratická chyba modelu se pak vezme průměr ze středních kvadratických chyb počítaných na validačních množinách každého z k trénování.

Výběr modelu

Máme-li různé modely kvantitativně ohodnocené jejich chybou (například střední kvadratickou), nejjednodušší je vzít model, jehož chyba je nejmenší. Tento způsob je vhodný zejména v případě použití nezávislé validační množiny, případně křížové

validace. V těchto případech by se totiž přeučenost modelu měla projevit vyšší spočtenou chybou.

Možností je ale také použití kritéria, které v kvantitativním vyjádření kvality modelu penalizuje jeho složitost. Nejznámějšími jsou *Akaikeho informační kritérium* (AIC) a *Bayesovské informační kritérium* (BIC). Každé vychází z jiných teoretických předpokladů, ale obě upravují logaritmus střední kvadratické chyby členem, který závisí na počtu parametrů modelu: čím složitější model, tím větší penalizace [1].

V uvedené monografii jsou kritéria vyjádřena následovně:

$$\mathrm{AIC} = \log\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2\right] + \frac{2p}{n}, \tag{2.12}$$

$$\mathrm{BIC} = \log\left[\frac{1}{n}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2\right] + \frac{[\log n]p}{n}. \tag{2.13}$$

V obou případech označuje p počet parametrů modelu, jejichž hodnoty se hledají trénováním, zbylé značení je stejné jako na začátku tohoto oddílu. Z uvedených rovností plyne, že BIC již od nevelkého počtu dat penalizuje počet parametrů více než AIC.

Použití zvoleného kritéria je k určení nejlepšího modelu poměrně jednoduché: jeho menší hodnota předpokládá lepší model.

Kapitola 3

Použité postupy a algoritmy

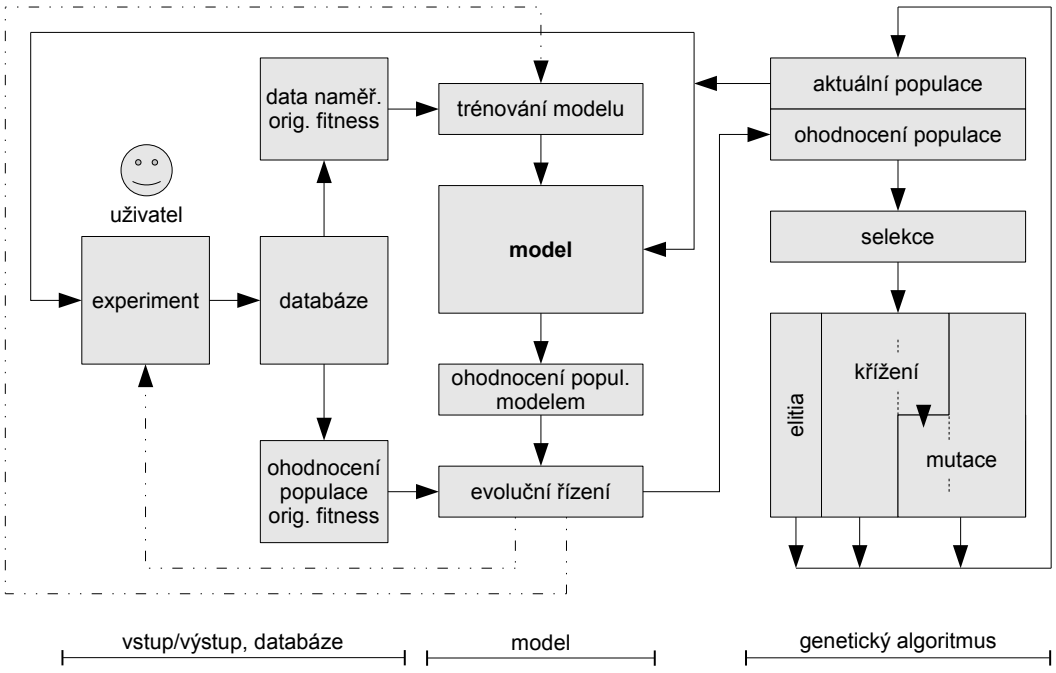
V předešlé kapitole jsme se zabývali obecnými metodami, které se týkají genetických algoritmů použitých k optimalizaci empirických funkcí. Tím jsme si připravili půdu pro konkrétní návrhy algoritmů, kterými se budeme zabývat v této části. Nejdříve představíme celý optimalizační systém a jeho komponenty poněkud neformálně. Dále následuje podrobný popis naší formy genetického algoritmu, který využívá náhradního modelu fitness funkce. Jak tento model vzniká a jak se používá je popsáno v částech následujících.

3.1 Systém evoluční optimalizace s náhradním modelem

Na začátku této kapitoly, dříve než se budeme podrobně věnovat algoritmům a jejich pseudokódům, představíme běh celého výsledného programu na základě interakce větších celků. Celý systém lze znázornit pomocí jednoduchého diagramu funkčních modulů, jak je tomu na obrázku 3.1. Jejich vzájemné vazby a distribuce potřebných dat jsou zobrazeny šipkami.

Popis obrázku začneme zprava. V pravé části je zjednodušeně znázorněn *genetický algoritmus*, v podstatě tak, jak byl popsán na začátku této práce v kapitole 2.1. Tvoří základ celého optimalizačního procesu, avšak bez zbytku programu by nepřinášel příliš nových myšlenek.

Zbytek diagramu úzce souvisí s použitým regresním modelem a tím, jak je zásoben potřebnými daty. Chování celé této části programu je určováno *evolučním*



Obrázek 3.1: Diagram funkčních modulů

řízením. To se aktivuje na popud genetického algoritmu ve chvíli, kdy je potřeba ohodnotit populaci.

Jak již bylo popsáno dříve, evoluční řízení rozhoduje o tom, zda se k ohodnocení populace (v generačním řízení) nebo konkrétního jedince (v individuálním řízení) použije model, nebo původní empirická cílová funkce. Tento proces je znázorněn dvojitě čerchovanými šipkami mezi *evolučním řízením* a jednotkou *trénování modelu*, resp. *experimentem*.

Popišme nejdříve ohodnocení modelem. Pokud při posledním ohodnocování originální empirickou funkcí přibyla nová trénovací data (*data naměř. orig. fitness* na obrázku), začíná se novým natrénováním modelu na všech dostupných datech z databáze. Výsledkem je nový *model*, který ohodnotí *aktuální populaci*, nebo její část; v diagramu jsou tyto výsledky naznačeny v rámci *ohodnocení popul. modelem*.

Ohodnocení jedinců originální empirickou funkcí probíhá zpravidla za asistence *uživatele*, který musí navržená řešení z *aktuální populace* ohodnotit nějakým *experimentem* a výsledky tohoto pokusu uložit do *databáze*. Teprve odtud jsou dostupné našemu programu v rámci *ohodnocení populace orig. fitness*.

Výsledné *ohodnocení* celé *populace* nakonec *evoluční řízení* předá zpět genetickému algoritmu.

V praktickém nasazení právě popsaného systému je důležitým faktorem doba měření empirické cílové funkce, která bývá často počítána na hodiny, nebo dokonce dny. V takovém případě celý program provede pouze jednu generaci genetického algoritmu s ohodnocením původní fitness a dočasně skončí.

V případě individuálního evolučního řízení načte populaci s nově naměřenými hodnotami fitness z databáze, genetickými operacemi z nich vygeneruje novou populaci, natrénuje model, vybere některé jedince na přehodnocení, vrátí je uživateli a skončí. Uživatel s těmito jedinci provede experiment, výsledky uloží do databáze a znovu spustí program, čímž se cyklus opakuje.

Generační řízení se liší pouze v tom, že místo výběru jedinců na přehodnocení nechá v určitých generacích proběhnout několik generačních cyklů pouze s ohodnocením modelem a uživateli vrací pro experimentální ohodnocení poslední vygenerovanou populaci.

Program zcela skončí při splnění určitých ukončovacích podmínek, které jsou popsány na konci následujícího oddílu. V diagramu není zobrazen žádný explicitní výstup. Je to z toho důvodu, že všechna důležitá data se ukládají do databáze, která je uživateli stále k dispozici. To je mimo jiné také důvodem, proč lze program po každé generaci snadno přerušit.

3.2 Genetický algoritmus využívající náhradního modelu

Zkonstruování podrobného genetického algoritmu, který využívá náhradního modelu cílové funkce, patří do druhého cíle této práce (viz 1.1). Jeho základ tvoří standardní genetický algoritmus, který je propojen s náhradním modelem podle individuálního nebo generačního evolučního řízení.

Základní kroky algoritmu jsou popsány pseudokódem v obrázku 3.2. Důležitým vstupem algoritmu je databáze trénovacích dat

$$\mathbf{D} = \{\mathbf{x}_k^{(\mathrm{d})}, \mathbf{x}_k^{(\mathrm{s})}, y_k\}_{k=1}^{N}. \tag{3.1}$$

Ve vzorci (3.1) odpovídá $\mathbf{x}_k^{(\mathrm{d})}$ diskrétním a $\mathbf{x}_k^{(\mathrm{s})}$ spojitým hodnotám nezávislých proměnných k-tého záznamu v databázi, v položce y_k je pak uložena hodnota empirické cílové funkce naměřená na tomto řešení $(\mathbf{x}_k^{(\mathrm{d})}, \mathbf{x}_k^{(\mathrm{s})})$. N je počet dat v databázi, dále v textu je pro tuto hodnotu také použito značení $|\mathbf{D}|$.

Algoritmus začíná přípravou úvodní populace přípustných řešení. Ta může být přímo zadána na vstupu (případně v databázi), nebo může být vygenerována náhodně. Součástí konkrétní implementace může být i specifikace pravděpodobnostních distribucí jednotlivých proměnných, které se pak použijí pro generování.

Velikost populace je při použití generačního evolučního řízení dána počtem řešení o, které je schopná najednou ohodnotit původní empirická cílová funkce. Pokud se použije individuální evoluční řízení, velikost populace je q-krát větší ($q \geq 1$), ovšem selekce vybírá rodiče pro křížení a mutaci pouze z jedinců, kteří byli ohodnoceni původní fitness.

Krokem (2) v pseudokódu začíná generace. Pokud by nebyl v databázi dostatek dat pro natrénování modelu, algoritmus funguje v podstatě jako obyčejný genetický algoritmus popsaný v části 2.1 s tím, že k ohodnocení používá pouze původní cílovou funkci. Její výsledky vždy ukládá do databáze. Pokud je dat dostatek, pokračuje se podle zvoleného evolučního řízení.

Individuální řízení nejdříve natrénuje z dostupných trénovacích dat model, a poté celou populaci velikosti $q \cdot o$ tímto modelem ohodnotí. Následuje výběr jedinců k přehodnocení původní cílovou funkcí. Uživatel může zvolit poměr, v jakém se použijí jednotlivé ze čtyř navržených metod výběru popsaných v kapitole 2.2. Nakonec dojde i k samotnému přehodnocení.

Pokud uživatel zvolil generační evoluční řízení, model se natrénuje a použije pouze po uběhnutí η generací v daném cyklu délky λ. V prvních generacích cyklu se použije původní cílová funkce.

Genetický algoritmus využívající náhradního modelu

Vstupy:

– specifikace optimalizační úlohy (diskrétní $v_1^{(\mathrm{d})}, \ldots, v_n^{(\mathrm{d})}$ a spojité $v_1^{(\mathrm{s})}, \ldots, v_r^{(\mathrm{s})}$ proměnné a jejich vlastnosti, omezení, o – velikost populace, která může být najednou ohodnocena originální cílovou funkcí, volitelně: p_0 – úvodní populace)

– parametry evolučního řízení (*individuální*, resp. *generační*; q – koeficient velikosti populace pro náhradní model, resp. λ – délka cyklu a η_0 – úvodní hodnota počtu originálně ohodnocených generací v cyklu η)

– parametry modelu ($s'_{\min}$ – min. počet dat pro práci s modelem, radiální funkce f_i, norma: eukleidovská nebo definovaná maticí $\mathbf{A}$, $s_{\min}$ – min. počet dat pro konstrukci jedné RBF sítě, k – počet množin pro křížovou validaci, e – typ chyby: MSE/AIC/BIC)

– databáze $\mathbf{D}$ trénovacích dat s ohodnocením přesnou cílovou funkcí

Algoritmus:

(1) $p \leftarrow$ úvodní populace: vezmi p_0 (pokud byla na vstupu) nebo náhodně vygeneruj novou splňující omezení; $\eta = \eta_0$

(2) **if** (malý počet dat v databázi $\mathbf{D}$, tzn. $|\mathbf{D}| < s'_{\min}$)

(3) ohodnoť p *originální fitness*, ulož výsledky do $\mathbf{D}$ a pokračuj krokem (11)

end if

if (*individuální* evoluční řízení)

(4) *model* $\leftarrow$ NatrénujModel($s_{\min}$, f_i, $||.||$, $\mathbf{D}$, r, e) – viz obr. 3.3

(5) ohodnoť p (velikosti $q \cdot o$) náhradním modelem

(6) vyber o jedinců a přehodnoť je *originální fitness* funkcí

else (*generační* evoluční řízení)

if (již proběhlo η z λ generací v cyklu)

(7) *model* $\leftarrow$ NatrénujModel($s_{\min}$, f_i, $||.||$, $\mathbf{D}$, r, e)

(8) ohodnoť p náhradním modelem

else

(9) ohodnoť p *originální fitness* funkcí

end if

(10) uprav η podle chyby predikce modelu

end if

(11) $p \leftarrow$ nová populace vygenerovaná genetickými operacemi (elitismem, selekcí, mutací a křížením) z o (vybraných) jedinců

(12) **if** (nebylo nalezeno vhodné řešení **and** doba výpočetu nepřekročila limit **and** ohodnocení populace se mění): pokračuj krokem (2)

Výstup: jedinci zpracovaní genetickým algoritmem

Obrázek 3.2: Pseudokód genetického algoritmu využívající modelu

Generace se zakončuje vygenerováním nové populace genetickými operacemi. Algoritmus končí při splnění některého z ukončovacích kritérií. Pokud dopředu existuje specifikace hledaného nejlepšího řešení, skončí se po jeho nalezení. Uživatel také může omezit využitelné výpočetní prostředky, například počet generací nebo počet ohodnocení (nejčastěji původní) cílovou funkcí. Poslední skupina ukončovacích kritérií sleduje vývoj několika posledních generací. Pokud se například ohodnocení nejlepšího jedince v populaci nemění po několik generací za sebou, algoritmus se ukončí také.

3.3 Příprava a trénování modelu

Zatímco jsme v předchozím oddíle popsali naší verzi rozšířeného genetického algoritmu, v této části se věnujeme algoritmu vytváření a trénování náhradního modelu cílové funkce, což je obsahem prvního cíle této práce. Zevrubně se budeme věnovat diskrétním proměnným, pro které nelze jednoduše použít RBF sítě.

Algoritmus trénovací procedury je popsán na obrázku 3.3. V prvním kroku vezme dostupná ohodnocená řešení z databáze a rozdělí je do shluků podle hodnot diskrétních proměnných. Důležité je, že potřebujeme, aby všechny shluky obsahovaly alespoň $s_{\min}$ dat.

Hierarchické shlukování s minimální velikostí shluků

Číslo $s_{\min}$ je zadáno uživatelem a jeho hodnota do značné míry závisí na konkrétní zpracovávané úloze. Vyšší hodnoty zaručí větší shluky, díky čemuž bude možné trénovat RBF sítě s větším množstvím komponent, a tedy s komplikovanějším tvarem. Pokud však data obsahují mnoho různých kombinací diskrétních hodnot, bývá vhodné $s_{\min}$ snížit. Řešení s různými diskrétními kombinacemi se totiž obvykle výrazně liší v hodnotách cílové funkce i pro podobné nebo stejné hodnoty spojitých proměnných. Vyšší $s_{\min}$ by v tomto případě donutilo shlukování slučovat řešení s dosti odlišnými diskrétními hodnotami, a tím také různými hodnotami fitness, což by model realizovaný RBF sítěmi nebyl schopný zachytit. Hodnota $s_{\min}$ musí splňovat nerovnost

$$s_{\min} \geq \left\lceil \frac{k}{k-1} p \right\rceil, \tag{3.2}$$

kde p je počet trénovaných parametrů modelu a k počet množin při křížové validaci. Uvedený počet dat je nutný k natrénování alespoň jedné radiální kompo-

nenty, a to také v případě trénování při křížové validaci.

Shlukování v našem algoritmu vychází z hierarchického shlukování, které je obsaženo v systému Matlab. Nejdříve se získají vektory všech kombinací diskrétních hodnot, které jsou zastoupeny v řešeních z databáze. V témže průchodu databází se také získají jejich počty. Pro tyto vektory se spočítají vzájemné vzdálenosti, na

---

NatrénujModel($s_{\min}$, f_i, $||.||$, $\mathbf{D}$, r, e)
Argumenty: $s_{\min}$ – minimální velikost shluků,
f_i – radiální funkce, $||.||$ – parametry normy,
$\mathbf{D}$ – databáze, r – počet spojitých proměnných
e – typ chyby: MSE, AIC nebo BIC
Algoritmus:
(1) rozděl data z databáze podle hodnot diskrétních proměnných do shluků o velikosti alespoň $s_{\min}$, $s_{\min} \geq \lceil \frac{k}{k-1} p \rceil$ (kde k je počet množin křížové validace a p počet parametrů jedné radiální komponenty f_i)
$m \leftarrow$ počet shluků;
$|cl_j| \leftarrow$ velikost j-tého shluku, $j = 1, \ldots, m$
$\{\mathbf{N}_j\}_{j=1}^{m} \leftarrow$ množiny diskrétních hodnot jednotlivých shluků
(2) $\{g_j^{\max}\}_{j=1}^{m} \leftarrow$ maximální počty komponent RBF sítí jednotlivých shluků (pro Gaussovy funkce $g_j^{\max} = \lfloor \frac{\lfloor \frac{k-1}{k} |cl_j| \rfloor}{2+r} \rfloor$)
for každý shluk cl_j, $j = 1, \ldots, m$
for počet komponent $g = 1, \ldots, g_j^{\max}$
(3) $mse[j, g] \leftarrow$ průměrná MSE (chyba) z k trénování použitím křížové validace
end for
(4) $g^\star \leftarrow$ počet komponent modelu s nejmenší hodnotou chyby typu e (MSE z $mse[j, g]$, AIC nebo BIC)
(5) $model_j \leftarrow$ model s $g^\star$ komponentami, nově natrénovaný na všech dostupných datech shluku cl_j
(6) $mse_j \leftarrow mse[j, g^\star]$
end for
Výstup: $\{model_j, mse_j, \mathbf{N}_j\}_{j=1}^{m}$

Obrázek 3.3: Pseudokód trénovací procedury

výběr je Hammingova metrika

$$d_{\mathrm{HAM}}(\mathbf{x}^{(\mathrm{d})}, \mathbf{y}^{(\mathrm{d})}) = \frac{\#_{j \in J}(x_j \neq y_j)}{n} \tag{3.3}$$

nebo Jaccardova metrika

$$d_{\mathrm{JACC}}(\mathbf{x}^{(\mathrm{d})}, \mathbf{y}^{(\mathrm{d})}) = \frac{\#_{j \in J}[(x_j \neq y_j) \wedge ((x_j \neq 0) \vee (y_j \neq 0))]}{\#_{j \in J}[(x_j \neq 0) \vee (y_j \neq 0)]}. \tag{3.4}$$

V rovnicích je J množina indexů diskrétních dimenzí, $n = |J|$ je délka vektorů $\mathbf{x}^{(\mathrm{d})}$ a $\mathbf{y}^{(\mathrm{d})}$ a $\#_{j \in J}()$ značí počet souřadnic j takových, že argument je logicky pravdivý. Označíme-li I indikátorovou funkci ($I(A) = 1 \Leftrightarrow A$ pravdivé, $I(A) = 0$ jinak), můžeme napsat také

$$\#_{j \in J}(A_j) = \sum_{j \in J} I(A_j).$$

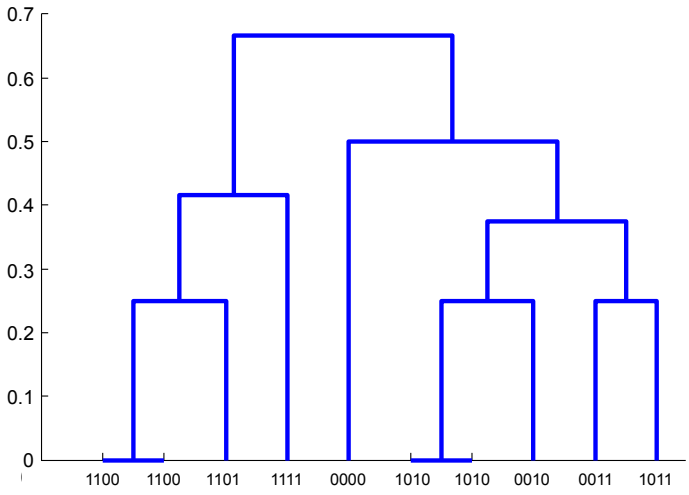
Klasické hierarchické shlukování pokračuje postupným slučováním nejbližších dosud získaných shluků, přičemž za shluky jsou na počátku prohlášeny všechny samostatné vektory. V každém kroku se sloučí vždy dva nejbližší shluky v jeden, takže pro n dat skončí shlukování po $(n - 1)$ krocích s jediným shlukem. Tento proces lze znázornit pomocí binárního stromu. V příkladu na obrázku 3.4 lze vidět hierarchické shlukování deseti náhodných čtyřbitových řetězců. Horizontální linie značí spojení dvou shluků; z jejich pozice lze na svislé ose odečíst Hammingovu vzdálenost, jakou měly shluky těsně před sloučením.

Shlukovací techniky v Matlabu neumožňují omezit velikost shluků. Náš algoritmus využívá slučovacího stromu, který postupně odspodu prochází. Každý list odpovídá jedné kombinaci diskrétních hodnot. Počty dat s jednotlivými diskrétními kombinacemi byly spočítány při průchodu databází, a každému listu lze tedy přiřadit jeho velikost danou tímto počtem. Velikost shluku, jemuž odpovídá netriviální podstrom, je daná součtem velikostí jeho listů.

Kdykoliv algoritmus narazí na shluk, jehož velikost je větší nebo rovna $s_{\min}$, uřízne podstrom odpovídající tomuto shluku a diskrétní kombinace sloučí do finálního shluku $\mathbf{N}_j$. Do $|cl_j|$ pak uloží počet odpovídajících dat v databázi, neboli velikost shluku. Pokud v kořeni zbývá shluk o velikosti menší než $s_{\min}$, spojí se s shlukem, který byl uříznut jako poslední. Proměnná m označuje celkový počet shluků.

Trénování sítí s radiálními bázovými funkcemi

Shluky získané v kroku (1) seskupují trénovací data s nejpodobnějšími kombinacemi diskrétních hodnot. Pro každý takový shluk cl_j je pak na spojitých proměn-



Obrázek 3.4: Dendrogram – hierarchické shlukování

ných odpovídajících dat natrénována jedna RBF síť. Maximální počet komponent každé sítě je shora omezen číslem

$$g_j^{\max} = \left\lfloor \frac{\lfloor \frac{k-1}{k} |cl_j| \rfloor}{p} \right\rfloor, \tag{3.5}$$

kde p je počet trénovaných parametrů každé radiální komponenty, závisející na počtu spojitých proměnných, a k počet množin pro křížovou validaci. Velikost shluku $|cl_j|$ je vynásobena členem $\frac{k-1}{k}$, aby byl zaručen dostatek trénovacích dat nejen pro trénování výsledné RBF sítě, ale i pro všechny iterace křížové validace. Praktické testy navíc ukázaly, že je vhodné trénovat sítě na mírně větším počtu dat než by přesně odpovídalo právě uvedenému vzorci. Jmenovatel proto vynásobíme ještě koeficientem q_g s doporučenou hodnotou $q_g = 1.2$, uživatel však toto číslo může zadat sám. Výsledný vztah pro Gaussovy funkce je

$$g_j^{\max} = \left\lfloor \frac{\lfloor \frac{k-1}{k} |cl_j| \rfloor}{q_g(2+r)} \right\rfloor. \tag{3.6}$$

Proměnná r udává počet spojitých proměnných. Parametry Gaussových funkcí zahrnují pro každou komponentu f_i její r-dimenzionální centrum $\mathbf{c}_i$, a dále dva skaláry: parametr β_i a váhu komponenty π_i. Celkem je tedy $p = 2 + r$.

V algoritmu následuje cyklus procházející všechny shluky cl_j, $j = 1, 2, \ldots, m$. V něm se postupně trénuje každá RBF síť. Nejprve se křížovou validací získá odhad střední kvadratické chyby $mse[j, g]$ pro každý počet komponent $g = 1, \ldots, g_j^{\max}$. V kroku (4) se pak vybere nejlepší počet komponent $g^\star$, pro který byla buď nejmenší chyba MSE, nebo pro které nabylo nejmenší hodnoty jedno z kritérií AIC nebo BIC.

Následuje opětovné natrénování sítě, tentokrát již bez křížové validace na všech dostupných datech. Pokud jsou použity náhodné startovní hodnoty parametrů pro trénování RBF sítě, je vhodné trénování několikrát opakovat a jako výslednou síť vybrat síť s nejmenší chybou. Tím zvýšíme pravděpodobnost nalezení globálního optima.

Vlastní trénování každé RBF sítě probíhá některou z běžných optimalizačních metod nejmenších čtverců, při které se hledají hodnoty parametrů nelineárních křivek prokládajících data. Tyto metody bývají citlivé na počáteční hodnoty. V našem algoritmu jsme pro Gaussovy funkce f_i nastavili:

- *Centra funkcí* $\mathbf{c}_i$. Nejdříve bylo vyzkoušeno shlukování dat metodou k středů a za centra funkcí jsme vzali středy výsledných shluků. Ukázalo se však, že v praktických aplikacích bývají data nerovnoměrně rozložena, a tak většina center vychází příliš blízko u sebe. Lepších výsledků jsme dosáhli při náhodném rovnoměrném rozložení center po prostoru vstupních dat. Pro jednotlivé spojité dimenze $l = 1, \ldots, r$ můžeme vyjádřit výběr počátečních hodnot $(\mathbf{c}_i^0)_l$ z rovnoměrného rozložení U vztahem

$$(\mathbf{c}_i^0)_l \sim U(\min_{k=1,\ldots,|cl_j|} (\mathbf{x}_k^{(\mathrm{s})})_l, \max_{k=1,\ldots,|cl_j|} (\mathbf{x}_k^{(\mathrm{s})})_l).$$

- *Váha π_i i-té radiální funkce.* Protože váhy v Gaussově funkci (2.5) přímo ovlivňují velikost amplitudy komponent, jako vhodné se ukazuje začít trénování s π_i rovnými průměrné hodnotě cílové funkce y_k trénovacích dat $(\mathbf{x}_k^{(\mathrm{s})}, \mathbf{x}_k^{(\mathrm{d})}, y_k) \in cl_j$ daného shluku

$$\pi_i^0 = \frac{1}{|cl_j|} \sum_{k=1}^{|cl_j|} y_k.$$

3.4 Ohodnocování populace modelem

Máme-li rozdělená trénovací data do shluků podle jejich diskrétních hodnot, pro každý shluk natrénovánu RBF síť a změřenou střední kvadratickou chybu, můžeme

takto vzniklý model

$$\{model_j, mse_j, \mathbf{N}_j\}_{j=1}^{m} \tag{3.7}$$

používat k ohodnocování jedinců. Ve výrazu (3.7) označuje $model_j$ sadu parametrů RBF sítě j-tého shluku (centra, parametry β a váhy), mse_j jeho odhad chyby predikce a $\mathbf{N}_j$ je množina diskrétních kombinací, které shluk obsahuje.

Ohodnocovaného jedince budeme označovat v souladu s předešlým značením dvojicí $(\mathbf{x}^{(\mathrm{s})}, \mathbf{x}^{(\mathrm{d})})$. Naším cílem je najít jeho přibližné ohodnocení $\hat{y}$.

Nejprve je potřeba určit index nejbližšího shluku c ve smyslu Hammingovy nebo Jaccardovy vzdálenosti (vzdálenosti budeme souhrnně označovat d_{DISCR}). Jeho nalezení lze vyjádřit jako

$$c = \arg \min_{j=1,\ldots,m} \frac{1}{|\mathbf{N}_j|} \sum_{\mathbf{y} \in \mathbf{N}_j} d_{\mathrm{DISCR}}(\mathbf{x}^{(\mathrm{d})}, \mathbf{y}). \tag{3.8}$$

Vezme se tedy takový index shluku j, pro který je nejmenší průměrná vzdálenost mezi diskrétními hodnotami ohodnocovaného jedince a diskrétními kombinacemi shluků. Pokud by se takto vybralo více shluků, vezme se ten s menší chybou mse_j.

Máme-li určen nejbližší shluk c, k získání ohodnocení $\hat{y}$ stačí dosadit odpovídající parametry RBF sítě do c-té modelové funkce f^c a spočítat její hodnotu pro spojité dimenze jedince

$$\hat{y} = f^c(\mathbf{x}^{(\mathrm{s})}).$$

V případě Gassových funkcí obsahuje $model_c$ sadu parametrů $\{\pi_i^c, \mathbf{c}_i^c, \beta_i^c\}_{i=1}^{g^\star}$ a uvedený výraz lze tedy napsat podrobněji

$$\hat{y} = f^c(\mathbf{x}^{(\mathrm{s})}) = \sum_{i=1}^{g^\star} \pi_i^c f_i^c(\mathbf{x}^{(\mathrm{s})}; \mathbf{c}_i^c, \beta_i^c) = \sum_{i=1}^{g^\star} \pi_i^c \ \mathrm{e}^{-\beta_i^c ||\mathbf{x}^{(\mathrm{s})} - \mathbf{c}_i^c||}.$$

Kapitola 4

Implementace

Tato kapitola je věnovaná konkrétní implementaci našich algoritmů. Po úvodní části představující použitý vývojový nástroj následuje podrobný popis implementace modelu, jeho parametrů a úprav, které vznikly jako reakce na reálnou implementaci. V závěru patří několik odstavců implementaci genetického algoritmu.

4.1 Prostředí

Pro implementaci našich algoritmů bylo zvoleno prostředí Matlab. Je to vývojové prostředí vhodné pro nejrůznější výpočty a vizualizace. Matlab používá svůj vlastní vysoko-úrovňový programovací jazyk. Svou syntaxí usnadňuje především práci s maticemi a vektory, jejichž zpracování je optimalizováno. Jako komerční nástroj je vyvíjen firmou MathWorks, v instalaci však najdeme zdrojové kódy k většině implementovaných funkcí.

K jádru systému je možné dokoupit různá rozšíření, tzv. toolboxy. V naší práci jsme využili hned čtyři: *Optimization Toolbox*, *Gentic Algorithm and Direct Search Toolbox*, *Statistics Toolbox* a *Database Toolbox*. První tři nám výrazně usnadnily práci s postupy, které jsou obecně známé, a nemuseli jsme je proto implementovat znovu. Poslední je použit spíše z implementačního důvodu, abychom mohli načítat naměřená data empirické funkce z databáze.

Systém Matlab byl použit ve verzi 7.5.0, označovaném také 2007b, základní funkčnost však byla ověřena také v předposlední verzi 7.7 (2008b).

4.2 Struktura a běh programu

V našem programu je potřeba rozlišovat dvě části. První z nich je trénování a použití regresního modelu, které bylo definováno jako první cíl práce a jejichž prototypové implementace jsou připraveny k testování a po drobných úpravách i použití v reálných aplikacích. Druhou částí je pak zbytek funkcí (včetně implementace genetického algoritmu), které slouží pouze k otestování modelu a nejsou pro reálné použití určeny.

Přirozenou programovací jednotkou Matlabu jsou funkce a do nich je rozdělen také náš program. V prostředí Matlabu se funkce ukládají do souborů se jménem stejným jako je název funkce, proto nebude činit problém popsané funkce ve zdrojových textech nalézt.

Inicializace a příprava dat

Všechny důležité parametry modelu i genetického algoritmu a ostatní nastavení jsou soustředěny do funkce `initialize(o, d)`. Prototypová implementace přijímá vstup z aplikace v oboru syntézy chemických katalyzátorů a jako parametr potřebuje na vstup dostat struktury `Optimization` a `Database`, které popisují tyto optimalizační úlohy. Jsou v nich definované jednotlivé dimenze řešení, rozdělení diskrétních a spojitých proměnných, omezení číselných proměnných, velikost populace, uložení dat v databázi apod. Další parametry naší metody lze snadno měnit přímo v souboru `initialize.m`.

Součástí funkce `initialize()` jsou také parametry pro přístup do databáze, ze které se načtou trénovací data obsahující řešení a jejich ohodnocení původní fitness funkcí. Pokud však máme data připravena jako strukturu Matlabu, lze je předat funkci jako nepovinný parametr a čtení z databáze se vynechá. To je praktické zejména v případě, kdy testujeme různé parametry na stále stejných datech, případně můžeme program používat i na počítači, z něhož přístup do databáze s daty nemáme. Na obrázku 3.1 funkce přibližně odpovídá levému hornímu rohu.

Další součásti programu

Patrně nejdůležitější z celého programu je funkce `fit_the_model()`, která z připravených dat konstruuje a trénuje model, a dále pak `model_fitness()`, která tímto modelem ohodnocuje populaci. Oběma se budeme zabývat v následujícím oddíle.

K jednoduché implementaci genetického algoritmu jsme využili již hotovou funkci `ga()` z Genetic Algorithm and Direct Search Toolbox, kterou jsme nepatrně upravili. Díky tomu je možné počítat počet vyhodnocení originální a modelovou fitness a pro naše potřeby jsme vylepšili možnost krokovat algoritmus po jednotlivých generacích.

Vlastní evoluční řízení, zjednodušené pro testování na umělé testovací fitness (více viz kapitola 4.4), je naprogramováno ve funkci `evoctrl_score()`, kterou je třeba zadat upravenému genetickému algoritmu jako fitness funkci. Testy optimalizace umělé testovací funkce s tímto algoritmem se budeme zabývat v následující kapitole.

Pokud se vrátíme k obrázku 3.1 ze strany 22, inicializace a příprava dat na něm odpovídá přibližně levému hornímu rohu, funkce `fit_the_model()`, `model_fitness()` a `evoctrl_score()` prostřední části a `ga()` sloupci vpravo.

4.3 Regresní model cílové funkce

V této sekci popíšeme zajímavější části trénovací a ohodnocovací funkce regresního modelu. Pozornost budeme věnovat zejména takovým částem kódu, které znamenaly využití složitějších funkcí Matlabu, nebo kde bylo nutno volit mezi různými alternativami.

Hierarchické shlukování

Trénovací funkce `fit_the_model()` po spočítání diskrétních kombinací tyto kombinace funkcí `hier_nom_cluster()` shlukuje. Využívá přitom hierarchického shlukování ze Statistics Toolbox: upravené verze funkce `pdist()`, počítající vzájemné vzdálenosti kombinací, a funkce `linkage()`, která tyto vzdálenosti využívá pro konstrukci slučovacího stromu, tzv. dendrogramu (viz kapitola 3.3).

Statistics Toolbox nabízí i jiné metody shlukování, například metodu k středů nebo pomocí směsí pravděpodobnostních rozdělení. Použití těchto způsobů jsme však zavrhli, neboť pro získání shluků minimální velikosti by vyžadovaly několikanásobný restart s různými parametry, navíc nedovolují využívat lokálních vlastností dat, takže velikost shluků by podstatně více kolísala a největší shluky by seskupovaly zbytečně mnoho různých dat.

Spojité dimenze

Funkce `fit_the_model()` pokračuje cyklem přes všechny získané shluky – pro každý trénuje jednu RBF síť. Ukázalo se, že jednotlivá data z reálných aplikací mají často ze všech potenciálních spojitých dimenzí použitu jenom jejich malou část (v dodaném příkladu asi jednu čtvrtinu). Výrazně se přitom liší v tom, které dimenze to jsou v konkrétním případě. K diskrétním dimenzím dat jsme proto přidali ještě dalších r rozměrů daných binárním vektorem $b \in \{0,1\}^r$ s hodnotou

$$b_i = \begin{cases} 1 & i\text{-tá spojitá dimenze je použita} \\ 0 & i\text{-tá spojitá dimenze není použita} \end{cases} \qquad i = 1, \ldots, r,$$

$$b = (b_1\ b_2\ \ldots\ b_r),$$

kde r je velikost sjednocení spojitých dimenzí všech dat. Tím se sice výrazně zvýšil počet různých diskrétních kombinací, zato mohou RBF sítě pracovat jenom s těmi proměnnými, které jsou pro data v daném shluku relevantní.

Při konstrukci každé z m RBF sítí je spočítán bitový součet vektorů b^k všech řešení z shluku,

$$B_j = b^1 \mid b^2 \mid \ldots \mid b^{|cl_j|} \qquad j = 1, \ldots, m$$

(znak $\mid$ označuje operaci bitového součtu, tzv. OR). Síť je pak vytvořena jen pro takové dimenze i, pro které nabývá i-tý bit vektoru B_j hodnoty 1. Tímto se jednak šetří čas trénování modelu, jednak se zpřesňuje jeho použití, neboť jsou omezeny redundantní informace.

Křížová validace

V implementaci našeho modelu jsme oproti algoritmu popsaném na obrázku 3.3 přidali možnost křížovou validaci vynechat a model vybírat pouze na základě popsaných kritérií, tedy vybrat model s nejmenší hodnotou AIC nebo BIC, případně model s nejmenší chybou na trénovacích datech MSE. I když v tomto případě hrozí dříve zmíněné přeučení, ušetří se výrazné množství času při trénování, což může být v některých případech žádoucí.

Parametrem je také možné nastavit, že se křížová validace automaticky nepoužije, pokud je pro nějaký shluk tak malé množství dat, že nelze natrénovat více než jednu komponentu.

Několikanásobné trénování

Praktické testy ukázaly, že pro dosažení lepších výsledků je často vhodné trénování jedné sítě několikrát opakovat a následně vybrat nejlepší model. Jde o již dříve popsaný případ, kdy se počáteční parametry RBF sítě určují náhodně. Není výjimkou, že některé trénování skončí po několika málo iteracích v lokálním minimu. Několikanásobným startem však podstatně zvyšujeme šanci, že některý pokus globální minimum najde.

Tuto možnost jsme přidali jednak do výše zmíněného případu, kdy se nepoužije křížová validace, jednak do konečného trénování v případě jejího použití. Při tomto konečném trénování již nemůže nastat přeučení, protože počet komponent již byl zvolen předtím podle validačních dat.

Trénování metodou nejmenších čtverců

Pro samotné trénování, tedy hledání přesných parametrů RBF sítí, jsme nejdříve použili ze Statistics Toolbox funkci `nlinfit()`. Po několika testech se však ukázalo, že je tato funkce velmi náchylná jednak k nepřesně zadaným počátečním hodnotám hledaných parametrů, jednak k množství trénovacích dat.

Podstatně lepších výsledků jsme začali dosahovat při použití funkce `lsqcurvefit()` z Optimization Toolbox, která, ač určená ke stejnému účelu, přinesla v obou zmíněných parametrech zlepšení. V naší verzi Matlab standardně používá variantu metody vnitřních bodů vycházející z klasické Newtonovy metody a využívající předpodmíněných sdružených gradientů. Ta je však stále poměrně citlivá na množství trénovacích dat. Z testů vyplynulo, že pro počet dat, který odpovídá počtu trénovaných parametrů, nebo je jen o málo větší, je vhodnější použít Levenberg-Marqardtovu metodu [14], kterou `lsqcurvefit()` nabízí také.

Použité funkci lze nastavit mnoho parametrů. Nejdůležitější z nich jsou patrně ukončovací kritéria, která v podstatě odpovídají popsaným kritériím ukončování genetických algoritmů. Je možné omezit počet iterací, počet vyhodnocení trénované funkce, určit minimální změnu jejích hodnot v jednom kroku nebo minimální vzdálenost dvou různých hodnot trénovaných parametrů. Zde platí, že první dvě kritéria odpovídají omezení výpočetních prostředků, zatímco druhá dvě ukazují, že trénování zřejmě dosáhlo minima (zda-li lokálního nebo globálního nelze jednoduše určit).

Potřebný počet iterací i funkčních vyhodnocení k dosažení minima roste s dimenzí a množstvím trénovacích dat a samozřejmě také s klesající minimální

změnou funkčních hodnot nebo minimálním krokem trénovacího algoritmu. Vyzkoušeny byly takové kombinace, že trénování celého modelu na reálných datech trvalo od několika málo minut po několik hodin.

4.4 Genetický algoritmus a evoluční řízení

V implementaci genetického algoritmu jsme oproti popisu v kapitole 3.2 některé věci zjednodušili.

Trénování modelu. Náhradní model jsme natrénovali pouze před celým během algoritmu, další trénování s nově „naměřenými“ daty již za běhu algoritmu nenastalo. Trénovací množina však byla již před startem algoritmu tak rozsáhlá, že nepředpokládáme, že bychom tím výsledky ještě zásadně zlepšili.

Úprava hodnot η v generačním evolučním řízení. Při použití generačního evolučního řízení jsme pevně zvolili hodnoty λ a η a další úprava těchto parametrů neproběhla. Toto pravděpodobně zhoršilo přesnost tohoto druhu optimalizace, na druhou stranu přineslo zajímavé výsledky v počtu ohodnocení původní cílovou funkcí.

Diskrétní hodnoty. Protože jsme genetický algoritmus mohli zkoušet pouze na umělé testovací ne-empirické funkci akceptující spojité vstupy, implementaci genetických operací pro řešení obsahující diskrétní proměnné jsme vynechali a optimalizace probíhala pouze nad řešeními složenými z proměnných spojitých.

I přes tato zjednodušení jsme algoritmus podrobili podrobným testům, kterým je věnována následující kapitola.

Kapitola 5

Výsledky testů

Tato předposlední kapitola se věnuje otestování našeho modelu. Na syntetických testovacích i reálných datech ukážeme, čeho jsme v naší práci dosáhli a jakým způsobem se model může v reálných aplikacích použít.

Testovací problémy se výrazně liší svou podstatou i výsledky. Zatímco v případě umělé testovací fitness jde o optimalizaci hladké funkce složené z parabol a sinusových křivek, ve druhém případě žádné explicitní vyjádření neexistuje, navíc jsme měli k dispozici pouze omezený počet dat a nová měření již nebylo možné udělat.

5.1 Testovací problém 1: Umělá testovací fitness

Modelovaná a optimalizovaná funkce v této části vychází z článku [24]. Valero a kol. v něm uvádí tuto funkci jako funkci podobnou chemickým empirickým funkcím. Z důvodu, že genetický algoritmus Matlabu fitness minimalizuje, jsme vzali její zápornou hodnotu. Výpočet funkční hodnoty $\vartheta(x_1, x_2, x_3, x_4, x_5)$ se dá vyjádřit jako

$$\vartheta(x_1, x_2, x_3, x_4, x_5) = -A(x_1, x_2) - B(x_2, x_3)C(x_3, x_4, x_5) \tag{5.1}$$

$$\begin{aligned}
A(x_1, x_2) &= 0.6g(x_1 - 0.35, x_2 - 0.35) + 0.75g(x_1 - 0.1, x_2 - 0.1) + \\
&\quad + g(x_1 - 0.35, x_2 - 0.1) \\
B(x_2, x_3) &= 0.4g(x_2 - 0.1, x_3 - 0.3) \\
C(x_3, x_4, x_5) &= 5 + 25\left[1 - \sqrt{1 + (x_3 - 0.3)^2 + (x_4 - 0.15)^2 + (x_5 - 0.1)^2}\right] \\
g(a, b) &= 100 - \sqrt{(100a)^2 + (100b)^2} + 50\frac{\sin\sqrt{(100a)^2 + (100b)^2}}{\sqrt{(100a)^2 + (100b)^2 + 0.0001}},
\end{aligned}$$

kde vstupní vektor musí splňovat omezení

$$\sum_{i=1}^{5} x_i = 1 \qquad \wedge \qquad x_i \in [0,1], \quad i = 1, \ldots, 5. \tag{5.2}$$

Výhodou této funkce je, že není empirická. Jak uvidíme dále, natrénovaný model tak bylo možné využít k otestování celého systému genetického algoritmu s náhradním modelem.

Trénování modelu

Abychom mohli natrénovat model, bylo nejdříve potřeba vytvořit trénovací množinu dat. Vygenerovali jsme 2000 náhodných pětirozměrných vektorů, které vyhovovaly omezením (5.2). Pro tato virtuální řešení jsme spočítali hodnotu výše definované funkce a výsledky uložili do databáze.

Na připravených datech jsem spustili náš program, který po inicializaci a načtení dat natrénoval model funkcí `fit_the_model()`. Protože se v datech nevyskytovaly žádné diskrétní proměnné a všechny rozměry řešení byly vždy využity, pro všechna data byl použit jediný shluk s jedinou RBF sítí. Z praktických důvodů jsme omezili počet komponent číslem 7, model jich však využil maximálně 5.

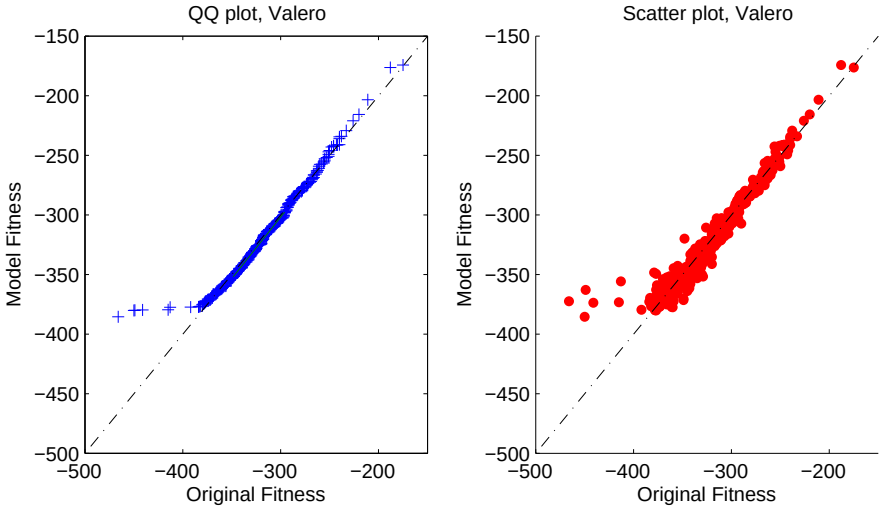
Model jsme trénovali dvanáctkrát s různým nastavením; podrobnosti jsou zaznamenány v tabulce 5.1. Tamtéž jsou také numerické výsledky trénování: průměrná chyba MSE na validačních množinách z křížových validací a MSE počítaná při závěrečném trénování na všech 2000 datech.

Jeden z modelů (v tabulce 5.1 v řádku 6) můžeme ilustrovat i graficky. Na obrázku 5.1 je QQ-plot a scatter plot na náhodných čtyř stech trénovacích datech, na obrázku 5.2 na čtyř stech nových datech testovacích, tedy odlišných od trénovací množiny.

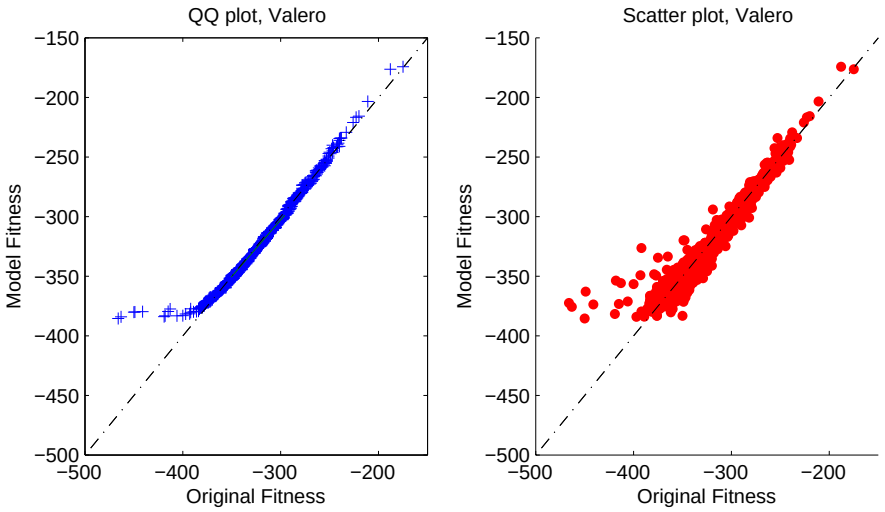
QQ-plot ukazuje rozložení kvantilů původní a modelované fitness, v ideálním případě by měly všechny znaky + ležet na hlavní diagonále grafu. Scatter plot porovnává přímo hodnoty obou funkcí pro jednotlivá řešení, a také zde by měly body ležet na hlavní diagonále. Z obou dvojic grafů je vidět, že model je schopen popsat většinu dat velmi dobře, pouze pokud originální fitness klesne přibližně pod hodnotu -400, model cílovou funkci nadhodnocuje. Vzhledem k tomu, že takových dat se v trénovací množině vyskytovalo méně než 2 %, je schopnost modelu aproximovat tuto umělou testovací funkcí velmi dobrá. Kvalitu modelu potvrzuje také to, že výsledky na trénovacích a testovacích datech se pouze minimálně liší.

pokus	krit.	f–evals	Iters	TolFun	#klast.	#komp.	valid. MSE	trén. MSE
1	AIC	4000	700	$1 \cdot 10^{-5}$	45	3	227.38	232.30
2	AIC	7000	2000	$5 \cdot 10^{-5}$	45	3	192.58	184.37
3	AIC	10000	5000	$1 \cdot 10^{-6}$	45	2	193.15	181.36
4	AIC	4000	700	$1 \cdot 10^{-5}$	15	3	207.46	223.73
5	AIC	7000	2000	$5 \cdot 10^{-5}$	15	4	191.82	181.92
6	AIC	10000	5000	$1 \cdot 10^{-6}$	15	2	192.79	181.42
7	BIC	4000	700	$1 \cdot 10^{-5}$	45	3	223.53	188.95
8	BIC	7000	2000	$5 \cdot 10^{-5}$	45	5	203.04	182.28
9	BIC	10000	5000	$1 \cdot 10^{-6}$	45	3	186.41	181.49
10	BIC	4000	700	$1 \cdot 10^{-5}$	15	2	207.88	181.58
11	BIC	7000	2000	$5 \cdot 10^{-5}$	15	3	208.50	182.64
12	BIC	10000	5000	$1 \cdot 10^{-6}$	15	2	197.82	177.48
průměr						2.92	**202.70**	**189.96**

Tabulka 5.1: Trénování modelu umělé testovací fitness funkce. Vysvětlivky zkratek viz tabulka 5.2



Obrázek 5.1: QQ a scatter plot natrénovaného modelu a originální fitness. Výsledky na *trénovacích* datech



Obrázek 5.2: QQ a scatter plot natrénovaného modelu a originální fitness. Výsledky na *testovacích* datech

Urychlení genetického algoritmu

Díky tomu, že umělá testovací fitness (5.1) umožňuje přesně ohodnotit jakékoliv navržené řešení, lze tuto funkci použít k otestování naší verze genetického algoritmu. V něm bude nahrazovat empirické ohodnocení experimentem. Optimalizace bude hledat její minimum, v celé této části 5.1 tedy platí: čím nižší hodnota fitness, tím lepší výsledek.

Minimum funkce ϑ je přibližně v bodě

$$x_{\min} = (0.351792, 0.098522, 0.298391, 0.150648, 0.100648),$$

kde nabývá po zaokrouhlení hodnoty

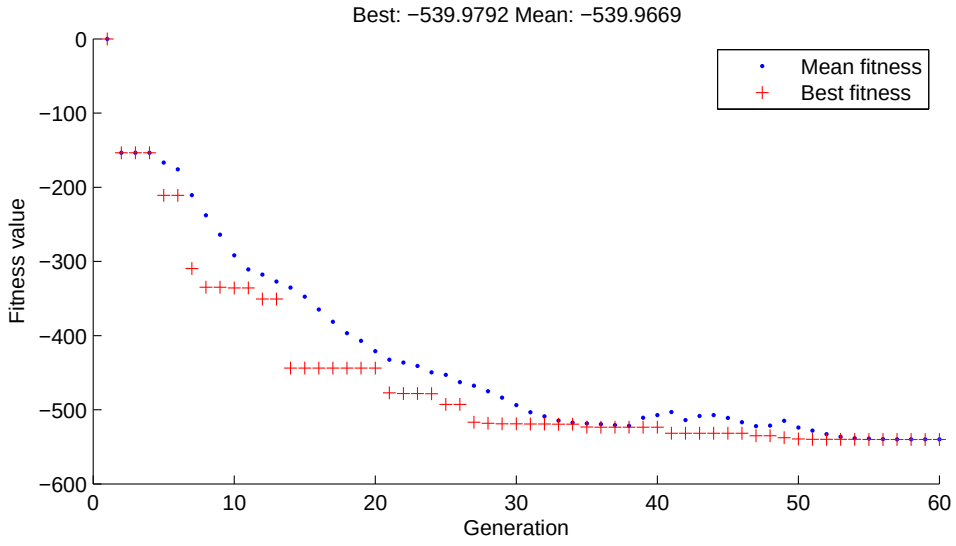
$$\vartheta(x_{\min}) = -547.724845.$$

Výsledky byly získány pomocí nelineární minimalizace s omezením, použita byla funkce `fmincon()` z Optimization Toolbox, která používá gradientní metodu sekvenčního kvadratického programování [8]. Jako počáteční hodnotu minimalizace jsme zvolili řešení nalezené genetickým algoritmem.

Podívejme se nejdříve na průběh genetického algoritmu **bez použití modelu**. Při všech následujících měřeních jsme používali velikost populace ohodnocované originální „empirickou“ fitness funkcí $o = 92$, minimální změna funkčních hodnot byla nastavena na $\texttt{TolFun} = 5 \cdot 10^{-7}$.

Typický vývoj střední a nejlepší fitness je na obrázku 5.3. Algoritmus často skončí s řešením, jehož funkční hodnota je o 5–15 procent horší než globální optimum. Charakteristické je pozvolné klesání fitness a celkový běh algoritmu po větší množství generací. Při stonásobném spuštění dosahoval algoritmus průměrně nejlepší fitness $\bar{\vartheta}_{\min} = -493.1415$, probíhalo v průměru 45.5 generací s 4188 vyhodnoceními originální cílovou funkcí. Takto špatné výsledky jsou nejspíše způsobeny tím, že vzhledem k náročnosti dané úlohy byla pro obyčejný genetický algoritmus zvolena příliš malá populace. Velikost však byla vzata přímo z reálné aplikace tak, aby se dala srovnat s genetickým algoritmem využívající náhradního modelu.

Algoritmus výrazně změní chování, pokud zapojíme náhradní model v režimu **individuálního evolučního řízení**. Velikost populace ohodnotitelná původní cílovou funkcí i minimální změna funkčních hodnot byly zachovány ($o = 92$, $\texttt{TolFun} = 5 \cdot 10^{-7}$). Koeficient q byl zvolen $q = 10$, takže velikost populace generovaná genetickými operacemi byla $|p| = oq = 920$.

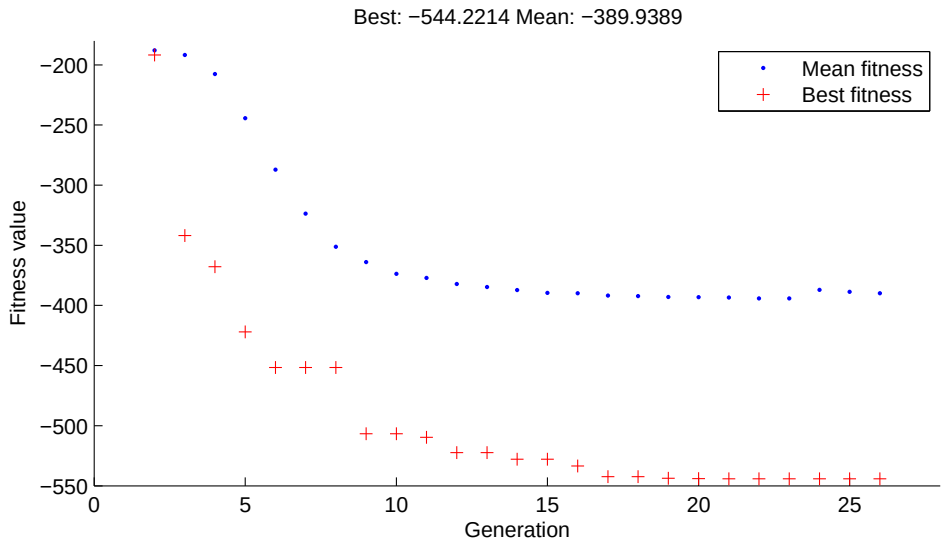


Obrázek 5.3: Vývoj průměrné a nejlepší fitness bez použití modelu

Vývoj průměrného a nejlepšího ohodnocení z jednoho běhu genetického algoritmu najdeme na obrázku 5.4. Na první pohled je patrné velmi rychlé nalezení poměrně slibných řešení, algoritmus prakticky vždy během prvních patnácti generací najde řešení s ohodnocením nižším než $\vartheta \leq -520$. Nejtypičtější je však konvergence k řešení, které je velmi blízké skutečnému optimu.

Při stonásobném opakování dosahoval algoritmus v průměru nejlepší fitness s hodnotou $\bar{\vartheta}_{\text{min}} = -542.0156$, což je pouze o jedno procento horší výsledek než optimum nalezené gradientní metodou. Průměrný počet generací byl 27.2, což odpovídá přibližně 2502 vyhodnocení původní cílové funkce.

Jestliže individuální řízení je typické kvalitou nalezeného řešení, pak **generační evoluční řízení** představuje nejúspornější variantu v počtu ohodnocení původní empirickou funkcí. V průměru dosahuje nejlepší fitness srovnatelných výsledků jako evoluce bez použití modelu, sto opakování dalo v průměru $\bar{\vartheta}_{\text{min}} = -498.8515$. Průměrný počet generací v tomto případě není příliš zajímavý údaj, neboť velmi závisí na parametrech generačního řízení λ a η, ale v počtu ohodnocení původní cílovou funkcí je tento způsob jasně nejlepší: průměrný počet ohodnocení je 1214, tedy asi třetinový ve srovnání s evolucí bez náhradního modelu.

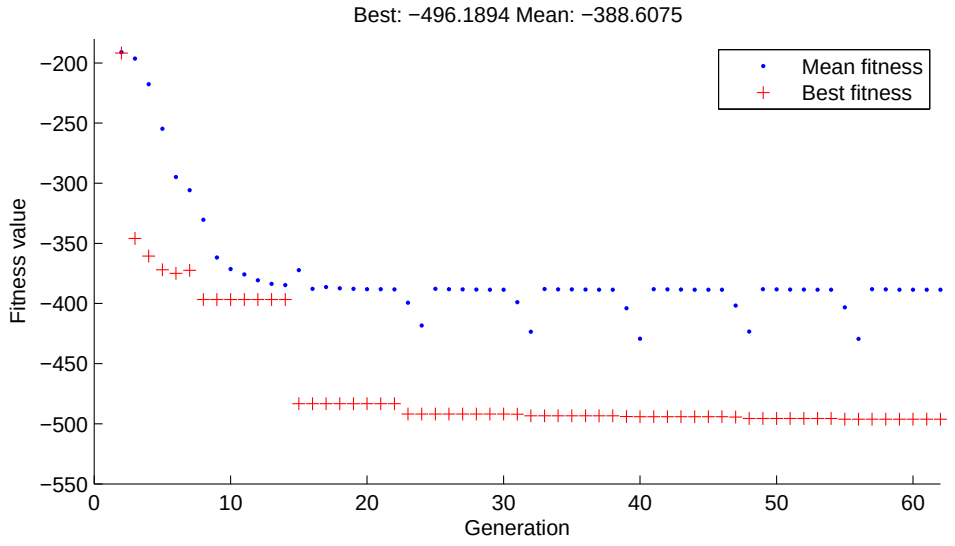


Obrázek 5.4: Vývoj průměrné a nejlepší fitness při individuálním evolučním řízení

Příklad vývoje nejlepší a průměrné fitness najdeme na obrázku 5.5. Abychom omezili velké rozdíly fitness hodnot nejlepších jedinců při ohodnocení modelem a skutečnou cílovou funkcí, upravili jsme generační řízení tak, aby se vždy elita ohodnotila originální funkcí. Bez této úpravy bylo častým jevem, že nejlepší fitness v populaci kolísala v závislosti na tom, která funkce byla k ohodnocení použita, což celkovou evoluci zpomalovalo.

Graficky znázorňují naše výsledky také grafy na následujících stranách. Na obrázku 5.6 jsou boxploty rozložení třech různých parametrů evoluce tak, jak byly naměřeny po stu doběhnutích genetického algoritmu bez použití modelu a s použitím individuálního a generačního evolučního řízení. Prostřední boxplot prvního grafu ukazuje, že téměř optimální řešení našel genetický algoritmus s individuálně řízeným modelem prakticky ve všech případech. Minimální výška třetího boxplotu v pravém grafu pak znázorňuje, že nízký počet ohodnocení původní cílovou funkcí je pro generační řízení opravdu charakteristický.

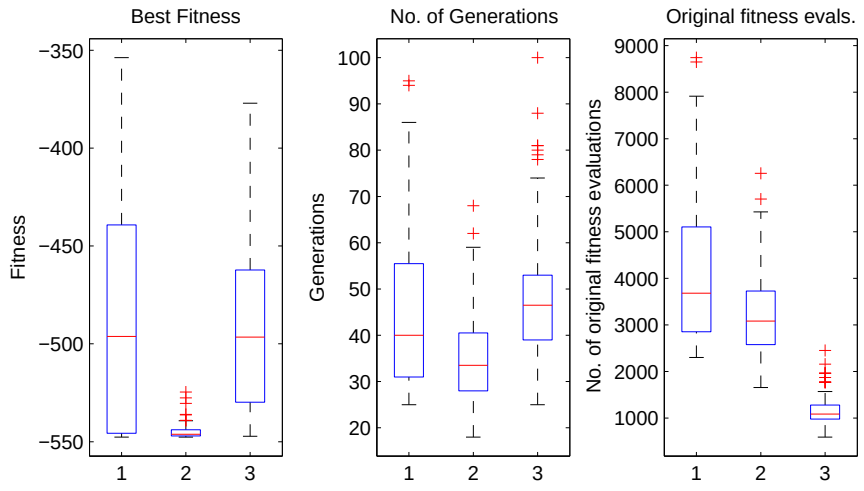
Následující graf na obrázku 5.7 zobrazuje průměrný průběh ohodnocení nejlepších jedinců v prvních 24 generacích ve třech zmíněných případech (bez modelu a s individuálním a generačním řízením). Je zřejmé, že generační řízení se i přes to, že používalo model ve dvou z osmi generací, v ohodnocení nejlepších jedinců neliší



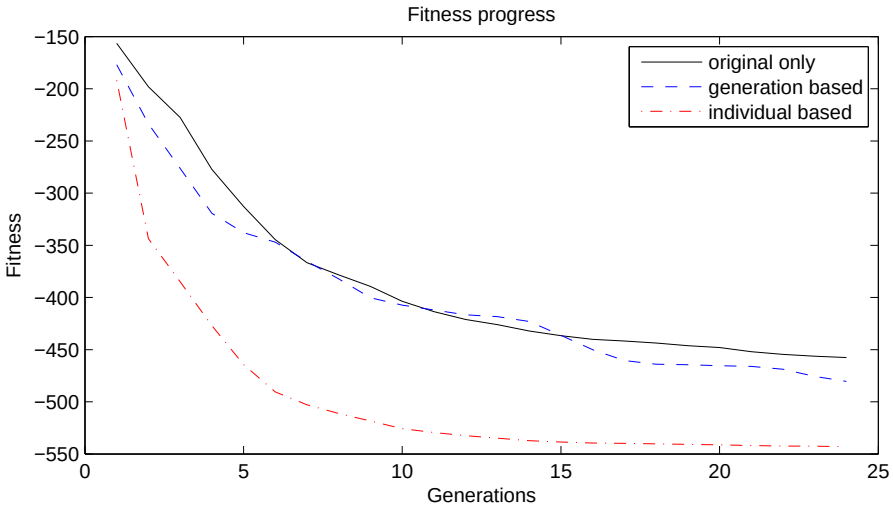
Obrázek 5.5: Vývoj průměrné a nejlepší fitness při generačním evolučním řízení

od běžného genetického algoritmu. Z grafu lze také vyčíst, že genetický algoritmus s individuálním řízením najde průměrný nejlepší výsledek generačního řízení (přibližně -500) za průměrně osm generací, což by v počtu ohodnocení původní fitness dokonce předčilo řízení generační ($8 \cdot 92 = 736$).

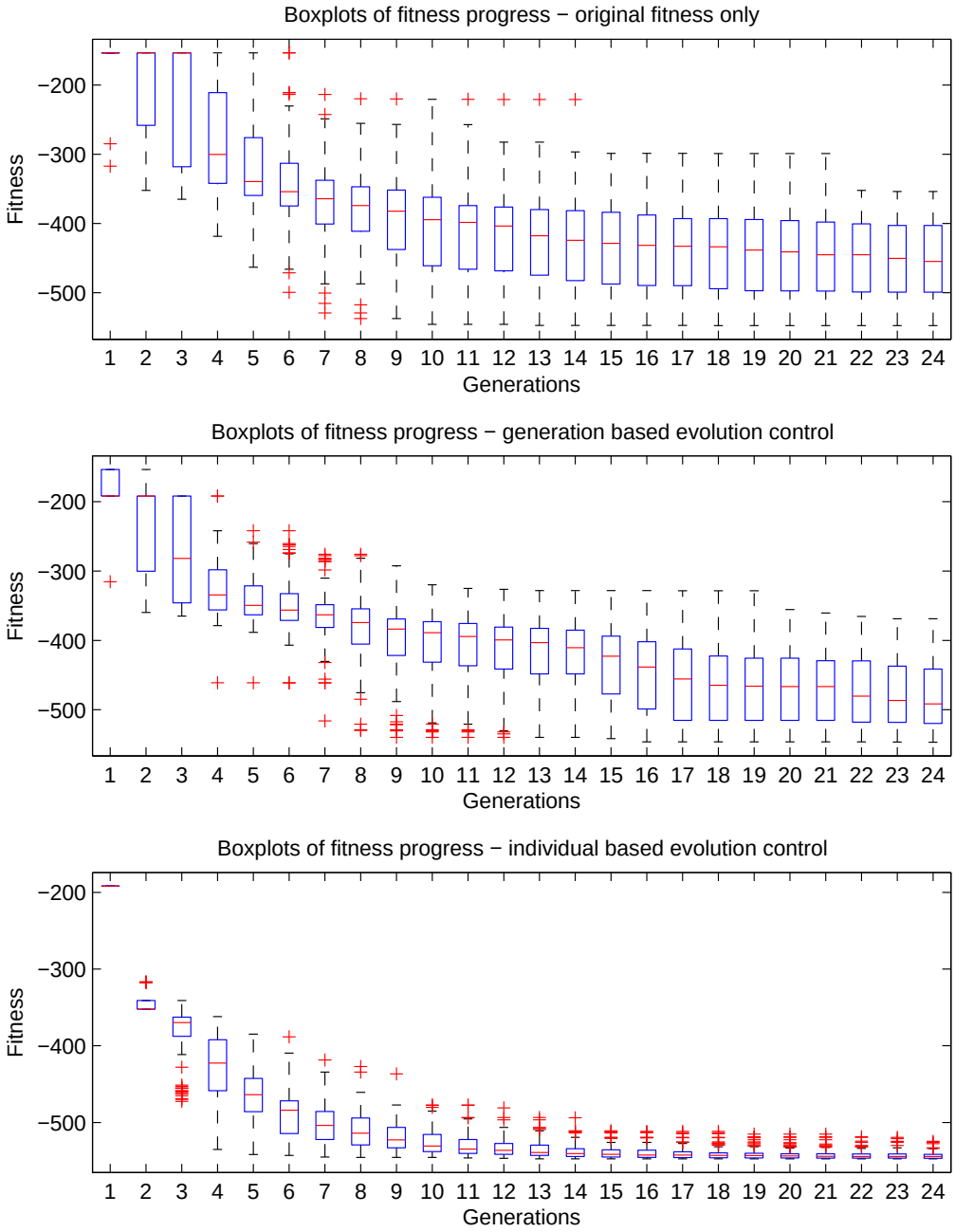
Vývoj ohodnocení nejlepších jedinců je také znázorněn na grafech na obrázku 5.8. Minimální výšky boxplotů v pravé polovině třetího grafu pouze potvrzují stabilitu konvergence individuálního evolučního řízení.



Obrázek 5.6: Rozložení nejlepších fitness (1. graf), celkového počtu generací (2. graf) a počtu ohodnocení emprickou funkcí (3. graf) bez použití modelu ($x = 1$) a s individuálním ($x = 2$) a generačním ($x = 3$) evolučním řízením



Obrázek 5.7: Vývoj nejlepší fitness – průměr ze 100 běhů genetického algoritmu



Obrázek 5.8: Vývoj fitness nejlepších jedinců v populacích při použití originální fitness, generačního a individuálního evolučního řízení

5.2 Testovací problém 2: Katalyzátor výroby kyseliny HCN

Druhá testovací úloha je z reálné aplikace: z výroby chemických katalyzátorů. Katalyzátor v ní urychluje a zesiluje chemickou reakci výroby kyseliny kyanovodíkové. Cílem optimalizace je nalézt složení takového katalyzátoru, aby reakce probíhala co nejefektivněji.

Řešení byla v našem konkrétním případě složena z jedenácti spojitých proměnných, které určovaly poměr použitých příměsí, a z jedné proměnné diskrétní. Ze spojitých dimenzí byla v každém řešení využita jen část: naplněna byla jedna až sedm z nich, nejčastěji to byly proměnné tři. Podle výše popsaného postupu jsme původně jedinou diskrétní proměnnou rozšířili o jedenáctirozměrný binární vektor indikující použité spojité dimenze.

Pro každé řešení (každé složení katalyzátoru)

$$(x_1^{(\mathrm{d})}, \ldots, x_{12}^{(\mathrm{d})}, x_1^{(\mathrm{s})}, \ldots, x_{11}^{(\mathrm{s})})_k$$

jsme měli také k dispozici naměřenou hodnotu y_k udávající kvalitu řešení, neboli výsledek empirické cílové funkce. Cílem bylo tuto empirickou funkci vhodně aproximovat.

Data obsahovala celkem 696 dat naměřených během osmi generací genetické optimalizace bez použití náhradního modelu. Počet různých (rozšířených) diskrétních kombinací byl 555, většina kombinací se tedy opakovala pouze jednou. Spojité proměnné splňovaly stejná omezení jako v předchozí úloze

$$\sum_{i=1}^{11} x_i = 1 \qquad \wedge \qquad x_i \in [0,1], \quad i = 1, \ldots, 11. \tag{5.3}$$

Trénování modelu

Model jsme pro měření konkrétních výsledků trénovali s různými parametry. Přehled numerických výsledků je v tabulce 5.2, naměřené chyby jsou v každém řádku průměrem ze čtyř pokusů. Na trénování mají zásadní vliv minimální velikosti shluků vznikající hierarchickým shlukováním dat. S těmito velikostmi bezprostředně souvisejí počty těchto shluků a také maximální počty komponent RBF sítí, které se pro tyto shluky trénují. Platí jednoduchý vztah: čím větší shluky, tím více radiálních komponent lze na datech jednoho shluku natrénovat, ale tím různorodější diskrétní kombinace se seskupí.

pokus	krit.	FunEvals	MaxIter	TolFun	#shluků	valid. MSE	trén. MSE
1	AIC	4000	700	$1 \cdot 10^{-5}$	45	222.04	152.48
2	AIC	7000	2000	$5 \cdot 10^{-5}$	45	226.20	134.47
3	AIC	10000	5000	$1 \cdot 10^{-6}$	45	224.25	128.41
4	AIC	4000	700	$1 \cdot 10^{-5}$	22	207.34	168.05
5	AIC	7000	2000	$5 \cdot 10^{-5}$	22	204.36	141.14
6	AIC	10000	5000	$1 \cdot 10^{-6}$	22	203.85	142.95
7	AIC	4000	700	$1 \cdot 10^{-5}$	15	229.68	182.07
8	AIC	7000	2000	$5 \cdot 10^{-5}$	15	224.16	170.17
9	AIC	10000	5000	$1 \cdot 10^{-6}$	15	226.87	174.20
10	BIC	4000	700	$1 \cdot 10^{-5}$	45	225.93	156.42
11	BIC	7000	2000	$5 \cdot 10^{-5}$	45	225.49	133.21
12	BIC	10000	5000	$1 \cdot 10^{-6}$	45	218.90	125.62
13	BIC	4000	700	$1 \cdot 10^{-5}$	22	209.81	159.84
14	BIC	7000	2000	$5 \cdot 10^{-5}$	22	205.05	144.78
15	BIC	10000	5000	$1 \cdot 10^{-6}$	22	203.04	140.45
16	BIC	4000	700	$1 \cdot 10^{-5}$	15	237.79	192.99
17	BIC	7000	2000	$5 \cdot 10^{-5}$	15	230.11	172.81
18	BIC	10000	5000	$1 \cdot 10^{-6}$	15	225.08	169.25
průměr						**219.45**	**154.97**

Tabulka 5.2: Trénování modelu empirické funkce katalyzátorů HCN. Vysvětlivky: *krit.* – použité kritérium výběru modelu; *FunEvals* – maximální počet vyhodnocení trénované funkce; *MaxIter* – maximální počet iterací trénovacího algoritmu; *TolFun* – minimální změna v trénované funkci; *#shluků* – počet diskrétních shluků; *valid. MSE* – průměrná střední kvadrat. chyba na validačních množinách z křížové validace; *trén. MSE* – střední kvadrat. chyba na trénovacích datech

Náš první návrh byl nastavit velikosti shluků co nejmenší tak, aby se minimalizoval počet různých diskrétních kombinací v nich sjednocených, a aby se tedy na jeden shluk natrénovala pouze jedna komponenta. Vyjádřeno v symbolech algoritmu z kapitoly 3.3,

$$s_{\min} = \left\lceil (1+\epsilon) \left\lceil \frac{k}{k-1}(r+2) \right\rceil \right\rceil. \tag{5.4}$$

kde ϵ je malé číslo zajišťující pro trénování o několik málo dat více než je jejich nezbytný minimální počet. Empiricky byla zjištěna optimální hodnota $\epsilon \approx 0.2$. Ve vzorci dále k značí počet množin křížové validace a r počet spojitých proměnných. V tabulce těmto pokusům odpovídají řádky 1, 2, 3, a 10, 11, 12.

Posléze jsme minimální velikost shluků nastavili trojnásobnou oproti hodnotě (5.4), do diskrétních shluků se tedy sloučilo přibližně třikrát více diskrétních kombinací (řádky 7–9 a 16–18). Trénování ale vybíralo model se třemi komponentami pouze pro dva z patnácti shluků, pro zbylých dvanáct použilo zpravidla dvě a pro jeden shluk pouze jednu komponentu. Model tedy více než dvě komponenty RBF sítí v podstatě nevyužil, pouze musel trénovat na rozmanitějších datech.

Trénování s parametrem $s_{\min}$ nastaveným na dvojnásobek hodnoty ze vzorce (5.4) pak přineslo zajímavé výsledky (v tabulce 5.2 řádky 4–6 a 13–15). Zatímco chyba měřená na trénovacích datech oproti prvnímu případu mírně vzrostla, podstatně důležitější průměrná chyba měřená na validačních datech z křížové validace znatelně poklesla.

Rozložení chyb pro jednotlivé sady parametrů zobrazuje také graf na obrázku 5.9. Také zde je zřetelně vidět, že model dokázal nejlépe popsat data při použití dvaceti dvou shluků a nejčastěji dvoukomponentových RBF sítí.

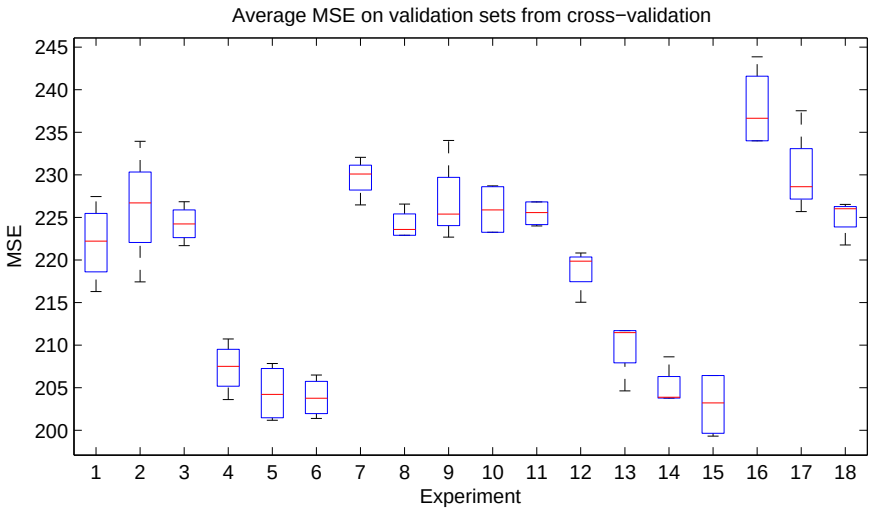
Model můžeme zhodnotit také pomocí QQ a scatter plotů, které byly použity v předchozím problému. Oproti grafům na straně 40 lze na obrázcích 5.10 a 5.11 jasně pozorovat, že popsání empirických dat pomocí RBF sítí je podstatně méně přesné. Celkové podhodnocování modelu je na QQ plotu patrné pro hodnoty původního ohodnocení větší než 30, v oblasti nejvyššího ohodnocení je chyba až 25 procent. Rozmístění bodů scatter plotů daleko nad i pod šikmou osou naopak zobrazuje podstatnou míru variability, kterou model nedokázal popsat.

I přes právě popsaná fakta se domníváme, že model by genetický algoritmus dokázal významným způsobem urychlit. To, že model dokáže postihnout celkový vývoj funkčních hodnot, můžeme vidět v grafech na obrázcích 5.12 a 5.13. V obou případech odpovídá i-tý z jedenácti grafů jedné spojité proměnné. Spojité křivky zobrazují funkční hodnoty modelu pro hodnoty i-té proměnné z definičního oboru

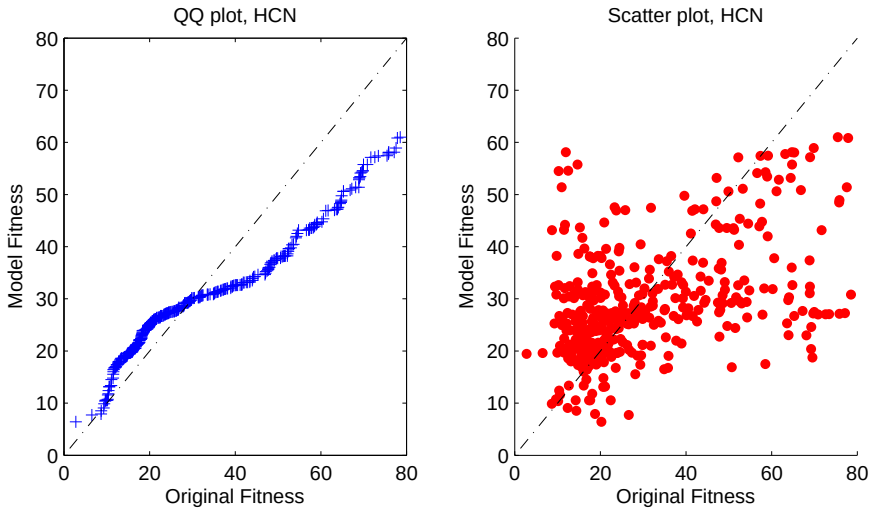
daného horizontální osou, hodnoty zbylých proměnných byly zafixovány v nule. Kroužky v i-tém grafu odpovídají průmětům skutečných dat a jejich hodnotám empirické fitness do i-té dimenze, znaky + promítají odpověď modelu pro stejné hodnoty nezávislých proměnných. Každému kroužku tedy odpovídá jeden znak +, který by měl být v ideálním případě ve středu odpovídajícího kroužku.

Poslední test, který jsme zpracovali, vyhodnocuje zpřesňování modelu novými daty získanými v průběhu genetického algoritmu ohodnocením (části) populace empirickou cílovou funkcí. Data z výroby katalyzátoru HCN k tomuto účelu dobře posloužila, neboť jejich součástí byl také údaj, ve které generaci genetického algoritmu byla naměřena.

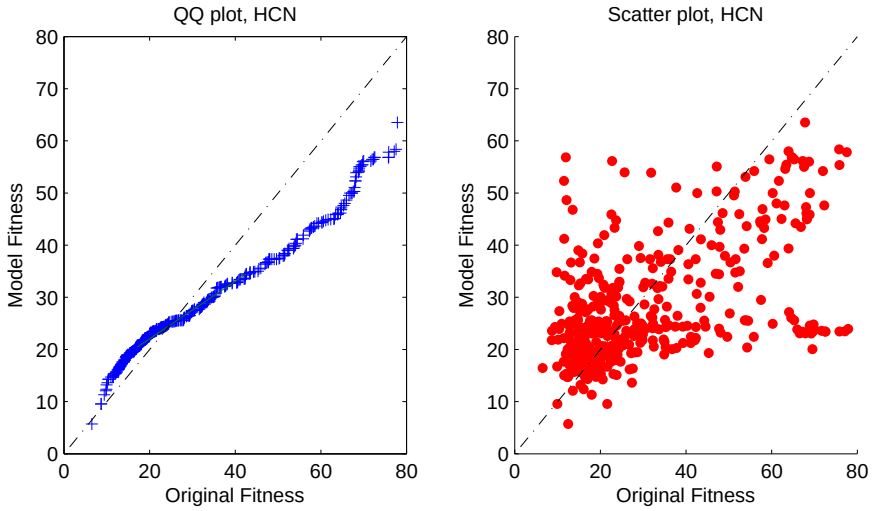
Aby výsledky byly porovnatelné, natrénovali jsme model postupně na datech z generací 1–4, 1–5, 1–6 a 1–7 a porovnali střední kvadratické chyby na osmé generaci. Výsledky udává tabulka 5.2 a scatter ploty porovnávající fitness skutečnou a spočítanou modelem jsou na obrázku 5.14. Zvláště z tabulky je patrné, že regresní schopnost modelu dle očekávání roste s množstvím použitých dat pro trénování.



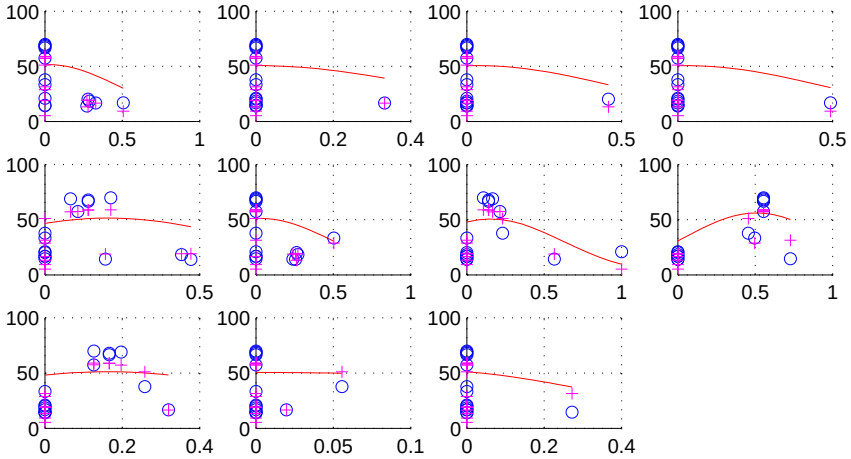
Obrázek 5.9: Rozložení průměrných středních kvadratických chyb naměřených na validačních množinách při křížové validaci



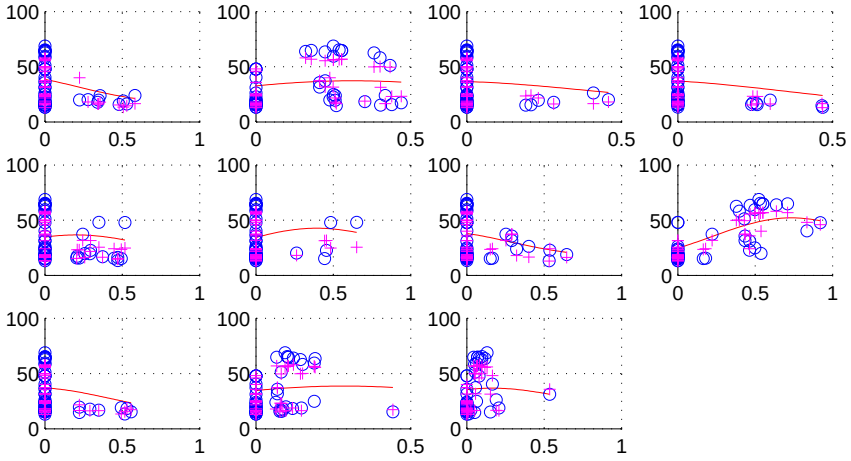
Obrázek 5.10: QQ a scatter plot natrénovaného modelu a originální fitness, náhodný výběr 400 ks trénovacích dat, model se 45 shluky



Obrázek 5.11: QQ a scatter plot natrénovaného modelu a originální fitness, náhodný výběr 400 ks trénovacích dat, model s 15 shluky



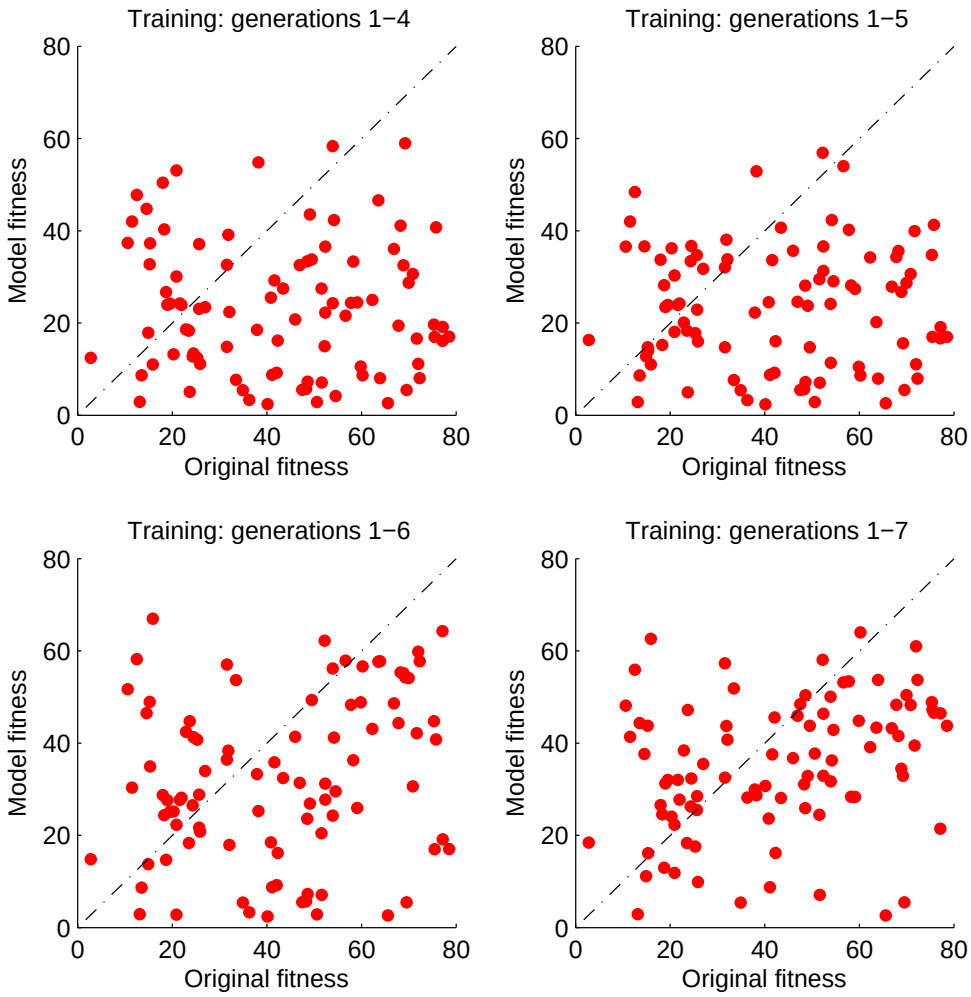
Obrázek 5.12: Průměty RBF sítě do každé z jedenácti spojitých proměnných, jeden ze 45 shluků. Vertikální osa: hodnoty fitness, horizontální viz text.



Obrázek 5.13: Průměty RBF sítě do každé z jedenácti spojitých proměnných, jeden z 15 shluků. Vertikální osa: hodnoty fitness, horizontální viz text.

trénovací množina	1–4 gen.	1–5 gen.	1–6 gen.	1–7 gen.
MSE na 8. generaci	1037.24	970.83	680.36	506.09

Tabulka 5.3: Trénování modelu novými daty v průběhu genetického algoritmu



Obrázek 5.14: Scatter ploty modelů natrénovaných na datech z generací 1–4, 1–5, 1–6 a 1–7, v grafech fitness měřená na 8. generaci

Kapitola 6

Závěr

6.1 Další vývoj

Oblast používání náhradních modelů v genetické optimalizaci se stále rozvíjí, a tak se nabízí poměrně mnoho směrů, kterými by se další pokračování této práce mohlo ubírat, navíc jistě v budoucnu přibudou další. Zkusme jmenovat alespoň některé naše současné návrhy.

Implementace genetického algoritmu s diskrétními proměnnými. Náš model, který nově zpracovává nejen proměnné spojité, ale také jejich kombinaci s proměnnými diskrétními, jsme na těchto kombinacích genetickým algoritmem neotestovali. Prvním přímým pokračováním práce tedy bude implementace genetických operací a upravení implementace genetického algoritmu tak, aby spojité i diskrétní hodnoty dokázal zpracovat.

Porovnání RBF sítí s jinými druhy regresních modelů. V zadání práce byly pro implementaci náhradního modelu určeny radiální bázové funkce. Jejich schopnost urychlovat genetický algoritmus by však bylo jistě zajímavé porovnat s dalšími druhy nelineárních regresních metod, například dopřednými neuronovými sítěmi, support vector machines nebo regresními stromy.

Hybridní model. Zhou a kol. [26] v jedné ze svých posledních prací kombinuje dva druhy modelů, tzv. lokální a globální. Možností kombinací různých modelů nebo druhů optimalizace, použití hierarchických přístupů a dalších metod je celá

řada a mohou ještě dále zlepšovat výsledky. Prozkoumání této oblasti a výběr vhodné kombinace je dalším možným pokračováním této práce.

Uživatelská přivětivost programu. Cílem této práce nebylo napsání optimalizačního softwaru řešícího obecné problémy. Naším cílem byly prototypové implementace modelu a jednoduchého genetického algoritmu, které ověřují, že naše obecné algoritmy a závěry v praxi opravdu budou fungovat. Dotažení do stavu použitelného v reálných aplikacích je naprosto přirozeným rozšířením započaté práce.

6.2 Shrnutí

Vraťme se na závěr ještě jednou k cílům práce, které jsme popsali v kapitole 1.1. Prvním bodem bylo vytvoření náhradního modelu fitness funkce do genetického algoritmu. Tyto modely nachází využití zejména v aplikacích, kde je vyhodnocení původní fitness funkce velmi nákladné. Jedná se o obecně používaný koncept, ale naším přínosem bylo zkonstruování takového modelu, který by dokázal využít kombinaci spojitých i diskrétních hodnot.

Po úvodu do celé problematiky a stručném popisu použitých metod jsme představili algoritmus trénování modelu v kapitole 3.3. Uveden byl podrobný pseudokód, v textu je samostatná část věnovaná zpracování diskrétních dimenzí, pro které jsme použili upravenou verzi hierarchického shlukování. V kapitole 4.3 jsou dále rozebrány problémy, na které jsme narazili při implementaci našeho modelu a jejichž vyřešení často zlepšilo původní teoretickou verzi.

Náš model jsme prostřednictvím prototypové implementace otestovali na dvou různých úlohách, které jsou popsány v kapitole 5. Jejich podstata i výsledky se značně liší. V prvním případě šlo o umělou testovací funkci, která má sice mnoho lokálních minim, ale je hladká, nepoužívá žádné diskrétní hodnoty a využito mohlo být v podstatě neomezené množství trénovacích dat. Díky tomu náš model dokázal aproximovat původní funkci velmi dobře, což dokládá kromě již uvedených výsledků a grafů také to, že se hodnoty modelu liší od původních hodnot jen přibližně o 2 % (model z šestého řádku tabulky 5.1 měřený na nových testovacích datech: MAE ≈ 7.5, průměrná fitness $\vartheta(\bar{\mathbf{x}}) = -322.40$).

Úloha z reálné aplikace představuje v mnohém opak. K dispozici byl pouze omezený počet dat, z nichž většina (přes 80 %) měla svou vlastní kombinaci diskrétních hodnot. V praxi často různé diskrétní kombinace znamenají zcela odlišné

vlastnosti materiálu, a tedy i naměřené hodnoty empirické cílové funkce. Tomu odpovídají také výsledky. Hodnoty, které spočítal model, se liší od původních měření o přibližně 41 % (poslední model z tabulky 5.2 měřený na datech z 8. generace: MAE $\approx$ 17.7, průměrná empirická fitness $\bar{y} = 42.74$), v některých případech i více. Grafické ověření však ukazuje, že celkový průběh empirické funkce model zachytil.

Druhým cílem bylo navrhnout konkrétní genetický algoritmus tak, aby dokázal našeho modelu využívat. Jeho detailní pseudokód najdeme v kapitole 3.2. I přes určitá zjednodušení, kterých jsme se v prototypové implementaci dopustili, jsme přínos použití modelu v genetickém algoritmu ověřili a důkladně otestovali na první testovací úloze s benchmarkovou fitness. Zapojením modelu dokázal genetický algoritmus nalézt řešení pouze o 1 % horší než globální optimum, což je oproti původně dosahované 10% odchylce výrazné zlepšení. Naopak, o 10 % horší řešení našel algoritmus s modelem za použití třetinového počtu vyhodnocení původní cílovou funkcí, druhá varianta algoritmu by dokonce vystačila s přibližně 18 %.

Výsledky naší práce považujeme za úspěšné. Z uvedených faktů je zřejmé, že na modelovém příkladě námi navržené řešení opravdu přineslo velmi výrazné zlepšení. Přínos na datech z reálné aplikace jsme nemohli posoudit genetickým algoritmem, protože jsme neměli možnost provádět nová experimentální měření. Věříme však, že i přes horší numerické výsledky by model dokázal proces optimalizace urychlit a ušetřit prostředky vynaložené na vyhodnocování empirické funkce.

Literatura

[1] Berk, R. A.: *Statistical learning from a regression perspective*. Springer, 2008.

[2] Bishop, C. M.: *Neural networks for pattern recognition*. Oxford University Press, 2005.

[3] Branke, J., Schmidt, C.: Faster convergence by means of fitness estimation. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 9(1):13–20, 2005

[4] Breiman, L., Friedman, J. H. a kol.: *Classification and Regression Trees*. Wadsworth International Group, 1984.

[5] Buhmann, M. D.: *Radial basis functions: theory and implementations*. Cambridge University Press, Cambridge, 2003.

[6] Deb., K.: *Multi-objective optimization using evolutionary algorithms*. John Wiley and Sons, 2001.

[7] Fogel, L. J., Owens, A. J. a Walsh, M. J.: *Artificial intelligence through simulated evolution*. John Wiley and Sons, 1996.

[8] Gill, P. E., Murray, W., Wright, M. H.: *Practical optimization*. Academic Press London, 1981.

[9] Györfi, L., Kohler, M. a kol.: *A distribution-free theory of nonparametric regression*. ASA, 2002.

[10] Hagan, M. T., Demuth, H. B., Beale, M.: *Neural network design*. PWS Publishing Co. Boston, MA, USA, 1997.

[11] Holeňa, M., Cukic, T., Rodemerck, U.: Optimization of Catalysts Using Specific, Description-Based Genetic Algorithms. *Journal of Chemical Information and Modeling*, 48(2):274–282, 2008.

[12] Jin, Y., Hüsken, M., Olhofer, M. a kol.: Neural networks for fitness approximation in evolutionary optimization. *Knowledge Incorporation in Evolutionary Computation*, Springer, s. 281–306, 2005.

[13] Larranaga, P., Lozano, J. A.: *Estimation of distribution algorithms.* Kluwer, 2002.

[14] Marquardt, D. W.: An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441, 1963.

[15] McLachlan, G. J., Peel, D.: *Finite mixture models.* John Wiley and Sons, 2000.

[16] Olhofer, M., Arima, T., Sonoda, T. a kol.: Optimisation of a stator blade used in a transonic compressor cascade with evolution strategies. *Adaptive Computing in Design and Manufacture*, s. 45–54, 2000.

[17] Ong, Y. S., Nair, P. B., Keane, A. J. a kol.: Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. *Knowledge Incorporation in Evolutionary Computation*, Springer, s. 307–322, 2005.

[18] Park, J., Sandberg, I. W.: Universal approximation using radial-basis-function networks. *Neural computation*, 3(2):246–257, 1991.

[19] Ratle, A.: Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. *Lecture notes in computer science*, Springer, s. 87–96, 1998.

[20] Rechenberg, I.: *Evolutionsstrategie.* Frommann–Hozboog Verlag, 1973.

[21] Schaefer, R.: *Foundations of Global Genetic Optimization.* Springer, 2007.

[22] Schölkopf, B., Smola, A. J.: *Learning with kernels.* MIT Press Cambridge, MA, 2002.

[23] Steinwart, I., Christmann, A.: *Support vector machines.* Springer, 2008.

[24] Valero, S., Argente, E. a kol.: DoE framework for catalyst development based on soft computing techniques. *Computers and Chemical Engineering*, 33(1):225–238, 2009.

[25] White, H.: *Artificial neural networks: approximation and learning theory*. Blackwell Publishers, Inc. Cambridge, MA, USA, 1992.

[26] Zhou, Z., Ong, Y. S., Nair, P. B. a kol.: Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 37(1):66–76, 2007.

Příloha: Obsah CD

Součástí práce je přiložený CD-ROM, který obsahuje

- stručnou uživatelskou a programátorskou dokumentaci
- zdrojové soubory prototypové implementace
- testovací data a příklady demonstrující použití a výsledky popsaných algoritmů
- tuto práci ve formátu PDF