ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

Bc. Pavol Kozák

Taguchiho metóda pre nastavenie parametrov algoritmu

Vedúci práce: prof. Ing. Ľudmila Jánošíková PhD.

Registračné číslo: 65/2017

Žilina, 2018

ŽILINSKÁ UNIVERZITA V ŽILINE

FAKULTA RIADENIA A INFORMATIKY

DIPLOMOVÁ PRÁCA

ŠTUDIJNÝ ODBOR: INTELIGENTNÉ INFORMAČNÉ SYSTÉMY

PAVOL KOZÁK

Taguchiho metóda pre nastavenie parametrov algoritmu

Žilinská univerzita v Žiline

Fakulta riadenia a informatiky

Školiace pracovisko: Katedra matematických metód a operačnej analýzy

Žilina, 2018

PRESKENOVANÉ PODPÍSANÉ ZADANIE

Čestné vyhlásenie

Čestne vyhlasujem, že som túto prácu napísal samostatne a že som uviedol všetky použité pramene a literatúru, z ktorých som čerpal.

V Žiline, dňa 13.4.2018

.....

podpis

Poďakovanie

Touto cestou vyslovujem poďakovanie pani prof. Ing. Ľudmile Jánošíkovej, PhD. za pomoc, odborné vedenie, cenné rady a pripomienky pri vypracovávaní mojej diplomovej práce.

ABSTRAKT V ŠTÁTNOM JAZYKU

KOZÁK, Pavol: *Taguchiho metóda pre nastavenie parametrov algoritmu*. [Diplomová práca]. – Žilinská univerzita v Žiline. Fakulta riadenia a informatiky; Katedra matematických metód a operačnej analýzy. – Školiteľ/Vedúci: prof. Ing. Ľudmila Jánošíková PhD. – Stupeň odbornej kvalifikácie: Inžinier v odbore Informačné systémy. – Žilina: FRI ŽU 2018. 63 s

Cieľom záverečnej práce bolo vytvoriť aplikáciu pre hľadanie čo najlepších hodnôt parametrov algoritmu založenú na Taguchiho metóde, v ktorej bude algoritmus fungovať ako čierna skrinka. Súčasťou aplikácie je štatistická metóda pre analýzu výsledkov a ich grafické znázornenie.

Kľúčové slová: Taguchiho metóda, ortogonálne polia, Calibra, Genetický algoritmus, AdaBoost, Mnist

ABSTRAKT V CUDZOM JAZYKU

KOZÁK, Pavol: *Fine-tuning of algorithms using the Taguchi method*. [Diploma Thesis]. – University of Žilina. Faculty of Management Science and Informatics; Department of mathematical methods and operations research. – Advisor: prof. Ing. Ľudmila Jánošíková, PhD. – Qualification degree: Engineer of information systems, study program: Information systems. Žilina: FRI ŽU, 2018. 63 s

The aim of this thesis was to create an application for finding the best parameter values for any algorithm, which works like black box. The application also contains a statistical method for analyzing the results and their graphical representation.

Key words: Taguchi methods, Orthogonal array, Calibra, Genetic algorithm, AdaBoost, Mnist

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek

DOE	design of experiments, plánovanie experimentov
FFD	full factorial design, úplný faktorový plán
OA	orthogonal array, ortogonálne pole

Úvod

Algoritmy spadajúce pod strojové učenie, neurónové siete, heuristiky, metaheuristiky využívajú parametre. Tieto parametre nijakým spôsobom neovplyvňujú riešenú úlohu, ale ovplyvňujú algoritmus, ktorý úlohu rieši. Nastavenie parametrov, nazývané aj ladenie parametrov, je veľmi dôležitá, ale aj časovo náročná úloha. Nastavenie týchto parametrov ovplyvňuje kvalitu riešenia. Práve čas strávený ladením parametrov môže ďaleko presiahnuť čas strávený programovaním a ladením algoritmu.

Jedným z možných prístupov ako nájsť čo najlepšie nastavenie parametrov je práve použitie Taguchiho metódy, ktorej úlohou je presne definovať hodnoty faktorov (parametrov) a preskúmať ich vplyv na výsledok, čím redukuje čas potrebný pri hľadaní najlepšieho nastavenia parametrov. Okrem toho dokáže určiť parametre, ktoré najviac ovplyvňujú výsledok, určiť hodnoty parametrov pre najlepší výsledok a odhadnúť výsledok algoritmu s najlepším nastavením parametrov.

Cieľom tejto práce je naprogramovať Taguchiho metódu a následne ju otestovať na algoritme strojového učenia nazývanom AdaBoost a genetickom algoritme.

1 Metódy pre nastavenie parametrov algoritmov

V algoritmoch strojového učenia a heuristikách sa využívajú parametre, tie majú zásadný vplyv na kvalitu riešenia. Väčšinou je potrebné nastaviť tieto parametre podľa typu a rozsahu úlohy. To znamená, že nájdenie vhodných parametrov pre jednu úlohu neznamená, že algoritmus vráti dobré riešenie aj pre inú úlohu.

V literatúre, ktorá popisuje konkrétny algoritmus, sa častokrát udáva, že odporúčané hodnoty pre parametre sú jednoducho dané, bez ďalšieho vysvetlenia alebo rozdelenia do skupín podľa typu úlohy. V lepšom prípade môže byť uvedené, že sa vykonali experimenty, podľa ktorých boli odvodené hodnoty parametrov, avšak bez uvedenia citlivosti týchto parametrov. Taktiež sa stáva, že hodnoty týchto parametrov sú určené podľa hodnôt uvedených v iných štúdiách bez overenia, či tieto parametre sú efektívne aj v konkrétnej úlohe. Iba v malom počte štúdií sú uvedené aj experimenty, podľa ktorých boli hodnoty parametrov určené.

Na určenie hodnôt sa môže použiť úplný faktorový plán (*full factorial design*). Ten vyberie *k* najvýznamnejších parametrov, pre tie sa určí *n* levelov. Následné sa vykoná n^k experimentov, t.j. spustení algoritmu s jednou kombináciou parametrov a vyberie sa najlepšia kombinácia hodnôt parametrov. Ide o určovanie pomocou hrubej sily, ktoré s rastúcim počtom parametrov a ich levelov začne byť nepraktické.

Používanejším a praktickejším prístupom je použitie metód pre automatické nastavenie parametrov metaheuristík. Metódy sú rozdelené do dvoch skupín. Prvá skupina využíva modely vyjadrujúce vzťah medzi algoritmom a jeho parametrami, druhá modely nevyužíva [5].

1.1 Metódy bez modelov

Metódy bez modelov sú založené na heuristických pravidlách, výber hodnôt parametrov býva často náhodný alebo určený jednoduchým experimentálnym plánom. Obvykle sa vykonávajú rýchlejšie než metódy s modelmi. Nevýhodou je, že len pomaly sa približujú k optimálnemu riešeniu.

Nájdenie dobrých hodnôt parametrov pre algoritmus aplikovaný na konkrétny problém je zložitá úloha, čoho hlavným dôvodom je, že nemôžeme urobiť žiadne

predpoklady o priestore riešení, napr. o jeho dimenzii alebo vzájomnej interakcii parametrov. Pre takýto typ úloh sa bežne používajú metódy, ktoré využívajú princípy, mechanizmy a vzory vynájdené a overené prírodou ako napríklad evolučný algoritmus.

V článku [5] sú uvedené aj ďalšie metódy:

Relevance Estimation and Value Calibration of Parameters (REVAC). Odhaduje rozdelenie sľubných vektorov parametrov pomocou teórie informácie.

ParamILS. Autori prezentujú dve implementácie: *BasicILS* a *FocussedILS*. Obe využívajú techniky lokálneho hľadania (*iterated local search*) s cieľom usmerniť vyhľadávanie na sľubné oblasti vo vyhľadávacom priestore. Metóda štartuje z jednej, užívateľom nastavenej kombinácie parametrov. V každej iterácii sa zmení hodnota jedného parametra. Proces skončí po dosiahnutí dostatočne dobrého riešenia alebo po uplynutí vymedzeného času.

Gender based Genetic Algorithm. Je genetický algoritmus využívajúci rodové rozdiely. Podľa autorov je väčšia pravdepodobnosť, že výber partnera bude mať vysoký vplyv na výslednú kvalitu, než napríklad prirodzený výber. Obe pohlavia využívajú rôzne prístupy pri výbere partnera.

Iný spôsob na systematické vyhľadávanie optimálneho definovania parametrov je použitie globálnej optimalizačnej techniky. Takýto prístup vyžaduje optimalizačnú funkciu, ktorá vyhodnocuje algoritmus ako čiernu skrinku. Príkladom takého prístupu je GENOCOP [11], vytvorený na University of North Carolina at Charlotte. GENOCOP je založený na genetickom algoritme, pričom algoritmus, ktorého parametre ladíme, je čierna skrinka. Vďaka tomu sa môže použiť na hľadanie optimálneho definovania parametrov. Metódy založené na tomto princípe predpokladajú, že vykonanie algoritmu nie je časovo náročné. Z tohto dôvodu dokáže GENOCOP vrátiť dobré riešenie po vygenerovaní veľkého množstva riešení, teda aj veľkého množstva spustení algoritmu. Výhodou tohto prístupu je, že vyhľadávanie nie je limitované počtom levelov a dokáže vyhľadávať celočíselné aj spojité hodnoty parametrov.

1.2 Metódy s modelmi

Základným cieľom metód s modelmi je určiť zaujímavé hodnoty alebo rozsahy hodnôt parametrov, ktoré sa majú skúmať. Často používaným nástrojom je grafická

interpretácia parametrov s ohľadom na dosiahnutú cieľovú *fitness* funkciu (životaschopnosť). Inými slovami, prístupy založené na modeloch vytvárajú modely schopné 1) predpovedať nastavenie nových hodnôt parametrov, ktoré sa oplatí skúmať, 2) odhadnúť dobré hodnoty parametrov pre nové problémy alebo inštancie problémov.

Metódy s modelmi pre optimalizáciu parametrov majú svoje korene v 1) tradičnom plánovaní experimentov a 2) globálnej optimalizácii.

Plánovanie experimentov (*design of experiments*, DoE) je klasická metóda pre vykonanie reprezentatívnych experimentov. Pokúša sa minimalizovať množstvo experimentov a zároveň zabezpečiť vysoko kvalitné výsledky. Experimenty majú vstupné premenné (faktory) a výstupné premenné (reakcie). Cieľom DoE je skúmať vplyv faktorov na výstupné premenné. DoE môže byť použitý na ladenie parametrov, ale bol pôvodne vynájdený kvôli fyzikálnym experimentom.

Racing algorithm (F-Race) využíva Friedmanov test a neparametrický test nulovej hypotézy. Pozostáva z dvoch fáz: 1) súhrnného testu a 2) párového porovnania jednotlivcov v turnaji. Veľkosť populácie sa zníži na niekoľko najlepších jedincov. F-Race predpokladá, že interakcie parametrov sú lineárne.

Calibra [2] kombinuje plánovanie experimentov a lokálne vyhľadávanie. Využíva Taguchiho metódu a vďaka DoE lineárnym zvyšovaním počtu parametrov sa počet experimentov nezvyšuje exponenciálne. Calibra môže byť použitá pre maximálne 4 parametre a 1 fixovaný, ktoré môžu byť spojité aj diskrétne, avšak spojité parametre musia byť prevedené na diskrétne.

2 Taguchiho metóda

Táto metóda je pomenovaná po japonskom inžinierovi a štatistikovi Genichi Taguchi (1924 - 2012), ktorý vytvoril metódu využívajúcu štatistické prístupy na zvýšenie kvality produktu. Táto metóda je natoľko univerzálna, že sa môže aplikovať v oblasti priemyslu, obchodu a reklamy [7].

Hlavnou úlohou Taguchiho metódy je zlepšenie kvality produktov a procesov, pričom kvalita závisí od niekoľkých faktorov.

2.1 Plánovanie experimentov

Plánovanie experimentov je proces definovania experimentov a ich vykonania. Definovanie experimentov predstavuje 1) určenie počtu experimentov a 2) v každom experimente definovanie hodnôt parametrov. Po vykonaní experimentov sú zozbierané dáta, ktoré sú analyzované štatistickými metódami, vďaka ktorým môžeme vytvoriť objektívne závery. S každým experimentom je spojené jeho definovanie a štatistická analýza, tieto aspekty sú úzko spojené.

2.1.1 Úplný faktorový plán

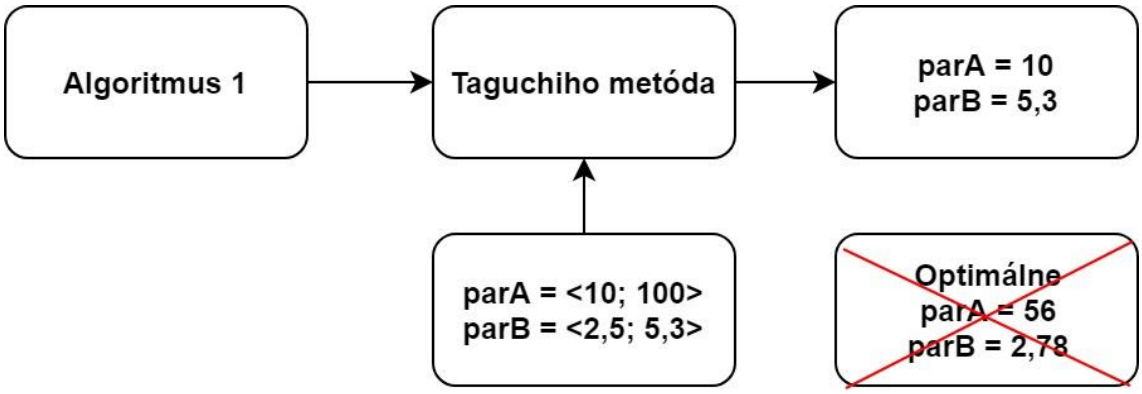
Pre plánovanie experimentov sa často používa úplný faktorový plán (*full factorial design*, FFD). Úplný faktorový plán vyberie k faktorov, ktoré najviac ovplyvňujú hodnotu riešenia. Následne je pre každý faktor zvolených 2 až n levelov (nazývané aj úrovne). Ak sú zvolené 2 levely, plán sa nazýva 2^k faktorový plán. Podobne plán obsahujúci 3 levely sa nazýva 3^k faktorový plán. Pri používaní Taguchiho metódy pre nastavenie parametrov algoritmu faktory vyjadrujú parametre a levely ich významné hodnoty na preskúmanie. Pri úplnom faktorovom pláne sú testované všetky kombinácie hodnôt parametrov. Napríklad preskúmanie 3 parametrov s 5 úrovňami zaberie $3^5 = 243$ experimentov, čo znamená 243 spustení algoritmu. Ak algoritmus obsahuje náhodné prvky, ako napríklad genetický algoritmus, je nevyhnutné experiment spúšťať pomocou replikácií, aby sme dostali zmysluplné výsledky. Takýto prístup ladenia parametrov sa môže použiť iba pri malých úlohách a časovo nenáročných algoritmoch s malým počtom parametrov.

2.1.2 Čiastočný faktorový plán

Úplný faktorový plán sa veľmi rýchlo stáva nepraktickým, pretože množstvo experimentov rastie exponenciálne s množstvom parametrov. Vhodnejšie je použitie čiastočného faktorového plánu (*fractional factorial design*). Tento plán dokáže vyvodiť závery založené na strategicky zvolenej podmnožine experimentov z úplného faktorového plánu. Tento prístup umožňuje skúmať veľké množstvo faktorov s relatívne málo experimentami. Práve Taguchiho metóda využíva čiastočné faktorové plány, ktoré sú založené na špeciálnych ortogonálnych poliach [8]. Tieto ortogonálne polia presne špecifikujú nastavenie faktorov v jednotlivých experimentoch.

Hlavnou úlohou experimentov nie je iba zistiť, ako jednotlivé faktory (parametre) ovplyvňujú výsledok, ale taktiež ako na seba jednotlivé faktory pôsobia. Ako bolo spomenuté vyššie, aj s malým počtom faktorov a malým počtom levelov pre každý faktor počet kombinácií levelov rýchlo rastie. V takom prípade sa vykonávajú iba niektoré experimenty a úlohou ortogonálnych polí je určiť, ktoré kombinácie levelov sa budú používať tak, aby sa z nich dali vyvodiť štatisticky spoľahlivé závery.

Taguchiho metóda sa môže použiť na maximalizáciu aj minimalizáciu výsledku algoritmu. Príklad maximalizácie môže byť hodnota účelovej funkcie optimalizačnej úlohy. Príklad minimalizácie môže byť zasa redukovanie času, ktorý potrebuje algoritmus strojového učenia na dosiahnutie istej presnosti. Avšak bez ohľadu na rozsah experimentu, najlepší výsledok je vždy definovaný jednou kombináciou z úplného faktorového plánu (obrázok 1). Takže pre odhalenie skutočného optima je potrebné iteratívne spúšťanie Taguchiho metódy, ktoré je bližšie popísané v algoritme Calibra [2].



Obrázok 1. Iterácia Taguchiho metódy

2.2 Ortogonálne polia

Taguchiho metóda využíva ortogonálne polia (OA) ktoré sa často používajú na plánovanie experimentov.

Ortogonálne pole obsahujúce n levelov a k faktorov sa označuje ako $OA_N(n^k)$ a predstavuje $N \times k$ maticu. Hodnoty jej prvkov sú čísla od 1 do n a musia spĺňať podmienku, že v každom páre stĺpcov sa každá z možných usporiadaných dvojíc prvkov vyskytuje rovnaký počet krát. Tabuľky 1 a 2 zobrazujú $OA_4(2^3)$ a $OA_8(2^7)$. Za povšimnutie stojí, že v každom páre stĺpcov tabuľky 1 sa každá zo štyroch usporiadaných dvojíc (1,1), (1,2), (2,1), (2,2) vyskytuje práve raz. Rovnako každý pár stĺpcov v tabuľke 2 obsahuje každú zo štyroch vyššie spomenutých dvojíc presne dvakrát.

Tabuľka 1. Ortogonálne pole $OA_4(2^3)$

Číslo riadku	Číslo stĺpca		
	1	**2**	**3**
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Tabuľka 2. Ortogonálne pole $OA_8(2^7)$

Číslo riadku	Číslo stĺpca						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2
3	1	2	2	1	1	2	2
4	1	2	2	2	2	1	1
5	2	1	2	1	2	1	2
6	2	1	2	2	1	2	1
7	2	2	1	1	2	2	1
8	2	2	1	2	1	1	2

Taguchiho OA sú zovšeobecnené latinské štvorce. Taguchi používa hodnoty prvkov 1, 2, ..., n, zatiaľ čo latinské štvorce používajú 0, 1, ..., n-1. Latinské štvorce sú $n \times n$ matice, naplnené n rozdielnymi symbolmi, pričom sa každý vyskytuje práve raz v každom stĺpci a riadku [10].

OA predstavuje plán experimentov s niekoľkými faktormi, kde stĺpce predstavujú faktory, hodnoty v stĺpcoch predstavujú testované levely daného faktora a riadky

reprezentujú experimenty. Konkrétnejšie *N* riadkov, ktoré obsahuje $OA_N(n^k)$, chápeme ako podmnožinu zo všetkých n^k experimentov úplného faktorového plánu, ktorý obsahuje k faktorov, z toho každý faktor obsahuje *n* levelov. $OA_N(n^k)$ predstavuje N/n^k časť úplného n^k faktorového plánu. Napríklad $OA_4(2^3)$, ktoré je zobrazené v tabuľke 1, predstavuje $4/2^3 = 1/2$ z 2^3 úplného plánu.

OA stále ostáva OA aj po vymazaní niekoľkých stĺpcov. Takýmto spôsobom dokážeme prispôsobiť OA pre ľubovoľný počet faktorov, počet experimentov sa ale nemení. Vymazaním niekoľkých stĺpcov dokážeme vytvoriť veľa rozdielnych plánov.

OA nemusia obsahovať len 2 levely faktorov, najčastejšie používané obsahujú od 2 do 5 levelov. Príklad $OA_9(3^4)$ je v tabuľke 3 a môže sa použiť pre 4 faktory, pričom každý má 3 levely. Niektoré OA môžu obsahovať zmiešané levely, označujú sa ako $OA_N(n^k \times m^j)$ a je to matica o *N* riadkoch a $k + j$ stĺpcoch. Pričom prvých *k* stĺpcov má *n* levelov, ďalších *j* stĺpcov má *m* levelov a v každom páre stĺpcov sa každá z možných usporiadaných dvojíc vyskytuje konštantný počet krát. Táto konštanta ale závisí od zvoleného páru stĺpcov. Príkladom zmiešaného ortogonálneho poľa je $OA_{18}(2^1 \times 3^7)$.

Tabuľka 3. Ortogonálne pole $OA_9(3^4)$

Číslo riadku	**Číslo stĺpca**			
	1	**2**	**3**	**4**
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Tabuľka 4 [16] opisuje počty experimentov, faktorov a levelov pre konštantné, ale aj zmiešané OA. Z tabuľky sa dá odvodiť, že počet experimentov s konštantným počtom levelov je $k * (n - 1) + 1$, kde *n* je počet levelov a *k* je počet faktorov, oproti úplnému faktorovému plánu n^k.

Tabuľka 4. Prehľad ortogonálnych polí

Ortogonálne pole	Počet experimentov	Počet faktorov	Maximálny počet faktorov			
			2 levely	3 levely	4 levely	5 levelov
OA_4	4	3	3	-	-	-
OA_8	8	7	7	-	-	-
OA_9	9	4	-	4	-	-
OA_{12}	12	11	11	-	-	-
OA_{16}	16	15	15	-	-	-
$OA_{16'}$	16	5	-	-	5	-
OA_{18}	18	8	1	7	-	-
OA_{25}	25	6	-	-	-	6
OA_{27}	27	13	-	13	-	-
OA_{32}	32	31	31	-	-	-
$OA_{32'}$	32	10	1	-	9	-
OA_{36}	36	23	11	12	-	-
$OA_{36'}$	36	16	3	12	-	-
OA_{50}	50	12	1	-	-	11
OA_{54}	54	26	1	25	-	-
OA_{64}	64	63	63	-	-	-
$OA_{64'}$	64	21	-	-	21	-
OA_{81}	81	40	-	40	-	-

2.3 Príklady použitia Taguchiho metódy

2.3.1 Optimalizácia výroby plastových produktov

Úlohou je optimalizovať kvalitu plastových produktov, ktorých výrobný proces má 3 faktory a 2 levely (tabuľka 5). Použitím Taguchiho metódy chceme definovať najlepšiu hodnotu každého faktora a prínos každého faktora ku kvalite produktu.

Tabuľka 5. Faktory a levely kvality plastových produktov

Faktor	Level 1	Level 2
Vstrekovací tlak	A_1= 250 psi	A_2 = 350 psi
Teplota tavenia	B_1= 150 °C	B_2= 200 °C
Čas procesu	C_1 = 6 sekúnd	C_1 = 9 sekúnd

Keďže výrobný proces obsahuje 3 faktory a 2 levely, najvhodnejšie bude OA_4 (tabuľka 1), ktoré obsahuje 3 stĺpce. Faktory môžu byť jednotlivým stĺpcom priradené náhodne. Plán experimentov je popísaný v tabuľke 6. Každý riadok OA opisuje jeden experiment s hodnotami levelov, pri ktorých sa má vykonať, konkrétne experimenty sú popísané v tabuľke 7.

Tabuľka 6. Plán experimentov

Popis faktorov	**Level** 1	2
Vstrekovací tlak	250 psi	350 psi
Teplota tavenia	150 °C	200 °C
Čas procesu	6 sek.	9 sek.

	Stĺpec			**Opakovanie**			
Experiment	1	2	3	1	2	3	•••
1	1	1	1	30			
2	1	2	2	25			
3	2	1	2	34			
4	2	2	1	27			

Tabuľka 7. Jednotlivé experimenty

Experiment 1	
Vstrekovací tlak	A_1= 250 psi
Teplota tavenia	B_1 = 150 °C
Čas procesu	C_1 = 6 sekúnd
Experiment 2	
Vstrekovací tlak	A_1 = 250 psi
Teplota tavenia	B_2 = 200 °C
Čas procesu	C_2 = 9 sekúnd
Experiment 3	
Vstrekovací tlak	A_2 = 350 psi
Teplota tavenia	B_1 = 150 °C
Čas procesu	C_2 = 9 sekúnd
Experiment 4	
Vstrekovací tlak	A_2 = 350 psi
Teplota tavenia	B_2 = 200 °C
Čas procesu	C_1 = 6 sekúnd

Vždy keď je to možné, experimenty by mali byť vykonávané v náhodnom poradí. Ak sa vykonáva viacero behov pre jeden experiment, môžu byť použité dva prístupy. Prvým z nich sú replikácie, kedy sa všetky experimenty aj ich behy vykonávajú náhodne. Druhým sú opakovania, kedy sa náhodne vyberie experiment a vykonajú sa všetky jeho behy. Tieto prístupy sú potrebné, keď experimenty zahrňujú náhodnosť (napr. genetický

algoritmus) a jeden experiment môže vrátiť rozdielne výsledky pri rovnakom nastavení parametrov.

Vykonali sa štyri experimenty (tabuľka 7), následne sa vyhodnotila kvalita výrobkov v každom experimente, ktorá je popísaná nižšie. Tieto výsledky sú uvedené aj v najpravejšom stĺpci plánu experimentov (tabuľka 6). Bola použitá len jedna replikácia pre každý experiment, z toho dôvodu sú výsledky uvedené len v jednom stĺpci. Z jednotlivých experimentov sme teda dostali takéto výstupy:

Y_1 =30, Y_2 =25, Y_3 =34, Y_4 =27

Pri analýze výsledkov Taguchiho metóda poskytuje niekoľko kľúčových postupov. Keď tieto kroky dodržiavajú rôzni jednotlivci, pravdepodobne dospejú k rovnakým záverom a zrýchlia analýzu výsledkov. Cieľom analýzy je odpovedať na nasledujúce otázky.

- Aké je optimálne nastavenie parametrov?
- Ktoré faktory ovplyvňujú variabilitu riešenia a ako veľmi?
- Aký bude výsledok pri optimálnom nastavení parametrov a ako jednotlivé faktory prispeli k tomuto výsledku?

Priemerný vplyv faktorov (*average effect*) na určitom leveli sa vypočíta tak, že súčet výsledkov experimentov, pri ktorých mal faktor daný level sa vydelí počtom týchto experimentov. Pre faktor *A* na leveli 1 sčítame výsledky experimentov 1 a 2 a vydelíme ich číslom 2. Podobne vypočítame vplyv ostatných faktorov tak, ako je popísané nižšie.

$$\bar{A}_1 = (Y_1 + Y_2)/2 = (30 + 25)/2 = 27{,}5$$

$$\bar{A}_2 = (Y_3 + Y_4)/2 = (34 + 27)/2 = 30{,}5$$

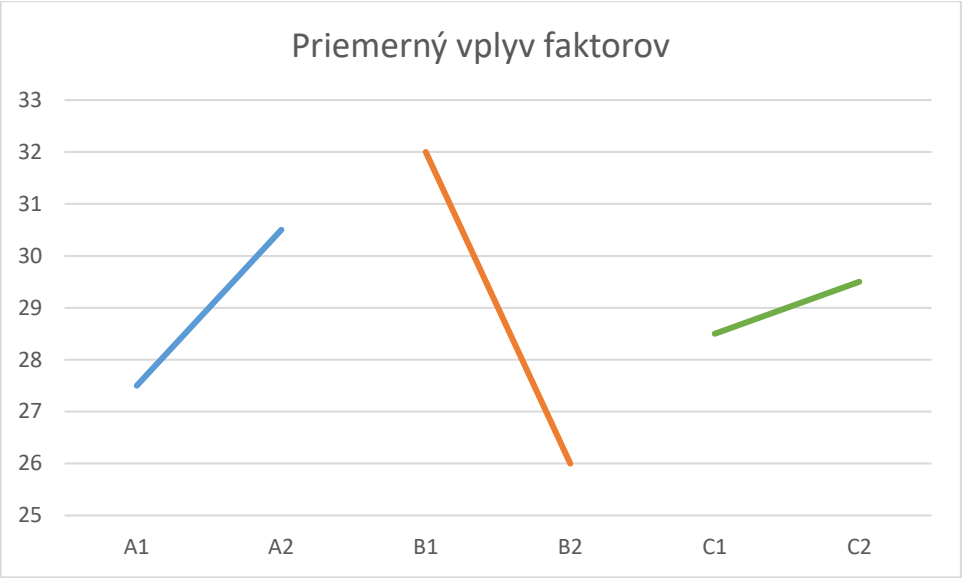
$$\bar{B}_1 = (Y_1 + Y_3)/2 = (30 + 34)/2 = 32$$

$$\bar{B}_2 = (Y_2 + Y_4)/2 = (25 + 27)/2 = 26$$

$$\bar{C}_2 = (Y_1 + Y_4)/2 = (30 + 27)/2 = 28{,}5$$

$$\bar{C}_2 = (Y_2 + Y_3)/2 = (25 + 34)/2 = 29{,}5$$

Priemerný vplyv faktorov môžeme vyniesť do grafu (obrázok 2).



Obrázok 2. Priemerný vplyv faktorov.

Pri optimalizácii výroby plastových produktov zlepšujeme kvalitu výsledného produktu. Kvalitu sa snažíme maximalizovať, teda tento problém zaraďujeme do skupiny „väčší je lepší“. Z priemerného vplyvu faktorov (obrázok 2) vyplýva, že kombinácia faktorov $A_2\,B_1\,C_2$ pravdepodobne vráti najlepší výsledok a predstavuje optimálnu kombináciu faktorov, s výnimkou možného účinku interakcií medzi faktormi. Optimálne nastavenie parametrov je:

A_2	čo predstavuje vstrekovací tlak	350 psi
B_1	čo predstavuje teplotu tavenia	150 °C
C_2	čo predstavuje čas procesu	9 sekúnd

Optimálne nastavenie parametrov sa zhodou okolností vyskytlo v treťom experimente. Ide o náhodu, väčšinou optimálny stav nebude jedným zo skúšobných experimentov, pretože Taguchiho metóda vykoná len malý počet experimentov z úplného faktorového plánu. V tomto malom príklade je ale šanca až 50%, keďže úplný faktorový plán má 8 experimentov a čiastočný 4. Optimálne nastavenie parametrov musí byť vždy v jednom z experimentov definovaných úplným faktorovým plánom. Pre odhad výsledku pri optimálnom nastavení parametrov sa používa nasledujúci výraz:

$$Y_{opt} = \frac{T}{N} + \left(\bar{A}_2 - \frac{T}{N}\right) + \left(\bar{B}_1 - \frac{T}{N}\right) + \left(\bar{C}_2 - \frac{T}{N}\right)$$

kde *T* je súčet všetkých výsledkov a *N* je počet všetkých experimentov.

Tento výraz predstavuje sčítanie priemerného výsledku a prínosu každého faktora ku priemernému výsledku. V našom prípade:

$$T = 116;\ N = 4;\ \ \bar{A}_2 = 30{,}5;\ \bar{B}_1 = 32{,}0;\ \ \bar{C}_2 = 29{,}5$$

$$Y_{opt} = 29 + (30{,}5 - 29) + (32 - 29) + (29{,}5 - 29)$$

$$Y_{opt} = 34$$

Čo je výsledok, ktorý sme dosiahli v experimente 3. Ak optimálne nastavenie parametrov nie je v jednom z už vykonaných experimentov, mal by byť vykonaný potvrdzujúci experiment. Je to nevyhnutným a dôležitým krokom v Taguchiho metóde, pretože potvrdzuje predpoklady použité pri analýze.

3 Calibra

3.1 Popis algoritmu Calibra

Algoritmus je založený na plánovaní experimentov a lokálnom vyhľadávaní. Plánovanie experimentov poskytuje spôsob ako sa zamerať na sľubné oblasti vo vyhľadávacom priestore. Najprv sa plánovanie experimentov použije na nájdenie sľubnej oblasti vyhľadávacieho priestoru (súboru hodnôt pre parametre) a na posúdenie zmeny vplyvu parametrov, to znamená, že vyhľadávanie nezačína v náhodnom bode.

Plánovanie experimentov je taktiež použité po inicializačnej fáze, aby udržalo vyhľadávanie v priestore navrhnutom analýzou výsledkov, ktorá poskytuje najlepšiu kombináciu levelov. Avšak táto kombinácia bola vypočítaná pomocou priemerného vplyvu faktorov a predpokladá, že vzťahy medzi faktormi sú lineárne. Ak najlepšia kombinácia levelov nebola zahrnutá v čiastočnom pláne experimentov, je vykonaný potvrdzujúci experiment pre porovnanie odhadovaného a skutočného výsledku. Predpoklad lineárnych väzieb môže spôsobiť, že najlepšia kombinácia levelov odhadnutá Taguchiho metódou sa líši od skutočného optima. Práve preto Calibra používa výsledky analýzy iba ako usmernenie na zúženie vyhľadávacieho priestoru a iniciovanie ďalšieho vyhľadávania. Keďže sa vyhľadávanie sústreďuje na užšie rozsahy pre každý parameter, lineárny predpoklad sa stáva menej reštriktívnym a predpokladaný "optimálny stav" sa približuje k skutočnému optimálnemu stavu aktuálneho experimentu.

Vzhľadom na akýkoľvek algoritmus s niekoľkými parametrami, ktoré treba doladiť, Calibra nastaví sériu experimentov založených na Taguchiho metóde. Tieto experimenty majú za cieľ nájsť najlepšiu hodnotu pre každý parameter. Najlepšia hodnota pre parameter závisí od toho, ako sa meria výkon algoritmu. Napríklad, ak sa výkon meria podľa kvality riešení na súbore problémov, najlepšie hodnoty parametrov sú tie, ktoré poskytujú riešenia s najvyššou kvalitou. Kvalita môže predstavovať čo najväčší alebo najmenší výsledok, časové kritérium a iné. Podľa výsledkov experimentov Calibra usmerňuje vyhľadávanie najlepších hodnôt pre parametre.

Calibra môže nastaviť maximálne 4 parametre. Ak algoritmus používa viac ako 4 parametre, použitím $OA_{16}(2^{15})$ môžeme odhadnúť 4 najdôležitejšie parametre. $OA_{16}(2^{15})$ dokáže spracovať až 15 parametrov, čo je pravdepodobne horná hranica počtu parametrov

pre ľubovoľný algoritmus. Pôvodná implementácia môže použiť iba jeden fixovaný parameter, moja implementácia môže použiť ľubovoľný počet fixovaných parametrov.

Pretože Calibra používa úplný aj čiastočný ($OA_9(3^4)$) faktorový plán a lokálne vyhľadávanie, musí byť definované vhodné kritérium ukončenia. Pri navrhovaní experimentov je zvykom najprv určiť počet pokusov a potom určiť plán, ktorý bude najvhodnejší. Calibra napodobňuje tento proces tým, že je potrebné zadať maximálny počet experimentov (volaní algoritmu), ktoré je možné vykonať na jemné doladenie algoritmu. Calibra využíva nasledujúce parametre (tabuľka 8) a premenné (tabuľka 7).

Tabuľka 8. Parametre algoritmu Calibra

MAX	Maximálny počet volaní algoritmu.
k	Počet parametrov.
u_i	Horná hranica intervalu pre parameter *i* (*i* = 1, ..., *k*).
l_i	Dolná hranica intervalu pre parameter *i*.
δ	Minimálny rozsah každého parametra.

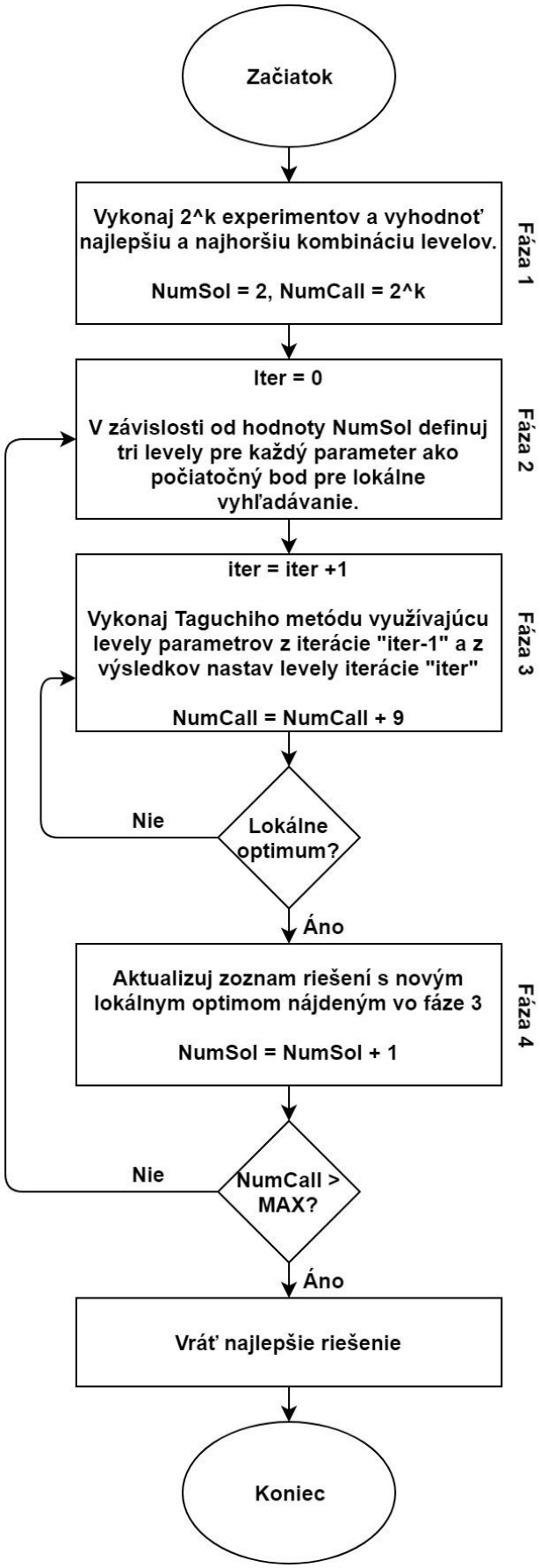
Tabuľka 9. Premenné algoritmu Calibra

iter	Počítadlo iterácií vykonaných Taguchiho metódou pri lokálnom vyhľadávaní.
$p_i^*(iter)$	Hodnota najlepšieho levelu pre parameter *i* v iterácii *iter*.
$p_i^-(iter)$	Hodnota najhoršieho levelu pre parameter *i* v iterácii *iter*.
$p_i^n(iter)$	Hodnota levelu *n* (*n* = 1, 2, 3) pre parameter *i* v iterácii *iter*.
P_i^s	Hodnota parametra *i* v riešení *s*.
NumSol	Počet nájdených riešení (hodnôt parametrov).
NumCall	Aktuálny počet volaní algoritmu.

Calibra pozostáva zo štyroch fáz (obrázok 3). V prvej fáze sa vykoná úplný faktorový plán pozostávajúci z 2^k experimentov, kde levely zodpovedajú prvému a tretiemu kvartilu v rámci intervalu hodnôt každého parametra.

Druhá fáza určuje počiatočnú oblasť riešenia (tri levely parametrov) pre lokálne vyhľadávanie. Táto oblasť je skonštruovaná tak, aby obsahovala najlepšie hodnoty levelov z prvej fázy. V ďalších krokoch je oblasť diverzifikovaná do iných oblastí. Calibra môže pracovať maximálne so 4 parametrami, z tohto dôvodu musia byť ostatné parametre fixované na konštantnú hodnotu.

Tretia fáza predstavuje lokálne vyhľadávanie. Jej úlohou je nájsť lokálne optimum postupným zmenšovaním intervalov každého parametra. Pôvodný algoritmus pracuje iba s celočíselnými premennými, takže spojité premenné musia byť prevedené na diskrétne.



Obrázok 3. Vývojový diagram algoritmu Calibra.

Napríklad hodnoty parametra v rozsahu od 5,879 do 8,954 sú pri pôvodnej implementácii prevedené na hodnoty od 5879 do 8954. Vyhľadávanie končí, ak interval každého parametra obsahuje jediný bod. Moja implementácia využíva aj spojité premenné a preto je potrebné zaviesť dodatkovú premennú δ, ktorá predstavuje minimálny rozsah každého parametra. Ak je hodnota $\delta = 0{,}5$ a rozsah parametra $A = < 5{,}64;\ 6{,}11 >$, podmienka pre minimálny rozsah parametra nie je splnená, pretože $6{,}11 - 5{,}64 = 0{,}47 < 0{,}5$. Ak je rozsah každého parametra menší ako δ, algoritmus pokračuje v ďalšej fáze.

Posledná fáza obsahuje podmienku, ktorá kontroluje, či môžu byť vykonané ďalšie experimenty a teda začne nové lokálne vyhľadávanie. Ak áno, algoritmus sa vráti do druhej fázy, inak skončí a vráti najlepšie nájdené riešenie. Nasledujúce kapitoly obsahujú detailný popis každej fázy.

3.2 Fáza 1 – Počiatočný experiment

Vykonaj 2^k experimentov zodpovedajúcich *k* parametrom s dvomi levelmi, ktoré predstavujú prvý a tretí kvartil v rámci intervalu hodnôt každého parametra. Dolný level pre parameter i sa vypočíta ako $l_i + (u_i - l_i)/4$ a horný ako $u_i - (u_i - l_i)/4$ pričom pôvodný algoritmus zaokrúhľuje hodnoty levelov na celé čísla, moja implementácia podporuje aj spojité hodnoty.

Hodnoty levelov najlepšieho a najhoršieho riešenia ulož do premenných $p_i^*(0)$ a $p_i^-(0)$ pre každý parameter i. Aktualizuj hodnotu *NumSol* = 2 a *NumCall* = 2^k.

3.3 Fáza 2 – Inicializácia lokálneho vyhľadávania

Nastav hodnotu $iter = 0$.

Ak $NumSol = 2$, potom pre každý parameter i nastav $p_i^1(iter) = p_i^*(0) - \varepsilon$, $p_i^2(iter) = p_i^*(0)$, $p_i^3(iter) = p_i^*(0) + \varepsilon$, kde

$$\varepsilon = \frac{min\{(p_i^*(0) - l_i), (u_i - p_i^*(0))\}}{2}.$$

Ak $NumSol = 3$, potom pre každý parameter i nastav $p_i^1(iter) = p_i^-(0) - \varepsilon$, $p_i^2(iter) = p_i^-(0)$, $p_i^3(iter) = p_i^-(0) + \varepsilon$, kde

$$\varepsilon = \frac{min\{(p_i^-(0) - l_i), (u_i - p_i^-(0))\}}{2}.$$

Ak $NumSol > 3$, potom vyber dve predchádzajúce riešenia, ktoré sú základom pre nové lokálne vyhľadávanie. Ak je $NumSol$ párne číslo, vyber dve najlepšie riešenia, ktoré ešte spolu neboli vybraté. Ak je $NumSol$ nepárne, vyber najlepšie a najviac diverzifikované riešenie, ktoré ešte spolu neboli vybraté. Diverzifikácia sa meria podľa hodnôt jednotlivých parametrov. Nech sú vybraté riešenia $s1$ a $s2$. Potom P_i^{s1} označuje hodnotu parametra i v riešení $s1$ a P_i^{s2} hodnotu parametra i v riešení $s2$.

Pre každý parameter i priraď $p_i^1(iter) = \min\{P_i^{s1}, P_i^{s2}\}$, $p_i^2(iter) = \frac{p_i^1(iter) + p_i^3(iter)}{2}$, $p_i^3(iter) = \max\{P_i^{s1}, P_i^{s2}\}$.

3.4 Fáza 3 – Lokálne vyhľadávanie

Aktualizuj $iter = iter + 1$.

Vykonaj experimenty podľa Taguchiho plánu $OA_9(3^4)$, pričom levely parametrov sú dané hodnotami $p_i^1(iter-1)$, $p_i^2(iter-1)$, $p_i^3(iter-1)$ pre každý parameter i. Nech $p_i^*(iter)$ je najlepšia hodnota parametra i po vykonaní experimentov. Následne priraď $p_i^1(iter) = p_i^*(iter) - \varepsilon$, $p_i^2(iter) = p_i^*(iter)$, $p_i^3(iter) = p_i^*(iter) + \varepsilon$, pričom $\varepsilon = \left|p_i^*(iter) - p_i^2(iter-1)\right|$. Ak sa $p_i^*(iter)$ rovná $p_i^2(iter-1)$ alebo nová hodnota zmení smer vyhľadávania, potom $\varepsilon = \min\left\{\left(p_i^*(iter) - p_i^1(iter-1)\right), \left(p_i^3(iter-1) - p_i^*(iter)\right)\right\}/2$.

Aktualizuj $NumCall = NumCall + 9$.

Pôvodný algoritmus kontroluje, či aktuálne riešenie je lokálnym optimom. Ak $p_i^1(iter) = p_i^2(iter) = p_i^3(iter)$ alebo ak $p_i^1(iter) = p_i^1(iter-1)$, $p_i^2(iter) = p_i^2(iter-1)$ a $p_i^3(iter) = p_i^3(iter-1)$ pre každý parameter i, aktuálne riešenie je lokálnym optimom. Moja implementácia obsahuje spojité hodnoty parametrov, a preto lokálne optimum spĺňa podmienku $\left|p_i^1(iter) - p_i^3(iter)\right| < \delta$.

Ak je aktuálne riešenie lokálne optimum, pokračuj vo fáze 4, inak pokračuj v lokálnom vyhľadávaním prechodom na začiatok fázy 3.

3.5 Fáza 4 – Aktualizácia nájdených riešení

Pridaj najlepšie nájdené riešenie v lokálnom vyhľadávaní $p_i^*(iter)$ do zoznamu riešení. Ak sa tam už nachádza, pridaj doplnok riešenia do zoznamu riešení (aby sa hľadanie diverzifikovalo). Doplnok pre parameter i sa vypočíta ako $p_i = l_i + (u_i - p_i^*(iter))$.

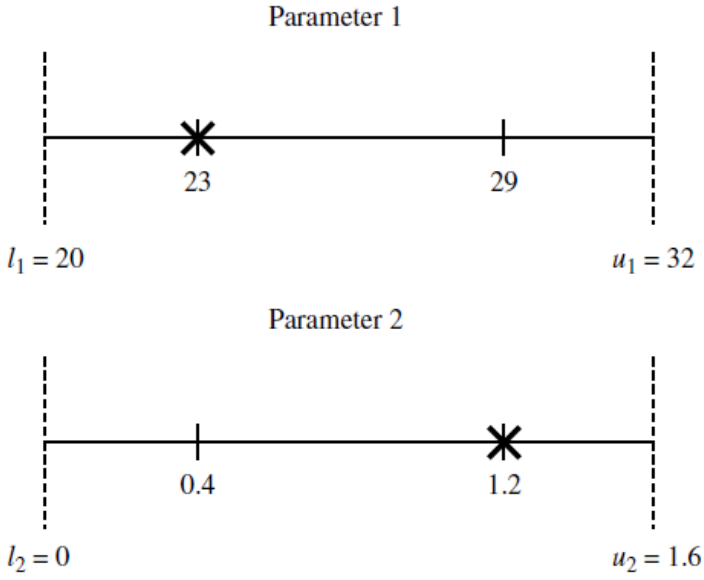
Aktualizuj $NumSol = NumSol + 1$.

Ak $NumCall > MAX$, zastav vyhľadávanie a vráť najlepšie hodnoty parametrov. Inak sa vrát späť na fázu 2.

3.6 Príklad použitia Calibra

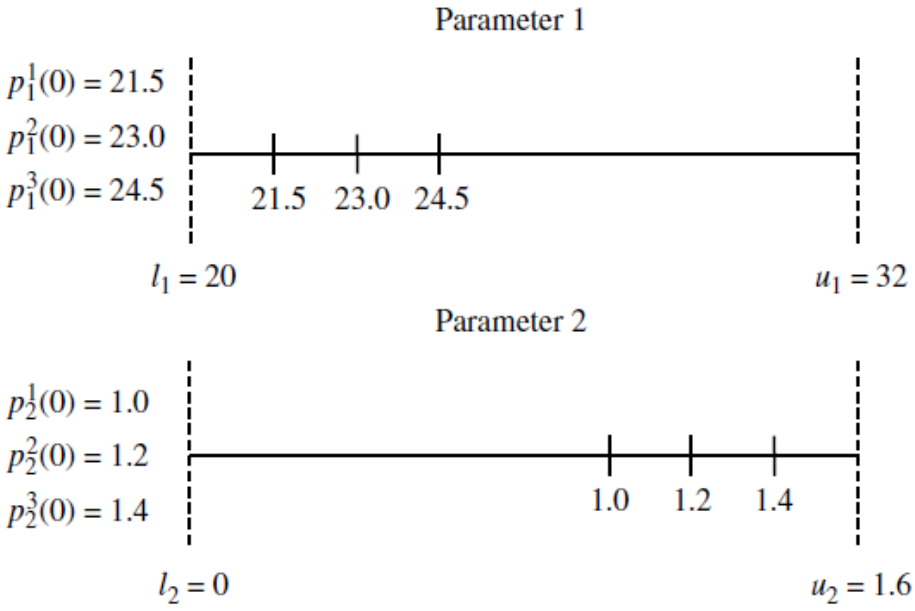
Jednotlivé fázy môžu byť znázornené aj graficky. Na obrázku 4 môžeme vidieť dva parametre znázornené horizontálnymi čiarami a ich horné a dolné hranice znázornené vertikálnymi čiarkovanými čiarami. Čiarky pretínajúce horizontálnu čiaru predstavujú levely parametrov v aktuálnej iterácii. Krížik predstavuje zatiaľ najlepšiu nájdenú hodnotu. Obrázok 4 ukazuje stav po ukončení prvej fázy pre algoritmus s dvoma parametrami, pričom rozsah prvého parametra je od 20 do 30 a druhého od 0 do 1,6.

Fáza 1 vykonáva experimenty s dvoma levelmi, ktoré predstavujú prvý a tretí kvartil každého parametra. Po vykonaní experimentov sa aktualizujú hodnoty $p_1^*(0) = 23$ a $p_2^*(0) = 1{,}2$. Tieto hodnoty sú použité na začiatku fázy 2.



Obrázok 4. Grafická reprezentácia parametrov po fáze 1 [2].

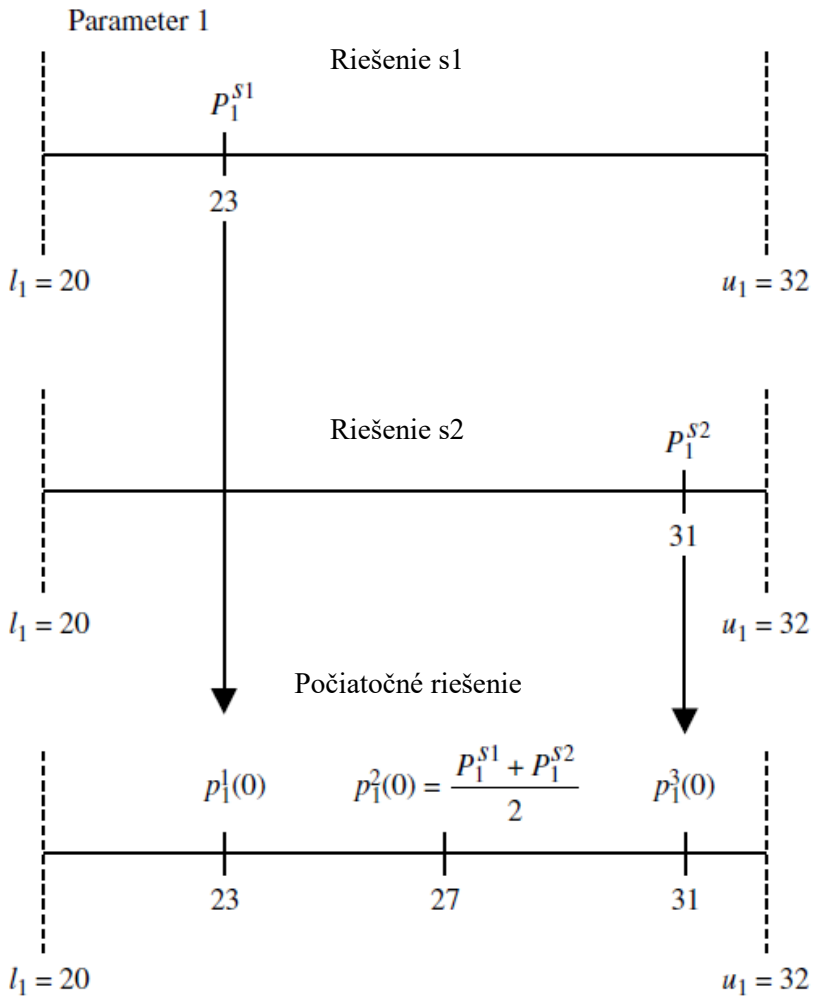
Na obrázku 5 je znázornený stav po fáze 2. Pretože $NumsSol = 2$, ε je vypočítané ako polovica vzdialenosti medzi hodnotou najlepšieho levelu a najbližšou hranicou. Táto hodnota ovplyvňuje tri počiatočné levely parametrov pre fázu 3. Na základe experimentov vo fáze 1 sa teda definuje sľubný priestor, v ktorom sa sústreďuje lokálne vyhľadávanie.



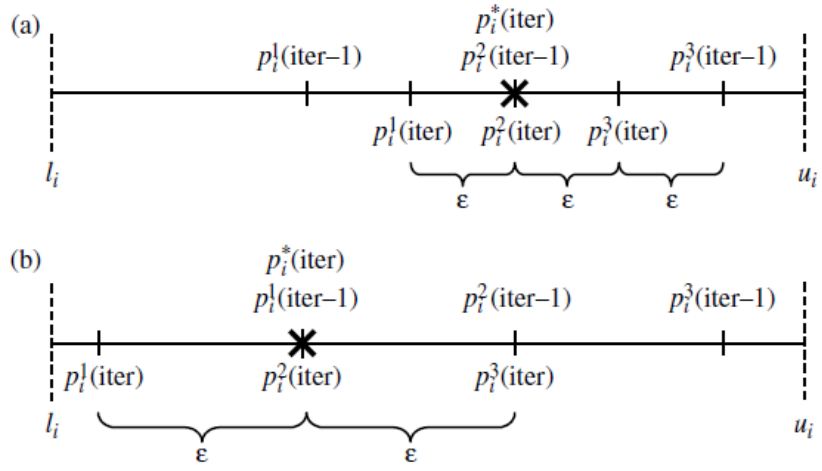
Obrázok 5. Príklad fázy 2 (*NumSol* = 2) [2].

Ak je hodnota $NumsSol > 3$, potom sa počiatočné úrovne parametrov vypočítajú pomocou dvoch lokálnych optimálnych riešení, ktoré spolu ešte neboli kombinované. Grafická reprezentácia tohto prípadu pre jeden parameter je znázornená na obrázku 6.

Určenie nových riešení (levelov pre parametre) v iterácii $iter$ vo fáze 3 závisí od najlepších hodnôt parametrov získaných pomocou Taguchiho metódy v iterácii $iter - 1$. Obrázok 7(a) zobrazuje situáciu, keď sa najlepšia hodnota v iterácii $iter - 1$ nachádza v strede rozsahu parametra ($p_i^*(iter) = p_i^2(iter - 1)$). Obrázok 7(b) zobrazuje situáciu, keď $p_i^*(iter) = p_i^1(iter - 1)$, čiže najlepšia hodnota je dolný level v predchádzajúcej iterácii.



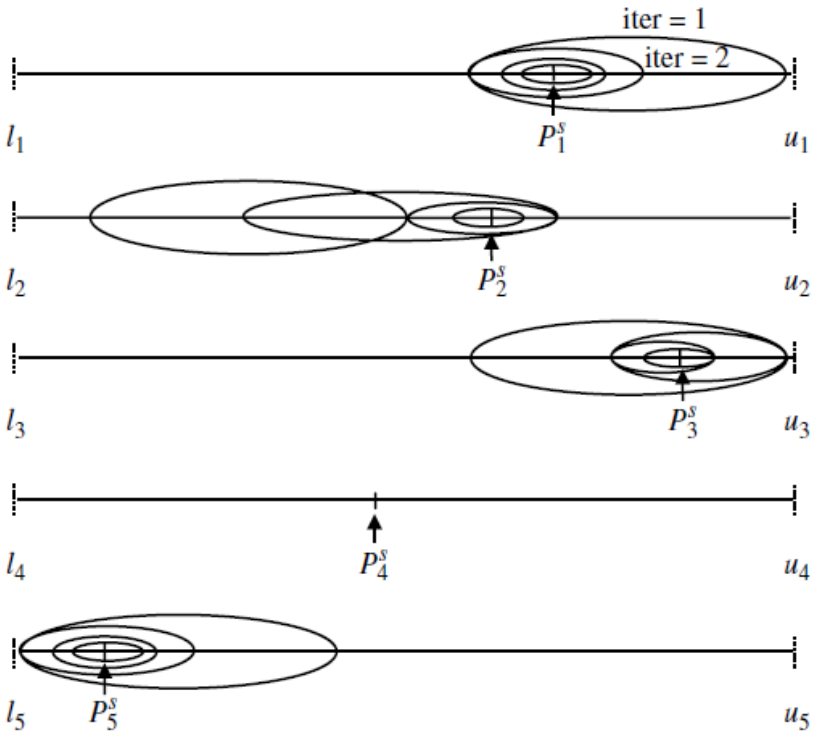
Obrázok 6. Príklad fázy 2 (*NumSol* > 3) [2].



Obrázok 7. Grafická reprezentácia určenia $p_1^n(iter)$ počas fázy 3 [2].

Príklad lokálneho vyhľadávania pomocou elíps je zobrazený na obrázku 8. Elipsa predstavuje oblasť, ktorá sa prehľadáva pri jednej iterácií lokálneho vyhľadávania. Na

obrázku je päť parametrov, pričom štvrtý je fixovaný. Vyhľadávanie končí, keď je elipsa dostatočne malá. Veľkosť elipsy je určená parametrom ε.

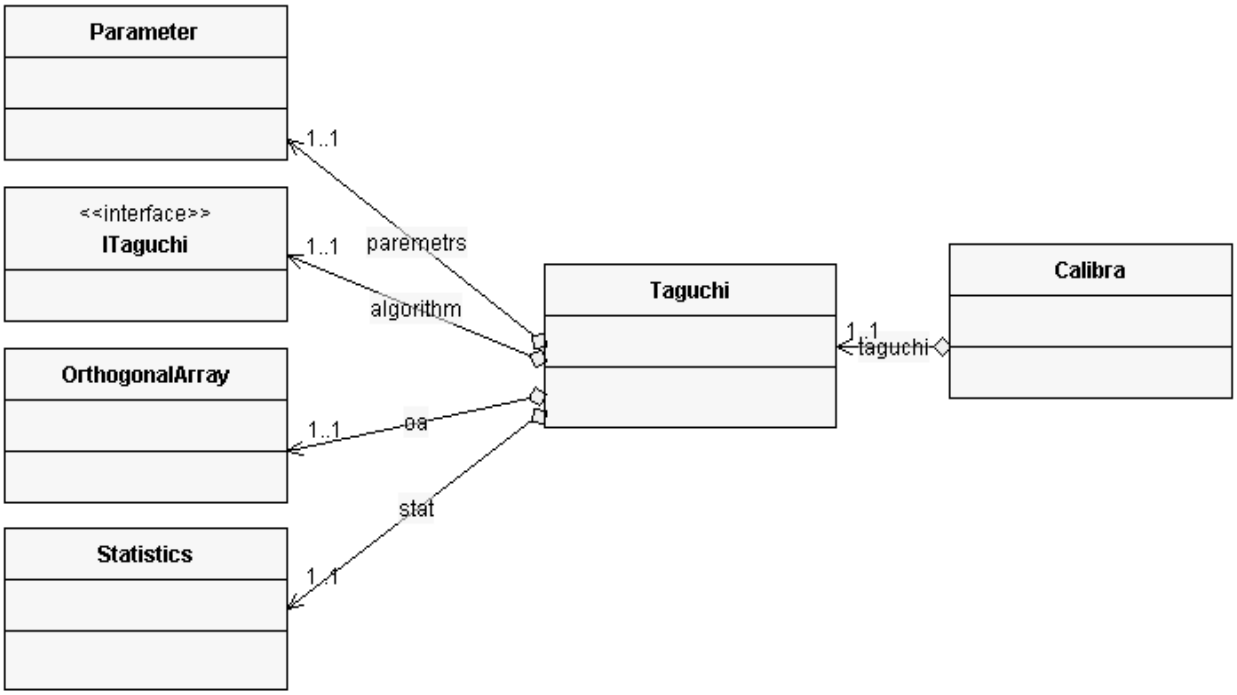


Obrázok 8. Lokálne vyhľadávanie vo fáze 3 [2].

4 Popis programu

4.1 Popis tried

Taguchiho metódu aj Calibra algoritmus som naprogramoval v jazyku Java pomocou vývojového prostredia NetBeans. Aplikácia pozostáva z piatich tried a jedného rozhrania (obrázok 9), kde môžeme vidieť základne vzťahy medzi triedami.



Obrázok 9. Calibra - diagram tried

4.1.1 Trieda *Parameter*

Trieda *Parameter* slúži na uchovanie jednotlivých parametrov a ich levelov.

Atribúty:	*parameterName*	Meno parametra.
	parameterType	Typ parametra int/double.
	parameterLevel	Pole, v ktorom sú uložené hodnoty levelov.

Metódy:	*getParameterValues*	Vráti pole hodnôt levelov.
	getParameterName	Vráti meno parametra.
	getNumberOfLevels	Vráti počet levelov daného parametra.
	getParameterType	Vráti typ parametra.

4.1.2 Trieda *OrthogonalArray*

Trieda *OrthogonalArray* slúži na načítanie správneho ortogonálneho poľa podľa počtu parametrov a levelov.

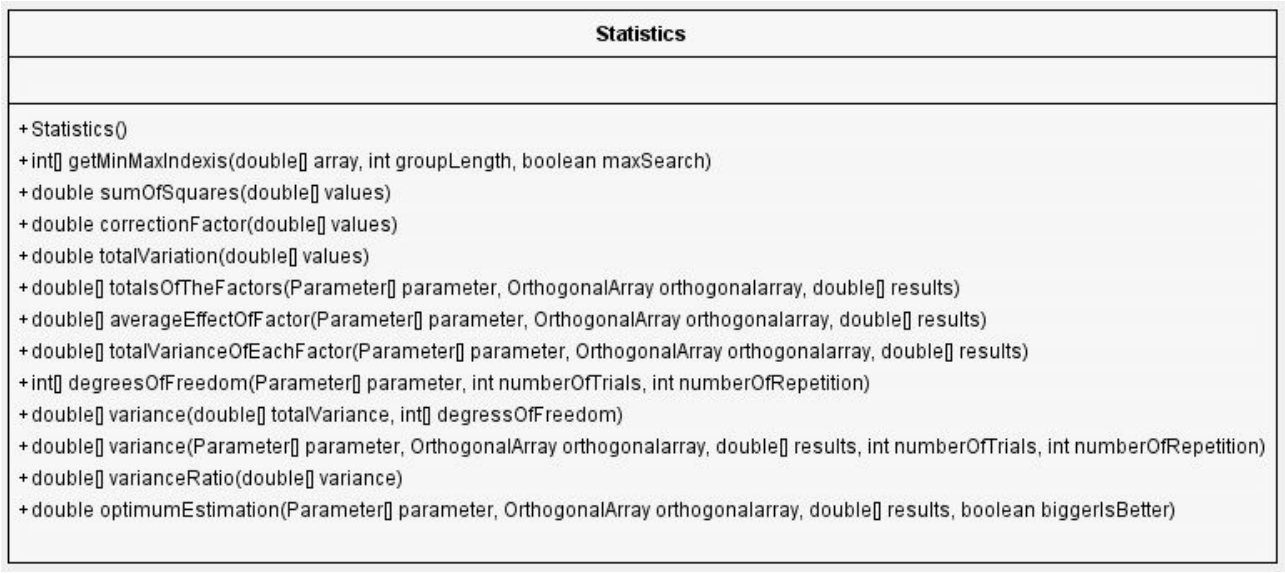
Atribúty:	*oa*	Matica ortogonálneho poľa.
	rows	Počet riadkov (experimentov) matice.
	factors	Počet faktorov.
	levels	Počet levelov.

Metódy:	*load*	Načíta OA zo súboru podľa počtu faktorov a levelov.
	cropOA	Oreže OA, ak je to potrebné (*factors* != *columns).*
	getOA	Vráti OA.
	getFactors	Vráti počet faktorov.
	getLevels	Vráti počet levelov
	getNumberOfRuns	Vráti počet experimentov pre jednu iteráciu.

4.1.3 Trieda *Statistics*

Trieda *Statistics* (obrázok 10) slúži na analýzu výsledkov. Detailný popis parametrov metód pre túto triedu je na obrázku 10. Správnosť štatistík je kontrolovaná unit testami.

Metódy:	*getMinMaxIndexis*	
	sumOfSquares	
	correctionFactor	
	totalVariation	
	totalsOfTheFactors	
	averageEffectOfFactor	
	totalVarianceOfEachFactor	
	degreesOfFreedom	
	variance	
	varianceRatio	
	optimumEstimation	

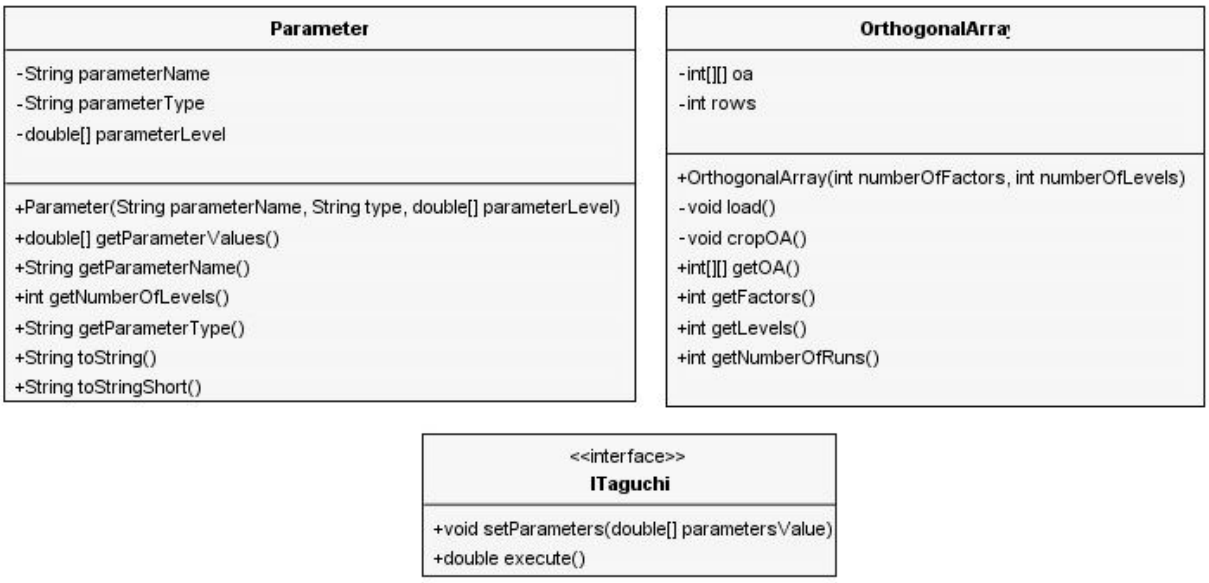


Obrázok 10. Trieda *Statistics*

4.1.4 Interface *ITaguchi*

Interface *ITaguchi* obsahuje dve metódy. Algoritmus, ktorého parametre ladíme a je zahrnutý v programovom balíku, musí tieto metódy implementovať.

Metódy:	*setParameters*	Nastavuje hodnoty parametrov.
	execute	Spustí algoritmus.



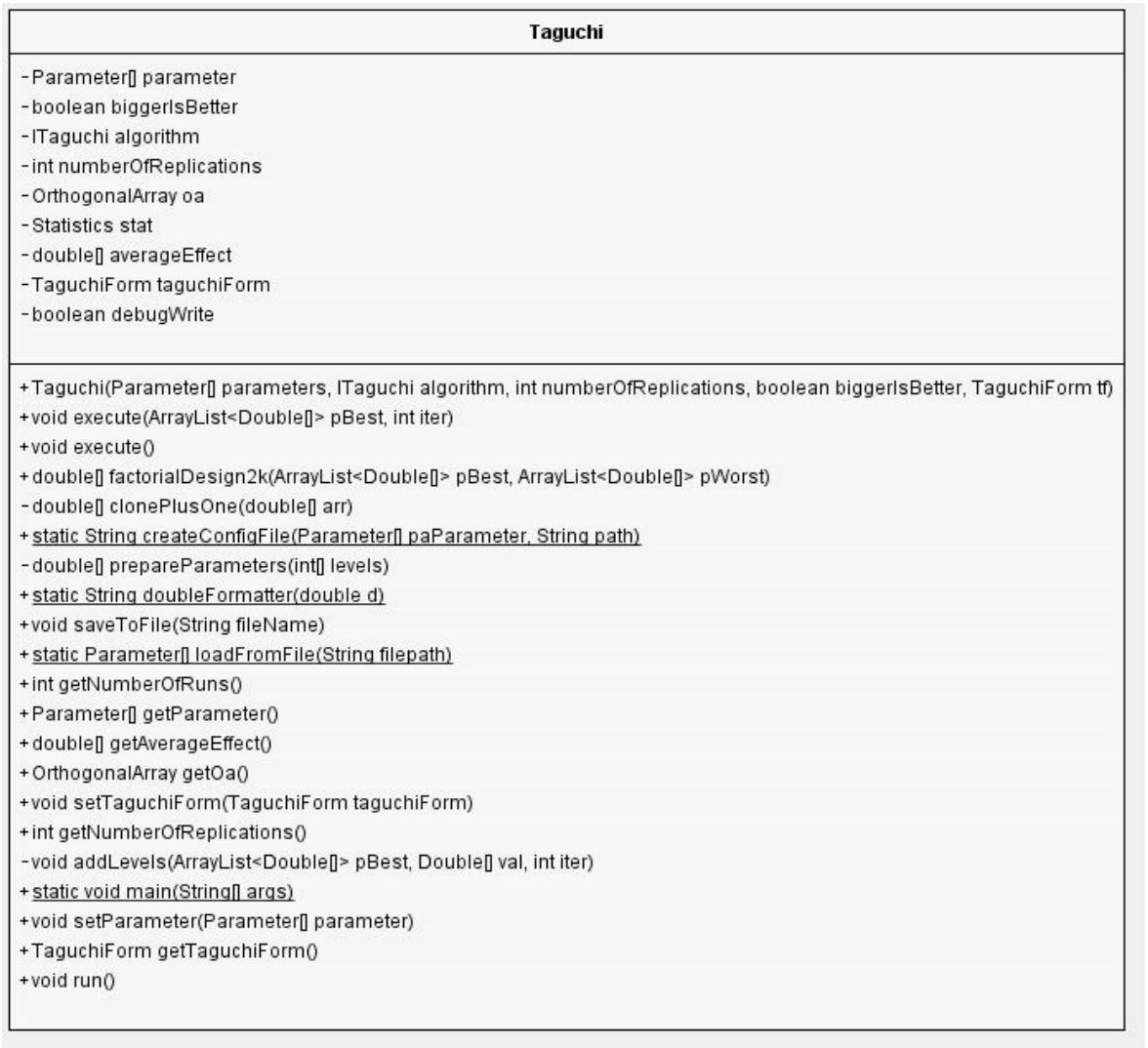
Obrázok 11. Triedy *Parameter*, *OrthogonalArray* a *ITaguchi*

4.1.5 Trieda *Taguchi*

Trieda Taguchi predstavuje Taguchiho algoritmus a využíva všetky vyššie spomenuté triedy. Pre ukladanie a načítanie aktuálnych nastavení do súboru som využil voľne dostupnú knižnicu *Gson* (https://github.com/google/gson).

Atribúty:	*parameters*	Pole všetkých parametrov.
	biggerIsBetter	Boolean premenná. Ak má hodnotu *True*, maximalizuj výstup z algoritmu, inak minimalizuj.
	algorithm	Algoritmus implementujúci *ITaguchi.*
	numberOfReplications	Maximálny počet replikácií.
	oa	Ortogonálne pole.
	stat	Inštancia štatistickej triedy.
	averageEffect	Priemerný vplyv každého parametra.
	taguchiForm	Referencia na GUI.

Metódy:	*Taguchi*	Konštruktor.
	execute	Vykoná experimenty podľa aktuálne OA.
	factorialDesign2k	Vykoná úplný faktorový plán.
	createConfigFile	Vytvorí konfiguračný súbor podľa šablóny.
	saveToFile	Uloží nastavenia do súboru.
	loadFromFile	Načíta nastavenia zo súboru.



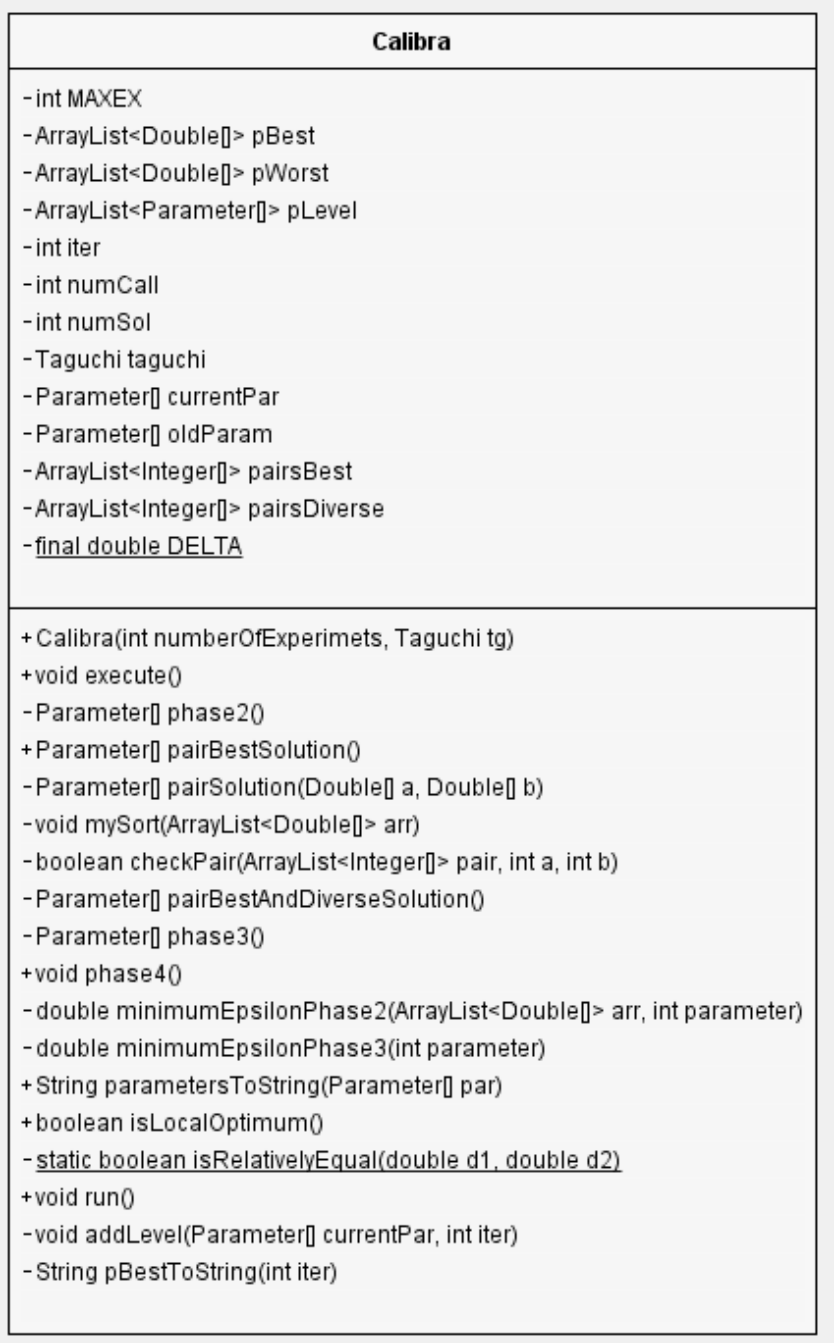
Obrázok 12. Taguchiho trieda

4.1.6 Trieda *Calibra*

Calibra predstavuje triedu ktorá vykonáva usmernené hľadanie s iteratívnym volaním Taguchiho metódy. Pôvodné parametre sú opísané v tabuľkách 8 a 9, okrem tých moja implementácia obsahuje niekoľko dodatkových atribútov.

Atribúty:	*taguchi*	Referencia na Taguchiho metódu.
	currentPar	Aktuálne hodnoty parametrov.
	oldParam	Predchádzajúce hodnoty parametrov.
	pairsBest	Zoznam najlepších riešení, ktoré bolí párované.
	pairsDiverse	Zoznam diverzných riešení, ktoré boli párované.

Metódy:	*Calibra*	Konštruktor.
	execute	Spustenie metódy.
	pairBestSolution	Metóda vyberá riešenia pre párovanie.
	pairSolution	Metóda pre párovanie riešení
	mySort	Zoraďuje riešenia podľa výsledku.
	checkPair	Kontroluje, či boli dané riešenia párované.
	pairBestAndDiverseSolution	Metóda vyberá riešenia pre párovanie.
	phase2, phase3, phase4	Jednotlivé fázy algoritmu.
	isLocalOptimum	Kontroluje, či je riešenie lokálnym optimom.
	isRelativelyEqual	Porovnáva dve hodnoty na presnosť δ.



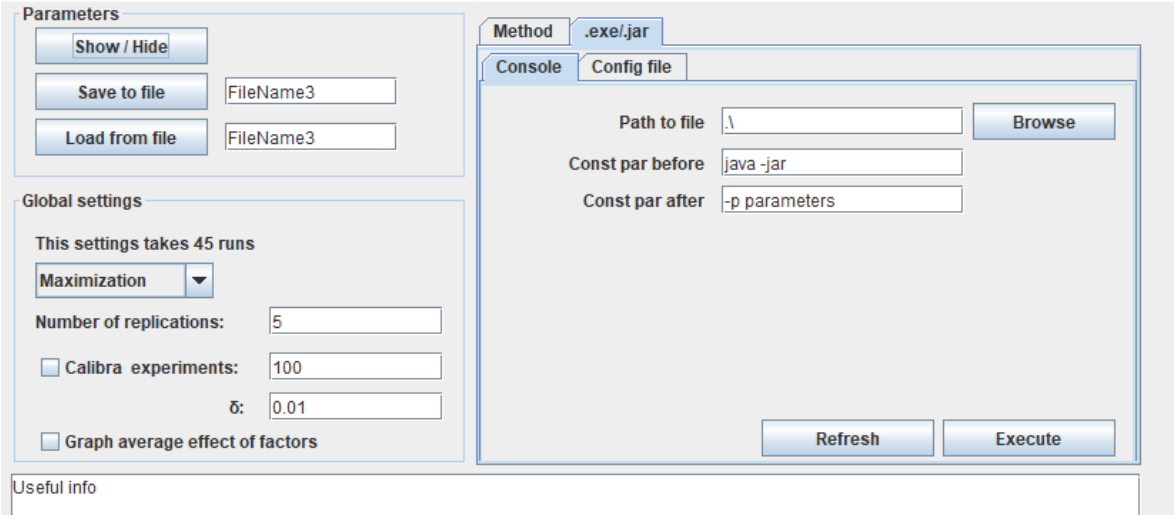
Obrázok 13. Calibra trieda

Program obsahuje aj balíček GUI ktorý predstavuje užívateľské rozhranie. Tento balíček využíva knižnicu *jFreeChart* (*http://www.jfree.org/jfreechart/*) použitú na vykresľovanie grafov.

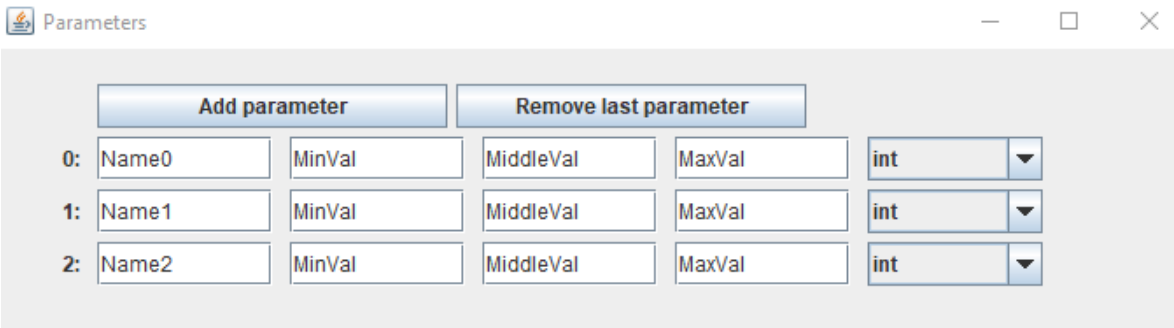
4.2 Aplikácia

Súčasťou programu je aj grafické Rozhranie (obrázok 14), na ktorom môžeme vidieť základné nastavenia ako zobrazenie a skrytie okna s parametrami pomocou tlačidla

Show / Hide a tlačidlá pre ich načítanie a uloženie do súboru, voľbu maximalizácie alebo minimalizácie výsledku cieľového algoritmu, počet replikácií a informačné okno. Aplikácia taktiež poskytuje možnosť použiť samotnú Taguchiho metódu. Taguchiho metóda sa vykoná, ak nie je označená možnosť *Calibra experiments*. Taguchiho metóda je výhodnejšia pri časovo náročných algoritmoch. Informácia *This settings takes 45 runs* udáva, koľko volaní algoritmu vyžaduje jedna iterácia Taguchiho metódy. Aplikácia taktiež dovoľuje spustenie algoritmu Calibra s obmedzeným počtom volaní algoritmu. Graf priemerného vplyvu faktorov bude predvedený v kapitole 5.1.2. Obrázok 15 predstavuje možnosti voľby parametrov.



Obrázok 14. Hlavné okno aplikácie



Obrázok 15. Upravovanie parametrov

Implementácia Taguchiho metódy môže aktuálne použiť maximálne 7 parametrov s 2 levelmi (OA_8) alebo 4 parametre s 3 levelmi (OA_9). Je však veľmi ľahko rozšíriteľná na použitie akéhokoľvek OA, a teda rôznych počtov parametrov a levelov. Pri použití dvoch parametrov sa stredný level parametra nevypĺňa. Program dokáže sám vybrať najvhodnejšie OA a upraviť ho podľa zvolených parametrov.

Aktuálna verzia aplikácie dokáže bez zásahu užívateľa spustiť algoritmy, ktoré sú súčasťou programového balíka, ale aj samostatne spustiteľné aplikácie, pre ktoré existuje možnosť, aby užívateľ vykonal experimenty sám, aj s niekoľkými replikáciami a následne uložil výsledky do súboru, ktoré aplikácia spracuje.

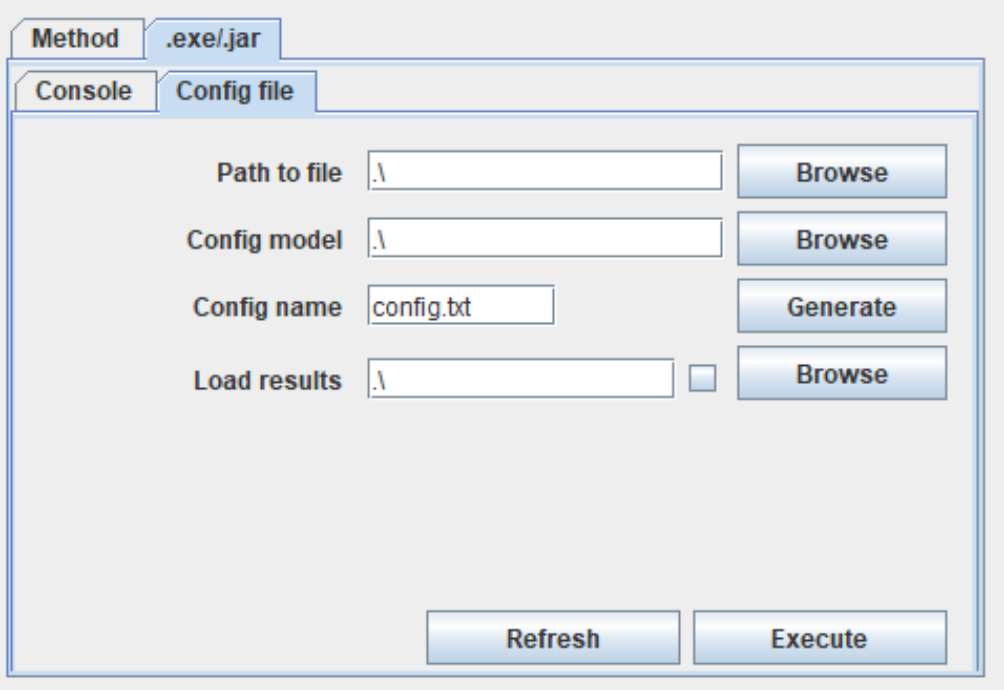
Aplikácia podporuje spustenie algoritmov, ktoré sú jej súčasťou, a teda sa musia nachádzať v rovnakom projekte a musia implementovať interface *ITaguchi*. Taktiež podporuje samostatne spustiteľné súbory ako napríklad *.exe*, *.jar* a iné. Takýmto algoritmom sa väčšinou môžu parametre zadávať prostredníctvom argumentov, a preto je potrebné, aby sa mená parametrov zhodovali s názvami argumentov. Napríklad algoritmus s názvom *GenOpt.jar* sa môže spúšťať z konzoly s použitím parametrov *pop*, *gen a time,* celý príkaz teda vyzerá takto: „*java -jar GenOpt.jar -pop 100 -gen 10 -time 30"*, pričom hodnota parametra *pop* je 100, hodnota parametra *gen* je 10 a *time* je konštantný parameter s hodnotou 30. Z tohto dôvodu sa musia aj ladené parametre v okne *Parameters* nazývať *gen* a *pop*. Konštantné parametre sú parametre, ktoré sa neladia, ale sú potrebné pre spustenie programu. Program taktiež podporuje zadávanie konštantných parametrov pred názvom programu, napr. programy s koncovkou *.jar* ako *GenOpt.jar* sa musia z konzoly spúšťať s konštantnými parametrami *java -jar* pred názvom programu. Príkaz teda vyzerá takto: „*java -jar GenOpt.jar ...*" Je potrebné, aby jediný výstup zo samostatne spustiteľného súboru bol výsledok algoritmu. Program následne sám vygeneruje a spustí príkazy cez konzolu a vyhodnotí najlepšiu kombináciu parametrov.

Samostatne spustiteľné programy môžu využívať aj konfiguračné súbory. Bolo by veľmi nepraktické editovať konfiguračný súbor po každom spustení algoritmu. Z tohto dôvodu program obsahuje možnosť ako tieto konfiguračné súbory generovať podľa šablóny. Ich jediné obmedzenie je, že všetky cesty musia byť absolútne. Napríklad program *multiBoost.exe* využíva konfiguračný súbor, šablóna pre tento konfiguračný súbor vyzerá nasledovne:

```
#multiboost.exe --configfile C:/Work/config[NUMBER].txt
fileformat arff
timelimit 120
traintest C:/Work/Train/train.arff C:/Work/Train/test.arff [PARAMETER0]
learnertype TreeLearner
baselearnertype HaarSingleStumpLearner [PARAMETER1]
csample num [PARAMETER2]
iisize 28x28
outputinfo C:/Work/results/result[NUMBER].dta
```

constant
seed 73

Za povšimnutie stoja hodnoty uvedené v hranatých zátvorkách ako *[NUMBER]* alebo *[PARAMETER0]*, ktoré budú nahradené konkrétnou hodnotou. *[NUMBER]* predstavuje číslo experimentu a *[PARAMETER0]* konkrétnu hodnotu parametra, ktorý je v zozname parametrov na prvom mieste. Na obrázku 16 môžeme vidieť možnosti generovania šablóny, *Path to file* predstavuje cestu k samostatne spustiteľnému súboru, *Config model* cestu k šablóne a *Config name* meno novo vytvorených šablón. Konfiguračné súbory sa generujú do adresára, v ktorom je program spustený. Ak je potrebné vykonať 9 experimentov a názov šablóny je *config.txt*, program vygeneruje 9 súborov s názvami *config0.txt*, *config1.txt*, ..., *config8.txt*. Program môže vykonať experimenty samostatne, avšak jediný výstup zo samostatne spustiteľného súboru musí byť výsledok. Experimenty môže vykonať aj užívateľ a výsledky zapísať do súboru, každý výsledok na nový riadok v takom poradí, v akom sú vygenerované šablóny. Ak je potrebných viacero replikácií, výsledky jednej konfigurácie parametrov sa píšu za sebou a sú oddelené medzerou. V políčku *Load results* sa vyplní cesta k súboru s výsledkami experimentov a program vyhodnotí najlepšiu kombináciu parametrov, ktorú vypíše v informačnom okne aplikácie.



Obrázok 16. Generovanie konfiguračných súborov

Alternatívny spôsob ladenia parametrov pre samostatne spustiteľné súbory je skombinovať dva predchádzajúce spôsoby. Môžeme rozšíriť programový balík o metódu, ktorá bude spúšťať samostatne spustiteľný súbor a z jej výpisu vyseparovať informáciu o výsledku algoritmu.

Calibra algoritmus používa ortogonálne pole, ktoré podporuje maximálne 4 parametre s 3 levelmi. Vzhľadom na to, že pre nájdenie jedného lokálneho optima pre cieľový algoritmus obsahujúci 4 parametre potrebujeme vykonať minimálne 34 experimentov (úplný faktorový plán $2^4 = 16$ experimentov, dvakrát Taguchiho metóda po 9 experimentov), ručné vykonávanie pre Calibra algoritmus by bolo veľmi nepraktické, a preto Calibra nepodporuje načítavanie výsledkov zo súboru.

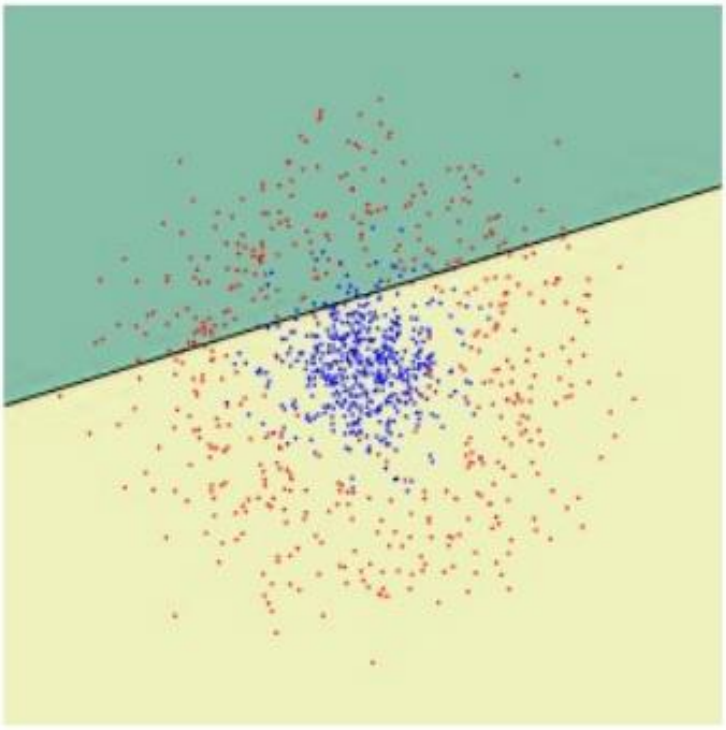
5 Testovanie

Taguchiho metóda je použitá na jemné doladenie parametrov pre algoritmus AdaBoost, ktorý môžeme natrénovať na klasifikáciu rôznych objektov. Trénovanie algoritmu je časovo náročné, a preto nie je vhodné použitie Calibra algoritmu.

Calibra algoritmus je otestovaný na genetickom algoritme, ktorý rieši úlohu o batohu. Obe úlohy sú popísané nižšie.

5.1 AdaBoost

Boosting je metóda pre dosiahnutie lepšieho výsledku akéhokoľvek učiaceho sa algoritmu. Základným predpokladom boostingu je, že nájdenie lineárnej kombinácie slabých klasifikátorov, ktoré ako celok vytvárajú jeden silný klasifikátor, je jednoduchšie ako nájdenie jedného silného klasifikátora. Boosting sa skladá z dvoch častí, v prvej sa iteratívne nájde niekoľko slabých klasifikátorov, v druhej sa z nich vytvorí jeden silný klasifikátor. Slabý klasifikátor (obrázok 17) dokáže klasifikovať objekty (rozhodnúť, či objekt do nejakej skupiny patrí alebo nepatrí) o niečo lepšie ako náhodné hádanie.



Obrázok 17. Slabý klasifikátor [18]

Algoritmus AdaBoost navrhli Freund a Schapire [1]. Bol to jeden z prvých algoritmov, ktoré kombináciou slabých klasifikátorov získali silný klasifikátor. Výsledný silný klasifikátor na vstupných dátach $x \in X$ je lineárnou kombináciou T slabých klasifikátorov:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

$H(x)$ – silný klasifikátor, α_t – koeficient slabého klasifikátora, h_t – slabý klasifikátor

Medzi jeho hlavné výhody patrí exponenciálne znižovanie chyby a to, že nemá sklony sa pretrénovať. Pretrénovaný klasifikátor si všíma špecifické vlastnosti datasetu, na ktorom je trénovaný a ktoré sa nevyskytujú v testovacích alebo reálnych dátach.

Algoritmus AdaBoost sa vykonáva iteratívne a v každej iterácii sa do lineárnej kombinácie pridá jeden slabý klasifikátor. Slabý klasifikátor je vybraný tak, aby bol minimalizovaný horný odhad chyby klasifikátora. Trénovacia množina pozostáva z dvojíc (x_i, y_i), pričom x_i predstavuje i-tú vzorku z množiny vstupných dát X a y_i správny výsledok klasifikácie x_i, pričom y_i môže nadobúdať jednu s dvoch hodnôt {-1, 1}, teda nepatrí a patrí. Vstupné dáta obsahujú $i = 1, \dots, m$ vzoriek. Pri učení sú využívané váhy pozorovaní v trénovacej množine $D_t(i)$, ktoré sú na začiatku inicializované rovnomerne na hodnotu $D_1(i) = 1/m$. V každej iterácii t algoritmu je potrebné vykonať nasledujúce kroky.

1. Nájsť najlepší slabý klasifikátor pri daných váhach $\boldsymbol{D}_t$.

2. Skontrolovať chybu tohto klasifikátora, ktorá musí byť menšia než 0,5.

2a. Spočítať koeficient slabého klasifikátora α_t .

2b. Aktualizovať váhy $\boldsymbol{D}_t$.

Algoritmus AdaBoost [17]:

Vstup: $(x_1, y_1), \dots, (x_m, y_m); x_i \in X, y_i \in \{-1, 1\}\ pre\ i = 1, \dots, m$

Inicializácia váh: $D_1(i) = 1/m$ pre $i = 1, ..., m$

For $t = 1, \dots, T$:

1. Nájdi $h_t = \arg min_{h_j} \varepsilon_j;\ \varepsilon_j = \sum_{i=1}^{m} D_t(i) I[y_i \neq h_j(x_i)\ ]$

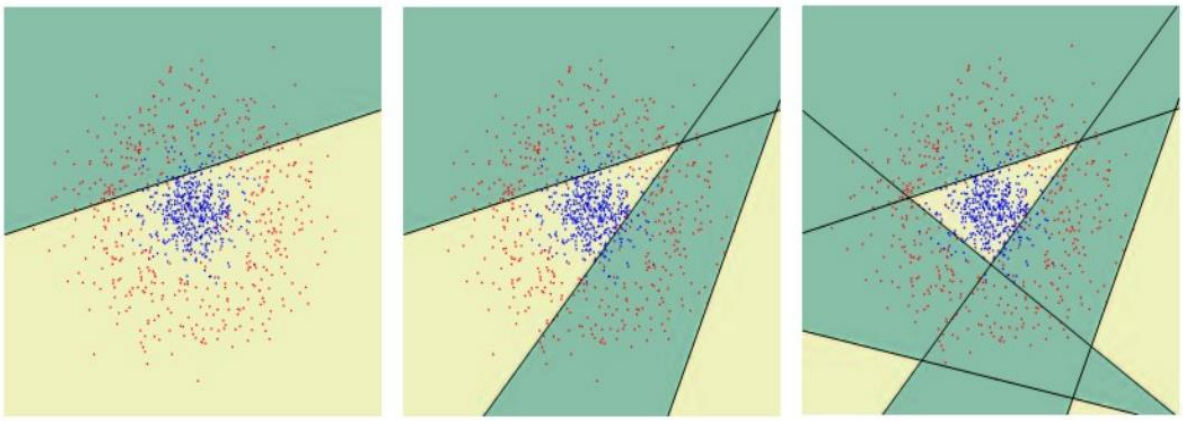
2. If $\varepsilon_t < \frac{1}{2}\ then$

2a. $\alpha_t = \frac{1}{2}\log(\frac{1-\varepsilon_t}{\varepsilon_t})$

2b. Pre i = 1, ..., m: Aktualizuj $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$; $Z_t = \sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$

Výsledný klasifikátor: $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$

Vybraný slabý klasifikátor musí mať najmenšiu váženú chybu na trénovacej množine. Funkcia $I[y_i \neq h_j(x_i)]$ v kroku 1 vracia 1, pokiaľ je výraz pravdivý, inak vráti 0. Podmienka v 2. kroku zabezpečuje, aby nájdený slabý klasifikátor bol lepší ako náhodný, a tým sa zabezpečuje konvergencia algoritmu. Aktualizácia váh v 2b. kroku spôsobí, že váha zle klasifikovaných meraní sa zväčší a váha dobre klasifikovaných sa zmenší. V nasledujúcom kroku sa bude hľadať slabý klasifikátor, ktorý bude musieť lepšie klasifikovať zatiaľ zle klasifikované dáta.



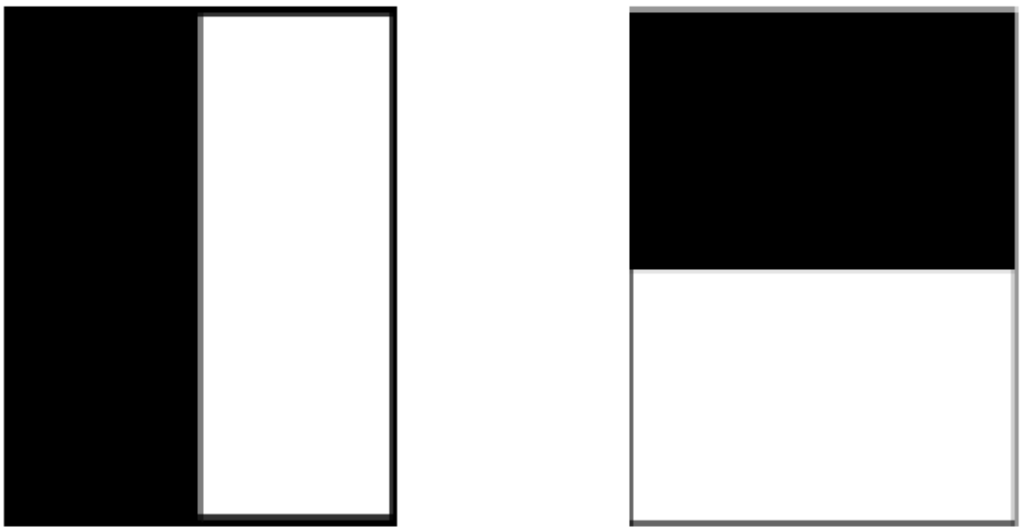
Obrázok 18. Iterácia AdaBoost t = 1, t = 3, t = 5 [18]

5.1.1 Popis problému

Úlohou je nájsť čo najlepšie parametre pre algoritmus AdaBoost, ktorý sa používa na klasifikovanie objektov. V mojom prípade budem klasifikovať konštantnú množinu obrázkov a zmenou hodnôt parametrov algoritmu AdaBoost chcem dosiahnuť čo najmenší počet zle klasifikovaných obrázkov (chybu klasifikácie).

Pri klasifikácii obrázkov sa často využívajú hárové vlnky (*Haar wavelet filter*) ako slabé klasifikátory. Príklad hárových vlniek je na obrázku 19. Hárovú vlnku si môžeme predstaviť ako štvoruholníkový filter rozdelený na dve časti. Majme teda vlnku o rozmere 10x10 pixelov, ktorá sa nachádza na ľubovoľnom mieste na obrázku ľubovoľného

rozmeru. Pri klasifikovaní obrázku väčšinou pracujeme s čiernobielymi obrázkami, takže každý pixel je reprezentovaný jediným číslom, ktoré vyjadruje odtieň sivej. Jedinou úlohou vlnky je sčítať hodnoty všetkých pixelov, ktoré sa nachádzajú v čiernej časti a v bielej časti vlnky. Výsledkom je rozdiel medzi sumou čiernych a bielych častí. Tento údaj spolu s hraničnou hodnotou sumy (*threshold*) dáva dostatočnú informáciu pre klasifikáciu objektu. Vlnky môžu mať rôzny tvar, natočenie a rozdelenie bielej a čiernej časti.



Obrázok 19. Príklad Hárových vlniek [4]

AdaBoost sa používa hlavne na klasifikáciu. V mojom prípade trénovanie prebieha na datasete Mnist. Tento dataset je voľne dostupný a bežne používaný [12], obsahuje obrázky číslic 0 až 9 s rozmerom 28x28 pixelov (obrázok 20). Trénovací dataset obsahuje 60 000 obrázkov, zatiaľ čo testovací 10 000. Úlohou je teda natrénovať silný klasifikátor pomocou algoritmu AdaBoost, ktorý dokáže klasifikovať obrázky nachádzajúce sa v dasatese Mnist s čo najmenším percentom zle klasifikovaných obrázkov.

Obrázok 20. Mnist dataset [15]

Algoritmus využíva 3 parametre, ktoré sú popísané v tabuľke 10. Maximálny počet iterácií predstavuje parameter *t* rovný maximálnemu počtu slabých klasifikátorov v silnom klasifikátore. Počet listov v strome je vlastnosť využívaná pre klasifikáciu viac ako dvoch objektov. Posledný parameter je počet hárových vlniek v jednom slabom klasifikátore, ktoré sa môžu odlišovať tvarom, rozdelením, threshold-om. Pred ladením parametrov som vyskúšal rôzne hodnoty parametrov (tabuľka 11), ktoré som čerpal zo zdrojov [7, 14]. Najlepší výsledok pred ladením parametrov som dosiahol s kombináciou parametrov

A = 1000; B = 10; C = 100

Výsledok pri tejto kombinácií faktorov je 3,29% zle klasifikovaných číslic.

Tabuľka 10. Parametre algoritmu AdaBoost

Parameter	Popis
A	Maximálny počet iterácií
B	Počet listov v strome
C	Počet hárových vlniek

Tabuľka 11. Experimenty pred ladením parametrov

A	*B*	*C*	Výsledok
1000	8	200	5,29%
1000	10	100	3,29%
1000	10	200	5,47%
1000	200	20	3,45%

5.1.2 Riešenie

V prvej sérii experimentov som pre každý parameter vybral tri levely (tabuľka 12). Túto sériu experimentov budem v ďalšom texte nazývať plán 1. Použitím Taguchiho metódy chceme definovať najlepší level každého parametra a prínos každého parametra k výsledku.

Tabuľka 12. Levely parametrov algoritmu AdaBoost, plán 1.

Level 1	Level 2	Level 3
A_1= 900	A_2 = 1000	A_3 = 1100
B_1= 9	B_2= 10	B_3= 11
C_1 = 90	C_2 = 100	C_3 = 110

Algoritmus AdaBoost spúšťaný pomocou programu *multiBoost.exe* [13] využíva konfiguračný súbor, ktorého šablóna bola ukázaná vyššie. Keďže používam 3 parametre s 3 levelmi, program sám vyberie $OA_9(3^4)$, ktoré musí vykonať 9 experimentov, a teda program vytvorí 9 konfiguračných súborov. Hodnoty parametrov v týchto súboroch a výsledky experimentov predstavujúce percento zle klasifikovaných objektov sú v tabuľke 13. Jeden trénovací beh algoritmu AdaBoost zaberie 2 hodiny. Počet replikácií je nastavený na 1.

Tabuľka 13. Hodnoty parametrov a výsledky experimentov v pláne 1.

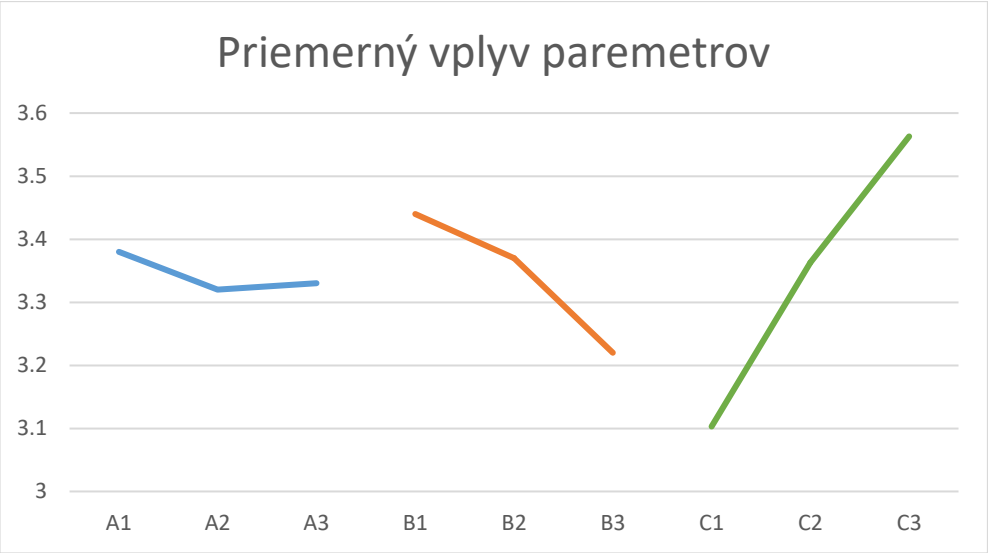
Experiment	1	2	3	4	5	6	7	8	9
A	900	900	900	1000	1000	1000	1100	1100	1100
B	9	10	11	9	10	11	9	10	11
C	90	100	110	100	110	90	110	90	100
Výsledok	3,35%	3,29%	3,50%	3,46%	3,68%	**2,82%**	3,51%	3,14%	3,34%

Pomocou analýzy výsledkov dokážeme určiť priemerný vplyv parametrov (obrázok 21). Keďže pri algoritme AdaBoost sa snažíme výsledok minimalizovať, z priemerného vplyvu faktorov vyplýva, že kombinácia faktorov $A_2\ B_3\ C_1$ pravdepodobne vráti najlepší výsledok. Hodnoty parametrov s týmito levelmi:

A_2 čo predstavuje 1000 iterácií.

B_3 čo predstavuje 11 listov v strome.

C_1 čo predstavuje 90 hárových vlniek.



Obrázok 21. Priemerný vplyv parametrov v pláne 1

Odhad výsledku pri tomto nastavení parametrov je **2,95%** a zhodou okolností ide o 6. experiment so skutočnou chybou klasifikácie **2,82%**. Keďže parametre *B* a *C* skončili na hraničných hodnotách, zmeníme rozsahy týchto parametrov. Rozsah parametra *A* zostane nezmenený. Nové hodnoty levelov sú popísané v tabuľke 14. Experimenty s takýmito hodnotami parametrov tvoria plán 2.

Tabuľka 14. Levely parametrov algoritmu AdaBoost, plán 2.

Level 1	Level 2	Level 3
A_1= 900	A_2 = 1000	A_3 = 1100
B_1= 11	B_2= 12	B_3= 13
C_1 = 70	C_2 = 80	C_3 = 90

Opäť vygenerujeme konfiguračné súbory a následne vykonáme ďalšie kolo experimentov, ktorých hodnoty parametrov a výsledky sú popísané v tabuľke 15.

Tabuľka 15. Hodnoty parametrov a výsledky experimentov v pláne 2.

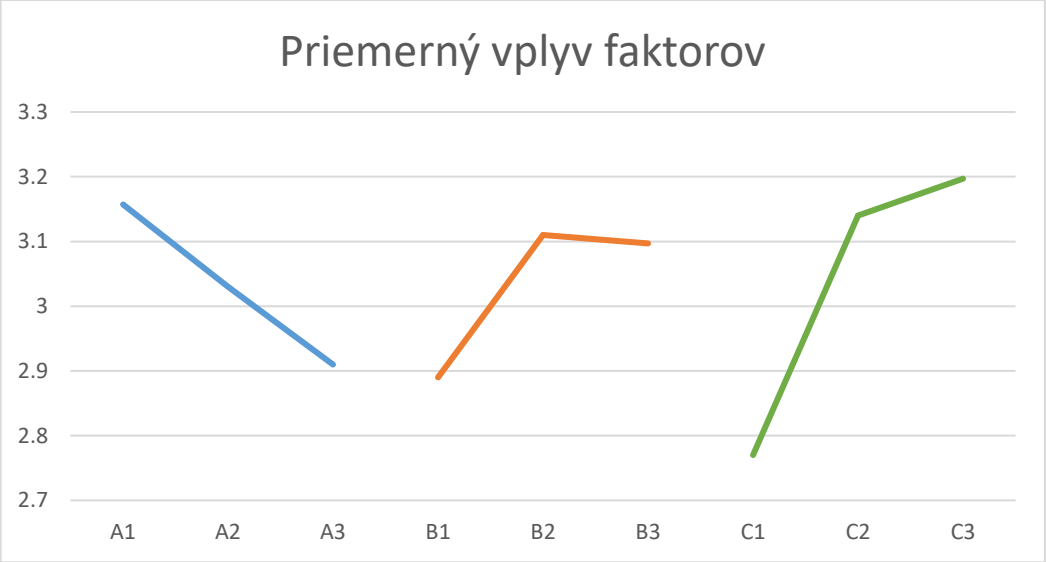
Experiment	1	2	3	4	5	6	7	8	9
A	900	900	900	1000	1000	1000	1100	1100	1100
B	11	12	13	11	12	13	11	12	13
C	70	80	90	80	90	70	90	70	80
Výsledok	2,81%	3,23%	3,43%	3,04%	3,34%	**2,72%**	2,82%	2,77%	3,14%

Pomocou analýzy výsledkov určíme priemerný vplyv parametrov (obrázok 22), z ktorého vyplýva, že kombinácia faktorov $A_3\,B_1\,C_1$ pravdepodobne vráti najlepší výsledok. Hodnoty parametrov s týmito levelmi sú:

A_3 čo predstavuje 1100 iterácií.

B_1 čo predstavuje 11 listov v strome.

C_1 čo predstavuje 70 hárových vlniek.



Obrázok 22. Priemerný vplyv parametrov v pláne 2

Odhad chyby pri tomto nastavení parametrov je **2,50%**, zatiaľ čo konfirmačný beh skončil s chybou **2,25%**. Všetky výsledky, konfiguračné modely a súbory sú v priečinku AdaBoost na priloženom DVD.

5.2 Genetický algoritmus

5.2.1 Popis problému

Druhou úlohou je nájsť optimálne parametre pre genetický algoritmus, ktorý rieši úlohu o batohu (*knapsack problem*)[3]. Genetický algoritmus som si vybral, pretože obsahuje parametre, od ktorých záleží výsledok riešenej úlohy, je ľahký na implementáciu a pochopenie, nie je presne definovaný a preto je dostatočne škálovateľný, napríklad môžem časovo obmedziť jeho vykonávanie. Úloha o batohu sa dá riešiť genetickým algoritmom, pričom ide o úlohu, v ktorej máme množinu predmetov I a o každom predmete musíme rozhodnúť, či ho vezmeme alebo nie. Takéto rozhodnutie typu áno/nie sa modeluje premennými, ktoré sú bivalentné a predstavujú ho premenné $x_i \; pre \; i \in I$. Cena každého predmetu je vyjadrená konštantou c_i, váha predmetu konštantou w_i a nosnosť batoha konštantou W.

Všeobecný tvar úlohy o batohu môžeme formulovať ako:

$$max \sum_{i \in I} x_i * c_i$$

$$za\ podmienok:$$

$$\sum_{i \in I} x_i * w_i \leq W$$

$$x_i \in \{0,1\}\ pre\ i \in I$$

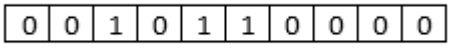
Úlohu budem riešiť genetickým algoritmom [9], ktorý patrí medzi tzv. evolučné algoritmy, ktoré sú inšpirované procesmi v prírode. Každé riešenie problému sa nazýva chromozóm a je tvorené binárnym reťazcom (génmi) danej dĺžky, ktorá je pre všetky chromozómy rovnaká. Populácia je množina chromozómov. Základná populácia je začiatočný stav, v ktorom všetky chromozómy musia byť riešením problému a je vygenerovaná náhodne. Z populácie vzniká krížením nová generácia, najlepšie chromozómy z tejto generácie nahradia pôvodnú populáciu. Tento vývoj riešenia zahŕňa:

Výber chromozómov na kríženie (pseudonáhodný výber)

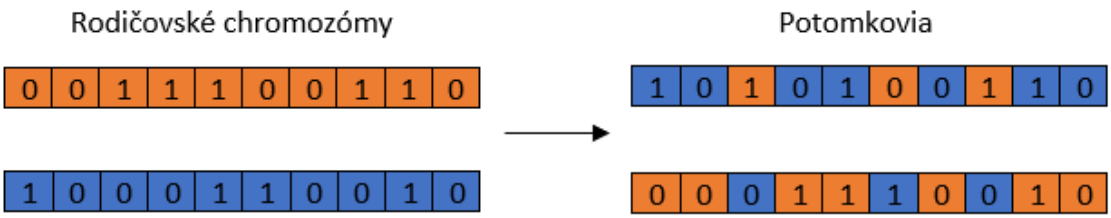
Kríženie chromozómov (výber podreťazca alebo jednotlivých génov)

Mutácia (náhodné zmutovanie malého počtu génov)

Moja implementácia reprezentuje chromozóm ako pole typu int, ktorého dĺžka sa rovná počtu predmetov a obsahuje hodnoty 0, ak predmet nie je v batohu a 1 - je v batohu (obrázok 23). Genetický algoritmus vyberá potomkov na kríženie náhodne a používa rovnomerné kríženie (obrázok 24). Tento druh kríženia vymení gény medzi jedincami s určitou pravdepodobnosťou, ktorá sa zadáva ako parameter algoritmu *pravdepodobnosť muttácie*. Po krížení náhodne zmením niekoľko génov oboch novo vytvorených jedincov, tento počet je daný parametrom *počet náhodne zmenených génov* . Týmito operáciami môžem dosiahnuť neprípustné riešenie, ktoré prekročí nosnosť batoha. Takéto riešenie nezahadzujem, ale používam metódu na sprístupnenie riešenia, ktorá vyhadzuje z batoha predmety s najmenšou cenou, až kým nie je hmotnosť batoha menšia alebo rovná ako nosnosť batoha.



Obrázok 23. Príklad chromozómu pre 10 predmetov



Obrázok 24. Rovnomerné kríženie

Genetický algoritmus využíva 4 parametre (tabuľka 16). Ak je veľkosť populácie 100 a veľkosť generácie 1000, z aktuálnej populácie vygenerujem 1000 nových chromozómov (riešení), tie predstavujú novú generáciu. Pri generovaní nových chromozómov vždy náhodne vyberiem dvoch rodičov z populácie. Pre každý gén vygenerujem náhodné číslo medzi 0 a 1. Ak je toto číslo menšie ako pravdepodobnosť mutácie, tak vymením gény vybraných chromozómov. Takto vzniknú dva nové chromozómy, ktorým zmením náhodné gény, počet takýchto zmien je definovaný premennou počet náhodne zmenených génov. Z novo vytvorenej generácie vyberiem 100 najlepších riešení, ktoré nahradia aktuálnu populáciu. Vykonávanie jedného behu genetického algoritmu je obmedzené na 30 sekúnd.

Tabuľka 16. Parametre genetického algoritmu

Parameter	Popis
A	Veľkosť populácie
B	Veľkosť generácie
C	Pravdepodobnosť mutácie
D	Počet náhodne zmenených génov

Pre úlohu o batohu som vytvoril generátor, ktorý vygeneruje jednu inštanciu úlohy. Parametre pre generátor sú popísané v tabuľke 17.

Tabuľka 17. Parametre generátora predmetov

Capacity	Maximálna nosnosť batoha.
numberOfObjects	Počet všetkých predmetov.
minMass	Minimálna váha predmetu.
maxMass	Maximálna váha predmetu.
minPrice	Minimálna cena predmetu.
maxPrice	Maximálna cena predmetu.
seed	Násada generátora, rovnaké parametre vygenerujú rovnaké predmety.

Genetický algoritmus rieši mnou vygenerovanú úlohu o batohu, ktorý má nosnosť 3500, počet predmetov je 800, hmotnosť predmetov je od 20 do 50 a ich cena od 1 do 50. Pred ladením parametrov som sa pokúsil ručne nájsť parametre pre genetický algoritmus s jednou replikáciou, hodnoty parametrov a výsledky sú popísané v tabuľke 17. Najlepší výsledok pred ladením parametrov som dosiahol s kombináciou parametrov

A = 100; B =1000; C =0,3; D =4

Výsledná hodnota batoha pri tejto kombinácii faktorov je **5071**.

Tabuľka 18. Experimenty pred ladením parametrov

A	*B*	*C*	*D*	výsledok
1000	1000	0,3	4	4228
1000	1000	0,6	8	3942
100	1000	0,3	4	5071
100	1000	0,6	8	4816
100	100	0,3	4	4230
100	100	0,6	8	3927

5.2.2 Riešenie

Samotný Calibra algoritmus obsahuje 5 parametrov, pričom 3 z nich (k, u_i, l_i) závisia od počtu a rozsahu parametrov genetického algoritmu. Taktiež je potrebné nastaviť počet replikácií, parametre som nastavil na hodnoty:

MAX = 1000, δ = 30, Počet replikácií = 5

Každý parameter genetického algoritmu obsahuje tri levely (tabuľka 19). Použitím algoritmu Calibra chceme nájsť optimálne nastavenie všetkých parametrov s presnosťou δ = 30. Ak algoritmus bude môcť vykonať dostatok experimentov, intervaly pre celočíselné aj spojité parametre budú mať rozsah 30.

Tabuľka 19. Hodnoty levelov genetického algoritmu

Level 1	**Level 2**	**Level 3**
A_1= 100	A_2 = 200	A_3 = 300
B_1= 500	B_2= 1000	B_3= 1500
C_1 = 0,4	C_2 = 0,6	C_3 = 0,8
D_1=2	D_2= 4	D_3 = 8

V prvom kroku Calibra vykoná 2^4 experimentov, tieto experimenty sú popísané v tabuľke 20, kde prvé tri stĺpce reprezentujú hodnoty parametrov a v poslednom je

priemerný výsledok z 5 replikácií. Následne sa vyhodnotí najlepšia a najhoršia kombinácia levelov, tieto hodnoty sú v tabuľke zvýraznené.

Tabuľka 20. Úplný faktorový plán, genetický algoritmus

A	*B*	*C*	*D*	výsledok
150	750	0.5	3	5008,6
150	750	0.5	5	4888,2
150	750	0.7	3	5022,4
150	750	0.7	5	4879,0
150	1250	0.5	3	5116,8
150	1250	0.5	5	4941,8
150	1250	0.7	3	5109,2
150	1250	0.7	5	4956,2
250	750	0.5	3	4894,2
250	750	0.5	5	4753,4
250	750	0.7	3	4910,6
250	750	0.7	5	4748,4
250	1250	0.5	3	5006,6
250	1250	0.5	5	4873,0
250	1250	0.7	3	5005,6
250	1250	0.7	5	4854,4

Následne sa vykoná hľadanie prvého lokálneho optima popísané v prvej polovici tabuľky 21, v tomto hľadaní je hodnota počtu náhodne zmenených génov konštantná a má hodnotu 3. Táto hodnota je vybraná podľa najlepšieho výsledku úplného faktorového plánu. Je konštatná, pretože Calibra počíta $\varepsilon = \frac{min\{(p_i^*(0) - l_i),(u_i - p_i^*(0))\}}{2}$ a po dosadení získame $\varepsilon = \frac{min\{(3-2),(8-3)\}}{2} = 1/2$. Keďže ide o celočíselný parameter, po zokrúhlení ε pomocou pretypovania na int získame $\varepsilon = 0$. Pretože počet náhodne zmenených génov je konštantný, v tabuľke 21 ho neuvádzam.

V jednom riadku tabuľky 21 sa nachádzajú 3 levely pre parametre *A*, *B* a *C*. V stĺpci *Najlepší experiment* sa nachádzajú najlepšie parametre jednej iterácie Taguchiho metódy pred analýzou a v stĺpci *Najlepší výsledok* je výsledok tohto nastavenia. V stĺpci *Najlepšia kombinácia* je predpokladaná najlepšia kombinácia dosiahnutá analýzou priemerného vplyvu faktorov, vo vedľajšom stĺpci *Reálny výsledok* je skutočná dosiahnutá hodnota overovacieho experimentu pre toto nastavenie a v poslednom stĺpci *Predpokladaný výsledok* je predpoklad Taguchiho metódy o výsledku pre najlepšiu kombináciu.

Tabuľka 21. Prvé a druhé hľadanie lokálneho optima

Veľkosť populácie	Veľkosť generácie	Pravdepodobnosť kríženia	Najlepší experiment	Najlepší výsledok	Najlepšia kombinácia	Reálny výsledok	Predpoklada -ný výsledok
<125, 150, 175>	<1125, 1250, 1375>	<0.45, 0.50, 0.55>	175 1375 0.5	5101	125 1375 0.5	5138,6	5144,7
<100, 125, 150>	<1250, 1375, 1500>	<0.48, 0.50, 0.53>	150 1500 0.5	5129.6	100 1500 0.5	5164,4	5166.8
<75, 100, 125>	<1375, 1500, 1625>	<0.49, 0.50, 0.51>	125 1625 0.5	5142.6	75 1625 0.5	5233.8	5215.4
<50, 75, 100>	<1500, 1625, 1750>	<0.49, 0.50, 0.51>	100 1750 0.5	5170.2	50 1750 0.49	5281.6	5317.8
<25, 50, 75>	<1625, 1750, 1875>	<0.49, 0.49, 0.50>	75 1875 0.49	5244.0	25 1875 0.48	5305.8	5321.7
<0, 25, 50>	<1750, 1875, 2000>	<0.48, 0.49, 0.49>	50 2000 0.48	5319.4	25 1750 0.4	5313.2	5319.4
<13, 25, 37>	<1625, 1750, 1875>	<0.49, 0.49, 0.50>	37 1875 0.49	5285.0	13 1750 0.5	5316.8	5333.5
<1, 13, 25>	<1688, 1750, 1812>	<0.49, 0.50, 0.51>	25 1812 0.5	5303.2	13 1688 0.49	5313.4	5332.7
<7, 13, 19>	<1626, 1688, 1750>	<0.49, 0.49, 0.50>	19 1750 0.49	5307.8	19 1626 0.5	5307.0	5328.5
<13, 19, 25>	<1564, 1626, 1688>	<0.49, 0.50, 0.51>	25 1688 0.5	5305.6	19 1564 0.49	5314.0	5322.5
<16, 19, 22>	<1502, 1564, 1626>	<0.49, 0.49, 0.50>	22 1626 0.49	5304.8	19 1502 0.48	5319.6	5321.4
<225, 250, 275>	<625, 750, 875>	<0.65, 0.70, 0.75>	275 875 0.7	4770.2	225 875 0.7	4818.4	4854.5
<200, 225, 250>	<750, 875, 1000>	<0.68, 0.70, 0.73>	250 1000 0.7	4844.8	200 1000 0.7	4873.4	4890.3
<175, 200, 225>	<875, 1000, 1125>	<0.69, 0.70, 0.71>	225 1125 0.7	4879.2	175 1125 0.7	4925.2	4919.4
<150, 175, 200>	<1000, 1125, 1250>	<0.69, 0.70, 0.71>	200 1250 0.7	4914.6	150 1250 0.7	4952.4	4978.2
<125, 150, 175>	<1125, 1250, 1375>	<0.70, 0.71, 0.71>	175 1375 0.7	4938.0	125 1375 0.7	5017.2	5025.5
<100, 125, 150>	<1250, 1375, 1500>	<0.69, 0.70, 0.71>	150 1500 0.7	4990.6	100 1500 0.7	5059.8	5061.2
<75, 100, 125>	<1375, 1500, 1625>	<0.70, 0.71, 0.71>	125 1625 0.7	5047.8	75 1625 0.7	5082.2	5101.6
<50, 75, 100>	<1500, 1625, 1750>	<0.70, 0.71, 0.71>	100 1750 0.7	5067.0	50 1750 0.7	5119.4	5126.7
<25, 50, 75>	<1625, 1750, 1875>	<0.70, 0.71, 0.71>	75 1875 0.7	5094.2	25 1875 0.7	5183.0	5188.6
<0, 25, 50>	<1750, 1875, 2000>	<0.71, 0.71, 0.71>	50 2000 0.7	5194.3	25 2000 0.7	5182.4	5194.3

V tabuľke si môžeme všimnúť, že hodnoty v stĺpci *Najlepšia kombinácia* sa stávajú základom pre nasledujúci interval parametra. Napríklad ak hodnoty parametra sú <125, 150, 175> a Taguchiho metóda predpokladá že prvý level s hodnotou 125 je najlepší, v nasledujúcom kroku bude hodnota 125 stredom intervalu a interval môže nadobudnúť hodnoty <100, 125, 150>. Ak by Taguchiho metóda predpokladala druhý level s hodnotou 150 ako najlepší, daný interval by bol menší a mohol by nadobúdať hodnoty <138, 150, 162>.

Vyhľadávanie prvého lokálneho optima skončilo, pričom algoritmus bol volaný 522-krát (11 iterácii * 9 experimentov jednej iterácie * 5 replikácii + 11 konfirmačných experimentov + 16 počiatočných). Následné hľadanie lokálneho optima začína s diverzifikovaným (najhorším) riešením v druhej polovici tabuľky 21. Hodnota počtu náhodne zmenených génov je aj v tomto lokálnom vyhľadávaní konštantná s hodnotou 5.

Najlepšie dosiahnuté riešenie s priemernou cenou batoha **5319,6** je v tabuľke 21 zvýraznené a bolo vypočítané s takýmito hodnotami parametrov:

A=19; B=1502; C=0,48; D=3

Optimálna hodnota batoha je **5462**.

Záver

Výsledkom diplomovej práce je implementácia Taguchiho metódy a Calibra algoritmu, úlohou ktorých je ladenie parametrov ľubovoľných algoritmov, bez potreby poznania ich fungovania. Taguchiho metóda je otestovaná na algoritme AdaBoost, pri ktorom dokáže úspešne minimalizovať výstup algoritmu. Calibra algoritmus je otestovaný na genetickom algoritme, ktorý rieši úlohu o batohu a úspešne dokáže maximalizovať výstup tohto algoritmu. Pričom Taguchiho metóda je vhodnejšia na jemné doladenie parametrov časovo náročných algoritmov, zatiaľ čo Calibra algoritmus je schopný odhaliť dobré nastavenie parametrov aj pre parametre obsahujúce rozsiahle intervaly hodnôt.

Vzhľadom na to, že v Taguchiho metóde funguje algoritmus, ktorému sú ladené parametre ako čierna skrinka, metódu je možné použiť na ladenie parametrov bez ohľadu na typ algoritmu alebo riešenej úlohy.

Zoznam použitej literatúry

[1] *AdaBoost* [online]. Dostupné na internete: https://en.wikipedia.org/wiki/AdaBoost [cit. 07.04.2018]

[2] Adenso-Díaz, B., Laguna, M. *Fine-Tuning of Algorithms Using Fractional Experimental Designs and Local Search*. Operations Research 54(1), 2006, pp. 99–114

[3] Buzna, Ľ., Janáček, J., Kohání, M., Szendreyová, A. *Diskrétna optimalizácia,* 2015, pp. 217

[4] Darshana, M., *Comparison of Feature Detection and Matching Approaches: SIFT and SURF* [online]. Dostupné na internete: https://www.researchgate.net/figure/Haar-wavelet filters-to-compute-the-responses-in-x-left-and-y-direction-right-The_fig1_314285930 [cit. 06.04.2018]

[5] Dobslaw, F. *Recent Development in Automatic Parameter Tuning for Metaheuristics.* In WDS'10 Proceedings of Contributed Papers, Part I, 2010, pp. 54-63

[6] *Genichi Taguchi* [online]. Dostupné na internete: https://en.wikipedia.org/wiki/Genichi _Taguchi [cit. 24.02.2018]

[7] *Haar training with multiboost* [online]. Dostupné na internete: https://groups.google .com/forum/#!topic/multiboost/B6tBC0Y0__0 [cit. 29.03.2018]

[8] Kacker, R. N., Lagergen, E. S., Filliben, J. J. *Taguchi's Orthogonal Arrays Are Classical Designs of Experiments*. Journal of Research of the National Institute of Standards and Technology 96(5), 1991, pp. 577-591

[9] Kramer, O., *Genetic Algorithm Essentials* pp. 11-17

[10] *Latin square* [online]. Dostupné na internete: https://en.wikipedia.org/wiki/Latin_ square [cit. 04.03.2018]

[11] Michalewicz, Z., Nazhiyat, G. *Genocop III: A Co-evolutionary Algorithm for Numerical Optimization Problem with Non Linear Constraints*. In Proceedings of the IEEE International Conference on Evolutionary Computation, 1995, pp. 647-651

[12] *MNIST database* [online]. Dostupné na internete: https://en.wikipedia.org/wiki/ MNIST_database [cit. 06.04.2018]

[13] *MultiBoost* [online]. Dostupné na internete: http://www.multiboost.org/ [30.03.2018]

[14] *MultiBoost config file* [online]. Dostupné na internete: https://www.lri.fr/~kegl/MNIST/Tree8Rsample100/config.txt [cit. 29.03.2018]

[15] Robinson, D., *Exploring handwritten digit classification: a tidy analysis of the MNIST dataset* [online]. Dostupné na internete: http://varianceexplained.org/r/digit-eda/ [cit. 29.03.2018]

[16] *Software Testing: Applying Taguchi Methods using Orthogonal Arrays* [online]. Dostupné na internete: http://www.theprofessionaldigest.com/search/label/biotechnology%20and%2“0marketing.%20Orthogonal%20Array [cit. 24.02.2018]

[17] Svoboda, P., *Vyhledávání osob ve fotografii* [online]. Dostupné na internete: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=116845[cit. 07.04.2018]

[18] Šochman, J., Matas J., *AdaBoost* [online]. Dostupné na internete: http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf [cit. 07.04.2018]

Zoznam príloh

Prílohy

Príloha A: Obsah DVD

Priložené DVD obsahuje:

- Práca v elektronickej podobe (formát PDF)
- Aplikácia obsahujúca Taguchiho metódu aj Calibra algoritmus
- Algoritmus AdaBoost spolu s datasetom Mnist, konfiguračnými súbormi a výsledkami experimentov
- Genetický algoritmus, ktorý rieši úlohu o batohu ako samostatný program