

# **Implementace vybraných evolučních algoritmů v prostředí .NET**

Implementation of Evolutionary Algorithms in .NET Framework

Jakub Krامل

---

Bakalářská práce  
2011



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Jakub KRAMPL**  
Osobní číslo: **A08059**  
Studijní program: **B 3902 Inženýrská informatika**  
Studijní obor: **Informační a řídicí technologie**

Téma práce: **Implementace vybraných evolučních algoritmů v prostředí .NET**

Zásady pro vypracování:

- 1. Zpracujte literární rešerši na téma evoluční algoritmy.**
- 2. Popište prostředí .NET.**
- 3. Analyzujte možnosti implementace v prostředí .NET.**
- 4. Navrhněte vlastní implementaci.**
- 5. Demonstrujte výsledky na ukázkové aplikaci.**

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. ZELINKA, Ivan. Umělá inteligence v problémech globální optimalizace. Praha : BEN, 2002. 192 s. ISBN 80-7300-069-5.
2. CLERC, Maurice. Particle swarm optimization. London : ISTE, 2006. 243 s. ISBN 1-905209-04-5.
3. PRICE, Kenneth V.; STORN, Rainer M.; LAMPINEN, Jouni A. Differential evolution: a practical approach to global optimization. Berlin : Springer, 2005. 538 s. ISBN 3-540-20950-6.
4. KVASNIČKA, V.; POSPÍCHAL, J.; TIŇO, P. Evoluční algoritmy. Bratislava : STU, 2000. 215 s. ISBN 80-227-1377-5.
5. ZELINKA, Ivan, et al. Evoluční výpočetní techniky. Praha : BEN, 2008. 534 s. ISBN 978-80-7300-218-3.

Vedoucí bakalářské práce:

**Ing. Erik Král**

Ústav bezpečnostního inženýrství

Datum zadání bakalářské práce:

**25. února 2011**

Termín odevzdání bakalářské práce:

**7. června 2011**

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



prof. Ing. Vladimír Vašek, CSc.  
*ředitel ústavu*

## **ABSTRAKT**

Práce nabízí pohled na vybrané evoluční algoritmy, konkrétně Samo-Organizující se Migrační Algoritmus (SOMA), Diferenciální evoluci (DE) a Rojení částic (PSO). Popisuje jednotlivé parametry, strategie a ukončovací kritéria algoritmů. V praktické části je popsána implementace zmíněných algoritmů prostřednictvím .NET Framework s grafickým uživatelským rozhraním vytvořeným pomocí Windows Presentation Foundation (WPF).

Klíčová slova: optimalizace, evoluční algoritmy, účelová funkce, SOMA, DE, PSO, .NET Framework, WPF

## **ABSTRACT**

Thesis provides an overview of selected evolutionary algorithms, specifically Self-Organizing Migrating Algorithm (SOMA), Differential Evolution (DE) and Particle Swarm Optimization (PSO). It describes individual parameters, strategies and stopping criteria. The practical part describes the implementation of these algorithms through .NET Framework with graphical user interface created using Windows Presentation Foundation (WPF).

Keywords: optimization, evolutionary algorithms, cost function, SOMA, DE, PSO, .NET Framework, WPF

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Eriku Královi především za seznámení s evolučními algoritmy a také za cenné rady a postřehy při tvorbě samotné aplikace.

**Prohlašuji, že**

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

**Prohlašuji,**

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

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD .....</b>	<b>10</b>
<b>I TEORETICKÁ ČÁST .....</b>	<b>11</b>
<b>1 EVOLUČNÍ ALGORITMY .....</b>	<b>12</b>
1.1 POPULACE.....	12
1.2 SPECIMEN .....	12
<b>2 SOMA: SAMO-ORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS .....</b>	<b>13</b>
2.1 PARAMETRY A TERMINOLOGIE.....	13
2.2 STRATEGIE.....	17
2.2.1 AllToOne .....	17
2.2.2 AllToAll.....	18
2.2.3 AllToAllAdaptive.....	19
2.2.4 AllToOneRand .....	20
2.2.5 Clusters .....	20
2.3 ZÁVISLOST NA ŘÍDICÍCH A UKONČOVACÍCH PARAMETRECH .....	20
<b>3 DE: DIFERENCIÁLNÍ EVOLUCE.....</b>	<b>22</b>
3.1 PARAMETRY A TERMINOLOGIE.....	22
3.2 KŘÍŽENÍ.....	23
3.2.1 Binomiální.....	23
3.2.2 Exponenciální.....	24
3.3 VARIANTY DIFERENCIÁLNÍ EVOLUCE .....	24
3.4 PRINCIP DIFERENCIÁLNÍ EVOLUCE.....	26
3.5 ZÁVISLOST NA UKONČOVACÍCH PARAMETRECH.....	26
<b>4 PSO: ROJENÍ ČÁSTIC.....</b>	<b>28</b>
4.1 PARAMETRY A TERMINOLOGIE.....	28
4.2 PRINCIP PSO.....	30
4.3 ZÁVISLOST NA UKONČOVACÍCH PARAMETRECH.....	31
<b>5 UKONČOVACÍ KRITÉRIA .....</b>	<b>32</b>
5.1 MAXDIST (QUICK) .....	32
5.2 STDDEV.....	32
5.3 MINDIV + MAXDIST (QUICK).....	33
<b>6 .NET FRAMEWORK .....</b>	<b>34</b>
<b>II PRAKTICKÁ ČÁST .....</b>	<b>35</b>
<b>7 POPIS IMPLEMENTACE ZÁKLADNÍCH TŘÍD .....</b>	<b>36</b>

7.1	ABSTRAKTNÍ TŘÍDA ALGORITHM.....	36
7.2	SOMA.....	37
7.3	DE .....	38
7.4	PSO.....	38
7.5	SPECIMEN .....	39
7.6	STOPPINGCRITERIA .....	40
7.7	STATS .....	41
7.8	XML .....	41
7.8.1	Import parametrů.....	41
7.8.2	DTD definice.....	42
7.8.3	Export výsledků.....	42
7.9	CSV .....	43
7.9.1	Import parametrů.....	43
7.9.2	Export výsledků.....	43
7.10	LOOP .....	43
7.11	GRAPH.....	44
7.12	VALIDAČNÍ PRAVIDLA .....	45
7.13	COMMANDS .....	45
7.14	GRIDVIEWSORTER .....	46
7.15	CF .....	47
7.16	CFLIST.....	47
<b>8</b>	<b>GRAFICKÉ ROZHRAŇÍ.....</b>	<b>48</b>
8.1	HLAVNÍ OKNO .....	48
8.1.1	Menu .....	48
8.1.2	Panel nástrojů .....	49
8.1.3	Obsah .....	50
8.1.4	StatusBar .....	51
8.2	VÝSLEDKOVÉ OKNO .....	52
8.3	EXPORT VÝSLEDKŮ .....	52
8.4	VZDÁLENOST OD EXTRÉMU .....	53
<b>9</b>	<b>POPIS OVLÁDÁNÍ .....</b>	<b>54</b>
9.1	MENU .....	54
9.2	PANEL NÁSTROJŮ .....	54
9.3	ALGORITMY A STRATEGIE.....	55
9.4	UKONČOVACÍ KRITÉRIA.....	55
9.5	SPECIMEN .....	55
<b>10</b>	<b>VLASTNÍ ÚČELOVÁ FUNKCE .....</b>	<b>57</b>
<b>11</b>	<b>VÝSLEDKY TESTŮ.....</b>	<b>59</b>
	<b>ZÁVĚR.....</b>	<b>61</b>
	<b>ZÁVĚR V ANGLIČTINĚ.....</b>	<b>62</b>
	<b>SEZNAM POUŽITÉ LITERATURY .....</b>	<b>63</b>
	<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....</b>	<b>65</b>

<b>SEZNAM OBRÁZKŮ</b> .....	<b>66</b>
<b>SEZNAM TABULEK</b> .....	<b>67</b>
<b>SEZNAM PŘÍLOH</b> .....	<b>68</b>

## ÚVOD

Evoluční algoritmy jsou vhodným nástrojem pro řešení a optimalizaci funkcí s více extrémy. Jak už název napovídá, jsou inspirovány mechanismem evoluce, a tak se pomocí křížení, mutačních procesů a dalších metod snaží postupně vyvinout co nejlepší řešení.

Hlavní předností evolučních algoritmů je schopnost řešit i velmi složité optimalizační problémy, u kterých se zaměřují na hledání globálních extrémů a nikoliv lokálních, jako je tomu u numerických metod. Mezi nevýhody lze řadit jistou závislost na náhodě, a tedy nemůžeme dopředu předvídat, zdali nalezené řešení je nejlepší možné. Proto je vhodné vyjít ze zkušenosti s těmito algoritmy.

Práce se zabývá třemi vybranými algoritmy: Samo-Organizující se Migrační Algoritmus (SOMA), Diferenciální evoluce (DE) a Rojení částic (PSO). Popsány jsou jak parametry jednotlivých algoritmů, tak i možné strategie, podle kterých se řídí a usměrňuje činnost optimalizačního běhu.

Dále jsou zmíněna některá ukončovací kritéria, jimiž lze ukončit běh evolučních algoritmů, jako jeden z flexibilních způsobů přístupu k vyhodnocení stavu aktuálních řešení.

Cílem je vytvořit aplikaci prostřednictvím .NET Framework 4.0 s grafickým, uživatelsky přívětivým, rozhraním vytvořeným pomocí Windows Presentation Foundation. Aplikace by měla nabídnout uživateli plnou kontrolu nad nastavením parametrů algoritmů a především možnost dynamicky načíst svou vlastní účelovou funkci. Dále umožnit výběr některého z předpřipravených ukončovacích kritérií, u kterých lze do jisté míry ovlivnit jejich chování. V neposlední řadě zobrazit vhodnou formou výsledky optimalizace a dovolit, v případě potřeby, jejich export do souboru.

## **I. TEORETICKÁ ČÁST**

## 1 EVOLUČNÍ ALGORITMY

Optimalizační algoritmy jsou velmi silným a mocným nástrojem k hledání optimálního řešení, kde analytická cesta je leckdy velmi složitá a někdy až nereálná.

Většinu problémů inženýrské praxe lze převést na matematický problém popsany odpovídajícím funkčním předpisem, jehož parametry se snažíme optimalizovat vzhledem k hodnotě účelové funkce.

V posledních dvou desetiletích se vyvinula skupina velmi výkonných algoritmů tzv. evoluční algoritmy, které hledají nejlepší možné řešení prostřednictvím populace jedinců v cyklech – generacích, podle daných pravidel. [1][2]

### 1.1 Populace

Populace je základním stavebním kamenem všech evolučních algoritmů a její členové se vytváří podle definovaného vzoru – Specimen. Můžeme si ji představit jako matici počet jedinců  $\times$  počet parametrů. Každý jedinec pak představuje možné řešení definovaného problému, ke kterému je navíc přiřazena hodnota účelové funkce, což představuje kvalitu příslušného jedince. [1]

	Jedinec 1	Jedinec 2	Jedinec 3	Jedinec 4	Jedinec 5
CV	0,91	-2,23	-1,97	-0,41	2,27
Parametr 1	1,5	0,17	-1,77	0,52	-1,69
Parametr 2	1,85	-2,33	-0,97	-0,08	2,42
Parametr 3	-2,44	-0,07	0,77	-0,85	1,54

Obr. 1. Znárodnění populace

### 1.2 Specimen

Specimen je vzorový jedinec, podle něhož se generuje počáteční populace. Pro každý parametr jedince má definován typ proměnné (reálný, celočíselný, diskretní, apod.) a rozmezí hodnot, ve kterých se může pohybovat. Při nesprávně zvoleném rozsahu můžeme získat neopodstatněná nebo fyzikálně nerealizovatelná řešení.

Počáteční populace jedinců je pak vytvořena na základě vztahu (1), kde  $rnd$  je náhodné číslo z intervalu  $<0; 1>$ ,  $i$  označuje jedince a  $j$  jeho parametr,  $Hi$  a  $Lo$  odpovídají hornímu a dolnímu limitu pro daný parametr. [1]

$$x_{i,j} = rnd \cdot (x_{i,j}^{Hi} - x_{i,j}^{Lo}) + x_{i,j}^{Lo} \quad (1)$$

## 2 SOMA: SAMO-ORGANIZUJÍCÍ SE MIGRAČNÍ ALGORITMUS

SOMA (Self-Organizing Migrating Algorithm) je algoritmus pocházející z roku 1999 a řadí se mezi algoritmy evoluční, ačkoliv během svých evolučních cyklů (migračních kol) nevytváří nové potomky. Vhodnější zařazení je však mezi algoritmy memetické, které jsou charakterizovány jako strategie soutěže a spolupráce.

Princip SOMA se dá interpretovat jako chování skupiny inteligentních jedinců, kteří spolupracují na řešení společného problému, např. hledání zdroje potravy. Je tedy založen na kooperativním prohledávání ohraničeného prostoru.

Samo-organizující činnost SOMA vyplývá ze vzájemného ovlivňování se jedinců z populace při určení směru, kterým se vydají v prohledávaném prostoru při hledání lepšího řešení. [1]

### 2.1 Parametry a terminologie

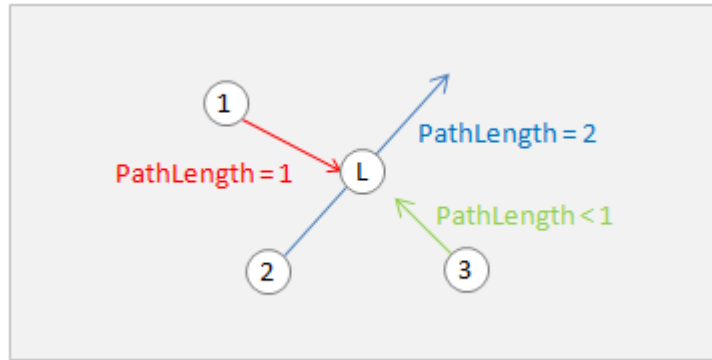
Parametry jsou velmi důležitým faktorem ovlivňující činnost a kvalitu celého algoritmu, proto je nutné věnovat jejich nastavení velkou pozornost.

#### **PathLength**

Parametr určující vzdálenost, ve které se od vedoucího jedince (leadera) zastaví aktivní jedinec. Pokud bude  $\text{PathLength} = 1$  a Step bude dělit PathLength celočíselně, zastaví se aktivní jedinec na pozici vedoucího jedince. Při  $\text{PathLength} = 2$  se zastaví až za vedoucím ve vzdálenosti, ze které startoval. V případě  $\text{PathLength} < 1$  skončí jeho pohyb ještě před jedincem, ke kterému migruje, což vede k degeneraci migračního procesu.

Maximum pro tento parametr není nijak omezeno, avšak testováním a zkušenostmi s tímto algoritmem byla zjištěna jako dostačující hodnota 3.

Jednotlivé vzdálenosti PathLength ilustruje obrázek (Obr. 2). [1]



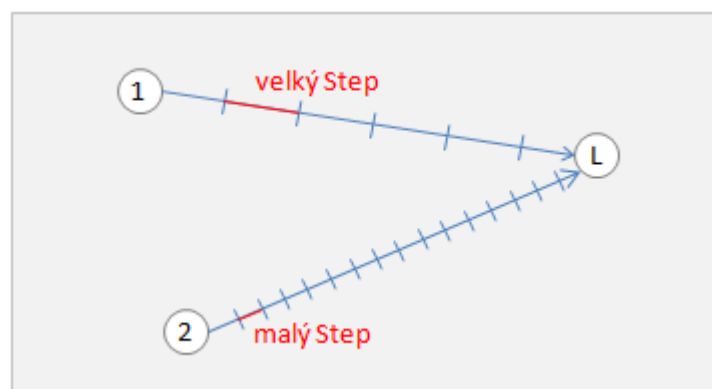
Obr. 2. Hodnoty parametru PathLength

### Step

Step určuje velikost kroku, se kterým aktivní jedinec migruje k vedoucímu jedinci. Jeho hodnota je volena v rozmezí  $\langle 0,11; PathLength \rangle$ .

Na obrázku (Obr. 3) je znázorněna malá a velká hodnota parametru Step na definované cestě směrem k vedoucímu jedinci. Je zde vidět patrný rozdíl v důslednosti prohledávání dané oblasti. Pro malou hodnotu Step je oblast velmi důkladně prohledávána a je tak daleko větší šance na nalezení globálního extrému. Avšak i větší krok má své uplatnění např. u jednoduchých účelových funkcí, kde tímto můžeme urychlit nalezení extrému.

Je také nutné zajistit, aby vzdálenost mezi vedoucím a aktivním jedincem (PathLength) nebyla celočíselným násobkem parametru Step. Pokud by k tomu došlo, každý jedinec by mohl být finálně přitažen vedoucím jedincem a proces migrace by tak mohl skončit v lokálním extrému. [1]



Obr. 3. Migrace jedince k leaderovi po malém a velkém kroku Step

## PRT

Jeden z nejdůležitějších a nejcitlivější parametrů značící pertubaci. Pohybuje se v intervalu  $\langle 0; 1 \rangle$ . Na základě jeho hodnoty se generuje pertubační vektor (PRTVector), který udává, jakým směrem se bude pohybovat aktivní jedinec.

Optimální hodnota se pohybuje kolem 0,1, avšak optimum může záviset jak na počtu parametrů účelové funkce, tak na počtu jedinců v populaci. Čím více se bude hodnota PRT blížit k jedničce, tím více bude algoritmus náchylný ke konvergenci k lokálním extrémům. [1]

## Dimenze

Dimenze problému neboli počet parametrů (argumentů) účelové funkce, které se snaží algoritmus optimalizovat na co nejlepší hodnotu vzhledem k účelové funkci. [1]

Obecná formulace účelové funkce:

$$f(x_1, x_2, \dots, x_n) \quad (2)$$

Ukázka dvourozměrné účelové funkce – Rosenbrockovo sedlo:

$$f(x_1, x_2) = 100 \cdot (x_1^2 - x_2)^2 + (1 - x_1)^2 \quad (3)$$

## PopSize

Parametr udávající počet jedinců v populaci, kteří se účastní migračních kol, za účelem nalezení globálního extrému. Teoreticky by populace mohla obsazovat pouze dva jedince, avšak evoluční algoritmus by se pak dal přirovnat ke klasickým deterministickým metodám. Z tohoto důvodu se volí jako minimální počet 10 jedinců.

V případě, že má účelová funkce velký počet parametrů, PopSize obvykle volíme jako 0,2 až 0,5 násobek parametru D. U jednoduchých funkcí můžeme volit malý počet jedinců. [1]

## Migrations

Udává počet migračních kol, ve kterých dochází k reorganizaci – obrození celé populace, za účelem nalezení lepších jedinců.

Za minimální počet se považuje 10 migrací, horní mez není nijak omezena a záleží pouze na složitosti daného problému.

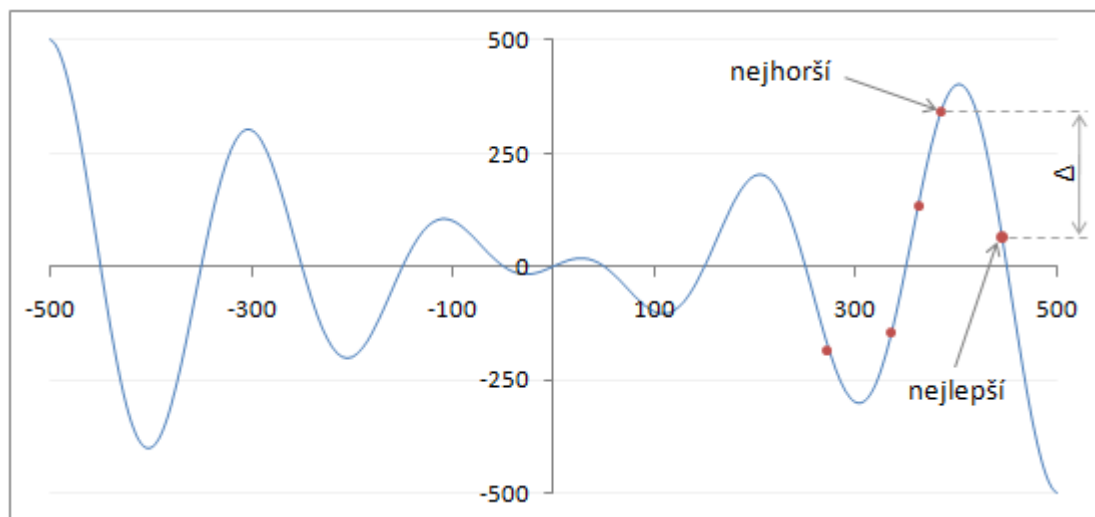
Migrations patří mezi ukončovací parametry algoritmu. [1]

### MinDiv

Uživatelé definovaný ukončovací parametr, který udává maximální přípustný rozdíl mezi hodnotou účelové funkce nejlepšího a nejhoršího jedince v aktuální populaci.

Vzhledem k tomu, že se jedná o ukončovací podmínku, tak při jejím splnění dojde k ukončení migrace, což při nevhodném zvolení (vysoká hodnota) může vést k předčasnému zastavení algoritmu, aniž by se dosáhlo globálního extrému. V případě nastavení na nulovou nebo malou hodnotu se v nejhorším případě provedou všechna migrační kola.

Pokud se nechceme zabývat nastavením hodnoty MinDiv, volíme ji jako zápornou.



Obr. 4. Princip parametru MinDiv

Obrázek (Obr. 4) znázorňuje princip činnosti parametru MinDiv, kde červené body jsou jedinci z populace a  $\Delta$  znázorňuje rozdíl mezi nejlepším a nejhorším jedincem z pohledu hodnoty účelové funkce. Jakmile bude hodnota  $\Delta$  menší jak MinDiv, dojde ke splnění ukončovací podmínky a migrace se ukončí. [1]

## Souhrn parametrů

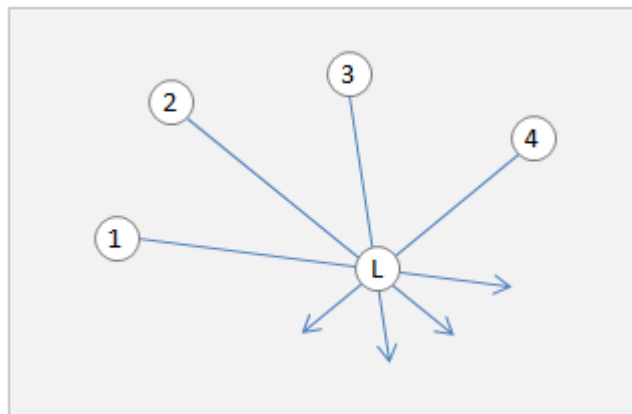
Tab. 1. Hodnoty parametrů SOMA [1]

Parametr	Doporučený rozsah
PathLength	<1,1; 5>
Step	<0,11; PathLength>
PRT	<0; 1>
D	dáno problémem
PopSize	<10; definuje uživatel>
Migrations	<10; definuje uživatel>
MinDiv	libovolně

## 2.2 Strategie

### 2.2.1 AllToOne

Strategie AllToOne se vyznačuje specifickým vedoucím jedincem – leaderem, ke kterému všichni jedinci z populace migrují.



Obr. 5. Strategie AllToOne

Po vytvoření počáteční populace je každý jedinec ohodnocen účelovou funkcí a podle její hodnoty je určen vedoucí – leader pro následující migrační kolo. Ostatní členové populace se následně začnou postupně pohybovat směrem k leaderovi po skocích následujícím způsobem:

1. Vygeneruje se prázdný pertubační vektor (PRTVector) o dimenzi  $D$  a sekvence náhodných čísel o počtu  $D$  z intervalu  $<0; 1>$ . Pokud je  $n$ -té náhodné číslo menší než hodnota parametru PRT, vloží si na  $n$ -tou pozici v pertubačním vektoru 1, v opačném případě 0.

2. Určí se nová pozice jedince dle vztahu (4), kde  $i$  určuje migrujícího jedince a  $j$  značí parametry,  $x_{i,j,start}$  odpovídá startovací pozici aktivního jedince,  $x_{L,j}$  je pozice leadera,  $t$  je vzdálenost, kterou jedinec doposud urazil na své cestě směrem k leaderovi (5) a  $PRTVector_j$  je parametr pertubačního vektoru.

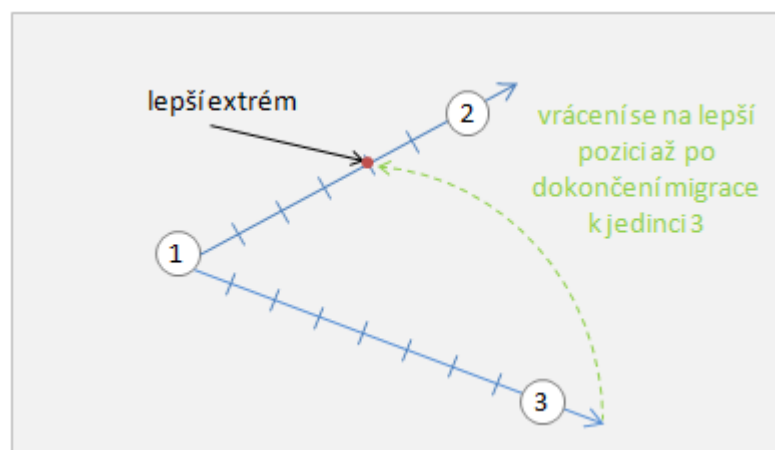
$$x_{i,j} = x_{i,j,start} + (x_{L,j} - x_{i,j,start}) \cdot t \cdot PRTVector_j \quad (4)$$

$$t \in \langle 0, Step, PathLength \rangle \quad (5)$$

3. Nová pozice se ohodnotí účelovou funkcí, a pokud je lepší než dosavadní, jedinec si pozici zapamatuje.
4. Následně putuje dále po skocích o velikosti Step, než dosáhne vzdálenosti PathLength. Potom se vrátí zpět na pozici, která byla zjištěna jako nejlepší z pohledu hodnoty účelové funkce.
5. Po dokončení migračního kola jsou tak všichni jedinci přemístěni (kromě leadera) a po zvolení nového vedoucího započne další cyklus migrací. Celý postup ilustruje obrázek (Obr. 9). [1]

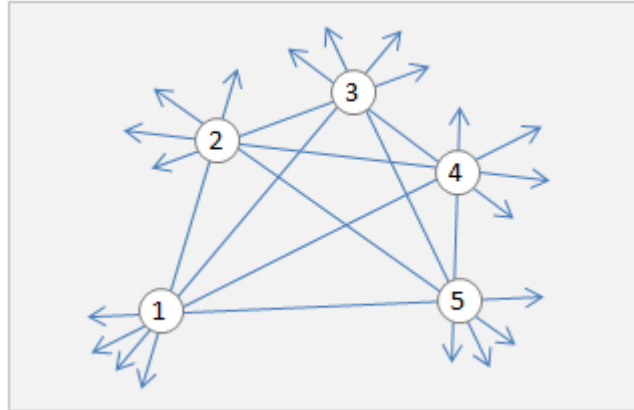
### 2.2.2 AllToAll

AllToAll postrádá vedoucího jedince. Jedinci z populace tedy nemigrují k nejlepšímu, ale postupně putují ke všem ostatním. Pokud při svém putování naleznou lepší extrém, dokončí migraci ke všem ostatním jedincům a teprve po té se vrátí zpět na pozici, kde byl nalezen nejlepší extrém a z této nové pozice v dalším migračním kole vychází (Obr. 6).



Obr. 6. AllToAll – migrace jedince

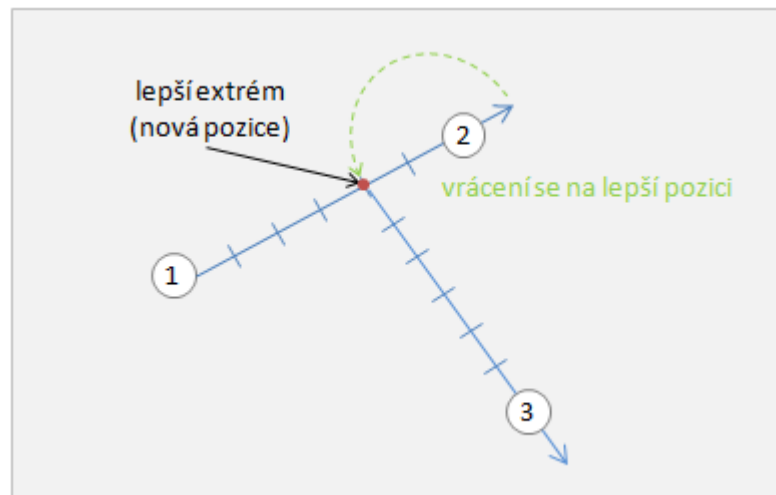
Při srovnání (Obr. 7) s AllToOne (Obr. 5) vidíme, že je zde prohledáváno daleko více prostoru, což zvyšuje pravděpodobnost nalezení globálního extrému, avšak za cenu větší časové náročnosti. [1]



Obr. 7. Strategie AllToAll

### 2.2.3 AllToAllAdaptive

Jedná se téměř o totožnou strategii s AllToAll. Rozdíl je pouze v místě návratu na lepší nalezenou pozici. Pokud tedy při svém putování naleznou lepší extrém, dokončí migraci k jedinci, ke kterému právě migrují a po té se vrátí zpět na pozici, kde byl nalezen lepší extrém a z této nové pozice v dalším migračním kole vychází (Obr. 8). [1]



Obr. 8. AllToAllAdaptive – migrace jedince

### 2.2.4 AllToOneRand

Strategie založená na principu migrace jedinců k leaderovi, kterým však není jedinec s nejlepší hodnotou z pohledu účelové funkce, ale náhodně zvolený jedinec z celé populace. [1]

### 2.2.5 Clusters

Varianta, která se dá aplikovat na kteroukoliv z předchozích strategií. Jedná se o uspořádání jedinců z populace do svazků, ve kterých probíhá samotný algoritmus SOMA. Svazky se mohou během migrací jedinců slučovat nebo rozpadat.

Kdo do jakého svazku patří, je dáno vztahem (6), kde  $IND_i$  je parametr jedince,  $CC_i$  je parametr leadera,  $HB_i$  a  $LB_i$  jsou hranice jednotlivých parametrů účelové funkce a  $CD$  je velikost svazku daná uživatelem.

V případě, že se některý z jedinců ocitne daleko od ostatních a není možné jej zařadit do existujících svazků, stává se sám o sobě svazkem a migruje ke všem ostatním jako ve strategii AllToAll. [1]

$$CD > \sqrt{\sum_{i=1}^{dim} \left( \frac{IND_i - CC_i}{HB_i - LB_i} \right)^2} \quad (6)$$

## 2.3 Závislost na řídicích a ukončovacích parametrech

Činnost algoritmu SOMA je ukončena po provedení všech migračních kol (parametr Migrations), nebo v případě dosažení maximálního přípustného rozdílu účelových funkcí nejhoršího a nejlepšího jedince z populace (parametr MinDiv).

Pravděpodobnost nalezení globálního extrému je popsána vztahem (7), kde  $N_{eval}$  je počet ohodnocení účelové funkce a  $L^n$  je počet možných hodnot účelové funkce pro  $n$  rozměrnou funkci.

$$P_{GE} = \frac{N_{eval}}{L^n} \quad (7)$$

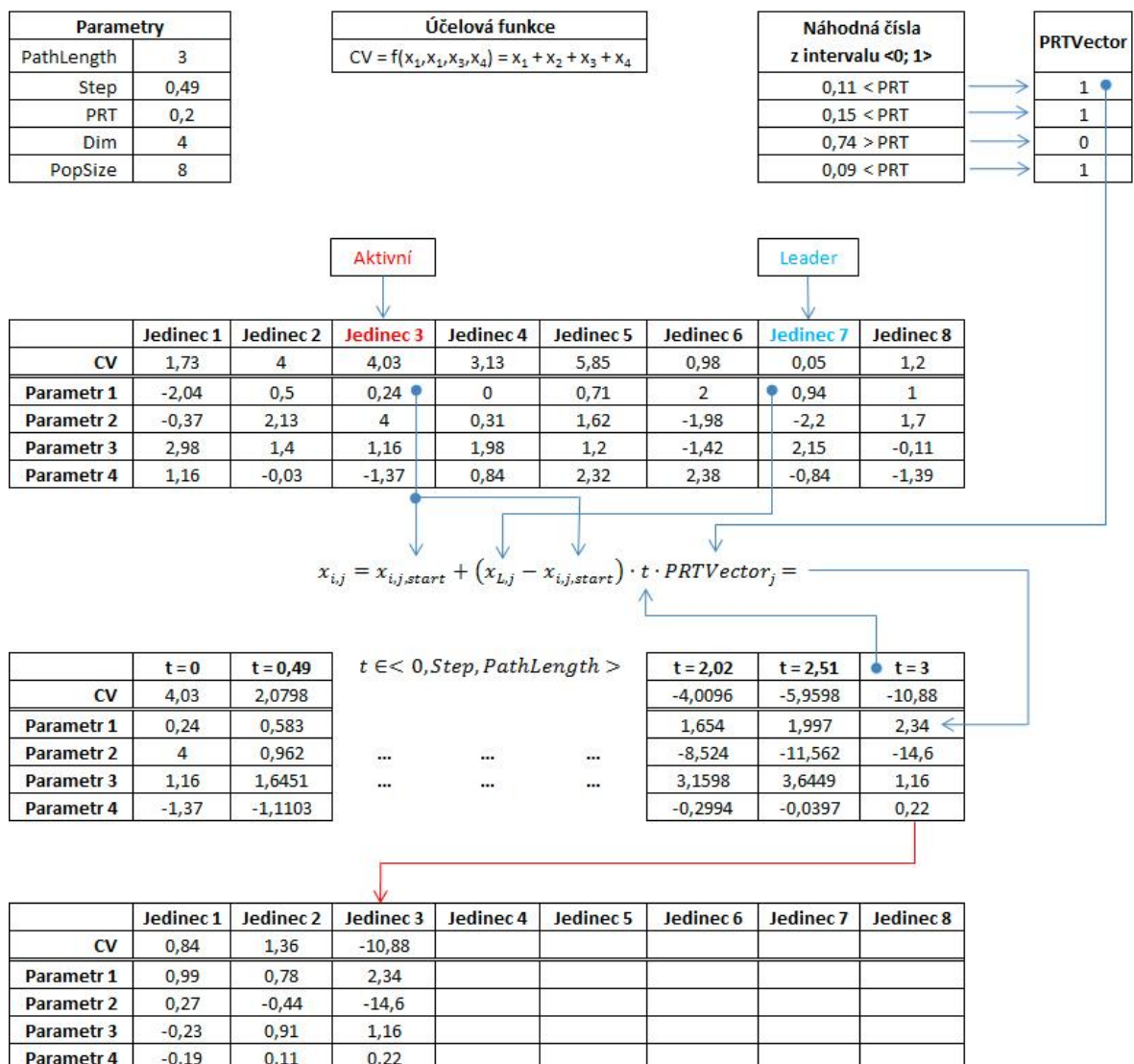
Pro variantu AllToOne, kdy jedinci z populace (kromě vedoucího) migrují k leaderovi a na své cestě provedou PathLength/Step skoků pro všechna migrační kola, je počet ohodnocení účelové funkce popsán vztahem (8).

$$N_{eval} = \frac{(PopSize - 1) \cdot PathLength \cdot Migrations}{Step} \tag{8}$$

U strategie AllToAll se pravděpodobnost zvyšuje s počtem jedinců v populaci (9).

$$N_{eval} = \frac{PopSize \cdot (PopSize - 1) \cdot PathLength \cdot Migrations}{Step} \tag{9}$$

Ze vztahů pro výpočet počtu ohodnocení účelové funkce (8) a (9) je zřejmé, že pokud zvětšíme PathLength, zvýšíme počet migračních kol, anebo snížíme Step, zvýšíme tím pravděpodobnost nalezení extrému. [1]



Obr. 9. Princip strategie AllToOne

### 3 DE: DIFERENCIÁLNÍ EVOLUCE

DE (Differential Evolution) je evoluční algoritmus, jehož vznik se datuje rokem 1995. Má podobné rysy jako genetické algoritmy, a to tvorbu potomků nebo využití generací. U tvorby nových potomků však nevyužívá dvou potomků, jak je tomu u genetických algoritmů, ale k vytvoření nového jedince využívá kombinaci čtyř dalších.

Principem vychází z tzv. genetického žhání, které se postupem času upravilo změnou reprezentace binární do dekadické a logických operací na vektorové. Těmito úpravami se dosáhlo algoritmu vhodného pro numerickou optimalizaci. Diferenciální evoluce pak ještě byla vylepšena objevením tzv. diferenciální mutace. [1]

#### 3.1 Parametry a terminologie

##### Dimenze

Dimenze, udává počet parametrů (optimalizovaných proměnných) účelové funkce. Závisí na definovaném problému, jenž je popsán účelovou funkcí a změnu lze provést pouze předefinováním celého problému. [1]

##### PopSize

Velikost populace – počet všech jedinců, kteří se účastní evolučního cyklu za účelem nalezení nejlepšího jedince. Platí zde omezení, aby algoritmus pracoval, nesmí být počet jedinců menší jak 4. Toto omezení vyplývá z principu diferenciální evoluce, kdy při vytváření nového jedince využíváme tří rodičů k vytvoření zkušebního vektoru v kombinaci se čtvrtým aktivním jedincem (Obr. 12).

V případě, že zvolíme malý počet jedinců, nemusíme dosáhnout dostatečně kvalitního výsledku, avšak při hodně velké populaci se značně prodlouží doba potřebná k vyšlechtění populace nové. [1]

##### Mutační konstanta – F

Mutační konstanta je důležitým řídicím parametrem při šlechtění jedince. Především pak ovlivňuje výpočet šumového vektoru, kde svou hodnotou určuje jak moc a kolik rodičů se bude podílet na vytvoření nového jedince.

Hodnotu mutační konstanty volíme z intervalu  $\langle 0; 2 \rangle$ .

U varianty DE/rand/1/exp (11) a při zvolení  $F = 0$ , bude nový potomek tvořen pouze jedním rodičem. Při  $F = 1$  vzniká riziko generování stejných jedinců, což vede ke stagnaci celé populace. [1]

### Práh křížení – CR

Práh křížení je dalším důležitým parametrem, tentokrát však při křížení, kdy na základě jeho hodnoty se kříží aktivní jedinec (dosud nepoužitý čtvrtý rodič) se šumovým zmutovaným vektorem a vznikne tak nový zkušební vektor.

Hodnotu prahu křížení volíme z intervalu  $\langle 0; 1 \rangle$ , avšak při zvolení  $CR = 0$  nedochází ke křížení, ale získáváme kopii aktivního jedince. Naopak při  $CR = 1$  bude nový jedinec tvořen pouze šumovým vektorem, a tak se na nově vzniklém jedinci nebude podílet čtvrtý rodič (aktivní jedinec), a bude tvořen pouze mutací tří náhodně zvolených rodičů. [1]

### Generations

Udává počet evolučních cyklů, ve kterých dochází k postupnému šlechtění celé populace. Přesný počet cyklů nelze jednoznačně říci, protože se liší v závislost na nastavení parametrů diferenciální evoluce, předepsané účelové funkci a jejím počtu argumentů a také na zkušenosti s diferenciální evolucí. [1]

### Souhrn parametrů

Tab. 2. Hodnoty parametrů diferenciální evoluce [1]

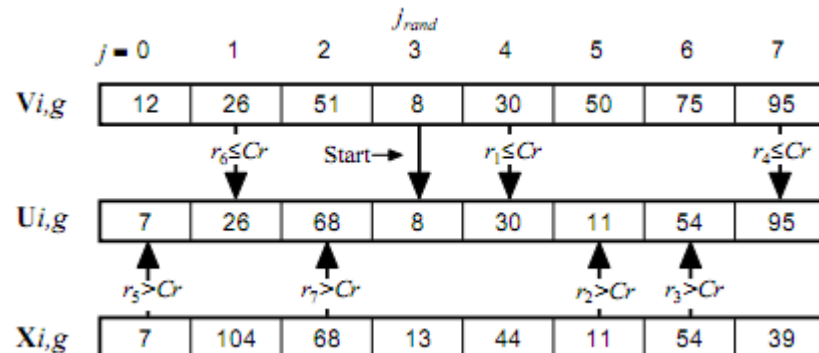
Parametr	Interval	Optimum
F	$\langle 0; 2 \rangle$	$\langle 0,3; 0,9 \rangle$
CR	$\langle 0; 1 \rangle$	$\langle 0,8; 0,9 \rangle$
D	dáno problémem	-
PopSize	$\langle 2D; 100D \rangle$	10D
Generations	definuje uživatel	-

## 3.2 Křížení

### 3.2.1 Binomiální

Náhodně se vybere jeden prvek  $j_{rand}$  z šumového vektoru  $V_{i,g}$ , který se přesune do vektoru zkušebního  $U_{i,g}$  a dále pak pro každou dvojici parametrů aktivního (cílového) jedince  $X_{i,g}$  a šumového vektoru se generuje náhodné číslo  $r_j$  z intervalu  $\langle 0; 1 \rangle$ . Pokud je větší jak CR,

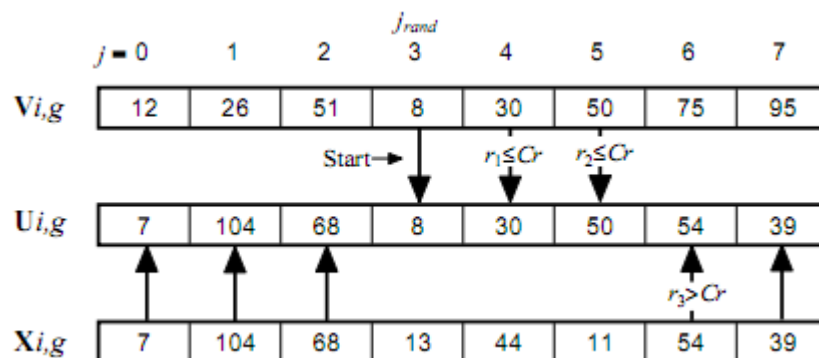
tak se do zkušebního vektoru na příslušnou pozici vloží hodnota aktivního jedince, v opačném případě se vezme parametr ze šumového vektoru. [1][3]



Obr. 10. Princip binomiálního křížení [3]

### 3.2.2 Exponenciální

Nejdříve se náhodně zvolí prvek  $j_{rand}$  z šumového vektoru  $V_{i,g}$ , jenž se přesune na odpovídající pozici vektoru zkušebního  $U_{i,g}$ . Pro každý prvek vpravo od něj (v kruhovém smyslu) se generuje náhodné číslo  $r_j$  z intervalu  $\langle 0; 1 \rangle$ , které se porovnává s CR. Dokud je náhodné číslo menší jak CR, tak se do zkušebního vektoru přesouvá prvek z šumového vektoru. Jakmile je  $r_j > CR$  jsou všechny zbývající volné pozice ve zkušebním vektoru zaplněny prvky z vektoru aktivního. [3]



Obr. 11. Princip exponenciálního křížení [3]

### 3.3 Varianty diferenciální evoluce

Diferenciální evoluce nabízí několik variant, jejichž odlišnost tkví především ve výpočtu šumového vektoru a následném sestaví vektoru zkušebního.

Značení jednotlivých variant je uváděno ve tvaru: DE/x/y/z. DE označuje diferenciální evoluci,  $x$  reprezentuje metodu výběru rodiče, který se použije jako základní vektor.

Např. DE/rand/y/z vybere základní vektor náhodně z celé populace, DE/best/y/z vybere nejlepší vektor. y udává počet různých vektorů, jenž pertubují základní vektor, z je typ použitého křížení (exp – exponenciální, bin – binomiální). [4][5]

**DE/best/1/exp**

$$v_j = x_{best,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (10)$$

**DE/rand/1/exp**

$$v_j = x_{r1,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (11)$$

**DE/rand-to-best/1/exp**

$$v_j = x_{i,j}^G + F \cdot (x_{best,j}^G - x_{i,j}^G) + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (12)$$

**DE/best/2/exp**

$$v_j = x_{best,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (13)$$

**DE/rand/2/exp**

$$v_j = x_{r5,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (14)$$

**DE/best/1/bin**

$$v_j = x_{best,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (15)$$

**DE/rand/1/bin**

$$v_j = x_{r1,j}^G + F \cdot (x_{r2,j}^G - x_{r3,j}^G) \quad (16)$$

**DE/rand-to-best/1/bin**

$$v_j = x_{i,j}^G + F \cdot (x_{best,j}^G - x_{i,j}^G) + F \cdot (x_{r1,j}^G - x_{r2,j}^G) \quad (17)$$

**DE/best/2/bin**

$$v_j = x_{best,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (18)$$

**DE/rand/2/bin**

$$v_j = x_{r5,j}^G + F \cdot (x_{r1,j}^G + x_{r2,j}^G - x_{r3,j}^G - x_{r4,j}^G) \quad (19)$$

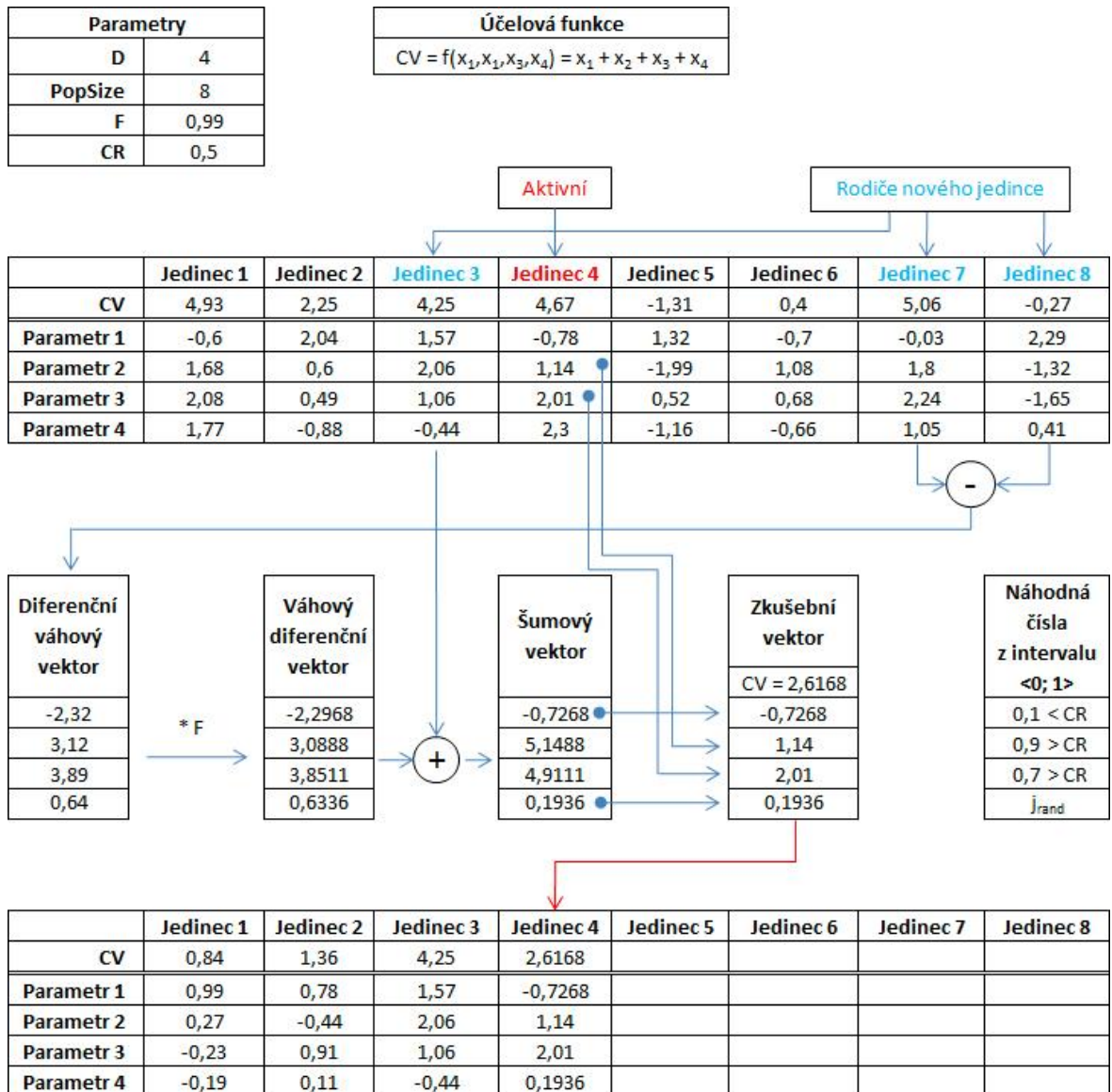
### 3.4 Princip diferenciální evoluce

Po vytvoření počáteční populace podle vzoru se začne provádět evoluční cyklus postupně pro každého jedince populace. Pro variantu (16) podle následujícího postupu:

1. K aktivnímu jedinci, který se právě účastní evolučního cyklu, se vyberou tři další různí jedinci z populace. Máme tedy celkem čtyři jedince, se kterými budeme pracovat.
2. Rozdílem druhého a třetího náhodného jedince obdržíme tzv. diferenční vektor.
3. Vynásobením diferenčního vektoru mutační konstantou  $F$  dochází k mutaci a získání tzv. váhového diferenčního vektoru.
4. Součtem váhového diferenčního vektoru a třetího náhodného jedince získáme tzv. šumový vektor.
5. Nyní se vytvoří tzv. zkušební vektor, jenž je dán kombinací aktivního (čtvrtého) jedince a šumového vektoru. Sestavení zkušebního vektoru je závislé na použité metodě křížení (3.2).
6. Zkušební vektor se ohodnotí účelovou funkcí a porovná se s dosavadní hodnotou účelové funkce aktivního jedince. Na pozici v nové populaci se přesune vektor s lepší hodnotou.
7. Celý postup se opakuje pro všechny jedince z populace. Názorná ilustrace je na obrázku (Obr. 12). [1]

### 3.5 Závislost na ukončovacích parametrech

Činnost diferenciální evoluce je ukončena po provedení všech evolučních cyklů – parametr Generations.



Obr. 12. Princip diferenciální evoluce – varianta DE/rand/1/bin

## 4 PSO: ROJENÍ ČÁSTIC

PSO (Particle Swarm Optimization) je stochastická optimalizační technika vyvinutá v roce 1995, která se, jako mnoho jiných algoritmů, inspirované v přírodě. V tomto případě je základ odvozen od chování ptačích a rybích hejn.

V mnoha věcech se podobá algoritmům genetickým, kde základ tvoří populace náhodných řešení, jejichž jedinci se nějakým způsobem podílí na hledání neoptimálnějšího řešení. U PSO se tato potenciální řešení (jedinci) nazývají particle – částice. Na rozdíl od genetických algoritmů nemá PSO tzv. evoluční metody, jako jsou křížení nebo mutace. [6]

### 4.1 Parametry a terminologie

#### Dimenze

Dimenze vyjadřuje počet argumentů účelové funkce, jejichž optimální kombinaci hledáme. U jednotlivých částic pak vyjadřuje jejich rozměr, ať už z pohledu pozice nebo rychlosti. Dimenzi nedefinuje uživatel, ale je dána definovaným problémem. [6]

#### Particles

Particles udává počet částic, které se účastní hledání nejlepšího řešení. U algoritmů SOMA a diferenciální evoluce je znám pod označením PopSize. Každá částice má rozměr  $D$  a je jí přiřazena hodnota účelové funkce, stejně jako u zmíněných algoritmů SOMA a DE. K těmto informacím je navíc ještě připojen údaj o rychlosti, kterou se částice pohybují v prostoru, jenž je dán vztahem (21).

Tak jako u PopSize platí, že čím více částic vytvoříme, tím více jich bude hledat nejvhodnější řešení, avšak za cenu větší časové náročnosti. [6]

#### vMax

Parametr vMax definuje maximální možnou rychlost během iteračního kola. Obvykle nastavujeme rozmezí, ve kterém se má rychlost parametru částice pohybovat.

Pro částici  $P_1$  o čtyřech parametrech  $\{x_1, x_2, x_3, x_4\}$  zvolíme rychlost pro parametr  $x_1$  v rozsahu  $\langle -10; 10 \rangle$  z čehož dostáváme  $v_{\text{Max}} = 20$ .

V případě zvolení malé rychlosti dochází sice k důkladnějšímu prohledání prostoru, avšak s podstatnou nevýhodou, kterou je menší prohledaná oblast. Naopak když zvolíme velkou

rychlost, bude prohledána větší plocha, ale ne tak důkladně a může se také stát, že částice se vlivem velké rychlosti dostane přes vymezenou oblast. [6]

### Učící se faktory – C1, C2

Učící se faktory C1 a C2 patří mezi parametry, které ovlivňují směr pohybu, kterým se částice vydá. Hodnoty těchto faktorů se obvykle volí z intervalu  $\langle 0; 4 \rangle$ . Často se však oba nastavují na hodnotu 2.

C1 udává, jak moc jsou částice přitahovány ke svému dosavadnímu nejlepšímu výsledku, který označujeme jako osobní nejlepší dané částice *pBest*. C2 se naopak snaží nasměrovat všechny částice tím směrem, který je nejbližší hledaného místa z pohledu celé populace. Tedy za tou částicí, značenou *gBest*, která má nejlepší hodnotu z pohledu účelové funkce. [6]

### Setrvačnost – $w_{start}$ , $w_{end}$

U složitějších optimalizačních problémů se v poslední fázi optimalizační procedury nedařilo kvůli velké rychlosti částic soustředit částice kolem nejslibnějšího řešení, a tak byl do výpočtu rychlosti částice (21) přidán parametr setrvačnost (20), který během iteračních kol postupně redukuje rychlost částic. [7]

$$w = w_{start} - \frac{(w_{start} - w_{end}) \cdot Iteration}{MaxIterations} \quad (20)$$

### MaxIterations

MaxIteration vyjadřuje počet cyklů, během kterých probíhá hledání nejlepšího řešení, a řadíme jej mezi ukončovací parametry.

U algoritmů SOMA a DE tento parametr nalezneme pod označením Migrations, resp. Generations. [6]

### MinDiv

Řadí se mezi ukončovací parametry a udává maximální povolený rozdíl mezi nejlepší a nejhorší částicí z pohledu hodnoty účelové funkce. Parametr MinDiv byl podrobně popsán u algoritmu SOMA.

## Souhrn parametrů

Tab. 3. Hodnoty parametrů PSO [6]

Parametr	Doporučený rozsah
D	dáno problémem
Particles	<20; 40>
vMax	dáno velikostí prohledávané oblasti
C1, C2	<0; 4>
$w_{start}$	0,9
$w_{end}$	0,4
MaxIterations	definuje uživatel
MinDiv	libovolně

## 4.2 Princip PSO

Princip algoritmu PSO si můžeme představit jako ptačí hejno, jehož jednotliví členové mají za úkol hledat zdroj potravy, který se nachází pouze na jediném místě, na ohraničeném území. Žádný jedinec však neví, kde přesně se zdroj potravy nachází, ale ví, jak daleko je od něj. Nejefektivnější způsob nalezení potravy se pak jeví v následování toho ptáka, který je nejbližší hledaného místa.

V PSO ptáci z hejna představují jednotlivá řešení zvaná particle – částice. Jednotlivé částice jsou popsány hodnotou účelové funkce a rychlostí, kterou se pohybují v daném prostoru. Následující postup demonstruje prohledávání oblasti. [6]

1. Na začátku iteračního kola se vypočítá setrvačnost dle vztahu (20).
2. Vypočítá se nová rychlost částice (21), která se následně využije pro přepočtení pozice (22).

$$v_{i,j} = w \cdot v_{i,j} + C1 \cdot rnd \cdot (pBest_{i,j} - x_{i,j}) + C2 \cdot rnd \cdot (gBest_j - x_{i,j}) \quad (21)$$

$$x_{i,j} = x_{i,j} + v_{i,j} \quad (22)$$

3. Nová pozice se ohodnotí účelovou funkcí a porovná se s dosavadní nejlepší nalezenou pro aktuální částici. Pokud je hodnota lepší, částice si pozici zapamatuje jako svoji nejlepší lokální  $pBest_{i,j}$ .
4. Následně se ověří, zdali pozice  $pBest_{i,j}$  není nejlepší vzhledem k celé populaci částic. V případě úspěchu se zaznamená jako  $gBest_j$ .

5. Celý postup se opakuje v každém iteračním kole pro všechny částice. [6]

### **4.3 Závislost na ukončovacích parametrech**

PSO je ukončeno po provedení všech iteračních cyklů (MaxIterations), anebo v případě splnění předepsaného rozdílu mezi nejlepší a nejhorší částicí z pohledu hodnoty účelové funkce (MinDiv).

## 5 UKONČOVACÍ KRITÉRIA

Ukončovací kritéria jsou nástrojem k ukončení činnosti optimalizačního algoritmu. Na rozdíl od maximálního počtu iterací jsou tato kritéria flexibilnější a přizpůsobivější vzhledem k aktuálnímu stavu optimalizace.

U algoritmů SOMA a PSO již bylo zmíněno kritérium MinDiv, v následující části jsou popsána některá další vybraná kritéria.

### 5.1 MaxDist (Quick)

MaxDist (Max Distance) vyjadřuje maximální přípustnou vzdálenost všech jedinců z populace od nejlepšího nalezeného řešení.

Pro  $i$ -tého jedince je vzdálenost od nejlepšího řešení dána vztahem:

$$dist_i = \sqrt{\sum_{j=1}^D (x_{i,j} - gBest_j)^2} \quad (23)$$

V případě, že budeme uvažovat jen určitou část populace např. 90%, pak se jedná o variantu MaxDistQuick. [8]

### 5.2 StdDev

StdDev (Standard Deviation) neboli směrodatná odchylka hodnot účelových funkcí celé populace. Pokud je menší než námi definovaná hodnota, je optimalizační proces ukončen. [8]

$$s = \sqrt{\frac{1}{PopSize - 1} \cdot \sum_{i=1}^{PopSize} (pBestCV_i - \bar{x})^2} \quad (24)$$

Aritmetický průměr:

$$\bar{x} = \frac{1}{PopSize} \cdot \sum_{i=1}^{PopSize} pBestCV_i \quad (25)$$

### 5.3 MinDiv + MaxDist (Quick)

Některá kritéria lze i vhodně kombinovat, např. při spojení MinDiv a MaxDist se nejdříve porovnává rozdíl hodnot účelové funkce nejlepšího a nejhoršího jedince s námi definovanou přípustnou hodnotou (MinDiv) a teprve po splnění se aplikuje druhé kritérium (MaxDist). [8]

## 6 .NET FRAMEWORK

.NET Framework je obrovská knihovna, která poskytuje vše potřebné pro vytváření webových nebo klientských aplikací. Rozhraní .NET obsahuje dvě hlavní části:

- CLR (Common Language Runtime)
- knihovna tříd

CLR je virtuální běhový systém, který v době provádění kódu aplikace poskytuje základní služby a nástroje, jako jsou správa paměti, správa vláken, vzdálená komunikace, bezpečnostní vrstvy pro přístup ke zdrojovému kódu a další.

Knihovna tříd je objektově orientovaná kolekce typů určená pro vývoj širokého spektra aplikací, např.:

- konzolové aplikace
- aplikace s grafickým uživatelským rozhraním – Windows Forms
- Windows Presentation Foundation (WPF)
- ASP.NET [9]

Windows Forms nabízí širokou škálu ovládacích prvků a standardních dialogových oken a programování je v jistých ohledech snazší. Na druhou stranu WPF klade pevné základy pro rozsáhlý budoucí vývoj a především poskytuje velkou volnost v úpravě grafického vzhledu aplikace prostřednictvím jazyka XAML.

Aplikace určené pro .NET se obvykle kompilují do zprostředkujícího kódu – jazyk MSIL, jenž se po spuštění aplikace překládá do nativního kódu, což umožňuje ochranu operačního systému před možnými chybami programů. [10]

*Tab. 4. Psaní aplikací pro systém Windows pomocí jazyka založeného na jazyku C [10]*

Dostupnost od roku	Jazyk	Rozhraní
1985	C	Rozhraní Windows API
1992	C++	Knihovna MFC
2001	C# nebo C++	Windows Forms
2006	C# nebo C++	Windows Presentation Foundation

## **II. PRAKTICKÁ ČÁST**



- `int gWorstIndex` index nejhoršího jedince
- `double[] gBestP` nejlepší nalezená kombinace parametrů
- `double[] gWorstP` nejhorší nalezená kombinace parametrů
- `double gBestCV` nejlepší hodnota účelové funkce
- `double gWorstCV` nejhorší hodnota účelové funkce

Účelová funkce a její parametry:

- `Cf Cf` zvolená účelová funkce
- `Specimen[] specimen` pole popisující parametry účelové funkce

Ukončovací kritérium:

- `StoppingCriteria Sc` zvolené ukončovací kritérium
- `double eps` dovolená odchylka řešení
- `double perc` procentuální část populace, která musí splnit eps

### Základní metody

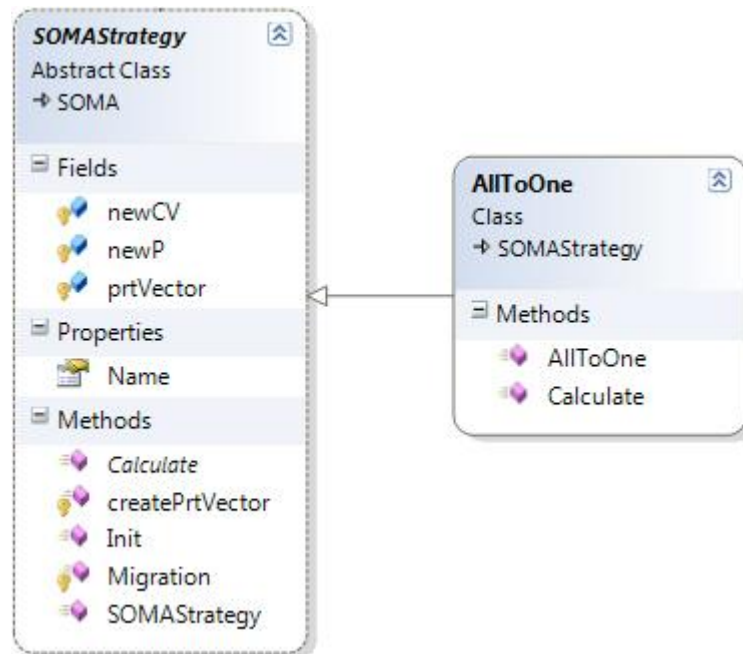
- `void Init` prvotní inicializace algoritmu
- `void InitLoop` inicializace opakování
- `void CreateNewPop` vytvoření nové populace jedinců
- `bool CalculateNextIteration` provedení jednoho iteračního kola
- `abstract bool CalculateNextParams` abstraktní metoda pro výpočet nové pozice
- `bool StoppingCriteria` kontrola ukončovacího kritéria
- `void SetGBest` uložení nejlepšího jedince
- `void SetGWorst` uložení nejhoršího jedince
- `void SetLoopsStats` sestavení statistik pro jedno opakování

## 7.2 SOMA

SOMA je odvozena od abstraktní třídy `Algorithm` a implementuje abstraktní metodu pro výpočet nové pozice jedince – `CalculateNextParams`.

`CalculateNextParams` nejdříve provede inicializaci strategie a teprve po té provede přepočítání pozice metodou `Calculate`. Inicializační metoda předává zvolené strategii informaci o nejlepším řešení a jedinci, který právě prochází migračním procesem.

Jednotlivé strategie algoritmu SOMA jsou založeny na abstraktní třídě SOMAStrategy. Každá odvozená třída (strategie) implementuje metodu Calculate (Obr. 14), která je specifická pro danou strategii. Calculate vrací novou pozici jako pole typu double.



Obr. 14. Abstraktní třída SOMAStrategy a odvozená strategie AllToOne

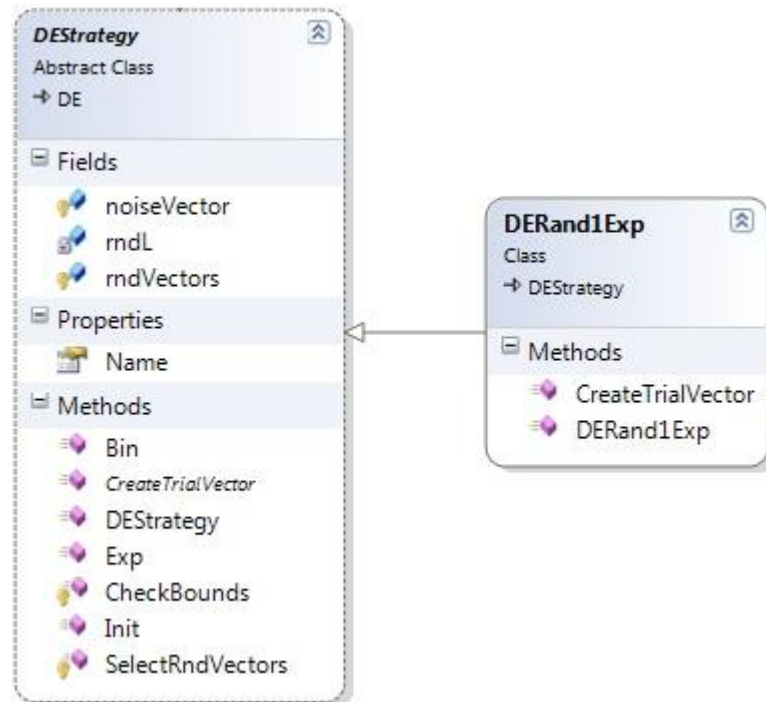
### 7.3 DE

Třída DE implementuje abstraktní třídu Algorithm a její metodu CalculateNextParams v níž je získání nové kombinace parametrů realizováno metodou CreateTrialVector z odvozené třídy DEStrategy.

DEStrategy je abstraktní třída seskupující strategie diferenciální evoluce. Strategie lze rozdělit na dvě skupiny podle typu křížení: binomiální a exponenciální. Binomiální je reprezentováno metodou Bin a exponenciální Exp. Obě metody vrací výsledek křížení jako pole typu double.

### 7.4 PSO

Stejně jako třídy SOMA a DE, tak i PSO dědí implementaci abstraktní třídy Algorithm. Realizace metody pro přepočítání pozice jedince (CalculateNextParams) je však v tomto případě trochu odlišná. Výpočet není prováděn prostřednictvím metod strategií, ale přímo pomocí CalculateNextParams, a to z důvodu, že je k dispozici pouze jediná strategie.



Obr. 15. Abstraktní třída *DEStrategy* a odvozená strategie *DE/Rand/1/Exp*

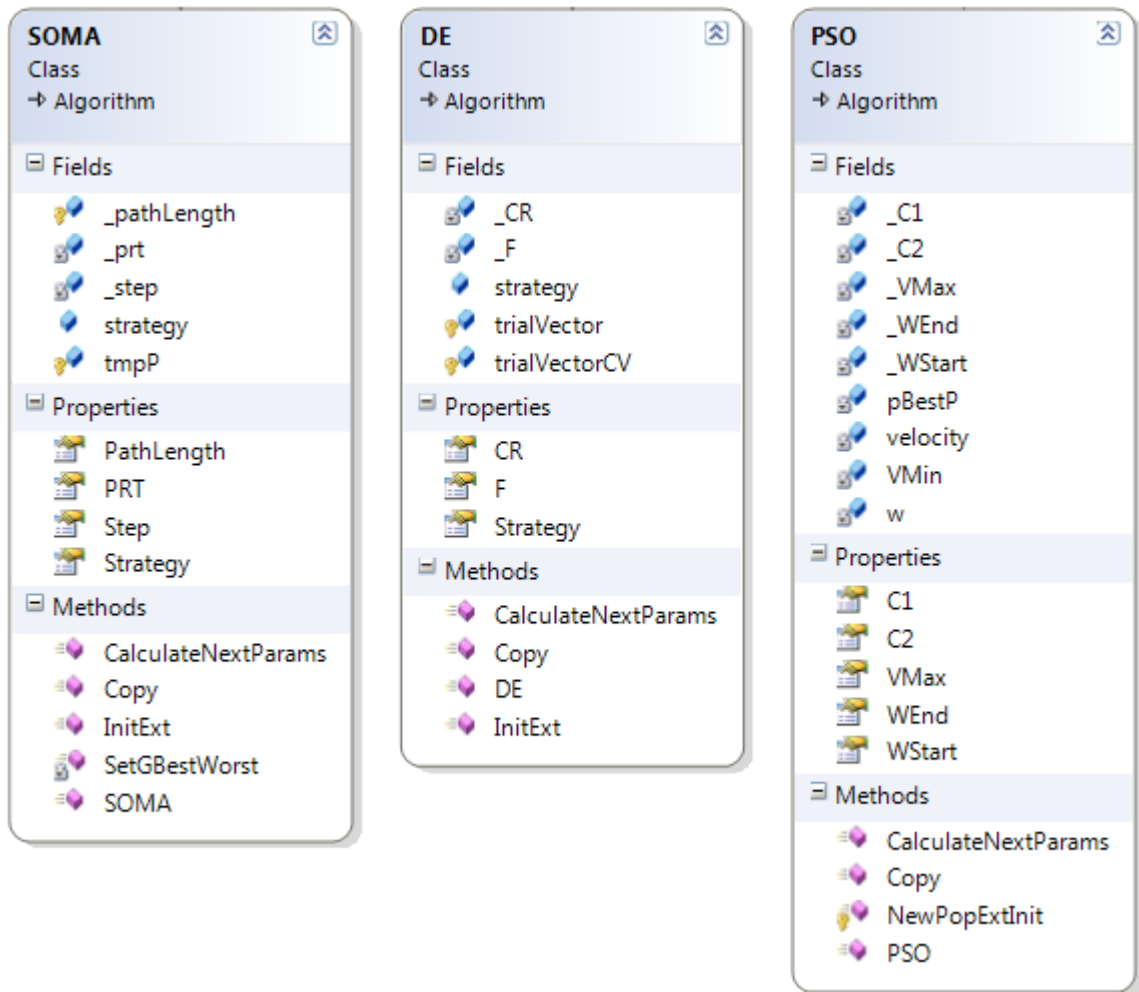
## 7.5 Specimen

Specimen popisuje vlastnosti jednotlivých parametrů účelové funkce. Implementuje rozhraní *ICloneable* pro možnost klonování existujícího objektu prostřednictvím metody *Clone* a *INotifyPropertyChanged* pro aktualizaci hodnot v grafickém prostředí aplikace pomocí metody *Notify*, jejímž vstupním parametrem je název vlastnosti, kterou chceme aktualizovat.

Každý parametr je popsán těmito vlastnostmi:

- název
- typ z množiny *SpecimenTypes*
- dolní mez
- horní mez

*SpecimenTypes* je výčtový typ nabývající hodnot: *Real*, *Integer*.



Obr. 16. Podrobné schéma tříd SOMA, DE a PSO

## 7.6 StoppingCriteria

Třída `StoppingCriteria` poskytuje algoritmům několik způsobů, kterými lze ukončit jejich činnost po dosažení uspokojivého výsledku ještě před dosažením maximálního počtu iterací.

Konstruktor přijímá tři argumenty: populaci jedinců (`pop`), dovolenou odchylku (`eps`) a část populace, která má odchylku splňovat (`perc`).

Pro vyhodnocení stavu řešení se využívají následující metody, které v případě splnění daného kritéria vrací booleovskou hodnotu `true`.

- `MinDiv (gWorstCV, gBestCV)`
- `MaxDist (gBestP)`
- `StdDev (gBestCV)`
- `MinDivMaxDist (gWorstCV, gBestCV, gBestP)`

## 7.7 Stats

Po dokončení všech iteračních kol a opakování probíhá vždy vytvoření statistik, které jsou vedeny jako kolekce třídy Stats. Prostřednictvím argumentu konstruktoru se předává instance třídy Algorithm, ze které se pomocí inicializačních metod sestaví statistiky.

Uložená data se využívají jak pro sestavení výsledkové stránky, tak i pro případný export dat do souboru.

### Základní vlastnosti a proměnné

- Specimen[] Specimen                      parametry specimena
- BestParams[] Bp                            nejlepší nalezené parametry
- Series[] DataItLeaders                    nejlepší řešení migračních kol
- List<Loop> LoopsStats                    statistiky jednotlivých opakování

Dále disponuje několika strukturami, které seskupují parametry algoritmu, účelové funkce a ukončovacího kritéria.

Exportu dat je vyhrazena metoda Export s argumentem FileType, což je výčtový typ nabývající hodnot CSV a XML. Výstupem je pole typu string reprezentující sestavený XML příp. CSV soubor.

## 7.8 XML

XML třída slouží pro načtení parametrů specimena a export výsledků optimalizačního běhu ve formátu XML.

K dispozici jsou dva přetížené konstruktory. Konstruktor s parametrem typu string, jenž udává název souboru, je určen pro načtení specimena. Druhý konstruktor se používá pro export dat a jeho argumentem je instance třídy Stats.

### 7.8.1 Import parametrů

Při načítání souboru s parametry je nutné nejdříve ověřit validitu souboru, zdali obsahuje všechny potřebné údaje a ve správném formátu. K ověření validity podle DTD definice slouží metoda IsValid, jenž využívá XmlReader, jehož konstrukturu předáme instanci třídy XmlReaderSettings s nastavením validace a parsování. Následně se pomocí metody Read (z třídy XmlReader) kontrolují jednotlivé elementy, zda odpovídají definici v DTD. Pokud je soubor v pořádku, vrací metoda návratovou hodnotu true.

Po úspěšné kontrole se provede parsování metodou TryParse, která využívá třídy XPathNavigator k pohybu ve zpracovávaném souboru a k identifikaci jednotlivých elementů. Po úspěšném zpracování parametru se vytvoří instance třídy Specimen s příslušným nastavením a jako nová položka se přidá do ListView seznamu, který se metodě TryParse předává jako vstupní parametr.

Celkový počet úspěšně načtených parametrů je k dispozici jako vlastnost s názvem Count.

### 7.8.2 DTD definice

DTD definuje strukturu, jejímž kořenovým elementem je <specimen>, jenž musí obsahovat minimálně jeden parametr <parameter> s názvem <name>, typem <type> a dolní <low> a horní <high> mezí, které už smí obsahovat pouze příslušnou datovou hodnotu.

#### DTD dokument

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!ELEMENT specimen (parameter+)>
3 <!ELEMENT parameter (name, type, low, high)>
4 <!ELEMENT name (#PCDATA)>
5 <!ELEMENT type (#PCDATA)>
6 <!ELEMENT low (#PCDATA)>
7 <!ELEMENT high (#PCDATA)>
```

#### XML soubor s jedním parametrem

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE specimen SYSTEM "./specimen.dtd">
3 <specimen>
4 <parameter>
5 <name>p1</name>
6 <type>Real</type>
7 <low>-5,12</low>
8 <high>5,11</high>
9 </parameter>
10 </specimen>
```

### 7.8.3 Export výsledků

Pro export výsledků optimalizačního procesu jsou k dispozici metody: Settings, Specimen, BestParams, DataItLeaders a LoopsStats s návratovou hodnotou XDocument. Každá ze zmíněných metod vytváří strukturu XML souboru za pomoci třídy XDocument. Třídě

XDocument se předávají informace o kódování a kořenovém prvku, kterému postupně pomocí XElement přidáváme podřízené elementy.

## 7.9 CSV

CSV je obdobou XML třídy s tím rozdílem, že pracuje se souborem, v němž jsou data na každém řádku oddělená oddělovacím znakem – středníkem.

Pro načtení parametrů využívá konstruktor s argumenty StreamReader (soubor, ze kterého načítáme) a char (oddělovací znak). Druhý přetížený konstruktor, jehož parametry jsou Stats (statistiky) a char (opět jako oddělovací znak), se využívá pro exportu výsledků.

### 7.9.1 Import parametrů

Import parametrů se provádí metodou TryParse, která přijímá jako argument referenci na seznam ListView, do kterého se parametry postupně vkládají.

Samotné načítání dat ze souboru se provádí pomocí ReadLine pro každý řádek, jenž se dále rozloží metodou Split na jednotlivé bloky dat podle oddělovacího znaku.

Do ListView se nakonec uloží nová instance třídy Specimen s načtenými hodnotami parametru.

Počet správně načtených parametrů můžeme získat prostřednictvím vlastnosti Count.

### 7.9.2 Export výsledků

Stejně jako v XML třídě, jsou i v CSV metody určené výhradně pro export dat a to: Settings, Specimen, BestParams, DataItLeaders a LoopsStats. Návrátovou hodnotou je v tomto případě StringBuilder, pomocí jehož metody AppendLine se postupně sestavují řádek po řádku soubory s výsledky.

## 7.10 Loop

Loop slouží pro uchování statistik z jednotlivých opakování optimalizačního procesu. Z důvodu možnosti seřazení provedených opakování, podle nejlepšího hodnoty účelové funkce, implementuje rozhraní IComparable<Loop> a jeho metodu CompareTo.

Každý Loop uchovává následující údaje:

- číslo opakování
- poslední provedenou iteraci

- zdali bylo splněno ukončovací kritérium
- nejlepší nalezené parametry jako pole struktury BestParams
- nejlepší hodnotu účelové funkce

Struktura BestParams popisuje každý parametr názvem, typem (SpecimenTypes) a hodnotou.

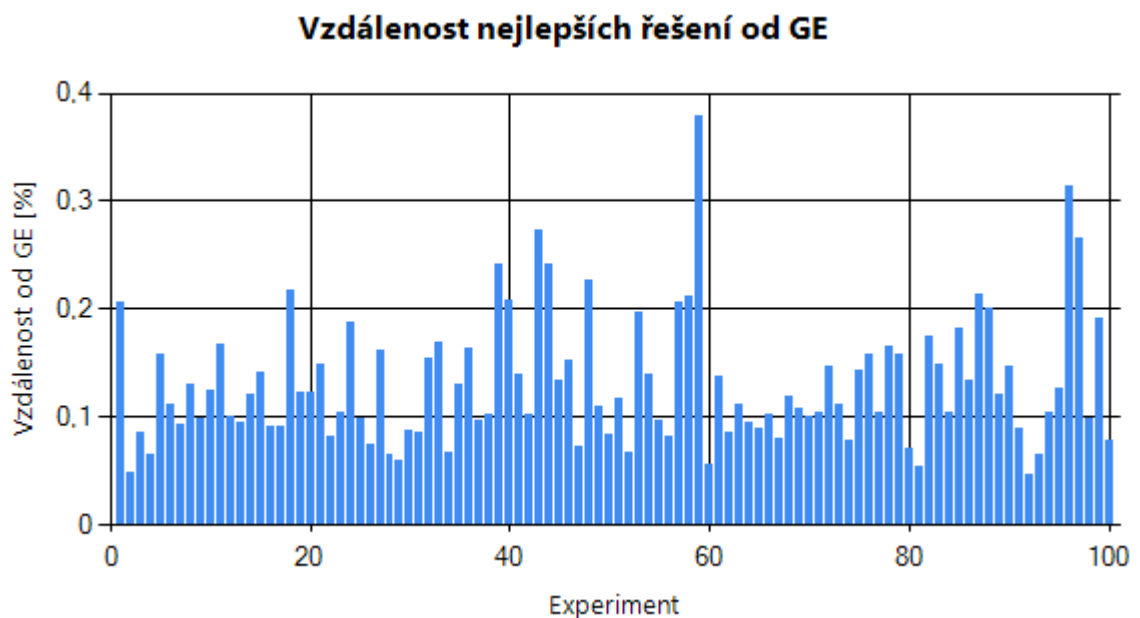
## 7.11 Graph

Třída Graph, jak už název napovídá, je jednoduchým, ale velmi užitečným nástrojem sloužícím k prezentaci výsledků ve formě přehledných grafů.

Prostřednictvím inicializační metody Init se předávají data k vykreslení (instance třídy Series), typ grafu (sloupcový, spojnicový, ...), název grafu a styl značek bodů.

Pro nastavení os X a Y jsou k dispozici dvě metody: AxisXInit a AxisYInit. Pomocí nich můžeme nastavit název osy a minimální a maximální hodnotu osy.

Samotné vykreslení grafu se provádí metodou Draw, která vrací instanci objektu WindowsFormHost, a tak je možné přiřadit graf do nějakého prvku okna.



Obr. 17. Ukázka použití třídy Graph

## 7.12 Validační pravidla

Validační pravidla jsou určena ke kontrole vstupních hodnot, které zadává uživatel z klávesnice. K dispozici je validace jak pro celočíselné hodnoty, reálné tak i řetězce. Pro celočíselné a reálné hodnoty je možné navíc nadefinovat dovolený rozsah.

K vytvoření uživatelského pravidla je zapotřebí vytvořit třídu, která dědí od `ValidationRules` a implementuje validační metodu `Validate`, v níž je možné kontrolovat, zdali je hodnota správná. Kontrola se provádí pomocí metody `TryParse`, jenž je součástí tříd `Int32` a `Double`, v případě řetězce se využívá metoda `IsNullOrWhiteSpace` z třídy `String`.

Pro kontrolu rozsahu u celočíselných a reálných hodnot je zapotřebí navíc nadefinovat vlastnosti `Min` a `Max`.

```
1 public int Min { get; set; }
2 public int Max { get; set; }
```

### Metoda `Validate` pro celočíselné hodnoty s rozsahem

```
3 // kontrola typu
4 if (!Int32.TryParse((string)value, out this.tmpValue)) {
5     return new ValidationResult(false, "Pouze celá čísla!");
6 }
7 // kontrola rozsahu
8 else if ((this.tmpValue < this.Min) || (this.tmpValue > this.Max)) {
9     return new ValidationResult(false, "Povolený rozsah: " + Min + " až "
10         + Max);
11 }
12 // platná hodnota
12 return new ValidationResult(true, null);
```

## 7.13 Commands

`Commands` je statická třída seskupující vlastní vytvořené příkazy aplikace.

K zaregistrování příkazu využívá `RoutedUICommand`, jehož konstruktoru se předává, jako jeden z argumentů, instanci třídy `InputGestureCollection`, pomocí níž se definuje vlastní klávesová zkratka a popis.

Konstrukce vlastních příkazů je využita jak v hlavním menu, tak i v panelu nástrojů.

## Vytvoření vlastního příkazu

```

1 public static RoutedUICommand Exit { get; private set; }
2 InputGestureCollection exitInputs = new InputGestureCollection();
3 exitInputs.Add(new KeyGesture(Key.F4, ModifierKeys.Alt, "Alt+F4"));
4 Exit=new RoutedUICommand("Exit","Exit",typeof(Commands),exitInputs);

```

## 7.14 GridViewSorter

GridViewSorter se stará o řazení záznamů podle zvoleného sloupce v seznamu ListView.

Pro rozlišení směru řazení využívá proměnnou výčtového typu ListSortDirection nabývající hodnoty Ascending (vzestupně) a Descending (sestupně) a pro zaznamenání sloupce, podle kterého se řadilo naposledy, proměnnou typu GridViewColumnHeader.

Smysl uchování naposledy setříděného sloupce je v tom, aby v případě kliknutí na ten samý sloupec se záznamy řadily opačně a při kliknutí na jiný sloupec byly záznamy řazeny vzestupně – výchozí řazení.

Název	Typ	Low	High
p3	Real	1	1.5
p2	Integer	0.5	2
p4	Real	2.5	3.1
p1	Real	0	4

*Obr. 18. ListView seřazené vzestupně podle sloupce High*

Obslužení události Click v záhlaví sloupců obstarává stejnojmenná metoda, která vyhodnocuje sloupec a směr seřazení a následně tyto údaje předá metodě Sort.

Sort je řadící metoda, v níž se nejdříve všechny položky ListView načtou do kolekce ICollectionView, jež nabízí třídící, řadící a seskupující funkce. V tomto případě je využita pouze vlastnost řazení – SortDescriptions. Pro správné fungování je nutné nejdříve odstranit předchozí seřazení pomocí Clear. Následně provedeme přidání nového způsobu řazení metodou Add a aktualizaci prostřednictvím Refresh.

```

1 ICollectionView dataView =
  CollectionViewSource.GetDefaultView(this.lv.Items);
2 SortDescription sd = new SortDescription(sortBy, direction);
3 dataView.SortDescriptions.Clear();
4 dataView.SortDescriptions.Add(sd);
5 dataView.Refresh();

```

## 7.15 Cf

Cf (Cost Function) reprezentuje účelovou funkci načtenou z DLL knihovny.

Implementuje generická rozhraní `IComparable<Cf>` pro možnost řazení podle názvu a `IEquatable<Cf>`, aby se v aplikaci vyskytovaly pouze účelové funkce s unikátním názvem.

Prostřednictvím parametru konstruktoru se předává `Type`, ze kterého se pomocí `InvokeMember` získá vlastnost s názvem účelové funkce a `GetMethod` se využívá pro načtení informací (`MethodInfo`) o metodě pro ohodnocení účelové funkce.

Metoda `GetCV` je určena pro výpočet účelové funkce a jako argument přijímá pole typu `double`. Samotný výpočet se provádí prostřednictvím `Invoke` nad objektem `MethodInfo`.

## 7.16 CfList

`CfList` je třída odvezená od generického typu `List<Cf>` a rozšiřuje klasický netříděný seznam o metody `Add` a `LoadAssembly`.

`Add` metoda je přizpůsobena jak pro načtení jednoho DLL souboru, tak i pro hromadné načítání z předem určené složky pomocí `Directory.GetFiles`, která prochází i podsložky a filtruje soubory s příponou `DLL`.

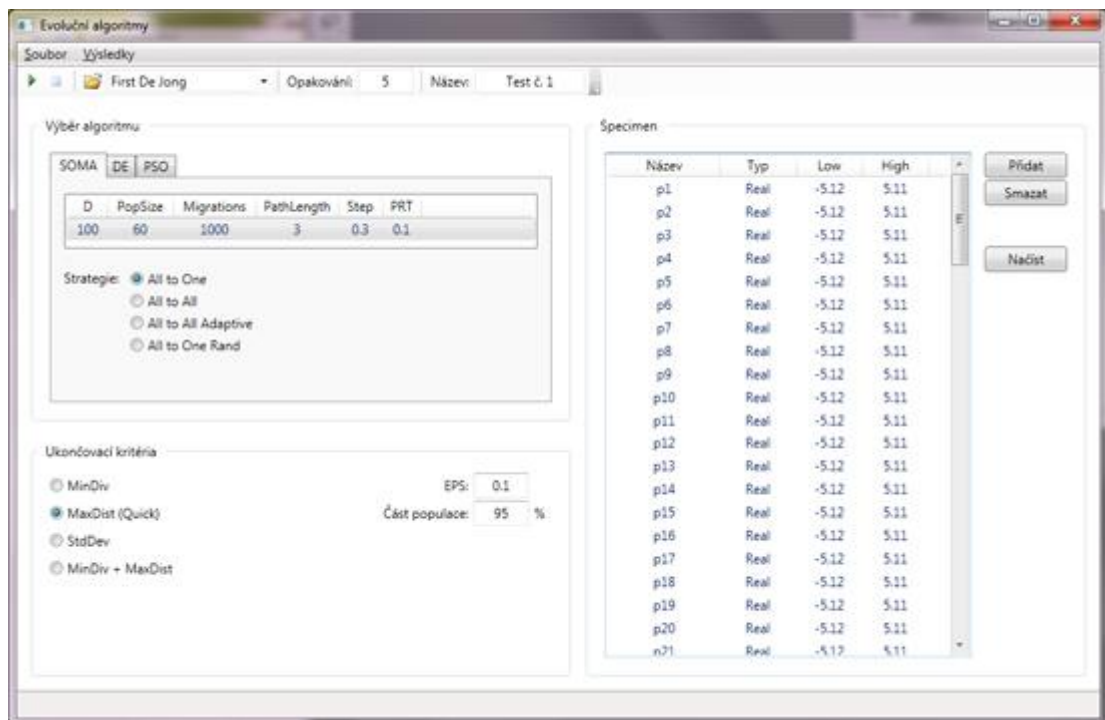
Pro všechny nalezené `DLL` soubory se zavolá metoda `LoadAssembly`, v níž se provede načtení assembly ze souboru pomocí `Assembly.LoadFrom` a prostřednictvím `GetTypes` se vyberou všechny dostupné typy. Z každého typu se vytvoří instance a ověří se, zdali obsahuje metodu `GetCV`, která je povinná a jestli již aplikace neobsahuje stejnojmennou účelovou funkci. Nakonec se jako nová instance třídy `Cf` přidá do kolekce všech účelových funkcí.

Vlastnost `CountCf` obsahuje počet úspěšně načtených účelových funkcí.

## 8 GRAFICKÉ ROZHRANÍ

### 8.1 Hlavní okno

Rozdělení prvků hlavního okna aplikace je vytvořeno pomocí DockPanel, ke kterému jsou vlastností DockPanel.Dock přichyceny hlavní části. Pro zachování požadovaného vzhledu (Obr. 19) je nutné přichytit objekty ve správném pořadí a to StatusBar, Menu, ToolBarTray a nakonec obsahovou část. Kdyby byl StatusBar přichycen až za obsahem, obsahová část by jej vlivem roztažení překryla.



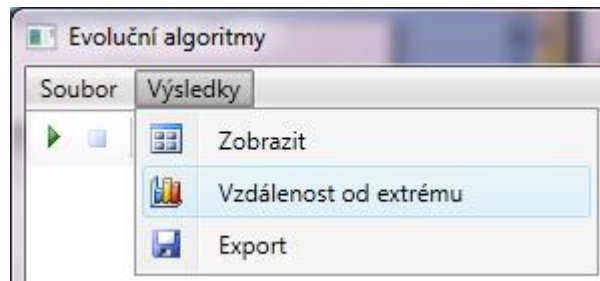
Obr. 19. Hlavní okno aplikace

#### 8.1.1 Menu

Struktura nabídky menu je tvořena objekty MenuItem, jenž mají nastaven atribut Command na odpovídající příkaz ze statické třídy Commands (7.13). Aby jednotlivé položky menu vykonávaly žádanou činnost, musí být příkazy navíc propojeny s některou z metod, která danou proceduru vykoná. Propojení lze realizovat pomocí CommandBinding.

```
1 <CommandBinding Command="c:Commands.ResView"
  Executed="ResViewCmd_Executed"
  CanExecute="ResViewCmdBinding_CanExecute" />
```

Atribut Executed se odkazuje na prováděnou metodu a CanExecute na metodu, která vyhodnocuje, kdy má být položka aktivní, např. výsledky lze exportovat až ve chvíli, kdy máme uložená data minimálně z jednoho experimentu.

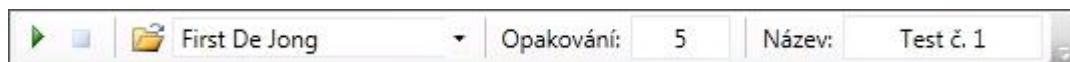


Obr. 20. Hlavní nabídka menu

### 8.1.2 Panel nástrojů

Panel nástrojů (ToolBarTray) je složen z tlačítek Button, jejichž vzhled je utvořen pomocí objektu Image, který nastavuje příslušný obrázek. Činnost tlačítek je, stejně jako v případě položek menu, realizována prostřednictvím příkazů Command.

Seznam účelových funkcí je vytvořen pomocí ComboBox a zadávací pole jako TextBox. U zadávacího pole pro opakování je navíc aplikováno validační pravidlo (7.12) s definovaným rozsahem.



Obr. 21. Panel nástrojů

#### Tlačítko spustit

Možnost spuštění běhu algoritmu je dostupná pouze tehdy, pokud je zvolena účelová funkce a vytvořen minimálně jeden parametr specimena. Aktivace a deaktivace tlačítka je docílena přiřazením vyhodnocovací metody atributu CanExecute. Pomocí DataTriggeru, jenž vyhodnocuje, zda je tlačítko aktivní, se obrázku navíc sníží sytost pro vizuální efekt.

```

1 <DataTrigger Binding="{Binding RelativeSource={x:Static
  RelativeSource.Self}, Path=IsEnabled}" Value="False">
2 <Setter Property="Opacity" Value="0.3" />
3 </DataTrigger>

```

Metoda RunCmd\_Executed provádí veškeré operace spojené se spuštěním zvoleného algoritmu. Aby během delších výpočtů hlavní okno „nezamrzlo“, je hledání nejlepšího řešení prováděno v pozadí aplikace použitím BackgroundWorker.

BackgroundWorker jsou nastaveny dvě události: DoWork a RunWorkerCompleted. DoWork provádí hlavní činnost – opakování a iterační kola. RunWorkerCompleted zabezpečuje vytvoření statistik a výsledkového okna po dokončení běhu DoWork.

```
1 BackgroundWorker worker = new BackgroundWorker();
2 worker.WorkerSupportsCancellation = true;
3 worker.RunWorkerCompleted += this.WorkerShutDown_Event;
4 worker.DoWork += this.WorkerDoWork_Event;
5 worker.RunWorkerAsync();
```

### Tlačítko zastavit

Tlačítko zastavit je zpřístupněno jen v případě, že běží optimalizační proces. Ukončení činnosti BackgroundWorker se provádí zavoláním metody CancelAsync.

### Načtení účelové funkce

Účelovou funkci lze vybrat pomocí dialogového okna OpenFileDialog, jenž umožňuje nahrát pouze soubory s příponou DLL.

### 8.1.3 Obsah

Obsahová část je tvořena objektem Grid, který je rozdělen na dva sloupce, aby se v případě maximalizace okna levá i pravá část roztáhly stejnoměrně na celou obrazovku.

### Volby a nastavení algoritmů

Výběr a nastavení algoritmů je rozděleno TabControl na záložky (Obr. 22), kterým jsou přiřazeny instance tříd SOMA, DE a PSO přes DataContext, a tak je možné přímo provázat vlastnosti algoritmů s položkami v ListView přes DisplayMemberBinding. Hodnota dimenze je propojena pomocí Data Binding s ListView specimena, ze kterého automaticky načítá počet položek.

```
1 <GridViewColumn Header="D" DisplayMemberBinding="{Binding
   ElementName=specimenLV, Path=Items.Count,
   UpdateSourceTrigger=Explicit}" />
```

### Ukončovací kritéria

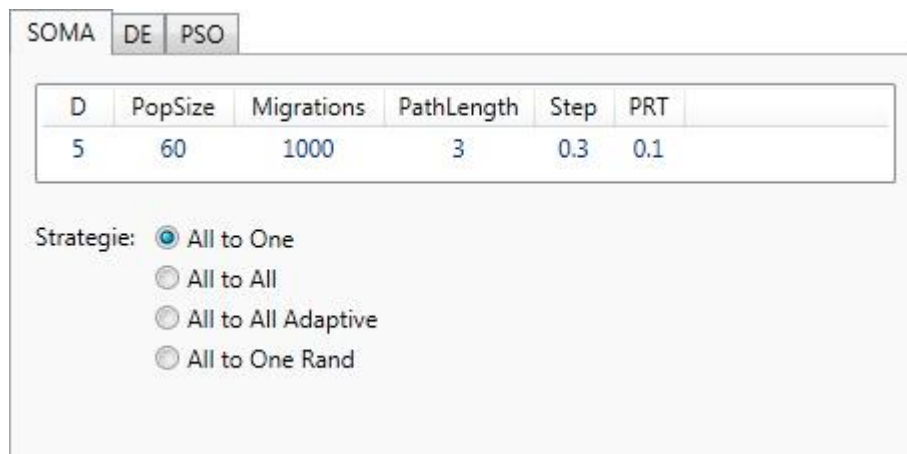
Volba kritérií je vytvořena pomocí prvků RadioButton, které jsou seskupeny pomocí GroupName. U některých je navíc k dispozici TextBox pro zadání části populace, jehož zpřístupnění se provádí vyhodnocením podmínek v MultiDataTrigger.

```
1 <MultiDataTrigger>
2 <MultiDataTrigger.Conditions>
```

```

3 <Condition Binding="{Binding ElementName=maxDist,Path=IsChecked}"
  Value="False" />
4 <Condition Binding="{Binding
  ElementName=minDivMaxDist,Path=IsChecked}" Value="False" />
5 </MultiDataTrigger.Conditions>
6 <Setter Property="IsEnabled" Value="False" />
7 </MultiDataTrigger>

```



Obr. 22. Výběr algoritmu pomocí TabControl

## Specimen

Seznam parametrů je tvořen objektem ListView a jednotlivé položky jsou ukládány jako instance třídy Specimen.

Položky lze přidávat po jednom, nebo hromadně ze souboru, který se zvolí přes dialogové okno OpenFileDialog. Vlastností Filter jsou povoleny pouze CSV a XML soubory.

```

1 OpenFileDialog ofd = new OpenFileDialog();
2 ofd.Filter = ".csv, .xml|*.csv;*.xml|.csv|*.csv|.xml|*.xml";

```

Tlačítko smazání je zpřístupněno, pouze pokud je vybrána minimálně jedna položka, což zajišťuje DataTrigger. Pro možnost výběru více položek, z důvodu hromadného mazání, je ListView nastaven SelectionMode na Extended.

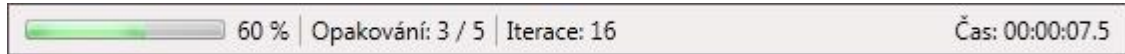
### 8.1.4 StatusBar

StatusBar během optimalizačního procesu zobrazuje aktuální stav prováděných operací.

Procentuální část je řešena pomocí objektu ProgressBar a aktuální hodnota je dána vztahem (26). Ostatní údaje jsou zobrazovány prostřednictvím objektu Label.

$$perc = \frac{(CurrLoop \cdot MaxIterations) + CurrIteration}{MaxIteration \cdot MaxLoops} \cdot 100 \quad (26)$$

O pravidelnou aktualizaci údajů se stará metoda UpdateProgressText.



Obr. 23. StatusBar

## 8.2 Výsledkové okno

Okno s výsledky je kvůli přehlednosti rozděleno TabControl na záložky, kde záložka prezentuje jeden provedený experiment.

Formát a uspořádání záložky je vytvořeno jako nový objekt UserControl, jenž obsahuje TabControl se záložkami zarovnanými ke spodní straně.

## 8.3 Export výsledků

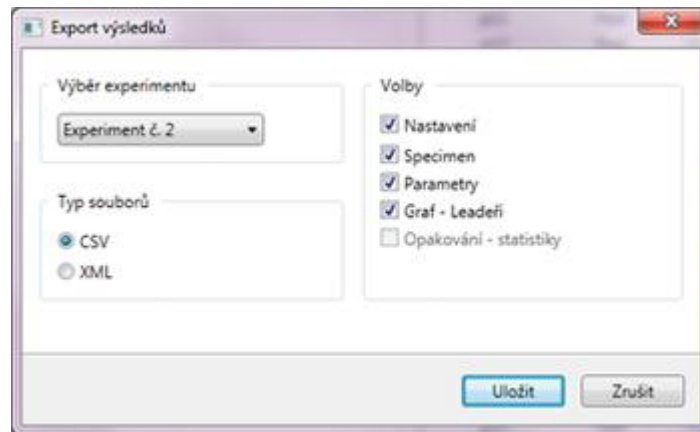
Export výsledků umožňuje získat data z aplikace ve formátu několika XML a CSV souborů. Pomocí zaškrtnutí CheckBoxu je možné zvolit části, které chceme exportovat, avšak „Opakování – statistiky“ je možné exportovat pouze v případě, že bylo ve vybraném experimentu provedeno více opakování. Aktivace a deaktivace této volby je uskutečněna pomocí DataTriggeru, který ověřuje počet opakování. Pokud je počet roven jedné, dojde k deaktivaci této volby.

```

1 <DataTrigger Binding="{Binding ElementName=expCB,
  Path=SelectedItem.LoopsStats.Count}" Value="1">
2 <Setter Property="IsChecked" Value="False" />
3 <Setter Property="IsEnabled" Value="False" />
4 </DataTrigger>

```

Tlačítko „Uložit“ obsluhuje metoda Save\_Click v níž se výběr cílové složky, pro uložení exportovaných souborů, realizuje pomocí třídy FolderBrowserDialog z Windows Forms, protože samotné WPF žádný nástroj pro výběr cílové složky neposkytuje. Po výběru složky se nad vybraným experimentem, který odpovídá třídě Stats, zavolá metoda Export se zvoleným typem souboru. Výsledné soubory se po té vytvoří zavoláním File.WriteAllText, jenž jako první parametr přijímá název souboru a jako druhý se předává obsah souboru ve formě řetězce.



Obr. 24. Export výsledků

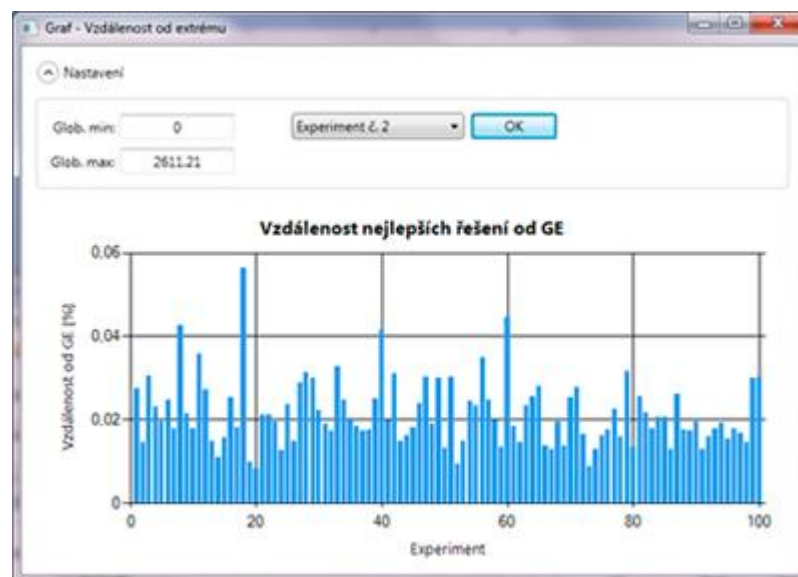
## 8.4 Vzdálenost od extrému

Oblast nastavení je realizována objektem Expander, jenž umožňuje skrytí oblasti nastavení za účelem lepšího využití místa pro jiný objekt, v tomto případě pro graf.

Pole pro zadávání globálního minima a maxima jsou ošetřeny validačním pravidlem, které dovoluje zadávat reálná čísla. Výpočet vzdálenosti od extrému provádí metoda CalculateDistFromGe dle vztahu (27). Vstupním parametrem je  $Min_{nalezené}$ .

$$dist = \frac{|Min - Min_{nalezené}|}{|Max - Min|} \cdot 100 \quad (27)$$

Vykreslení grafu obstarává třída Graph.

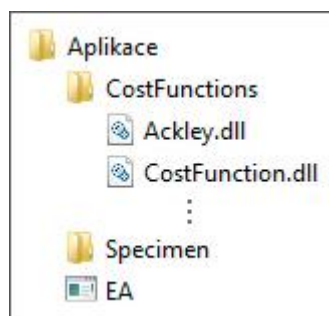


Obr. 25. Vzdálenost od extrému

## 9 POPIS OVLÁDÁNÍ

Aplikaci je distribuována formou spustitelného EXE souboru, takže ji není nutné nijak složitě instalovat. Ke spuštění je však nutné mít na počítači nainstalován Microsoft .NET Framework 4.

Po spuštění se automaticky načtou účelové funkce z adresáře CostFunctions i jeho podadresářů, jenž se nachází na stejné úrovni, jako samotná aplikace. Pokud tedy chcete mít k dispozici vlastní účelové funkce hned po spuštění, umístěte je právě do tohoto adresáře.



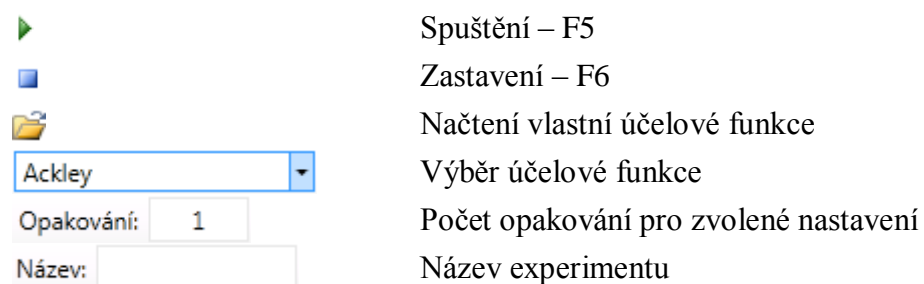
Obr. 26. Struktura adresářů aplikace

### 9.1 Menu

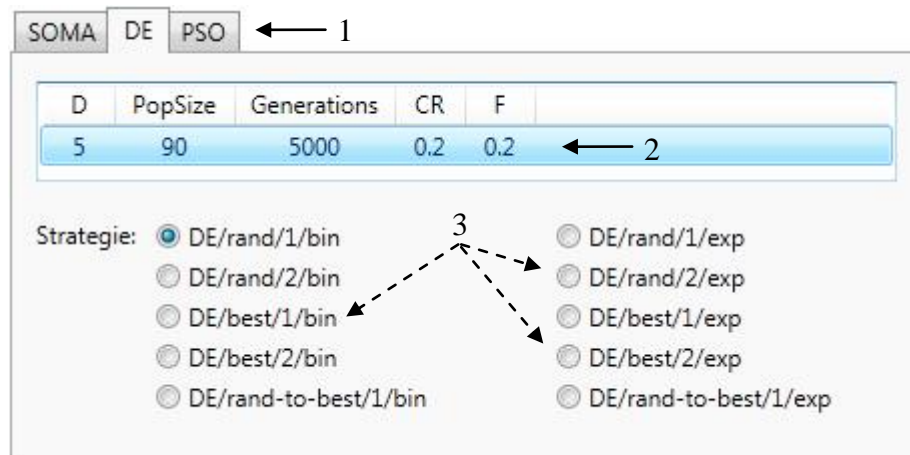
Menu obsahuje dvě hlavní položky: Soubor a Výsledky. V části „Soubor“ je možné aplikaci ukončit, což lze provést i pomocí klávesové zkratky Alt + F4. Položka „Výsledky“ obsahuje:

- zobrazení výsledků v novém okně
- vytvoření grafu, jenž prezentuje vzdálenost od extrému – vhodné pro testovací funkce
- export výsledků do CSV, nebo XML souborů

### 9.2 Panel nástrojů



### 9.3 Algoritmy a strategie



Obr. 27. Volba a nastavení algoritmu

1. Volba algoritmu SOMA/DE/PSO se provádí pouhým přepnutím záložky. Na obrázku (Obr. 27) je vybrána diferenciální evoluce – DE.
2. Nastavení a úprava parametrů se provádí dvojklikem na zvýrazněnou oblast.
3. Výběr strategie.

### 9.4 Ukončovací kritéria

Uživatel má na výběr čtyři ukončovací kritéria. Pro každé kritérium je nutné nadefinovat hodnotu EPS, která udává dovolenou odchylku řešení. U kritérií MaxDist (Quick) a MinDiv + MaxDist lze navíc zvolit část populace, jenž má předepsanou hodnotu EPS splňovat. Pokud nechceme využít žádné kritérium, nastavíme EPS na -1.



Obr. 28. Výběr a nastavení ukončovacího kritéria

### 9.5 Specimen

Parametry lze postupně přidávat prostřednictvím tlačítka „Přidat“, což může být při větším počtu parametrů nepraktické. Z toho důvodu je možné nahrát parametry ze souboru ve formátu CSV, nebo XML. Umístění souboru na disku nehraje roli. Pro testovací funkce jsou předpřipraveny soubory ve složce Specimen.

V případě potřeby lze provést úpravu názvu, typu, nebo mezí vybraného parametru. Editační okno se zobrazí po dvojkliku na příslušný parametr.

Název	Typ	Low	High
p1	Real	-5.12	5.11
p2	Real	-5.12	5.11
p3	Real	-5.12	5.11
p4	Real	-5.12	5.11
p5	Real	-5.12	5.11

Přidat  
Smazat  
Načíst

Obr. 29. Parametry specimena

## 10 VLASTNÍ ÚČELOVÁ FUNKCE

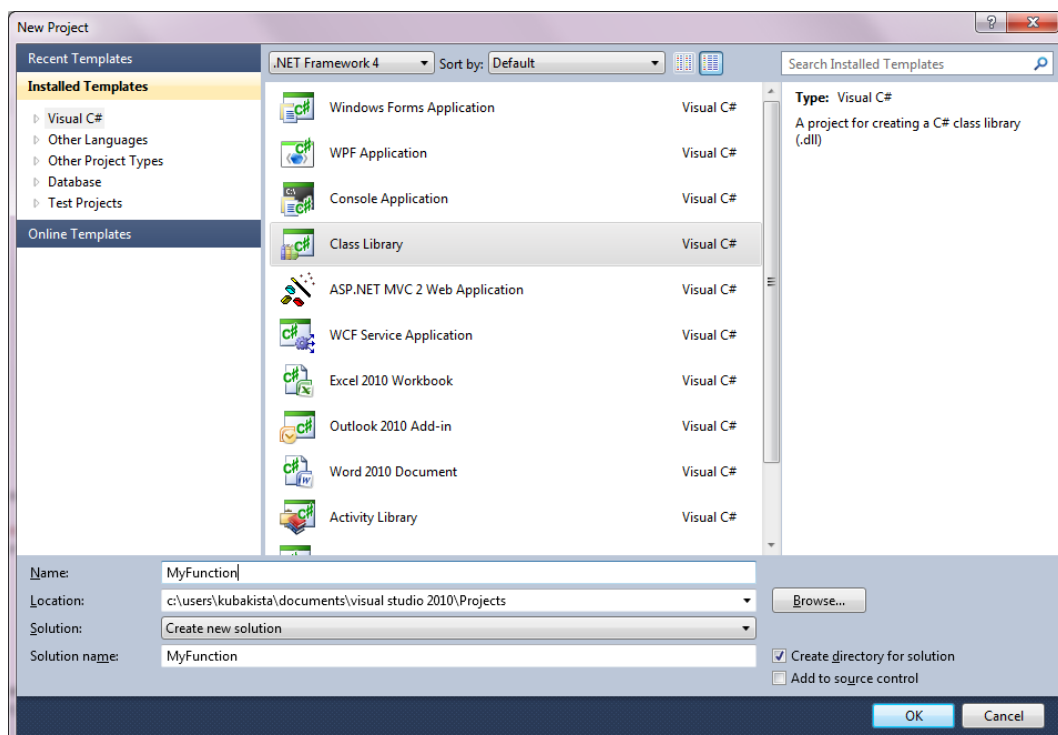
Aplikace umožňuje načtení vlastních účelových funkcí ve formátu DLL knihoven, které však musí splňovat určitá kritéria:

1. Název jmenného prostoru a třídy je libovolný, ale název metody, která obstarává výpočet účelové funkce, musí být „GetCV“.
  - a. Jediným parametrem metody je pole typu double.
  - b. Návratovou hodnotou je double.
2. Pro lepší identifikaci účelových funkcí v aplikaci je dobré navíc nadefinovat název účelové funkce jako vlastnost typu string s názvem „Name“. Pokud název neuvědeme, vygeneruje se automaticky název ve tvaru „Účelová funkce #1“.

V jedné DLL knihovně je možné definovat i více účelových funkcí.

### Postup ve Visual Studio 2010

Vytvoříme nový projekt (File – New Project), vybereme příslušný programovací jazyk (C#, Basic, ...) a zvolíme typ „Class Library“ (Obr. 30). Následně napíšeme jednoduchou třídu splňující již zmíněná kritéria a po úspěšném přeložení zdrojového kódu se nám automaticky vytvoří DLL knihovna (soubor s příponou DLL).



Obr. 30. Vytvoření projektu DLL knihovny ve Visual Studio 2010

### Účelové funkce ve Visual C#

```
1 namespace TestFunctions {
2     public class MyFunction {
3         public string Name { get; private set; }
4         private double cv;
5         public MyFunction() { this.Name = "Moje funkce"; }
6         public double GetCV (double[] p) {
7             this.cv = 0;
8             for (int i = 0; i < p.Length; ++i) {
9                 this.cv += Math.Abs(p[i]);
10            }
11            return this.cv;
12        }
13    }
14 }
```

### Účelová funkce ve Visual Basic

```
1 Public Class MyFunction
2     Private _name As String
3     Public Property Name() As String
4     Get
5     Return _name
6     End Get
7     Private Set(ByVal value As String)
8     _name = value
9     End Set
10 End Property
11 Public Sub New()
12     Name = "Moje funkce"
13 End Sub
14 Function GetCV(ByVal p() As Double) As Double
15     Dim i As Integer
16     Dim cv As Double
17     cv = 0
18     For i = 0 To (p.Length - 1)
19         cv += Math.Pow(p(i), 2)
20     Next
21     Return cv
22 End Function
23 End Class
```

## 11 VÝSLEDKY TESTŮ

Funkčnost aplikace byla otestována prostřednictvím několika experimentů provedených na testovacích účelových funkcích pro všechny algoritmy.

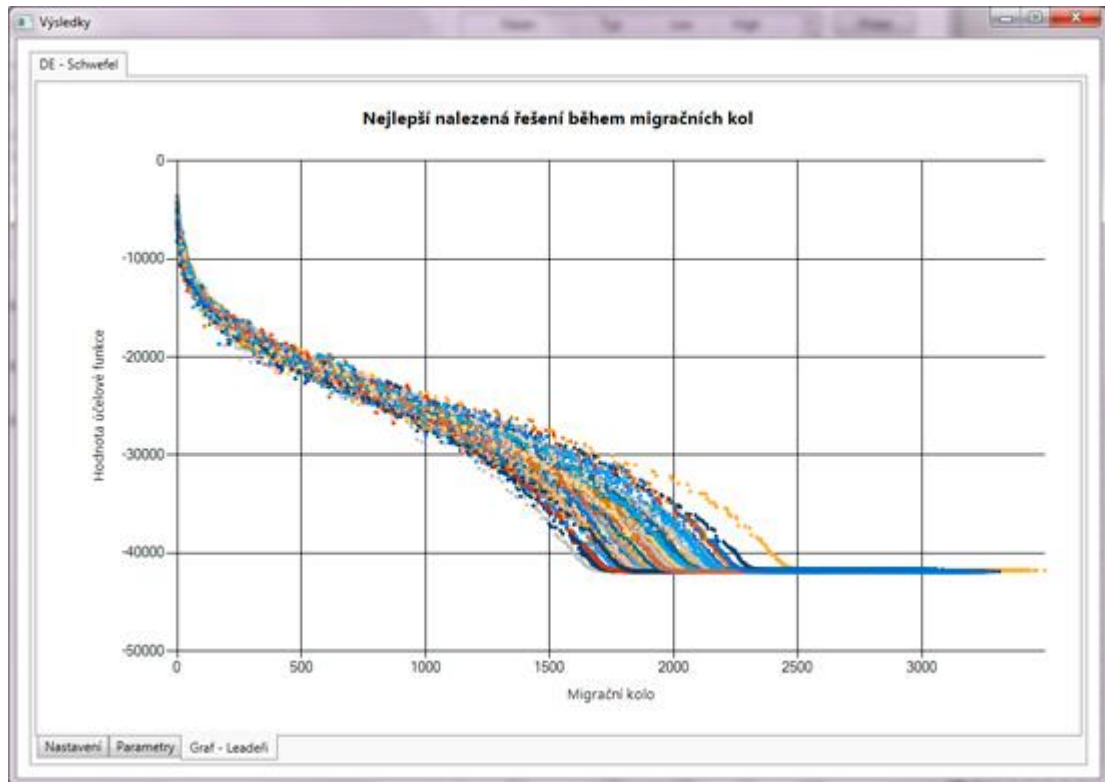
Nastavení parametrů a zvolená strategie SOMA a DE byly použity z testů provedených v [1], PSO bylo nastaveno podle doporučených rozsahů (Tab. 3) s tím, že počet migračních byl nastaven stejně jako u DE. Pro každý experiment bylo provedeno 100 opakování, ze kterých se vybralo vždy nejlepší řešení (Tab. 5).

Pro zajímavost, testy byly prováděny na počítači s procesorem Intel Core i5 s frekvencí jádra 3,20 GHz a doba jednoho experimentu, tedy 100 opakování pro stejné nastavení, se pohybovala v rozmezí 30 - 60 minut.

Tab. 5. Nalezené extrémy u testovacích funkcí

Testovací funkce	Vzdálenost od globálního extrému v %		
	SOMA	DE	PSO
1 <sup>st</sup> De Jong	0,01	0,053	0,024
3 <sup>rd</sup> De Jong	0,24	0,98	0,56
4 <sup>th</sup> De Jong	0,00081	1,77	0,0011
Ackley's function	0,012	$1,99 \times 10^{-15}$	0,1
Egg holder <sup>1</sup>	-58927,25	-29637,74	-37304,29
Griewangk's function	0,45	0	0,44
Masters's cosine wave function	5,84	26,68	12,83
Michalewicz's function	0,3	59,85	4,41
Pathological function <sup>1</sup>	32,37	38,36	33,56
Rana's function <sup>1</sup>	-24914,55	-18193,55	-22231,81
Rastrigin's function	0,051	0,043	0,2
Rosenbrock's saddle	0,026	0,026	0,025
Schwefel's function	0,00097	0,00097	17,56
Stretched V sine wave function	16,33	17,88	14,2
Test Function (Ackley)	1,44	1,72	1,3

<sup>1</sup> Vzdálenost v % nebyla určena, protože není znám globální extrém, a tak jsou uvedeny absolutní hodnoty.



Obr. 31. Grafické zpracování výsledků – DE/rand/1/bin, Schwefel’s function

	A	B	C	D	E	F	G	H	I	J
1	Nejlepší řešení během migračních kol									
2	Loop 1		Loop 2		Loop 3		Loop 4		Loop 5	
3	x	y [-]	x	y [-]	x	y [-]	x	y [-]	x	y [-]
4	0	742,5529	0	648,9134	0	697,4948	0	688,2503	0	697,5135
5	1	678,1586	1	603,965	1	627,8736	1	611,0908	1	663,6838
6	3	657,3125	2	557,1204	3	621,8592	4	610,8465	3	579,7525
7	4	643,8941	6	544,3849	4	601,3067	6	589,2276	4	517,8313
8	5	596,0589	7	533,0837	6	541,6208	7	541,1566	8	516,268
9	6	513,0346	9	514,5239	7	495,7347	8	535,4233	9	502,1598
10	7	487,9168	11	459,4399	9	458,9431	10	521,1687	10	497,9693
11	10	463,0661	15	441,8049	12	433,2026	11	481,1881	11	443,2584
12	15	443,8359	16	421,3022	14	426,7104	12	462,4413	12	406,622
13	16	417,373	17	406,7688	16	398,3543	13	457,939	14	403,2579
14	18	389,7932	18	385,7136	17	368,062	14	450,6951	17	348,8591
15	19	373,4591	20	362,6525	20	360,4539	15	428,6372	19	328,1171
16	20	372,9045	23	349,9093	22	339,016	17	362,7049	22	318,4888
17	21	353,136	25	348,1198	23	313,4558	18	345,4396	27	308,971

Obr. 32. Zobrazení části exportovaných výsledků v MS Excel 2007

## ZÁVĚR

Aplikace nabízí praktické řešení tří vybraných algoritmů: Samo-Organizující se Migrační Algoritmus (SOMA), Diferenciální evoluce (DE) a Rojení částic (PSO). Řešení je realizováno prostřednictvím .NET Framework s využitím grafického uživatelského rozhraní Windows Presentation Foundation.

Dle vlastního uvážení je možné zvolit hodnoty parametrů evolučních algoritmů a navíc vybrat a nastavit ukončovací kritérium. Vzorového jedince (specimen) lze vytvořit postupným přidáváním parametrů anebo hromadně z CSV nebo XML souboru.

Kromě několika přichystaných testovacích účelových funkcí je možné využít i vlastní, které však musí být ve formátu DLL knihoven.

Za velkou výhodu lze považovat jistou formu zautomatizovaného procesu, kdy lze zadat počet opakování pro zvolené nastavení. Výsledkem pak bude nejlepší nalezené řešení ze všech opakování. Za zmínku také stojí možnost výsledky exportovat ať už jako CSV soubory, nebo XML.

Samozřejmě vždy je co vylepšovat a jinak tomu není ani v tomto případě. U tvorby vzorového jedince (specimen), kde se aktuálně nachází dva možné typy parametrů a to celočíselné a reálné, by bylo možné doplnit další. Konkrétně by se jednalo o logické a především diskrétní hodnoty [4].

V případě samotného běhu optimalizačního procesu evolučních algoritmů by se dalo uvažovat nad vhodným způsobem zobrazení (grafickým, textovým) nejlepšího aktuálního řešení, které by již v průběhu optimalizace naznačilo uživateli, v jakém stádiu se nachází optimalizace definovaného problému.

Se stále více rozmáhajícími se vícejádrovými procesory se otevírají další možnosti, jak přistupovat k provádění výpočtů, ať už z pohledu rozdělení výpočtů účelové funkce do více vláken anebo paralelního provádění migračního mechanismu jedinců při hledání globálního extrému. U vícevláknového zpracování však bude nutné zajistit určitou nezávislost mezi výpočty, aby se nevytratil princip jednotlivých strategií a především účinnost evolučních algoritmů.

## ZÁVĚR V ANGLIČTINĚ

The application provides a practical solution to the three selected algorithms: Self-Organizing Migrating Algorithm (SOMA), Differential Evolution (DE) and Particle Swarm Optimization (PSO). The solution is carry through .NET Framework using the graphical user interface – Windows Presentation Foundation.

User can set parameters of evolutionary algorithms. User can also select the stopping criterion and set its conditions. The specimen can be created by consecutively adding parameters or by loading them from CSV or XML file.

Except a few prepared experimental cost functions, you can use your own. However it must be in DLL.

The great advantage is some form of automatic process. You can specify the number of loops for the chosen configuration and the result will be the best solution from all the loops. There is also possibility to export the results as CSV or XML files in this application.

Of course there is always room for improvement in any application. For example there are currently two possible types of parameters (integer and real) to create specimen in this one and it would be possible to add more. Concretely, these would be logical and especially discreet values [4].

There could be also considered to add representation (by graph or by text) of the best current solution in running optimization process of evolutionary algorithms which can indicate at what stage is currently the optimization of defined problem.

There are other ways of performing calculations with multi-core CPUs. For example there is possibility to distribute calculation of cost function in multithreaded processing or parallel implementation of the migration mechanism for individuals seeking of global extreme. There is also necessary to ensure some independence between the calculations, so it will not lose the principle of individual strategies and especially evolutionary algorithms efficiency in multithreaded processing.

**SEZNAM POUŽITÉ LITERATURY**

- [1] ZELINKA, Ivan. Umělá inteligence v problémech globální optimalizace. Praha : BEN, 2002. 192 s. ISBN 80-7300-069-5.
- [2] KVASNIČKA, V.; POSPÍCHAL, J.; TIŇO, P. Evoluční algoritmy. Bratislava : STU, 2000. 215 s. ISBN 80-227-1377-5.
- [3] PRICE, Kenneth V.; STORN, Rainer M.; LAMPINEN, Jouni A. Differential evolution: a practical approach to global optimization. Berlin : Springer, 2005. 538 s. ISBN 3-540-20950-6.
- [4] ONWUBOLU, Godfrey C.; DAVENDRA, Donald. Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization. Berlin : Springer, 2009. 211 s. ISBN 978-3-540-92150-9.
- [5] CHAKRABORTY, Uday K. Advances in Differential Evolution. Berlin : Springer, 2008. 338 s. ISBN 978-3-540-68827-3.
- [6] HU, Xiaohui. Particle Swarm Optimization: Tutorial [online]. 2006 [cit. 2011-05-09]. PSO Tutorial. Dostupné z WWW: <<http://www.swarmintelligence.org/tutorials.php>>.
- [7] PARSOPOULOS, Konstantinos E.; VRAHATIS, Michael N. Particle Swarm Optimization and Intelligence: Advances and Applications. [s.l.] : IGI Global snippet, 2010. 310 s. ISBN 978-1-61520-666-7.
- [8] ZIELINSKI, Karin; LAUR, Rainer. Stopping Criteria for a Constrained Single-Objective Particle Swarm Optimization Algorithm. Informatica : An International Journal of Computing and Informatics [online]. 2007, 31, 1, [cit. 2011-05-11]. Dostupný z WWW: <[http://www.informatica.si/PDF/31-1/16\\_Zielinski-Stopping%20Criteria%20for%20a%20Constrained...pdf](http://www.informatica.si/PDF/31-1/16_Zielinski-Stopping%20Criteria%20for%20a%20Constrained...pdf)>.
- [9] Overview of the .NET Framework [online]. Microsoft, c2011 [cit. 2011-05-25]. .NET Framework Conceptual Overview. Dostupné z WWW: <<http://msdn.microsoft.com/en-us/library/zw4w595w.aspx>>.
- [10] PETZOLD, Charles. Mistrovství ve Windows Presentation Foundation. Brno : Computer Press, 2008. 928 s. ISBN 978-80-251-2141-2.
- [11] CLERC, Maurice. Particle swarm optimization. London : ISTE, 2006. 243 s. ISBN 1-905209-04-5.

- [12] ZELINKA, Ivan, et al. Evoluční výpočetní techniky. Praha : BEN, 2008. 534 s. ISBN 978-80-7300-218-3.
- [13] FEOKTISTOV, Vitaliy. Differential evolution: in search of solutions. Berlin : Springer, 2006. 195 s. ISBN 0-387-36895-7.
- [14] ZELINKA, Ivan, et al. Evolutionary Algorithms and Chaotic Systems. Berlin : Springer, 2010. 521 s. ISBN 978-3-642-10706-1.
- [15] NASH, Trey. C# 2010 : Rychlý průvodce novinkami a nejlepšími postupy. Brno : Computer Press, 2010. 624 s. ISBN 978-80-251-3034-6.
- [16] WATSON, Ben. C# 4.0 : Řešení praktických programátorských úloh. Brno : Zoner Press, 2010. 656 s. ISBN 978-80-7413-094-6.
- [17] .NET Framework Class Library [online].Microsoft, c2011 [cit. 2011-05-25]. Dostupné z WWW: <<http://msdn.microsoft.com/en-us/library/gg145045.aspx>>.

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

API	Application Programming Interface
CF	Cost Function
CPU	Central Processing Unit
CV	Cost Value
CSV	Comma-Separated Values
DLL	Dynamic-Link Library
MFC	Microsoft Foundation Class
MSIL	Microsoft Intermediate Language
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

**SEZNAM OBRÁZKŮ**

Obr. 1. Znázornění populace.....	12
Obr. 2. Hodnoty parametru PathLength .....	14
Obr. 3. Migrace jedince k leaderovi po malém a velkém kroku Step .....	14
Obr. 4. Princip parametru MinDiv .....	16
Obr. 5. Strategie AllToOne.....	17
Obr. 6. AllToAll – migrace jedince.....	18
Obr. 7. Strategie AllToAll .....	19
Obr. 8. AllToAllAdaptive – migrace jedince .....	19
Obr. 9. Princip strategie AllToOne .....	21
Obr. 10. Princip binomiálního křížení [3] .....	24
Obr. 11. Princip exponenciálního křížení [3] .....	24
Obr. 12. Princip diferenciální evoluce – varianta DE/rand/1/bin .....	27
Obr. 13. Schéma provázanosti tříd algoritmů .....	36
Obr. 14. Abstraktní třída SOMAStrategy a odvozená strategie AllToOne .....	38
Obr. 15. Abstraktní třída DEStrategy a odvozená strategie DE/Rand/1/Exp.....	39
Obr. 16. Podrobné schéma tříd SOMA, DE a PSO.....	40
Obr. 17. Ukázka použití třídy Graph.....	44
Obr. 18. ListView seřazené vzestupně podle sloupce High .....	46
Obr. 19. Hlavní okno aplikace .....	48
Obr. 20. Hlavní nabídka menu .....	49
Obr. 21. Panel nástrojů .....	49
Obr. 22. Výběr algoritmu pomocí TabControl .....	51
Obr. 23. StatusBar .....	52
Obr. 24. Export výsledků.....	53
Obr. 25. Vzdálenost od extrému .....	53
Obr. 26. Struktura adresářů aplikace.....	54
Obr. 27. Volba a nastavení algoritmu.....	55
Obr. 28. Výběr a nastavení ukončovacího kritéria.....	55
Obr. 29. Parametry specimena .....	56
Obr. 30. Vytvoření projektu DLL knihovny ve Visual Studio 2010 .....	57
Obr. 31. Grafické zpracování výsledků – DE/rand/1/bin, Schwefel’s function .....	60
Obr. 32. Zobrazení části exportovaných výsledků v MS Excel 2007 .....	60

**SEZNAM TABULEK**

Tab. 1. Hodnoty parametrů SOMA [1] .....	17
Tab. 2. Hodnoty parametrů diferenciální evoluce [1] .....	23
Tab. 3. Hodnoty parametrů PSO [6] .....	30
Tab. 4. Psaní aplikací pro systém Windows pomocí jazyka založeného na jazyku C [10].	34
Tab. 5. Nalezené extrémny u testovacích funkcí .....	59

## SEZNAM PŘÍLOH

P I      CD-ROM