VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OPTIMALIZAČNÍ METODY PRO SIMLIB/C++

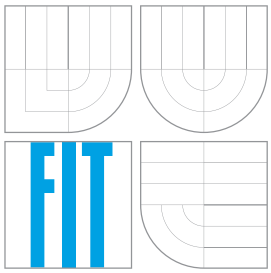
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN GODULA

BRNO 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

OPTIMALIZAČNÍ METODY PRO SIMLIB/C++

OPTIMIZATION METHODS FOR SIMLIB/C++

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN GODULA
AUTHOR

VEDOUCÍ PRÁCE — Dr. Ing. PETR PERINGER
SUPERVISOR

BRNO 2008

Abstrakt

Práca sa venuje metódam pre globálnu optimalizáciu. Teoreticky oboznamuje s pojmom optimalizácia, jej cieľom a popisuje zvolené optimalizačné metódy, navrhuje spôsob ich realizácie s ohľadom na kompatibilitu so simulačnou knižnicou SIMLIB/C++. Popisuje ich implementáciu v jazyku C++, demoštruje ich funkčnosť na rozdielne obtiažných testovacích problémoch a zhodnocuje ich úspešnosť.

Kľúčové slová

Optimalizácia, simulované žíhanie, Nelder-Meadova metóda, gradientná metóda, genetický algoritmus.

Abstract

Bachelor's thesis is concerned with methods for global optimization. It deals with concept of optimization theoretically, and describes its goals and chosen optimization methods, suggests process of their implementation with regard to compatibility with simulation library SIMLIB/C++. It describes their implementation in C++ language and demostrates their functionality on different test problems and evaluates their rate of success.

Keywords

Optimization, simulated annealing, Nelder-Mead method, gradient method, genetic algorithm.

Citácia

Martin Godula: Optimalizační metody pro SIMLIB/C++, bakalárska práca, Brno, FIT VUT v Brne, 2008

Optimalizační metody pro SIMLIB/C++

Prehlásenie

Prehlasujem, že som túto bakalárskou prácu vypracoval samostatne pod vedením pana Dr. Ing. Petra Peringera. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Martin Godula
13. mája 2008

Pod'akovanie

Týmto by som chcel pod'akovat' Dr. Ing. Petrovi Peringerovi za poskytnutú pomoc a konzultácie pri tvorbe tejto práce.

Obsah

Kapitola 1

Úvod

Techniky operačnej analýzy sú každodenne využívané na získanie čo najviac efektívnych riešení problémov ako časové plánovanie, alokácia zdrojov, priemyselné plánovanie, maximalizácia zisku a mnohých iných. Za týmto účelom sú využívané rôzne optimalizačné metódy, ktoré sú často inšpirované nahliadnutím do úplne iných oborov alebo procesmi bežne pozorovateľnými v prírode. Príkladmi môžu byť veľmi populárne optimalizačné metódy ako *simulované žíhanie* so základom v metalurgii alebo *genetický algoritmus* inšpirovaný procesom evolúcie.

Cieľom práce je navrhnutie rozšírenia optimalizačnej časti simulačnej knižnice SIMLIB/C++ o optimalizačné metódy simulovaného žíhania, Nelder-Meadovej metódy a gradientnej metódy. Pre tieto metódy je podrebné navrhnúť hierarchiu tried s využitím vhodných návrhových vzorov. Výsledné riešenie je nutné otestovať na vybraných testovacích príkladoch.

Nasledujúca kapitola je venovaná úvodu do problematiky. Zoznamuje so základnými pojmami ako operačná analýza a optimalizácia. Zaoberá sa rozborom zvolených optimalizačných metód pre neobmädzenú optimalizáciu. Ďalej sa zameriava na obmädzené optimalizačné problémy a ich riešenie prostredníctvom penalizačnej metódy. Venuje sa návrhovým vzorom, ich významu a oboznamuje so základnými návrhovými vzormi, ktoré sú zaradené do príslušných kategórií. V závere kapitoly je stručne predstavená simulačná knižnica SIMLIB/C++. Kapitola 3 sa venuje rozboru problému, na základe ktorého je vybraný vhodný návrhový vzor, pomocou ktorého je navrhnutá hierarchia tried reprezentujúcich jednotlivé metódy. Kapitola 4 predstavuje rozhrania tried pre jednotlivé metódy, popisuje spôsob implementácie samostatných metód, pričom priestor je venovaný najmä častiam líšiacim sa od štandardnej implementácie a podproblémom, ktoré je potrebné prispôsobiť cieľu využitia metód. Ďalej popisuje riadiace parametre jednotlivých optimalizačných metód. Kapitola 5 je venovaná testovacím úlohám, na ktorých boli implementované metódy otestované, graficky zobrazuje výsledky testov a rozoberá dosiahnuté výsledky. Záverečná kapitola sa následne venuje zhrnutiu celej práce ako celku. Rozoberá dosiahnuté výsledky a prezentuje námety na budúci vývoj projektu.

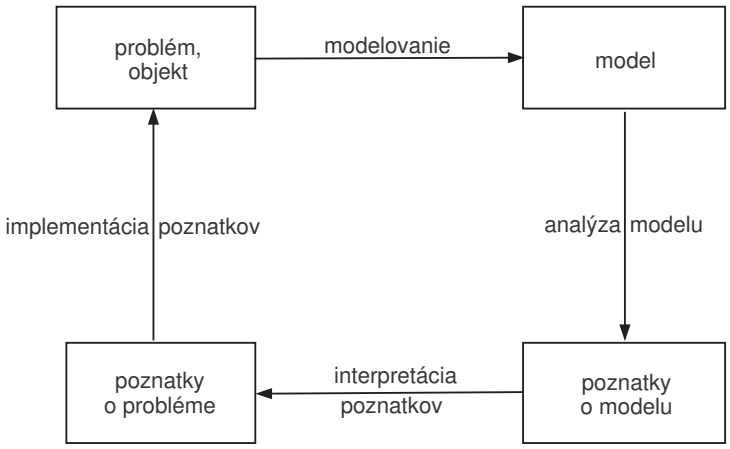
Kapitola 2

Uvedenie do problematiky

Cieľom tejto kapitoly je oboznámenie s pojmami operačná analýza, obmädzená a neobmädzená optimalizácia. Ďalej sa zameriava na popis zvolených optimalizačných metód pre jednotlivé typy optimalizácie a ich základné princípy. Venuje sa úvodu do návrhových vzorov, ich významu a kategorizácie, pričom sú tu predstavené a zaradené základné najčastejšie používané návrhové vzory. V závere je stručne predstavená simulačná knižnica SIMLIB/C++.

2.1 Operačná analýza

Operačná analýza [3] je veda zaoberajúca sa spôsobom formulovania matematických modelov komplexných inžinierskych problémov a spôsobom ich analyzovania s cieľom získania náhľadu na ich možné riešenia. Tieto problémy sú väčšinou spojené s optimalizovaním maxima (zisk, využitie prostriedkov) alebo minima (náklady, znižovanie rizika). Zámerom využívania operačnej analýzy je získanie najlepšieho možného riešenia matematicky, pričom jeho aplikácia v praxi má za následok zlepšenie výkonu daného systému.



Obrázok 2.1: Proces operačnej analýzy

2.2 Optimalizácia

Optimalizácia [3] je veda zaoberajúca sa štúdiom problémov, v ktorých je snaha o nájdenie maximálnej alebo minimálnej hodnoty určitej reálnej funkcie.

$$\text{min, max} \quad f(x_1, \ldots, x_n)$$

$$\text{za podmienok:} \quad g_i(x_1, \ldots, x_n) \begin{Bmatrix} \leq \\ = \end{Bmatrix} 0 \quad i = 0, \ldots, n$$

Optimalizačný problém sa teda typicky skladá z troch častí:

- **Objektívna funkcia** $f(x_1, \ldots, x_n)$, ktorú sa snažíme minimalizovať alebo maximalizovať. Takmer všetky optimalizačné problémy majú iba jednu objektívnu funkciu. Menej častými prípadmi sú problémy bez objektívnej funkcie, ktoré sa nazývajú *realizovateľné problémy* (cieľom je iba nájdeme hodnôt premenných, ktoré spĺňajú obmädzujúce podmienky), alebo problémy s viacerými objektívnymi funkciami (ktoré sa však dajú preformulovať pomocou váženej kombinácie jednotlivých objektívnych funkcií alebo nahradením niektorých z nich obmädzujúcimi podmienkami).
- **Súbor rozhodovacích premenných** $x_1, \ldots, x_n$, na ktorých je závislá hodnota objektívnej funkcie.
- **Súbor obmädzujúcich podmienok** $g_i, i = 0 \ldots n$, ktoré však nie sú nevyhnutnou súčasťou optimalizačného problému. Na základe prítomnosti alebo neprítomnosti obmädzujúcich pomienok sa optimalizačné problémy delia na dve skupiny (obmädzené a neobmädzené). Tieto podmienky definujú aké hodnoty môžu premenné objektívnej funkcie nadobúdať. Ak výber hodnôt rozhodovacích premenných splňuje všetky obmädzujúce podmienky hovorí sa o takzvanom realizovateľnom riešení. Optimálne riešenie je následne realizovateľným riešením, pre ktoré je hodnota objektívnej funkcie aspoň tak dobrá ako ľubovoľného iného realizovateľného riešenia.

2.3 Metódy pre neobmädzenú optimalizáciu

Postupne budú predstavené metóda simulovaného žíhania, Nelder-Meadova metóda, gradientná metóda a genetický algoritmus. Bude venovaný priestor ich základným princípom, budú rozobrané podproblémy jednotlivých metód, stručne popísané ich algoritmy a matematické výpočty súvisiace s nimi.

2.3.1 Simulované žíhanie

Metóda simulovaného žíhania [3] patrí medzi stochastické optimalizačné algoritmy, ktoré majú základ vo fyzike. Je založená na simulácii procesu ohrievania a následného kontrolovaného ochladzovania kovu. Atómy kovu sa zohriatím dostanú z pôvodného stavu (lokálneho minima vnútornej energie). Postupným ochladzovaním sa atómy usadzujú do čo najlepšej polohy s čo najnižšou energiou. Výsledkom tohoto procesu je dosiahnutie lepšej štruktúry a vlastností kovu.

Na začiatku je náhodne vybrané určité počiatočné riešenie. V každom ďalšom kroku simulovaného žíhania je snaha o nahradenie aktuálneho riešenia náhodným susedným riešením. Okrem akceptovania lepších riešení metóda simulovaného žíhania umožňuje aj

akceptovanie horšieho riešenia s určitou pradepodobnosťou závisiacou od aktuálnej teploty (s postupne klesajúcou teplotou šanca na akceptovanie horšieho riešenia klesá). Akceptácia horších riešení umožňuje algortimu uniknutie z lokálneho minima.

Chladiaci rozvrh

Jeho úlohou je znižovnie teploty algortimu. Existuje mnoho spôsobov jeho realizácie pričom najviac používaný je exponenciálny chladiaci rozvrh:

$$T_{k+1} = \alpha T_k$$

kde $0 < \alpha < 1$ je *chladiaca miera* (odporúčané hodnoty sú 0.8 až 0.9). Vo všeobecnosti pomalé chladiace rozvrhy majú väčšiu šancu na nájdenie globálneho optima, ale cenou za to je omnoho väčšia časová náročnosť algoritmu. Ďalšími parametrami, ktoré riadia chladiaci rozvrh, sú počiatočná teplota T_{max} a minimálna teplota T_{min}.

Generovanie susedného stavu

Hrá obrovskú úlohu vzhľadom na celkový výkon algoritmu. Veľkosť množiny, z ktorej je nový stav generovaný, by mala byť dostatočne veľká nato, aby umožnila rýchly pohyb k lepším stavom a súčasne by nemala byť príliš veľká nato, aby zapríčinila náhodný pohyb v prehľadávanom priestore. Bežne používanou technikou je generovanie riadené teplotou, čo znamená, že veľkosť množiny nových stavov sa s klesajúcou teplotou zmenšuje.

Akceptácia nového riešenia

Vychádza z jednoduchého algoritmu Metropolis, ktorý môže byť využitý na efektívne simulovanie skupiny atómov vo vyváženom stave pri danej teplote. V každom kroku tohto algoritmu je každému atómu dané malé náhodné posunutie a je vypočítaná výsledná zmena v energii ΔE. Ak $\Delta E \leq 0$, posunutie je akceptované a konfigurácia je použitá ako počiatočný stav pre nasledujúci krok algoritmu. Prípad $\Delta E > 0$ je riešený pravdepodobnostne, čiže pravdepodobnosť, že konfigurácia bude prijatá je $P(\Delta E)e^{-\Delta E/k_B T}$. Náhodné uniformne distribuované číslo v intervale $(0, 1)$ je porovnané s $P(\Delta E)$. Ak je náhodné číslo menšie, nová konfigurácia je akceptovaná, v druhom prípade je použitá originálna konfigurácia ako počiatočná pre nasledujúci krok. Opakovaním tohoto jednoduchého kroku je simulovaný tepelný pohyb atómov pri teplote T. Použitím objektívnej funkcie na mieste energie a reprezentovaním konfigurácie súborom parametrov x_i, je možné generovať postupnosť riešení optimalizačného problému pri určitej teplote [8].

Algoritmus simulovaného žíhania

1. **Inicializácia.** Voľba ľubolného počiatočného riešenia x_0 a nastavanie teploty na počiatočnú hodnotu $T > 0$.
2. **Markovov reťazec** o dĺžke N.
   (a) **Generovanie nového stavu** zo stavu aktuálneho a aktuálnej teploty.
   (b) **Akceptácia nového stavu.** V prípade ak vylepšuje stav predchádzajúci, inak s pravdepodobnosťou $e^{f(x_t)-f(x_{t+1})/T}$
3. **Zníženie teploty** podľa daného chladiaceho rozvrhu.

4. **Ukončovacie podmienky.** Ak teplota dosiahla najnižšiu možnú hodnotu, alebo aktuálne riešenie je dostatočne presné.

2.3.2 Metóda Nelder-Mead

Je algoritmom pre riešenie nelineárnych optimalizačných problémov v mnohorozmernom priestore, prvýkrát publikovaná v roku 1965 J. A. Nelderom a R. Meadom. Metóda pre svoju činnosť využíva len porovnávanie hodnôt objektívnej funkcie a nevyžaduje žiadne informácie o deriváciach.

Pri optimalizácii problému s n riadiacimi premennými si Nelder-Meadova metóda [3, 1] uchováva súbor o veľkosti $n+1$ odlišných riešení $y_1 \dots y_{n+1}$, pričom riešenie y_1 obsahuje najlepšiu hodnotu objektívnej funkcie, y_2 druhú najlepšiu a tak ďalej. S každou iteráciou je snaha o nahradenie najhoršieho riešenia y_{n+1} lepším. Kritickým bodom akejkoľvek optimalizačnej metódy nevyužívajúcej derivácie je spôsob získania smeru hľadania lepšieho riešenia bez pomoci parciálnych derivácií. Metóda Nelder-Mead tento problém rieši spôsobom vzdiaľovania sa aktuálne najhoršiemu riešeniu smerom k ostatným. Metóda je riadená štyrmi parametrami: α pre reflexiu, β pre kontrakciu, γ pre expanziu a δ pre priblíženie. Tieto parametre by mali vyhovovať nasledujúcim podmienkam:

$$\alpha > 0, \quad 0 < \beta < 1, \quad \gamma > 1, \quad \gamma > \alpha, \quad 0 < \delta < 1$$

Algoritmus metódy Nelder-Mead

1. **Inicializácia.** Náhodný výber $n+1$ rôznych riešení $\{y_1, \dots, y_{(n+1)}\}$, vyhodnotenie $\{f(y_1), \dots, f(y_{(n+1)})\}$, inicializácia indexu iterácií $t \leftarrow 0$.

2. **Centroid.** Zoradenie súboru riešení y_i do nezlepšujúceho sa radu s ohľadom na hodnotu ohodnocovacej funkcie, následne výpočet centroidu z najlepších n riešení:

$$c = \frac{1}{n} \sum_{i=1}^{n} y_i$$

3. **Ukončovacie podmienky.**

   - Ak sú všetky hodnoty objektívnej funkcie pre riešenia $\{f(y_1), \dots, f(y_{(n)})\}$ dostatočne blízko k objektívnej hodnote centroidu $f(c)$.
   - Ak sú všetky body $\{y_1, \dots, y_n\}$ dostatočne blízko u seba.
   - Ak počet iterácii dosiahol maximálnu možnú hranicu.

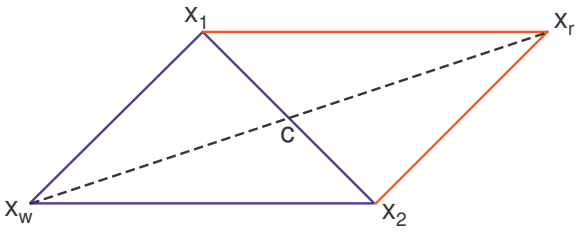
4. **Smer pohybu.** Výpočet smeru pohybu od bodu s najhorším ohodnotením s využitím centroidu:

$$\Delta x \leftarrow c - y_{n+1}$$

5. **Reflexia (obrázok 2.2).** Výpočet reflexného bodu:

$$x_r = c + \alpha \Delta x$$

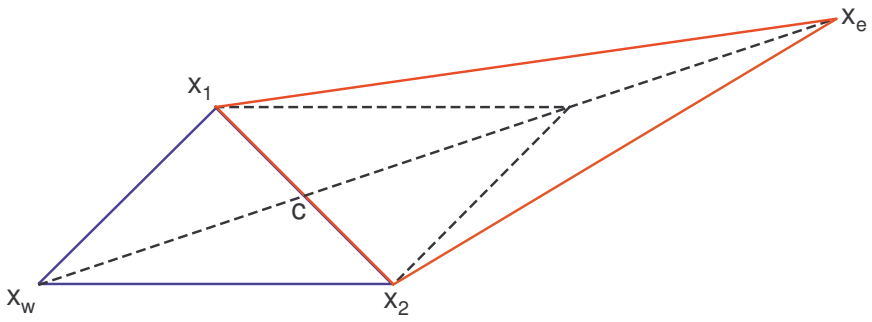
   Ak $f(x_r) \leq f(x_1)$ algoritmus prejde na krok 6 (Expanzia). Ak $f(x_r) \geq f(x_n)$ nasleduje krok 7 (Kontrakcia). V ostatných prípadoch je reflexný bod akceptovaný $y_{n+1} = x_r$ a nastáva prechod na novú iteráciu.



Obrázok 2.2: Reflexia. Reflexný bod x_r, centroid c, najhoršie ohodnotený bod x_w, dva najlepšie ohodnotené body x_1, x_2

6. **Expanzia (obrázok 2.3).** Výpočet expanzného bodu:

$$x_e = c + \gamma \Delta x$$



Obrázok 2.3: Expanzia. Expanzný bod x_e, centroid c, najhoršie ohodnotený bod x_w, dva najlepšie ohodnotené body x_1, x_2

V prípade, že $f(x_e) \leq f(x_r)$, je akceptovaný expanzný bod x_e, inak je akceptovaný reflexný bod x_r.

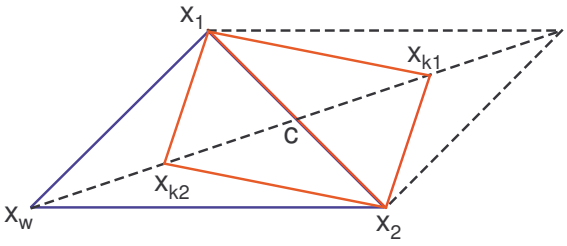
7. **Kontrakcia (obrázok 2.4).** Výpočet kontrakčného bodu s využitím jednej z nasledujúcich možností:

   - Ak $f(x_r) \leq f(y_{n+1})$, $x_c = c + \beta \Delta x$
   - Ak $f(x_r) \geq f(y_{n+1})$, $x_c = c - \beta \Delta x$

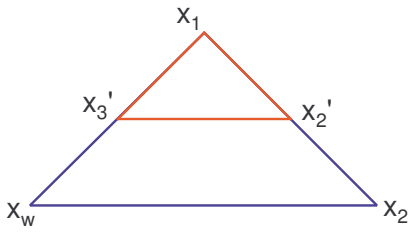
   Ak niektorá z týchto možností je lepšia ako aktuálna najhoršia, je akceptovaná a prejde sa na nasledujúcu iteráciu.

8. **Priblíženie (obrázok 2.5).** Posun aktuálnych riešení smerom k aktuálne najlepšiemu y_1:

$$y_1 \leftarrow \delta(y_1 + y_i) \quad \text{pre všetky} \quad i = 2, \ldots, n+1$$



Obrázok 2.4: Kontrakcia. Vonkajší kontrakčný bod x_{k1}, vnútorný kontrakčný bod x_{k2}, centroid c, najhoršie ohodnotený bod x_w, dva najlepšie ohodnotené body x_1, x_2



Obrázok 2.5: Priblíženie. Najhoršie ohodnotený bod x_w, dva najlepšie ohodnotené body x_1, x_2 a x_2', x_3' sú priblížené body x_2, x_w

2.3.3 Gradientná metóda

Gradientná metóda [3] je optimalizačnou metódou, ktorej princíp spočíva v odvodení smeru prehľadávania z gradientu v aktuálnom bode. Hoci sa gradientná metóda radí medzi priamé, nie je pre väčšinu príkladov veľmi efektívna. Metóda poskytuje dostatočne dobrý počiatočný progres, avšak s približovaním sa k stacionárnemu bodu sa metóda stáva pomalou a neposkytuje ideálne výsledky pre väčšinu nelineárnych problémov.

Algoritmus gradientnej metódy

1. **Inicializácia.** Voľba ľubovoľného počiatočného riešenia x_0, nastavenie presnosti ϵ, inicializácia indexu iterácií $t \leftarrow 0$.

2. **Gradient.** Výpočet gradientu objektívnej funkcie $\Delta f(x_t)$ v aktuálnom bode x_t.

$$\| \Delta f(x_t) \| = \sqrt{\sum_j \frac{\partial f}{\partial x_j}}$$

3. **Ukončovacie podmienky:**

   - Ak normovaný gradient $\| \Delta f(x_t) \| < \epsilon$
   - Ak bol dosiahnutý maximálny možný počet iterácií riešenia $t = t_{max}$.

4. **Výpočet** λ_{t+1} pomocou jednosmerného lineárneho prehľadávania:

$$\min f(x_t + \lambda \Delta x_{t+1})$$

5. **Nový bod** x_{t+1} a prechod na novú iteráciu (krok 2) $t \leftarrow t + 1$.

$$x_{t+1} = x_t + \lambda \Delta x_{t+1}$$

2.3.4 Genetický Algoritmus

Genetický algoritmus [5] je technika využívaná na hľadanie riešení optimalizačných problémov, inšpirovaná evolučnou biológiou. Genetický algoritmus je založený na uchovávaní populácie možných riešení, ktorá je postupne iteratívne podrobovaná genetickým operáciam selekcie, kríženia a mutácie s cieľom vytvorenia novej generácie populácie s lepšími vlastnosťami (kondíciou) pre nasledujúcu iteráciu. Členovia s lepšou kondíciou majú väčšiu šancu na prežitie ako členovia s menšou. Genetický algortimus je metastratégiou, pričom riešenie jednotlivých podproblémov algoritmu je otvorené a potrebné ho prispôsobiť konkrétmenu problému pre dosiahnutie optimálnych výsledkov.

Selekcia

Je výber časti populácie pre vytvorenie novej generácie. Pravdepodobnosť výberu určitého jednotlivca je závislá od jeho kondície. Existuje množstvo rôznych metód pre realizáciu selekcie. Jednou z najbežnejšie používaných je takzvaná *ruletová metóda*:

$$p_i = fit_i / fit_{pop}$$

Kríženie

Je kombináciou páru rodičovských riešení s cieľom vytvorenia páru potomkov. Nasledujúci príklad demonštruje kríženie dvoch rodičov, reprezentovaných binárnymi vektormi, pomocou ich rozdelenia na rovnakom mieste a následným spojením prvej časti vektoru jedného rodiča s druhou časťou druhého a naopak.

$$\begin{aligned} parent_1 &= (1,0,0,0, \;|\; 1,1,0,1) \\ parent_2 &= (1,1,0,1, \;|\; 0,0,0,1) \\ child_1 &= (1,0,0,0, \;|\; 0,0,0,1) \\ child_2 &= (1,1,0,1, \;|\; 1,1,0,1) \end{aligned}$$

Mutácia

Genetická operácia uskutočnujúca čiastočnu zmenu na určitom riešení. Nastáva s určitou pravdepodobnosťou (definovanou pri implementácii algoritmu).

$$(1,0,0,0,1,1,0,1)$$
$$(1,1,0,0,1,1,0,1)$$

Postup jednoduchého genetického algoritmu

1. **Štart**. Generovanie náhodnej populácie o veľkosti n chromozónov (vhodných riešení pre problém).
2. **Kondícia**. Vyhodnotenie kondície $f(x)$ každého chromozónu x populácie.
3. **Nová populácia**. Vytvorenie novej polpulácie opakovaním nasledujúcich krokov, pokiaľ nie je nová populácia kompletná.
   (a) **Selekcia**. Výber dvoch rodičovských chromozónov z populácie podľa ich kondície (lepšia kondícia znamená vyššiu pravdepodobnosť výberu).
   (b) **Kríženie**. S pravdepodobnosťou kríženia je vykonané kríženie rodičov s cieľom sformovania potomstva. Ak nebolo kríženie, vykonané potomkovia sú presné kópie rodičov.
   (c) **Mutácia**. Vykonanie mutácie na potomstve s pravdepodobnosťou mutácie.
   (d) **Akceptácia**. Prijatie nového potomstva do populácie.
4. **Nahradenie**. Použitie novovygenerovanej populácie pre ďalší chod algoritmu.
5. **Ukončujúce podmienky**. Ak sú ukončujúce podmienky splnené, je chod algoritmu ukončený a výsledkom je najlepšie riešenie v aktuálnej populácii.
6. **Nová iterácia**. Prechod na novú iteráciu (krok 2).

2.4 Obmädzená optimalizácia

Sa zaoberá optimalizačnými problémami, kde okrem súboru rozhodovacích premenných a objektívnej funkcie do hry vstupuje aj súbor obmädzujúcich podmienok. Cieľom takejto optimalizácie nie je iba nájdenie takej konfigurácie parametrov, pre ktoré objektívna funkcia dosahuje optimálnej hodnoty, ale je nutné aj splnenie obmädzujúcich podmienok. Existuje viacero metód pre riešenie tohoto typu optimalizácie, napríklad metóda Lagrangeových násobiteľov, penalizačné metódy, bariérové metódy.

2.4.1 Penalizačná metóda

Penalizačná metóda [3] je algoritmom určeným na riešenie obmädzených optimalizačných problémov. Je založená na transformácii obmädzujúcich podmienok na sériu neobmädzených optimalizačných problémov, ktorých postupné riešenie konverguje ku globálnemu optimu. Takto predefinovaný optimalizažný problém je možné riešiť optimalizačnými metódami spomínamými v predchádzajúcej sekcii. Metóda potrebuje pre svoju činnosť vytvorenie takzvanej penalizačnej funkcie, ktorej hodnota je závislá na splnení obmädzujúcich podmienok. Ak sú všetky splnené jej hodnota je nulová, avšak s mierou nesplnenia týchto podmienok jej hodnota narastá. Hodnota penalizačnej funkcie je násobená penalizačným násobiteľom a následne pripočítaná k objektívnej funkcii. Tento súčet sa stáva objektívnou funkciou pre sériu optimalizácií bez obmädzujúcich podmienok, pričom hodnota penalizačného násobiteľa záčína na relatívne malej hodnote a postupne sa zvyšuje. V dôsledku takéhoto postupu je vplyv penalizačnej funkcie na celkovú objektívnu funkciu v počiatočných iteráciach malý, čím je umožnené nájdenie optima, ktoré však nemusí spĺňať obmädzujúce podmienky. V ďalších iteráciach sa vplyv penalizačnej funkcie zvyšuje, čo má za následok

posun riešenia v smere k splneniu obmädzujúcich podmienok. Toto je dosiahnuté po určitom množstve iterácií pri dostatočne veľkej hodnote penalizačného násobiteľa, samozrejme len v prípade ak je vôbec možné splnenie obmädzujúcich podmienok.

Matematický popis penalizačnej metódy

$$\min, \max F(x) = f(x) \pm \mu \sum_i p_i(x)$$

kde + pre minimalizáciu, − pre maximalizáciu, penalizačný násobiteľ $\mu > 0$ a p_i sú funkcie zodpovedajúce jednotlivým obmädzujúcim podmienkam, pričom ak je podmienka splnená $p_i(x) = 0$, inak $p_i(x) > 0$.

Tvorba penalizačnej funkcie

$$\begin{array}{lrcll} \text{Pre} & g_i(x) & \leq & b_i & \max\{0, g_i(x) - b_i\}, \quad \max^2\{0, g_i(x) - b_i\} \\ \text{Pre} & g_i(x) & \geq & b_i & \max\{0, b_i - g_i(x)\}, \quad \max^2\{0, b_i - g_i(x)\} \\ \text{Pre} & g_i(x) & = & b_i & |b_i - g_i(x)|, \quad |b_i - g_i(x)|^2 \end{array}$$

2.5 Návrhové vzory

Návrhové vzory [2] pomenúvajú, abstrahujú a identifikujú kľúčové špecifiká všeobecných návrhových štruktúr s cieľom vytvorenia znovupoužiteľného objektovo-orientovaného návrhu. Návrhový vzor identifikuje zúčastnené triedy, inštancie a ich úlohy, spoluprácu a rozdelenie zodpovedností.

Vo všeobecnosti je návrhový vzor tvorený štyrmi základnými elementami:

- Názov vzoru, ktorý umožňuje zvládnutie popisu návrhového problému, jeho riešení a následkov jedným alebo dvoma slovami.
- Problémom popisované situácie, v ktorých je vhodné použitie daného návrhového vzoru.
- Riešenie popisujúce elementy, ktoré vytvárajú návrh, ich vzťahy, zodpovednosti a spoluprácu. Riešenie nepopisuje konkrétny návrh alebo implementáciu, pretože návrhový vzor je niečo ako šablóna, ktorá môže byť použitá v mnohých rôznych situáciách. Miesto toho návrhový vzor poskytuje abstraktný popis návrhového problému a ako ho všeobecne pripravené triedy a objekty riešia.
- Následky, ktoré sú výsledkom výhod a nevýhod aplikácie určitého návrhového vzoru, hrajú dôležitú úlohu pri porovnávaní alternatívnych návrhov.

Návrhové vzory sú klasifikované podľa dvoch kritérií [2]. Prvý je účel návrhového vzoru, vyjadrujúci činnosť, ktorú vzor vykonáva. Na základe tohoto triedenia môžu mať návrhové vzory vytvárací, štrukturálny alebo behaviorálny účel. Druhým kritériom delenia je priestor aplikácie návrhového vzoru, ktorý špecifikuje jeho zameranie buď na triedy alebo objekty. Triedne návrhové vzory riešia statické vzťahy, vytvárané dedičnosťou, medzi triedami a ich podtriedami. Objektové návrhové vzory sú zamerané na vzťahy medzi objektami, ktoré sa môžu za behu meniť a sú viacej dynamické. Väčšina návrhových vzorov patrí do tejto kategórie.

Vytváracie návrhové vzory

Venujú sa procesu vytvárania objektu. Časť tohoto procesu presúvajú na triedu alebo objekt v závislosti na tom či sú tieto návrhové vzory triedne (*Factory Method*) alebo objektové (*Abstract Factory, Builder, Prototype, Singleton*).

Štrukturálne návrhové vzory

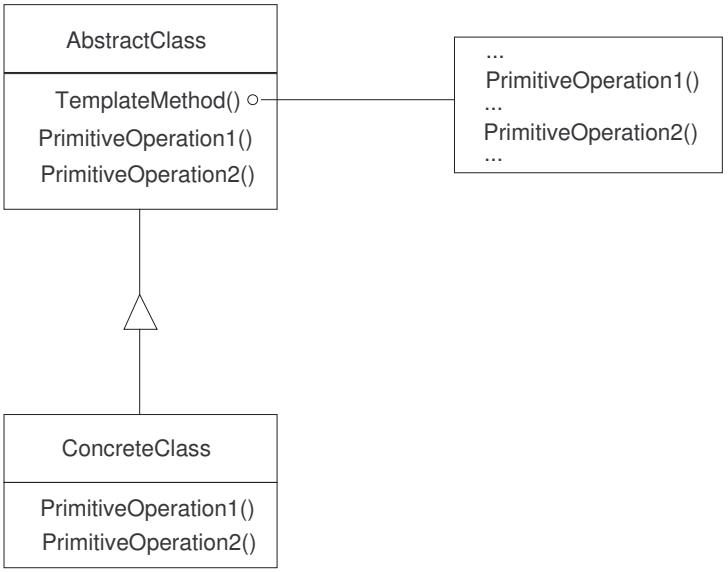
Triedne štruktuálne vzory využívajú dedičnosť na tvorenie odvodených tried (*Adapter*), zatiaľ čo objektové popisujú spôsoby zostavenia objektov (*Adapter, Bridge, Composite, Decorator, Facade, Proxy*).

Behaviorálne návrhové vzory

Triedne behaviorálne vzory využívajú dedičnosť na popis algoritmov a kontroly ich toku (*Interpreter, Template Method*). Objektové následne definujú spoluprácu skupiny objektov a realizáciu úlohy, ktorú žiadny samostaný objekt nedokáže vykonať (*Chain of Responsibility, Command, Iterator, Mediator, Memento, Flyweight, Observer, State, Strategy, Visitor*).

Návrhový vzor *Template method*

Úmyslom tohoto návrhového vzoru, patriaceho do skupiny behaviorálnych návrhových vzorov, je vytvorenie triedy reprezentujúcej kostru určitého algoritmu, ktorá bude dereferovať niektoré kroky algoritmu podtriedam. Návrhový vzor *Template pattern* [2] umožňuje redefiníciu určitých podkrov algoritmu bez zmeny jeho štruktúry.



Obrázok 2.6: Diagram návrhového vzoru *Template method*. Trieda *AbstractClass* definuje metódu *TemplateMethod* tvoriacu kostru algoritmu a abstraktné primitívne operácie, implementované v podtriedach *ConcreteClass*.

Prípady použitia návrhového vzoru *Template method*:

- Implementácia stálej časti algortimu raz a implementáciu častí, ktoré sa môžu líšit', prenechat' na podtriedy.
- Implementácie spoločného správania podtried do jednej spoločnej triedy s cieľom zamädzenia duplikácie zdrojového kódu.
- Obmädzenie vytvárania podtried len na určité prípady.

2.6 SIMLIB/C++

SIMLIB/C++ [7] je simulačnou knižnicou pre jazyk C++, vyvíjaná na Fakulte Informačních technológií, Vysokého Učení Technického v Brne od roku 1991. Zdrojové kódy sú lincencované pod GNU LGPL. Knižnica umožňuje vytváranie simulačných modelov priamo v jazyku C++ s využitím predefinovaných simulačných nástrojov poskytovaných knižnicou. Umožňuje objektovo-orientovaný popis modelov založených na simulačnej abstrakcii. Súčasná verzia umožňuje popis spojitých, diskrétnych, kombinovaných, 2D/3D vektorových a fuzzy modelov.

Kapitola 3

Návrh tried optimalizačných metód

Táto kapitola popisuje objektovo orientovaný návrh tried pre optimalizačné metódy simulovaného žíhania, metódy Nelder-Mead gradientnej metódy a genetického algoritmu. Návrh vychádza z návrhového vzoru *Template method*. Nasledujúce časti kapitoly sa venujú rozboru problému a jeho zvolenému riešeniu.

3.1 Rozbor problému

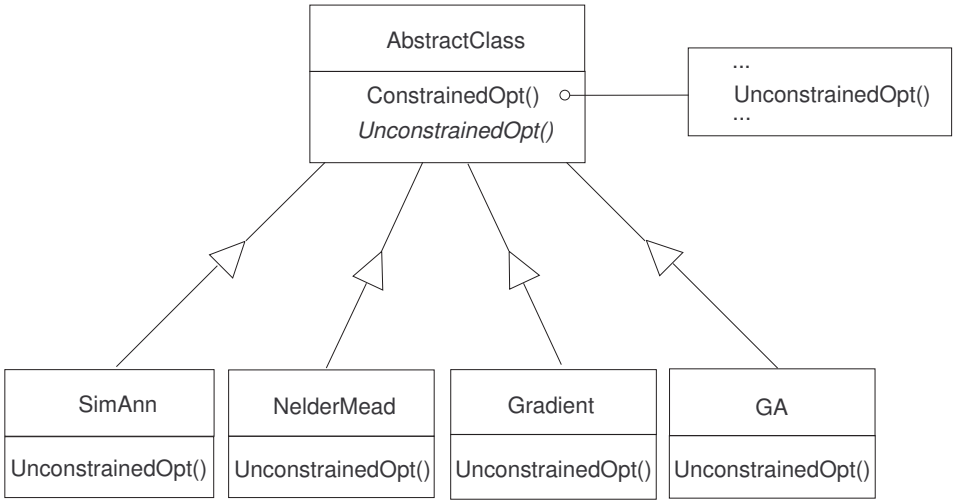
Je potrebné, aby triedy pre zvolené optimalizačné metódy spĺňali tieto požiadavky:

- Riešenie neobmädzeného optimalizačného problému, v každej triede realizované príslušnou metódou.
- Riešenie obmädzeného optimalizačného problému penalizačnou metódou, pričom podúlohou tejto metódy je aj neobmädzený optimalizačný problém. Táto časť je pre všetky triedy zhodná s rozdielom riešenia spomínanej podúlohy, ktorá zodpovedá príslušným metódam.

S požiadavkov je možné usúdiť, že by bolo vhodné vytvorenie abstraktnej triedy realizujúcej výpočet penalizačnej metódy a podtried, z nej vychádzajúcich, implementujúcich samostatné metódy pre neobmädzenú optimalizáciu. Takýto postup zodpovedá návrhovému vzoru *Template method*.

3.2 Navrhnutá hierarchia tried

Na základe návrhového vzoru *Template method* je navrhnutá abstraktná trieda *AbstractMethod* definujúca kostru pre jednotlivé optimalizačné problémy. Táto trieda implementuje obmädzenú optimalizáciu s využitím penalizačnej metódy. Navyše deklaruje rozhranie pre neobmädzenú optimalizáciu, ktorá je zároveň aj podúlohou využívanou algoritmom penalizačnej metódy. Implementáciu tejto podúlohy prenechá triedam, odvodeným z abstraktnej triedy *AbstractMethod*.



Obrázok 3.1: Diagram navrhnutej hierarchie tried.

3.3 Triedy pre genetický algoritmus

Táto časť textu sa venuje návrhu tried pre genetický algoritmus, ktorý nie je súčasťou zadania práce. Z dôvodu veľkej popularity tejto metódy však boli navrhnuté jednoduché triedy, ktoré by mohli túto metódu realizovať.

Trieda pre implementáciu genetického algoritmu sa vytvorí podobne ako u ostatných metód odvodením od abstraktnej triedy *AbstractMethod*. Úlohou tejto triedy bude samotné riešenie optimalizačného problému a manipulácia s riadiacimi parametrami metódy. Základom genetického algoritmu je určitá populácia, nad ktorou sú aplikované genetické operácie s cieľom dosiahnutia novej kvalitnejšiej generácie. Každá populácia je tvorená určitým počtom jedincov. Z týchto faktov je možné usúdiť, že pre implementáciu samotného genetického algoritmu by bolo vhodné navrhnutie ďalších dvoch tried, ktoré by implementovali populáciu, jednolivca populácie a operácie s nimi.

Trieda *Individual*

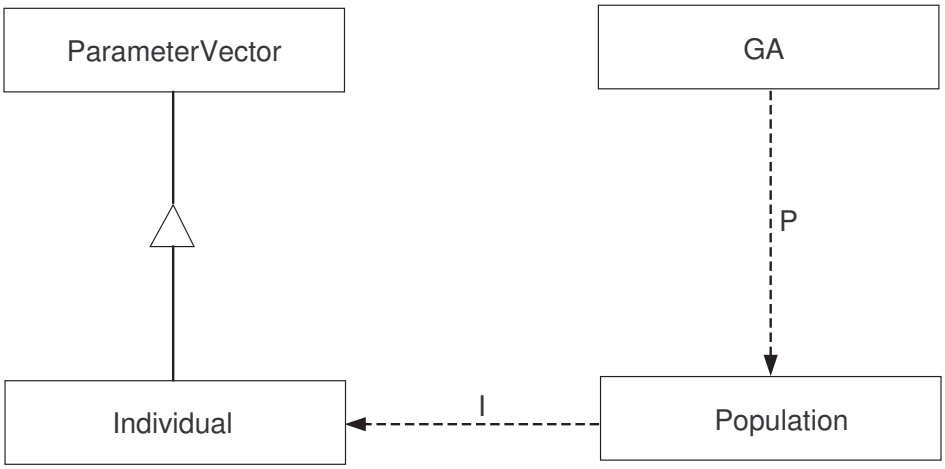
Úlohou triedy je reprezentácia jednotlivca populácie a genetických operácií nad ním definovaných. Pre tento účel je využiteľná trieda *ParameterVector* (implementovaná v SIMLIB/C++), ktorá reprezentuje vektor riadiacich premenných určitého optimalizačného problému. K tejto triede je potrebné doimplementovať členské funkcie realizujúce operácie využívané genetickým algoritmom a to:

- Náhodná inicializácia jedinca.
- Vyhodnotenie kondície jedinca.
- Mutácia.

Trieda *Population*

Úlohou triedy *Population* je reprezentácia populácie a genetických operácii nad ňou definovaných:

- Selekcia.
- Kríženie.
- Mutácia.
- Akceptácia.
- Ohodnotenie kondície populácie.
- Operátor [ ] pre sprístupnenie jedinca populácie.



Obrázok 3.2: Diagram tried genetického algoritmu. Plná čiara zobrazuje dedičnosť, zatiaľ čo prerušovaná je použitá, ak je trieda používaná alebo obsiahnutá inou triedou. Popisok pri tejto čiare pomenúva premennú, cez ktorú je referovaná trieda prístupná.

Kapitola 4

Implemetácia

Navrhnuté riešenie bolo implementované v jazyku C++ pod operačným systémom FreeBSD, pričom zdrojový kód je prenositeľný aj na iné operačné systémy. Deklarácie tried pre jednotlivé metódy sa nachádzajú v hlavičkovom súbore *opt.h*, konštanty (použité ako implicitné hodnoty pre riadiace parametre metód) sú definované v súbore *consts.h*. Implementácia tried pre jednotlivé metódy sa nachádza v moduloch *absmet.cc*, *simann.cc*, *neldermead.cc* a *gradient.cc*.

Popis implementácie jednotlivých metód je rozedelený do štyroch častí. Prvá popisuje spôsob implementácie častí algoritmu, ktoré nie sú pevne dané, prípadne rozdiely v zvolenej implementácii od štandardného spôsobu. Ďalšie časti postupne popisujú pseudokód algoritmu, riadiace parametre metódy a verejné rozhrania implementovaných tried.

4.1 Simulované žíhanie

Metóda je realizovaná triedou *SimAnn*, ktorá umožňuje riešenie neobmädzených optimalizačných problémov, ako aj nastavenie všetkých riadiacich parametrov. Metóda je implementovaná štandardným spôsobom, čiže algoritmus je tvorený dvoma cyklami. Vonkajší má na starosti znižovanie teploty, vnútorný realizuje obdobu algoritmu Metropolis. Klúčovými časťami algoritmu sú chladiaci rozvrh a generovanie susedného stavu popísané nižšie.

Chladiaci rozvrh

Je realizovaný exponenciálne $T_{k+1} = \alpha T_k$ privátnou metódou *tempDec()*. Rozvrh je riadený troma parametrami $minT$, $maxT$ a $decT$, pričom pri použití implicitného konštruktora triedy sú ich hodnoty nastavené na $minT = 0.1$, $maxT = 50$ a $decT = 0.9$.

Susedný stav

Generovaný v závislosti na aktuálnej teplote systému:

$$x_{t+1} = x_t + range(x) * (Random() - 0.5) * actT/maxT$$

Generovanie susedného stavu je implemntované metódou *moveToNextPoint()*.

Pseudokód

```
double SA(f, p)
   opt = f(p);
   act = p;
   act_fx = opt;
   while actT > minT
      badCount = 0;
      while badCount < maxBadCount  // obdoba algoritmu Metropolis
         new = newPoint(act);    // generovanie susedného bodu
         new_fx = f(new);
         if new_fx < act_fx         // nový bod je lepší ako aktuálny
            act = new;
            act_fx = new_fx;
            if new_fx < opt         // nové optimum
               p = new;
               opt = new_fx;
         else                       // akceptácia horšieho bodu
            if exp((act_fx - new_fx)/actT) < Random()
               act = new;
               act_fx = new_fx;
         else badCount = badCount + 1;  // počet neakceptovaných riešení
       actT = actT * decT;        // zníženie teploty
   return opt;
```

Riadiace parametre metódy

minT - spodná hranica teploty, po jej dosiahnutí je algoritmus ukončený.

maxT - počiatočná teplota.

decT - koeficiet, ktorým je teplota znižovaná v jednotlivých iteráciách.

maxBadCount - maximálny počet neakceptovaných riešení v algoritme Metropolis.

Verejné Rozhranie triedy *SimAnn*

Konštruktory triedy, umožňujúce nastavenie riadiacich parametrov metódy. K dispozícii je aj implicitný konštruktor:

```
SimAnn(double minT, double maxT, double decT, int maxIter);
SimAnn(double minT, double maxT, double decT, int maxIter,
int maxConIter, int minConIter, double penMul, double escFac);
```

Deštruktor:

```
~SimAnn();
```

Členská funkcia vykonávajúca samotné riešenie neobmädzeného optimalizačného problému metódou simulovaného žíhania. Parametrami funkcie sú vektor riadiacich premenných a ukazateľ na objektívnu funkciu:

```
double Solve(opt_function_t f, ParameterVector &p);
```

Členská funkcia umožňujúca nastavovanie riadiacich patametrov metódy:

```
void SetParams(double minT, double maxT, double decT, int maxIter);
```

4.2 Metóda Nelder-Mead

Implementovaná triedou *NelderMead*, spôsobom prezentovaným v kapitole 2 s odľahčením ukončovacích podmienok. Tie sú realizované iba naplnením maximálneho možného počtu iterácií algoritmu, alebo v prípade ak je rozdiel hodnoty objektívnej funkcie centroidu a vrcholov simplexu menší ako požadovaná presnosť (parameter metódy).

Pseudokód

```
double NelderMead(f, p)
   D = p.size();                    // počet riadiacich premenných
   points[D + 1] = RandomInit();    // vrcholy simplexu
   sortPoints();                    // usporiadanie vrcholov podla f()
   centroid = computeCentorid();
   for i = 0 to maxIter
      if StopCondition()
         break;
      sortPoints();
      centroid = computeCentroid();  // výpočet centroidu
      awf = centorid - points[D+1];  // vzdialenost' od najhoršieho bodu
      pointR = centroid + alpha * afw;  // reflexný bod
      if f(pointR) <= f(points[1])
         pointE = centroid + gamma * afw;  // expanzný bod
         if f(pointR) => f(pointE)
            points[D+1] = pointE;
         else
            points[D+1] = pointR;
      else
         if f(pointR) < f(points[D])
            if f(pointR) < f(points[D+1])
               pointC = centroid + beta * afw;   // kontrakčný bod
            else
               pointC = centroid + beta * afw;
            if f(pointC >= points[D+1])
               for i = 2 to D + 1 // posunutie smerom k najlepšiemu vrcholu
                  points[i] = points[1] + delta * (points[i] - points[1]);
            else
               points[D+1] = pointC;
         else
            points[D+1] = pointR;
   p = f(points[1]);  // najlepší vrchol
   return f(p);
```

Riadiace parametre metódy

alpha - riadiaci parameter pre reflexiu.

beta - riadiaci parameter pre kontrakciu.

gamma - riadiaci parameter pre expanziu.

delta - riadiaci parameter pre priblíženie.

eps - presnost'.

maxIter - maximálny počet iterácií algoritmu.

Verejné rozhranie triedy *NelderMead*

Konštruktory triedy, umožňujúce nastavenie riadiacich parametrov metódy. K dispozícii je aj implicitný konštruktor:

```
NelderMead(double eps, int maxIter, double alpha,
           double beta, double gamma, double delta);
NelderMead(double eps, int maxIter, double alpha, double beta, double gamma,
double delta, int maxConIter, int minConIter, double penMul, double escFac);
```

Deštruktor:

```
~NelderMead();
```

Členská funkcia vykonávajúca samotné riešenie neobmädzeného optimalizačného problému Nelder-Meadovou metódou. Parametrami funkcie sú vektor riadiacich premenných a ukazateľ na objektívnu funkciu:

```
double Solve(opt_function_t f, ParameterVector &p);
```

Členská funkcia umožňujúca nastavovanie riadiacich patametrov metódy:

```
void SetParams(double eps, int maxIter, double alpha,
               double beta, double gamma, double delta);
```

4.3 Gradientná metóda

Implementovaná triedou *Gradient*. Hlavnou podúlohou algoritmu je výpočet gradientu, ktorý je realizovaný s využitím dvoch bodov - aktuálneho bodu a bodu vypočítaným z pozície aktuálneho bodu posunutej o veľkost' kroku aktuálnej iterácie. Veľkost' kroku je na počiatku daná jedným z parametrov metódy a v každej iterácii je zmenšovaná (násobením koeficientom závislým od maximálneho počtu iterácií). Takto získaný gradient je normalizovaný a následne je vykonané lineárne prehladávanie v smere gradientu. Výpočet metódy je ukončený, ak je dosiahnutý maximálny počet iterácií, alebo v aktuálnej iterácií nebolo nájdené nové optimum.

Pseudokód

```
double Gradient(f, p)
   opt = f(p);
   for i = 1 to maxIter
      oldOpt = opt;
      act_p = p;
      delta = p.Range() * stepSize / 10;  // posunutie bodu
      new_p = act_p + delta;
      g = (f(new_p) - f(act_)) / delta;  // výpočet gradientu
      if f(new_p) < opt                  // nové optimum
         opt = f(new_p);
         p = new_p;
      normalizeGradient(g);      // normalizácia gradientu
      tmpStep = stepSize;
      for j = 1 to maxSubSteps   // lineárne vyhladávanie
         new_p = p + p.Range() * tmpStep * g;
         if f(new_p) < opt
            opt = f(new_p);      // nové optimum
            p = new_p
         else
            tmpStep = tmpStep * rho;  // zmenšenie subkroku
      stepSize = stepSize * (maxIter - 1)/maxIter;  // zmenšenie kroku
   return opt;
```

Riadiace parametre metódy

maxIter - maximálny počet iterácií.

maxSubSteps - maximálny počet podkrokov (dĺžka lineárneho vyhľadávnia).

stepSize - počiatočná veľkosť kroku.

rho - koeficient zmenšenia pre lineárne vyhľadávanie.

Verejné rozhranie triedy *Gradient*

Konštruktory triedy, umožňujúce nastavenie riadiacich parametrov metódy. K dispozícii je aj implicitný konštruktor:

```
Gradient(int maxIter, int maxSubSteps, double StepSize, double rho);
Gradient(int maxIter, int maxSubSteps, double StepSize, double rho,
         int maxConIter, int minConIter, double penMul, double escFac);
```

Deštruktor:

```
~Gradient();
```

Členská funkcia vykonávajúca samotné riešenie neobmädzeného optimalizačného problému gradientnou metódou. Parametrami funkcie sú vektor riadiacich premenných a ukazateľ na objektívnu funkciu:

```
double Solve(opt_function_t f, ParameterVector &p);
```

Členská funkcia umožňujúca nastavovanie riadiacich patametrov metódy:

```
void SetParams(int maxIter, int maxSubSteps, double StepSize, double rho);
```

4.4 Penalizačná metóda

Metóda je implementovaná v triede *AbstractMethod* v spolupráci s každou z metód pre neobmädzenú optimalizáciu. V prípade, že metóda nedokázala nájsť realizovateľné riešenie danej úlohy, je ako optimum vrátená hodnota konštanty NaN.

Pseudokód

```
double PF(f, p, pf)
   penMul = penMul0;
   i = 1;
   opt = f(p) + penMul * pf(p);
   while i < maxConIter
      oldOpt = opt;
      opt = UnconstrainedOptimization();  // neobmädzená optimalizácia
      penMul = penMul * escFac;  // zväčšenie penalizačného násobitela
      if oldOpt <= opt && i > minConIter  // ukončujúce podmienky
         break;
      i = i + 1;
   if pf(p) == 0  // splnenie obmädzujúcich podmienok
      return opt;
   else           // nesplnenie obmädzujúcich podmienok
      return NaN;
```

Riadiace parametre metódy

PenMul - počiatočná hodnota penalizačného násobiteľa.

escFac - stupňovací faktor, hodnota, ktorou je násobený penalizačný násobiteľ v jednotlivých iteráciách.

maxConIter - maximálny počet iterácií.

minConIter - minimálny počet iterácií. Keďže hodnota penalizačného násobiteľa začína na relatívne malej hodnote, je vplyv penalizačnej funkcie na nájdené riešenie v prvých iteráciách malý. Z tohoto dôvodu môžu prvé iterácie algoritmu nájsť rovnaké nedostatočne presné riešenie, čo znamená splnenie ukončujúcich podmienok. Dôvod zavedenia tohto parametru je vyhnutie sa práve týmto prípadom.

Verejné rozhranie triedy *AbstractMethod*

Konštruktor triedy, umožňujúci nastavenie riadiacich parametrov metódy. K dispozícii je aj implicitný konštruktor:

```
AbstractMethod(int maxConIter,int minConIter,double penMul,double escFac);
```

Deštruktor:

```
~AbstractMethod();
```

Členská funkcia vykonávajúca samotné riešenie obmädzeného optimalizačného problému penalizačnou metódou. Parametrami funkcie sú ukazateľ na objektívnu funkciu, vektor riadiacich premenných a ukazateľ na penalizačnú funkciu:

```
double SolveCon(opt_function_t f, ParameterVector &p, opt_function_t pf);
```

Členská funkcia umožňujúca nastavovanie riadiacich patametrov metódy:

```
void SetConsParams(int maxConIter, int minConIter,
                   double penMul, double escFac);
```

Kapitola 5

Testovacie príklady

Implementované metódy boli podrobené trom optimalizačným úlohách s rôznou obtiažnosť'ou: Rosenbrocková funkcia [4], Rastriginová funkcia [4] a Bukinova funkcia [6]. Prvé dve funkcie sú štandardne využívanými testami pre metódy globálnej optimalizácie, tretia funkcia patrí medzi novšie menej často využívané testovacie úlohy. Všetky úlohy sú funkciami dvoch premenných a okrem definičného oboru riadiacich parametrov pre nich nie sú zadané žiadne obmädzujúce podmienky.

Pre každú testovaciu úlohu bolo vykonaných 1000 experimentálnych optimalizácií každou metódou. Experimenty, zodpovedajúce jednotlivým testovacím úlohám, boli vykonávané vždy s rovnakými riadiacimi parametrami jednotlivých metód. Zaznamenanými výsledkami pre každý experiment boli vypočítané optimum a počet vyhodnotení objektívnej funkcie, potrebných pre jeho dosiahnutie.

Za hodnotiace kritéria v testoch boli zvolené správnosť a presnosť riešenia a výpočtová zložitosť jednotlivých metód. Pod výpočtovou zložitost'ou sa rozumie počet vyhodnotení objektívnej funkcie, keďže pre jej výpočet je v praktických príkladoch potrebné vykonanie simulácie optimalizovaného modelu, čo je obvykle časovo veľmi náročná operácia.

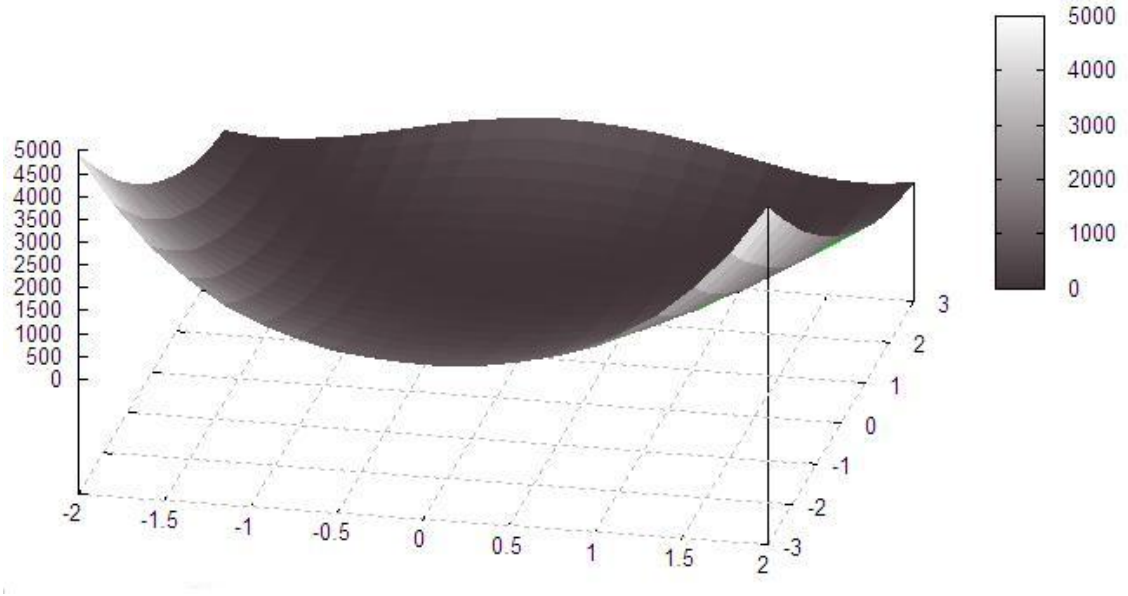
Implementácia testov sa nachádza v súboroch *test1.cc*, *test2.cc* a *test3.cc*. Súbor *test-stats.cc* obsahuje pomocné funkcie pre štatistické vyhodnotenie testov, pričom ich deklarácie sú uvedené v hlavičkovom súbore *test-stats.h*.

5.1 Popis testovacích funkcií

Rosenbrockova funkcia dvoch premenných

Jedná sa o veľmi známu nekonvexnú testovaciu funkciu, tiež známu aj pod názvami Rosenbrockovo údolie alebo Rosenbrockova banánová funkcia [4], vďaka tvaru jej vrstevníc. Jej optimalizácia sa stáva veľmi nepríjemnou, ak je počiatočný bod zvolený príliš ďaleko od globálneho minima, ktoré sa nachádza vnútri dlhého údolia s veľmi strmými svahmi a takmer plytkým dnom. Nájdenie údolia je triviálnou úluhou, avšak konvergencia do globálneho minima je veľmi náročná, a preto je táto funkcia často používaná na ohodnotenie výkonnosti optimalizačných algoritmov. Funkcia môže spôsobiť problémy najmä gradientným metódam, ktoré môžu v niektorým prípadoch úplne zlyhať v hľadaní minima. Funkcia má globalné minimum v bode $x = 0$, $y = 0$ kde $f(x, y) = 0$. Funkcia nemá žiadné lokálne minimá. Hodnoty parametrov sú v teste obmädzené na $-100 \leq x_i \leq 100 \quad \text{pre} \quad i = 1, 2$.

$$f(x, y) = (y - x)^2 + 100(y - x^2)^2$$



Obrázok 5.1: Graf Rosenbrockovej funkcie.

Rastriginova funkcia dvoch premenných

Funkcia je založená na *De Jongovej funkcii* [4], ku ktorej bola pridaná modulácia funkciou *cosínus*, čo má za následok vznik častých pravidelne rozložených lokálnych miním. Globálne minimum má hodnotu $f(x, y) = 0$ v bode $x = 0$, $y = 0$. Optimalizácia tejto funkcie je považovaná za náročnú pre väčšinu metód.

$$f(x, y) = nA + [x^2 - Acos(\omega x)] + [y^2 - Acos(\omega y)]$$

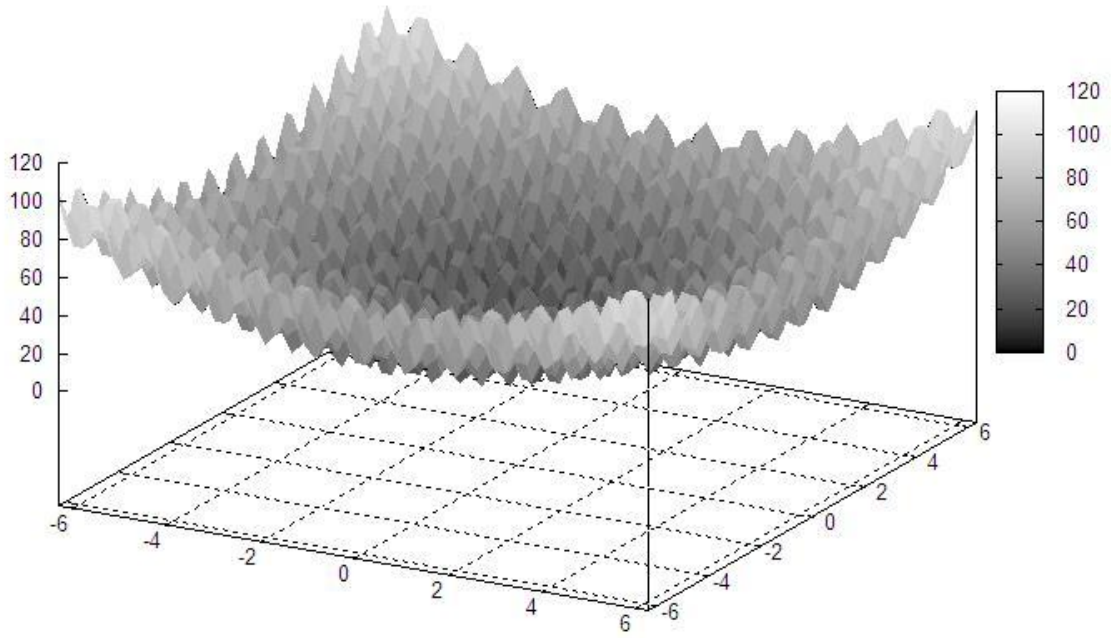
kde n je počet dimenzií, A udáva strmosť lokálnych extrémov a ω je frekvenciou výskytu lokálnych miním. V tomto teste sú ich hodnoty nastavené na $n = 2$, $A = 10$, $\omega = 2\pi$. Hodnoty parametrov sú obmädzené na $-2\pi \leq x_i \leq 2\pi$, $i = 1, 2$.

Bukinova funkcia

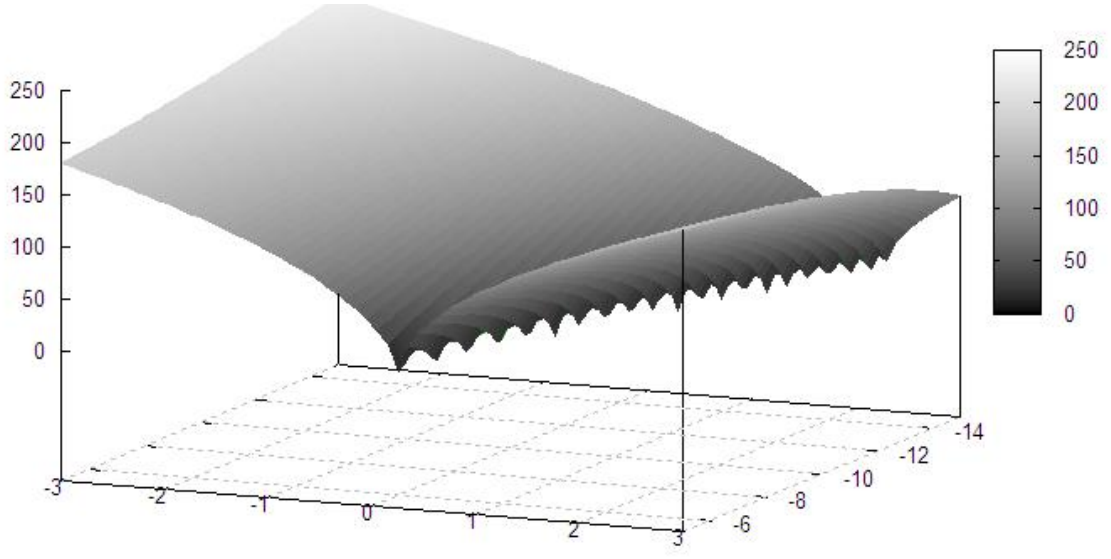
Táto funkcia je takmer identická v okolí svojich minimálnych bodov (s jemne stúpajúcimi svahmi) a to ju robí extrémne náročnú pre akúkoľvek optimalizačnú metódu. Funkcia má niekoľko lokálnych miním a globálne minimu $f(x, y) = 0$ v bode $x = -10$, $y = 1$. Funkcia je definovaná následovne [6]:

$$f(x, y) = 100\sqrt{|y - 0.01x^2|} + 0.01|x + 10|$$

kde $x \in [-15, -5]$ a $y \in [-3, 3]$.



Obrázok 5.2: Graf Rastriginovej funkcie.



Obrázok 5.3: Graf Bukinovej funkcie.

5.2 Dosiahnuté výsledky

Výsledky každého testu sú pre jednotlivé metódy zobrazené na dvoch histogramoch. Oba histogramy majú totožnú x-ovú os, ktorá výsledky jednotlivých optimalizačných metód rozdeľuje do vhodne zvolených intervalov podľa charakteru testovanej funkcie. Intervaly zobrazujú jednu z nasledujúcich informácii:

- rozdiel nájdeného optima od skutočného globálneho optima
- vzdialenosť nájdeného optimálného bodu od skutočného optimálneho bodu

Prvý histogram vyjadruje úspešnosť a presnosť nájdenia globálneho minima, čiže na y-ovej osi je nanesený počet nájdených riešení spadajúcich do jednotlivých intervalov.

Druhý histogram zobrazuje výpočtovú zložitosť, ktorá je na na y-ovej osi vyjadrená priemerným počtom vyhodnotenií objektívnej funkcie, potrebných pre dosiahnutie riešenia v jednotlivých intervaloch.

Grafy výsledkov pre jednotlivé testy sú zobrazené na obrázkoch 5.4, 5.5 a 5.6.

Použité parametre metód pre Rosenbrockovu funkciu:

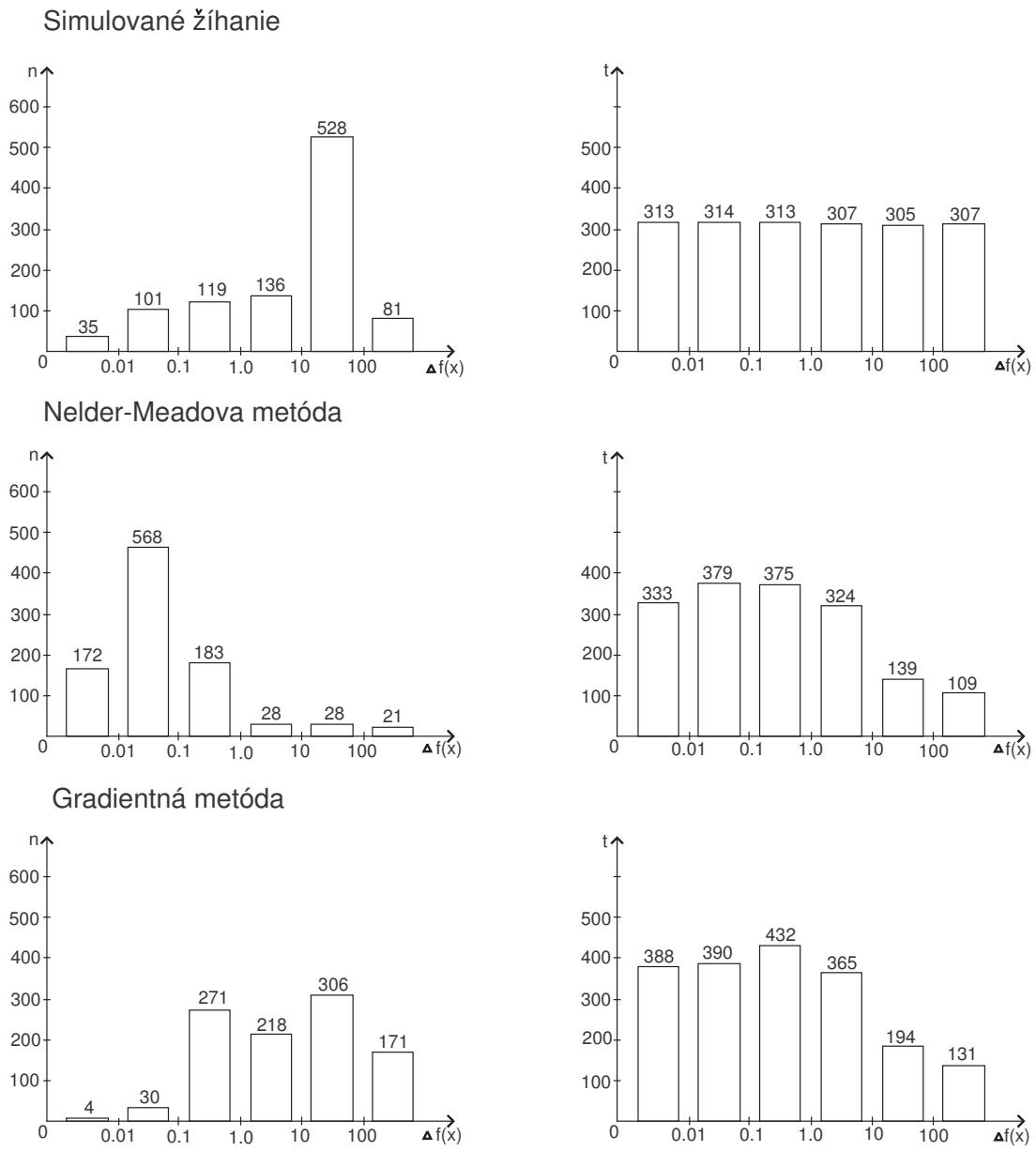
Simulované žíhanie: $maxT = 50$, $minT = 0.1$, $decT = 0.9$, $maxBadCount = 5$
Nelder-Meadova metóda: $maxIter = 200$, $eps = 0.01$, $alpha = 1$, $beta = 0.5$, $gamma = 2$, $delta = 0.5$
Gradientná metóda: $maxIter = 250$, $maxStepSize = 0.01$, $maxSubSteps = 3$, $rho = 0.5$

Použité parametre metód pre Rastriginovu funkciu:

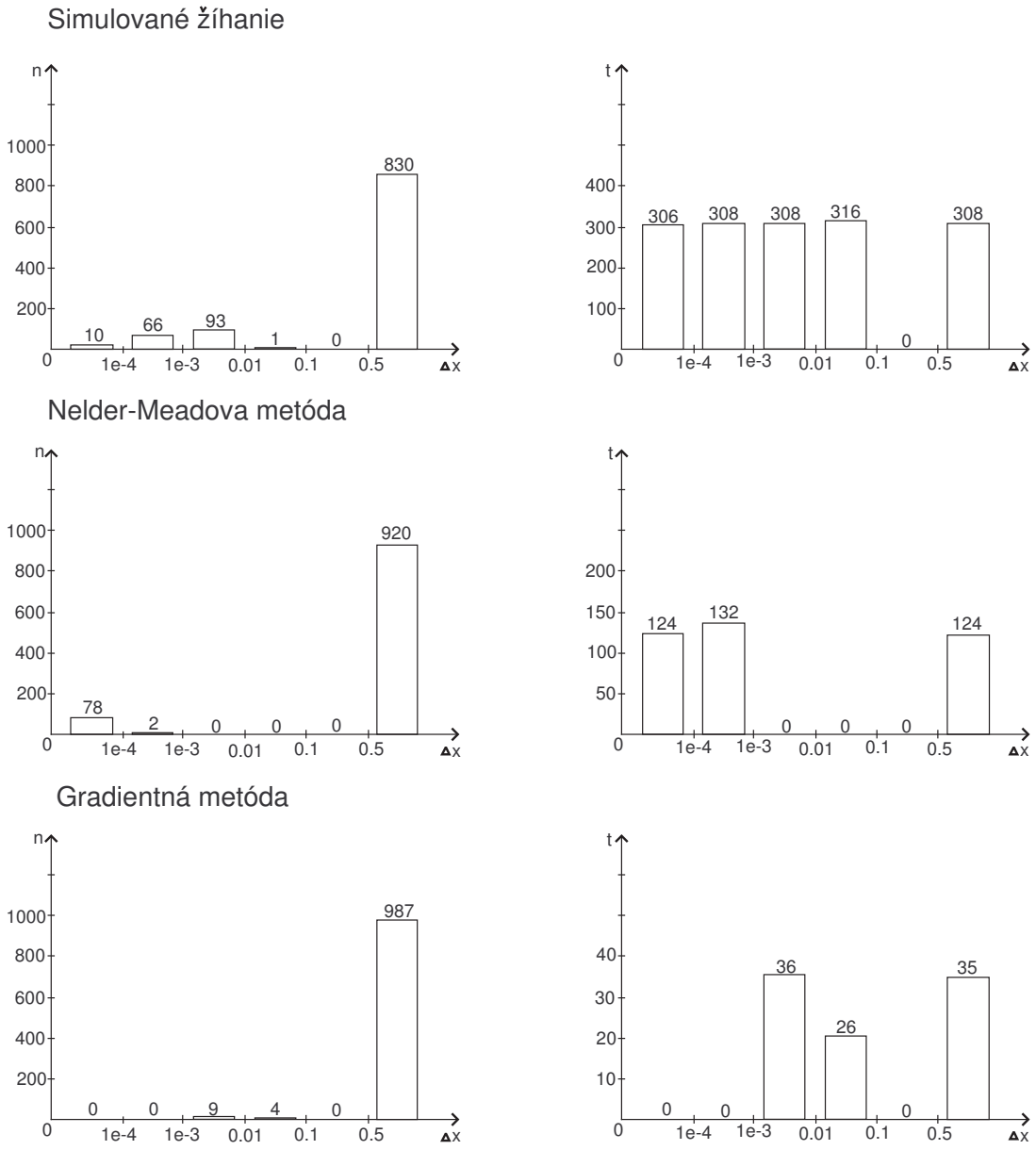
Simulované žíhanie: $maxT = 50$, $minT = 0.1$, $decT = 0.9$, $maxBadCount = 5$
Nelder-Meadova metóda: $maxIter = 200$, $eps = 0.01$, $alpha = 1$, $beta = 0.5$, $gamma = 2$, $delta = 0.5$
Gradientná metóda: $maxIter = 250$, $maxStepSize = 0.0001$, $maxSubSteps = 3$, $rho = 0.5$

Použité parametre metód pre Bukinovu funkciu:

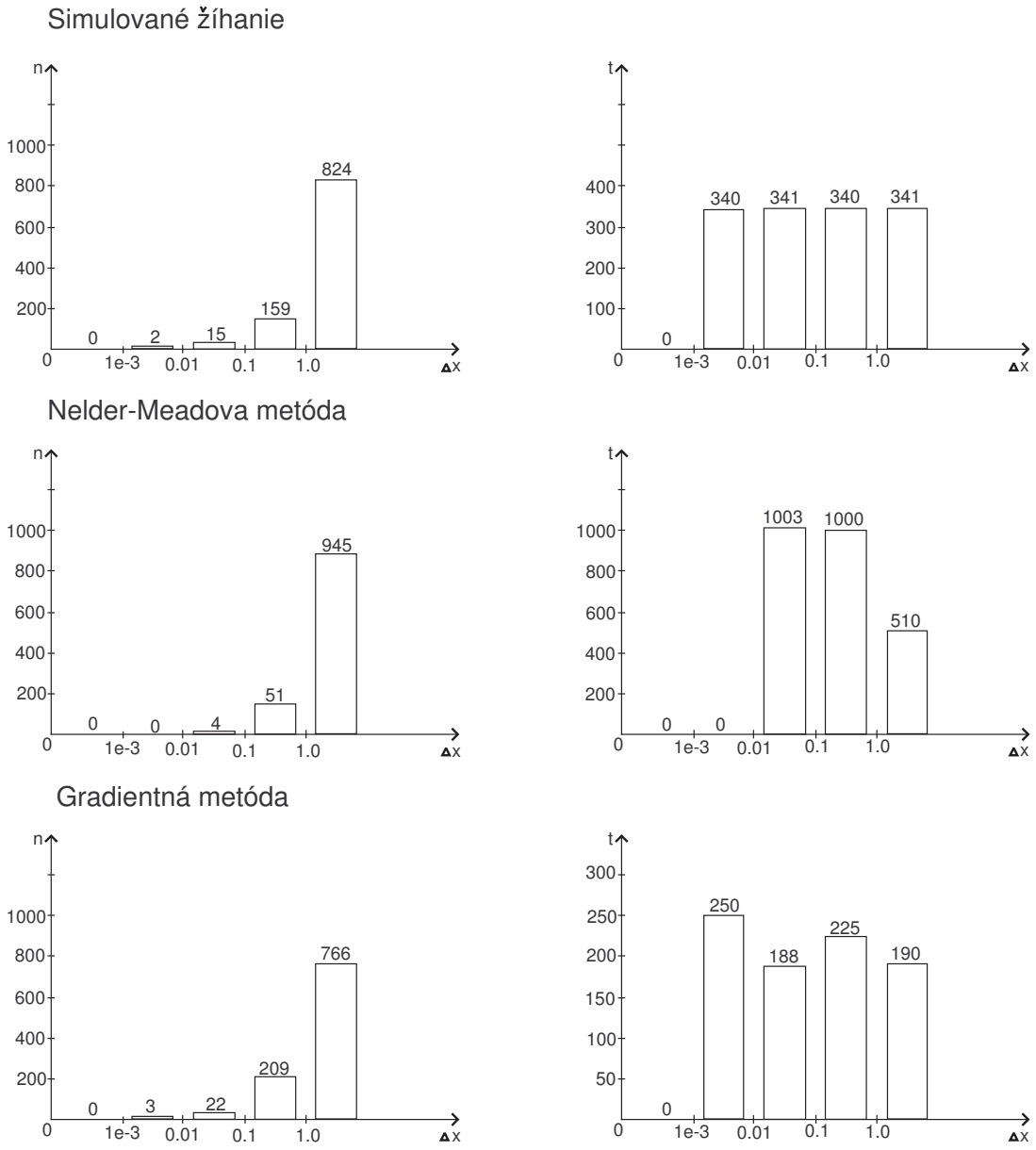
Simulované žíhanie: $maxT = 50$, $minT = 0.1$, $decT = 0.9$, $maxBadCount = 5$
Nelder-Meadova metóda: $maxIter = 200$, $eps = 0.01$, $alpha = 1$, $beta = 0.5$, $gamma = 2$, $delta = 0.5$
Gradientná metóda: $maxIter = 250$, $maxStepSize = 0.01$, $maxSubSteps = 6$, $rho = 0.2$



Obrázok 5.4: Histogramy výsledkov pre Rosenbrockovu funkciu. x-ové osi histogramov zobrazujú intervaly, do ktorých spadajú rozdiely nájdeného optima od skutočného optima $\Delta f(x)$. Na y-ovej osi histogramu na ľavej strane je vyjadrený počet výsledkov spadajúcich do jednotlivých intervalov riešení. y-ová os histogramu na pravej strane zobrazuje priemerný počet vyhodnotení objektívnej funkcie, ktorý bol potrebný pre dosiahnutie výsledkov spadahúcich do príslušného intervalu.



Obrázok 5.5: Graf výsledkov pre Rastriginovu funkciu. x-ové osi histogramov zobrazujú intervaly, do ktorých spadajú veľkosti vzdialeností nájdených optimálnych bodov od skutočného optima Δx. Na y-ovej osi histogramu na ľavej strane je vyjadrený počet n výsledkov spadajúcich do jednotlivých intervalov. y-ová os histogramu na pravej strane zobrazuje priemerný počet vyhodnotení objektívnej funkcie t, ktorý bol potrebný pre dosiahnutie výsledkov spadahúcich do príslušného intervalu.



Obrázok 5.6: Graf výsledkov pre Bukinovu funkciu. x-ové osi histogramov zobrazujú intervaly, do ktorých spadajú veľkosti vzdialeností nájdených optimálnych bodov od skutočného optima Δx. Na y-ovej osi histogramu na ľavej strane je vyjadrený počet n výsledkov spadajúcich do jednotlivých intervalov. y-ová os histogramu na pravej strane zobrazuje priemerný počet vyhodnotení objektívnej funkcie t, ktorý bol potrebný pre dosiahnutie výsledkov spadahúcich do príslušného intervalu.

5.3 Zhodnotenie výsledkov testov

V úvodnom teste, Rosenbrockovej funkcii, si výborne počínala Nelder-Meadová metóda, ktorej sa vo väčšina prípadov podarilo dopracovať do globálneho minima a to aj s veľmi dobrou dosiahnutou presnosťou. Menej úspešnými boli ďalšie dve metódy (simulované žíhanie, gradientná metóda), ktoré sa síce boli schopné dopočítať do oblasti globálneho minima, avšak nedokázali k nemu dostatočne presne konvergovať.

V teste realizovanom Rastriginovou funkciou si relatívne dobre počínali metódy Nelder-Mead a simulované žíhanie, ktoré dosiahli globálne minimum v približne 8% pokusov. Naopak gradientná metóda sa nedokázala vysporiadať s častými výskytmi lokálnych miním. Na histograme zobraujúcom počet vyhodnotení objektívnej funkcie je zjavné, že metóda bola veľmi rýchlo schopná dosiahnúť lokálneho minima, z ktorého už však nebola schopná pokračovať.

Výsledky testu Bukinovej funkcie potvrdili náročnosť optimalizácie tejto úlohy. Najúspešnejšie si v počínala gradientná metóda, ktorej sa podarilo presné nájdenie globálneho minima v 3% prípadov. Najhoršie dopadla v predchádzajúcich testoch najúspešnejšia metóda Nelder-Mead aj napriek výrazne najvyššiemu počtu vyhodnotení objektívnej funkcie. Z tohoto faktu by sa dalo usúdiť, že táto metóda mala problém v oblasti okolia globálneho minima, kde sa jej nedarilo dostatočne konvergovať.

Záverom by sa výsledky testov dali zhodnotiť ako uspokojivé. Pri dostatočne veľkom počte experimentov boli metódy takmer vo všetkých úlohách určiť globálne minimum, s výnimkou gradientnej metódy v teste na Rastriginovej funkcii. Testovacie príklady predvedené v tejto kapitole sú iba názorné, keďže pre každý test boli jednotlivé metódy testované iba s jednou sadou riadiacich parametrov. Inou voľbou parametrov by mohli byť dosiahnuté presnejšie výsledky, ale cenou za to by bolo zvýšené množstvo vyhodnotení objektívnej funkcie, čo by malo za následok výrazné zvýšenie časovej náročnosti optimalizácie.

Kapitola 6

Záver

Cieľom práce bolo rozšírenie optimalizačnej časti knižnice SIMLIB/C++, kde boli doposiaľ implementované metódy Hooke-Jeeves a simulované žíhanie v zjednodušenej verzii. Ďalej tu boli pripravené triedy pre prácu s riadiacimi parametrami optimalizačných problémov. Bolo navrhnuté rozšírenie tohoto stavu o metódy simulované žíhanie, Nelder-Meadova metóda, gradientná metóda pre neobmädzenú optimalizáciu a penalizačnú metódu pre optimalizáciu obmädzenú. Pre tieto metódy bola navrhnutá hierarchia tried s využitím návrhového vzoru *Template method*. Takto navrhnuté riešenie bolo implementované v jazyku C++.

Presnosť a efektivita implementovaných metód bola otestovaná na troch rôzne náročných príkladoch (Rosenbrockova funkcia, Rastriginova funkcia a Bukinova funkcia). Prvé dve z nich sú štandardne využívané na testovanie metód pre globálnu optimalizáciu. Z graficky prezentovaných výsledkov testov možno usúdiť, že metódy sa s uvedenými problémami vysporiadali uspokojivo. Oblasť globálneho minima boli schopné dosiahnúť relatívne spoľahlivo, avšak metódy prejavili určité nedostatky v presnej konvergencii do globálneho minima. Navrhnuté riešenie gradientnej metódy, na rozdiel od ostatných, preukázalo značné problémy s riešením úloh s veľkým množstvom lokálnych miním.

Z hľadiska budúceho vývoja by bolo knižnicú vhodné rozšíriť o nové metódy, najmä v súčastnosti veľmi široko využívanú metódu genetického algoritmu a o nové metódy pre obmädzenú optimalizáciu, keďže penalizačná metóda nepatrí medzi najefektívnejšie a pre svoju činnosť potrebuje veľký počet vyhodnotení objektívnej funkcie.

Literatúra

[1] Bürmen, A., Puhan, J., Tuma, T.: Grid restrained nelder-mead algorithm. http://www.springerlink.com/content/641r825303572365/fulltext.pdf, 2006.

[2] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley, 1997. ISBN 0-201-63361-2.

[3] Rardin, Ronald L.: *Optimization in Operations Research.* Prentice Hall, 2000. ISBN 0-02-398415-5.

[4] Molga M., Smutnicki, C.: Test functions for optimization needs. http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf, 2005.

[5] Obitko, M.: Introduction to genetic algorithms. http://cs.felk.cvut.cz/ xobitko/ga/, 1998.

[6] Sudhanshu, Kumar Mishra: Some new test functions for global optimization and performance of repulsive particle swarm method. http://mpra.ub.uni-muenchen.de/2718/, 2007.

[7] Peringer, P.: SIMLIB C/C++. http://www.fit.vutbr.cz/~peringer/SIMLIB/, 2008.

[8] Kirkpatrick, S., Gelatt, C. D., Vecchi, M. P.: Optimization by simulated annealing. http://www.cs.virginia.edu/cs432/documents/sa-1983.pdf, 1983.