České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Tomáš Jiran**

Studijní program: Otevřená informatika (bakalářský)
Obor: Softwarové systémy

Název tématu: **Algoritmus kontinuální evoluce**

Pokyny pro vypracování:

Nastudujte a v jazyce Java implementujte optimalizační algoritmus kontinuální evoluce (CEA). Po konzultaci s vedoucím práce demonstrujte funkčnost implementace na vybraných problémech. Připravte a otestujte aplikaci optimalizačního algoritmu na testovací funkce BBOB (Black-Box Optimization Benchmarking).

Seznam odborné literatury:

Z.Buk: Continual Evolution Algorithm, dissertation thesis, CTU, Prague, 2012
http://coco.gforge.inria.fr/doku.php?id=bbob-2013

Vedoucí: Ing. Zdeněk Buk, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

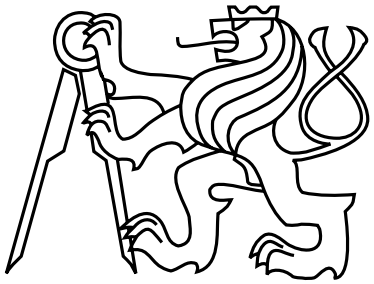
L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 2. 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Bakalářská práce

Algoritmus kontinuální evoluce

Tomáš Jiran

Vedoucí práce: Ing. Zdeněk Buk, Ph.D

Studijní program: Otevřená informatika

Obor: Softwarové systémy

25. května 2015

Poděkování

Především bych chtěl poděkovat vedoucímu mé práce Ing. Zdeňku Bukovi, Ph.D. za všechny jeho rady, čas a za vytvoření přátelského pracovního prostředí.

Také bych chtěl poděkovat své rodině za všechnu jejich podporu během mých studií.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 23. 5. 2014 .....

Abstract

The main topic of this thesis is the implementation and benchmarking of the Continual Evolution Algorithm (CEA). It's an optimization algorithm, which belongs to so-called evolutionary algorithms. It combines an evolutionary optimization approach with a local search method so as to decrease the number of fitness evaluations needed to find a good enough solution for a specific optimization problem.

The thesis covers a brief introduction to evolutionary algorithms and the Continual Evolution Algorithm. Then, chosen implementation of the algorithm, executed experiments and obtained results are presented.

Key words: optimization, evolutionary algorithm, Continual Evolution Algorithm, BBOB, JCOOL

Abstrakt

Tato práce se zabývá především implementací a testováním algoritmu kontinuální evoluce (CEA). Jedná se o optimalizační algoritmus, který patří mezi takzvané evoluční algoritmy. Kombinuje evoluční optimalizační přístup s lokální prohledávací funkcí, což umožňuje snížit počet fitness ohodnocení nutných k nalezení vhodného řešení daného optimalizačního problému.

Práce pokrývá stručný úvod do problematiky evolučních algoritmů a algoritmu kontinuální evoluce. Dále pojednává o zvolené implementaci tohoto algoritmu, o provedených experimentech a obdržených výsledcích.

Klíčová slova: optimalizace, evoluční algoritmus, algoritmus kontinuální evoluce, BBOB,

Obsah

Seznam obrázků

Kapitola 1

Úvod

Zde jsou uvedeny základní cíle práce. Hlavním cílem bylo implementovat algoritmus kontinuální evoluce a otestovat jeho funkčnost využitím Black-Box Optimization Benchmarking (BBOB) [3] (o BBOB viz kapitolu 4.1).

Práce je rozdělena do následujících pěti kapitol:

- **Evoluční přístup** poskytuje základní úvod do evolučních algoritmů.
- **Algoritmus kontinuální evoluce** udává klíčové vlastnosti algoritmu kontinuální evoluce.
- **Implementace** vysvětluje, jaká implementace byla zvolena.
- **Experimenty** prezentuje experimenty, které byly použity k otestování, a jejich výsledky.
- **Závěr** uzavírá práci a shrnuje obdržené výsledky.

Kapitola 2

Evoluční přístup

Tato kapitola obsahuje obecný úvod do evolučních algoritmů.

2.1 Přírodní evoluce

Tato sekce uvádí pouze klíčový princip a termíny *přírodní evoluce*, z níž evoluční algoritmy čerpají svou inspiraci. Obeznámený čtenář může tuto sekci přeskočit.

Všechny živé organismy mají pomocí *genů* (bloků DNA) zakódovány své vrozené rysy. Kompletní množina genetického materiálu jedince se nazývá *genom*. *Genotyp* je pak určitá konkrétní množina genů v genomu, tj. reprezentace jedince. *Fenotyp* je pak výsledkem genotypu a prostředí, konkrétním jedincem.

Proces evoluce stojí na principu *přírodního výběru* jedinců v určité populaci. Jedinci, kteří jsou díky svému genomu a naučenému chování lépe přizpůsobeni soutěži o zdroje, tj. mají tzv. lepší *fitness* (zdatnost), tak spíše přežijí a předají své výhodné geny svým potomkům. To časem vede k postupnému vzrůstu celkové zdatnosti populace.

2.2 Evoluční systémy

Tato sekce čtenáře seznamuje s elementárním konceptem evolučních systémů.

Evoluční přístup může být typicky užitečný pro řešení takových problémů, jejichž řešení nelze snadno spočítat analyticky nebo kde stavový prostor řešení je příliš rozsáhlý. Evoluční komputace je výzkumnou oblastí v rámci počítačových věd. Je speciální případem komputace, který svou inspiraci čerpá z procesu *přírodní evoluce* [7, p. 17].

Nejprve je třeba specifikovat, co definuje *evoluční systém*. Existuje obecný konsenzus o tom, které rysy evoluční systémy vykazují [9, p. 13]:

- populace jedinců soupeřících o omezené zdroje,
- dynamická změna populace, přítomná kvůli rození a umírání jedinců,
- koncept *fitness* (zdatnosti), který reflektuje schopnost jedince přežít a rozmnožit se, a

- koncept variační dědičnosti, tj. potomstvo je podobné svým rodičům, ale obecně ne identické.

V evolučních výpočtech se využívá následujícího základního konceptu. Díky své charakterizaci lze evoluční systém považovat za proces s danými počátečními podmínkami, kde se jedinci pohybují v čase komplexním evolučním stavovým prostorem [9, p. 13]. V evoluční komputaci je přirozené nahlížet na jedince jako na oddělená kandidátní řešení určitého optimalizačního problému. Například máme-li nalézt globální extrém dané n-dimenzionální funkce, můžeme na počátku volit jedince jako náhodné souřadnice funkce. Jedinci tak představují počáteční kandidátní řešení problému.

2.3 Evoluční algoritmy

V této sekci jsou shrnuty základní principy evolučních algoritmů.

Evoluční algoritmy se inspirují evolučními systémy (viz předchozí sekce). Jedním z nejznámějších evolučních algoritmů je například *genetický algoritmus*.

Pro každý genetický algoritmus jsou známy následující komponenty [7, p. 34]:

- *Reprezentace jedince*
- *Populace*
- *Fitness funkce*
- *Rekombinace*
- *Selekce*

Jedinec navíc může *zmutovat*, tj. jeho genotyp se může náhodně odlišit při rekombinaci od genotypů svých rodičů.

2.3.1 Reprezentace jedince

Genetická reprezentace popisuje elementy genotypu a jak jsou tyto elementy mapovány do fenotypu [8, p. 31]. Volba konkrétní reprezentace závisí na problému, který máme řešit. Příklady reprezentací jsou: **diskrétní** (jedinec je popsán sekvencí l diskrétních hodnot), **reálné** (genotyp se skládá z množiny n čísel z domény reálných čísel) a **stromové** (vhodné pro popis hierarchických větvených struktur).

2.3.2 Populace

Populace sestává z jedinců. Jedinci kódují kandidátní řešení daného optimalizačního problému. Na počátku jsou genomy jedinců populace generovány obvykle náhodně. Dále se uplaňuje přírodní výběr jedinců podle jejich fitness hodnoty.

2.3.3 Fitness funkce

Fitness funkce popisuje kvalitu nalezených řešení (jedinců). Čím vyšší má jedinec svou fitness hodnotu, tím více se blíží hledanému optimálnímu řešení. Například hledáme-li co nejrychlejší algoritmus mezi kandidátními algoritmy, jako fitness hodnotu algoritmu lze volit jeho průměrnou výpočetní dobu.

2.3.4 Rekombinace

Rekombinace je operace vytvoření nového jedince. Genotyp nového jedince vychází z genotypů jeho rodičů. Například máme-li binární reprezentaci jedinců, kde rodičovské genotypy jsou kódovány například jako 10000000 a 11111111, tak genotyp potomka může být, získáváli polovinu genotypu od prvního rodiče a polovinu od druhého (jedná se o tzv. *one-point crossover*), 10001111. V praxi se používají i *n-point crossovers*.

2.3.5 Mutace

Mutace je náhodné odlišení genotypu jedince při rekombinaci od genotypů svých rodičů.

2.3.6 Selekce

Selekce je operace výběru jedinců, kteří přežijí či kteří budou mít potomky. Čím vyšší má jedinec svou fitness hodnotu, tím vyšší je pravděpodobnost, že bude vybrán k reprodukci. Zohledněn ale může být např. věk jedince.

2.3.7 Schéma evolučního algoritmu

Obecné schéma evolučního algoritmu v pseudokódu může vypadat např. takto [14, p. 6]:

```
Begin
  INICIALIZACE populace náhodnými kandidátními řešeními;
  EVALUACE každého kandidáta;
  Repeat Until (je splněna UKONČOVACÍ PODMÍNKA) Do
    1. SELEKCE rodiců;
    2. REKOMBINACE párů rodičů;
    3. MUTACE vzniklých potomků;
    4. EVALUACE nových kandidátů;
    5. SELEKCE jedinců pro další iteraci;
  endDo
End.
```

Tedy před vlastní optimalizací je vygenerována populace jedinců, neboli kandidátních řešení na daný problém. Pro některé problémy mohou být počáteční kandidátní řešení zvolena i náhodně z určitého stavového prostoru obsahujícího hledané řešení. Poté se provede fitness ohodnocení každého kandidáta, které je určeno mírou, do jaké se kandidát svými určitými vlastnostmi blíží hledanému optimu.

Dále následuje vlastní cyklus evolučních kroků (iterací). V každém kroku se podle vypočtených pravděpodobností zvolí z populace rodiče. Rodiče pomocí procesu rekombinace vytváří jednoho nebo více potomků. Vzniklí potomci mohou podstoupit mutaci. Nově vzniklým jedincům je vyhodnocena jejich fitness hodnota a na základě pravděpodobností pro eliminaci jedinců z populace jsou vybráni jedinci pro další iteraci.

Cyklus je opakován, dokud není splněna ukončovací podmínka, obvykle dokud není nalezeno dostatečně vhodné řešení nebo nebyl vyčerpán předem stanovaný maximální počet celkových fitness ohodnocení za dobu běhu algoritmu.

Kapitola 3

Algoritmus kontinuální evoluce

Zde jsou popsány základní charakteristiky jednoho z evolučních algoritmů, *algoritmu kontinuální evoluce* (Continual Evolution Algorithm, CEA).

3.1 Motivace

U mnoha problémů evoluční komputace hraje klíčovou roli čas aplikace. Protože výpočet fitness hodnocení jedince může být časově nákladný, je cílem algoritmu kontinuální evoluce minimalizovat právě počet provedených ohodnocení fitness během výkonu algoritmu. Toho lze dosáhnout dvěma způsoby:

- Správnou regulací populace, aby v ní setrvávali pouze jedinci se slibnou fitness nebo jedinci, jejichž fitness se během jejich života dostatečně zlepšuje díky procesu učení (optimalizace lokální prohledávací funkcí).
- Vhodným lokálním prohledáváním, které může vést ke snížení počtu iterací celého evolučního procesu a tím i ke snížení potřebného počtu fitness vyhodnocení.

Další výhodou CEA je, že umožňuje reprezentovat jak strukturální část jedince, tak část behaviorální.

Kombinace evolučního algoritmu a procesu učení jedinců během doby jejich života v populaci se nazývá *hybridizace*. Jedním z hybridních evolučních algoritmů je právě algoritmus kontinuální evoluce.

3.2 Lokální prohledávání

Evoluční algoritmus je tzv. *hybridní*, jestliže pro hledání řešení kombinuje evoluční algoritmus a lokální prohledávací metodu, tzv. *adaptaci*, přičemž evoluční algoritmus je aplikován na celou populaci, ale lokální prohledávání na každého jedince zvlášť.

Lokální prohledávání je metoda, která iterativně zkoumá množinu bodů v sousedství současného řešení a nahrazuje současné řešení lepším sousedním, existuje-li. Tři komponenty ovlivňují fungování lokálního prohledávacího algoritmu. Jsou to [14, p. 10]:

- *pivot rule:* definuje kritéria pro akceptování zlepšujícího se bodu (fitness jedince).
- *depth condition:* definuje ukončovací podmínku lokálního prohledávání pro vnější smyčku.
- *neighborhood generating function:* definuje množinu bodů, které mohou být dosaženy aplikací.

Pseudokód lokálního prohledávání může vypadat např. takto [14, p. 11]:

```
// je dáno počáteční řešení i a sousedství funkce n
Begin
  best = i;
  iterations = 0;

  Repeat Until ( depth condition je splněna ) Do
    count = 0;

    Repeat Until (pivot rule je splněno) Do
      generovat dalšího souseda j
      count = count + 1;
      If (f(j) je lepší než f(best)) Then
        best = j;
      endIf
    endDo

    i = best;
    iterations = iterations + 1;
  endDo
End.
```

3.3 Kódování jedinců

CEA umožňuje uchovávat jak strukturální část jedinců (obvykle se jedná o topologii), tak behaviorální (vlastnosti jedince změněné lokální prohledávací funcí). Strukturální část jedince se během jeho života nemění, behabiorální se měnit může procesem učení (lokální prohledávací metodou). Jedinec je vlastně reprezentován třemi složkami:

- svou **strukturou S**, která se mění pouze při rekombinaci a případné mutaci, ale ne při adaptaci (lokálním prohledávání)
- svým **instinktem I**, tj. svým vrozeným chováním. Instinkt se používá při rekombinaci.
- svým **chováním B**, tj. svým naučeným chováním. Chování se může změnit adaptací a používá se pro rekombinaci, takže CEA tedy vlastně stojí na Lamarckově teorii evoluce.

3.4 Proces stárnutí a elitismus

Dalším klíčovým principem algoritmu kontinuální evoluce je koncept *stárnutí* jedinců v populaci. Myšlenkou procesu stárnutí je poskytnout jedincům určitý čas, během kterého mohou využít proces učení (adaptace), než aby byly předčasně eliminováni z populace. Některá řešení se totiž mohou zpočátku jevit jako nevhodná, ale adaptací se mohou později výrazně vylepšit, co se týče jejich fitness hodnoty. Jedinci si uchovávají svůj věk (zpočátku je nula), který reflektuje počet cyklů, po které se jedinci nacházejí v populaci evolučního algoritmu.

Elitismus představuje způsob, jak zajistit, aby velmi slibný jedinec nebyl vyhozen z populace kvůli svému věku. Využívá se různých implementací, jak toho dosáhnout, např. lze při každém cyklu evolučného algoritmu nastavit věk nejlepšího jedince na nulu.

3.5 Řízení velikosti populace

Pro řízení velikosti populace slouží operátory rekombinace a eliminace. Selekce jedinců pro reprodukci je odvozována z pravděpodobnosti reprodukce jedince, eliminace z pravděpodobnosti jeho smrti. Pravděpodobnosti jsou počítány s přihlédnutím na velikost populace.

3.6 Schéma algoritmu kontinuální evoluce

Nakonec uveďme, jak může vypadat schéma implementovaného algoritmu kontinuální evoluce:

```
inicialize()          // inicializuje se populace náhodnými řešeními

while(not TERMINATION_CONDITION) {
    evaluate();         // ohodnotí jedince v populaci
    saveFittest();      // elitismus
    adapt();            // lokální prohledávací metoda
    eliminate();        // selekce jedinců pro eliminaci z populace a jejich eliminace
    reproduce();        // reprodukce zvolených jedinců a jejich přidání do populace
    update();           // inkrementace věku
  }
```

Kapitola 4

Implementace

Tato kapitola stručně pojednává o mnou zvolené implementaci algoritmu kontinuální evoluce.

4.1 BBOB

Tato sekce slouží jako úvod k *Black-Box Optimization Benchmarking* (BBOB), který byl implementovaným algoritmem CEA využit pro účely testování.

Kvantifikace a porovnání výkonnosti (benchmarking) numerických optimalizačních algoritmů je důležitým aspektem výzkumu v optimalizaci. Dosažení tohoto cíle umožňuje platforma COCO (COmparing Continuous Optimisers)[3], a to pro algoritmy v jazycích C, Java a Python. Využívá tzv. Black-Box Optimization Benchmarking (BBOB) [2], tj. při porovnávání výkonnosti numerických optimalizačních algoritmů se pro danou shodnou množinu vstupů srovnávají jejich výstupy bez uvažování vlastní vnitřní struktury algoritmů.

COCO poskytuje pro účely BBOB sadu testovacích funkcí a vlastní metodologii testování.

4.1.1 Testovací funkce

Výběr testovacích funkcí se snaží reflektovat, alespoň do určité míry a s několika vyjímkami, obtížnější část problému distribuce funkcí, které se vyskytují v praxi[10]. Všechny poskytnuté funkce mají navíc následující vlastnosti[2, p. 3]:

- *dimenze*: Každá funkce je škálovatelná svojí dimenzí z množiny {2, 3, 5, 10, 20, 40}.
- *instance*: Pro každou funkci mohou být generovány rozdílné instance z množiny instancí {1, ..., 5, 41, ..., 50}, pro něž jsou odlišně voleny náhodně přiřazované konstanty.
- *minimum*: Všechny funkce mají své globální optimum v $[-5, 5]^{dim}$, kde *dim* je dimenzionalita funkce.

Pro účely testování bylo použito všech 24 poskytnutých funkcí bez šumu (noiseless) pro všechny dimenze z dané množiny dimenzí a všechny instance z dané množiny instancí. Mezi funkcemi se vyskytují jak unimodální (mají jen jeden lokální extrém), tak multimodální funkce (mají mnoho lokálních extrémů).

Vizualizace všech 24 BBOB funkcí je uvedena v příloze D.

4.1.2 Testovací metodologie

Pro každou funkci lze zjistit její *optimální funkční hodnotu*, tj. hodnotu jejího globálního minima. Podle BBOB dokumentace [2, p. 3] se smí tato optimální hodnota použít pro účel ukončení experimentu, ale samozřejmě ne jako vstupní informace algoritmu. Vstupní informací algoritmu nesmí být ani jakákoliv známá charakteristika funkce, pro kterou právě probíhá vyhodnocování, např. její identifikátor. Dále musí být pro všechny experimenty identické nastavení parametrů algoritmu. [2, p. 3]. Implementovaný CEA všem těmto požadavkům vyhovuje.

Ukončovací kritérium algoritmu není předdefinováno, jeho volba je relevantní částí algoritmu. Implementovaný algoritmus CEA se ukončuje po dosažení maximálního počtu provedených fitness ohodnocení nebo dříve, pokud bylo nalezeno řešení, které se od hledaného optima funkce liší o méně než povolenou předdefinovanou maximální odchylku poskytnuté knihovny.

Nyní uvedu příklad zjednodušeného pseudokódu vyvolávání experimentů na BBOB testovacích funkcích, který vyhovuje COCO metodologii[2, p. 3]:

```
dimensions = {2, 3, 5, 10, 20, 40};    // dimenze funkcí
functions = {1, 2, ..., 24};           // identifikátory funkcí
instances = {1, ..., 5, 41, .., 50};   // instance funkcí

for dim in dimensions
  for fun in functions
    for ins in instances {
      maxEvaluations = 50*dim;                // max. počet fitness ohodnocení
      bbob.init(dim, fun, ins);               // inicializace experimentu
      CEA_OPTIMIZER(bbob, maxEvaluations);    // provedení experimentu algoritmem
    }
```

4.2 JCOOL

Zde se nachází úvod k použité knihovně *Java Continuous Optimization Library* (JCOOL), z níž CEA využívá pro lokální prohledávání některé algoritmy. JCOOL je dostupná např. v repozitáři Ing. Jana Drchala, Ph.D.[6].

JCOOL byla předložena jako open-source Java knihovna pro kontinuální optimalizaci v roce 2009 na ČVUT. Jedná se o framework pro implementaci vlastních metod pro kontinuální optimalizaci, specifikaci testujících funkcí a statistické vyhodnocování. Cílem JCOOL bylo oddělit implementaci algoritmů od implementace testujících funkcí. Umožňuje použití každého z implementovaných algoritmů k vyřešení každé z implementovaných testujících funkcí v rámci stejného prostředí a podmínek[4, p. 2].

JCOOL framework definuje rozhraní *OptimizationMethod*, který specifikuje 3 hlavní metody optimalizačního procesu: inicializaci, která je volána dříve, než k optimalizaci dochází,

a předává referenci na implementaci rozhraní optimalizační funkce *ObjectiveFunction*, ukončovací podmínky a optimalizační krok, který implementuje iteraci optimalizačního procesu.

Pro účely demonstrace funkčnosti CEA byly vybrány 2 numerické metody z JCOOL: *Quasi-Newton* (QN) a *Conjugate Gradient* (CG), dále 1 inspirovaná přírodou (*nature-inspired*): *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) a některé další.

4.2.1 Numerické metody

Numerické metody jsou založeny na matematických principech, které jsou využity za účelem iterace stavovým prostorem funkce k nalezení jejího lokálního minima.

Mezi využité numerické metody z JCOOL knihovny patří např. metoda *Conjugate Gradient*.

Conjugate Gradient je jednou z tzv. gradientních metod. Gradientní metody používají k nalezení lokálního minima informaci získanou komputací tzv. *gradientu* funkce v určitém bodě. Gradient je vektor, který pro daný bod x ukazuje ve směru největšího růstu funkční hodnoty *f(x)*[11, p. 10]. Jedná se o transpozici totální derivace reálné funkce[13].

Mezi další použitou numerickou a gradientní metodu patří i tzv. *Quasi Newton metoda* [4, p. 10].

4.2.2 Metody inspirované přírodou

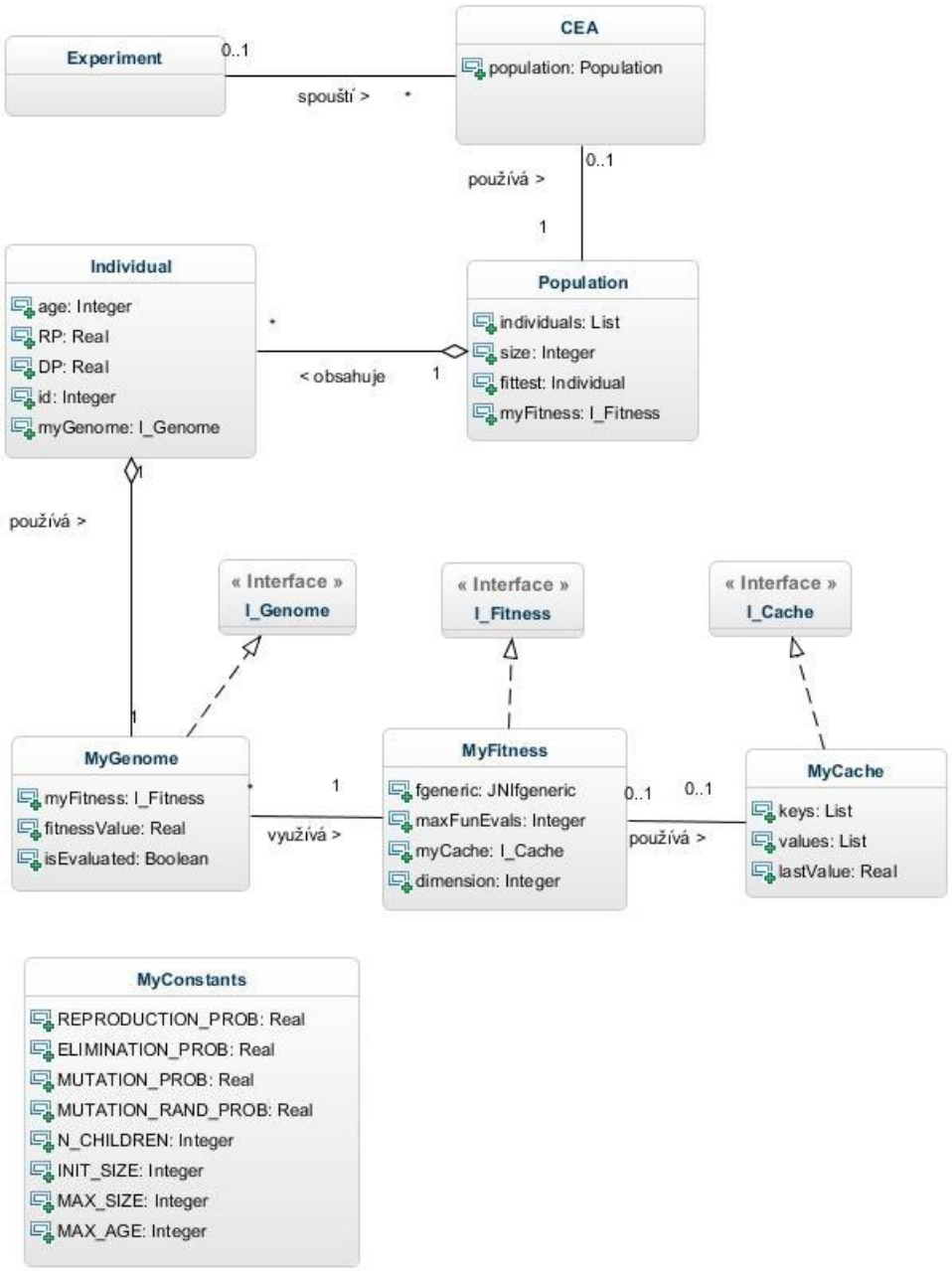
Důležitou charakteristikou tzv. *metod inspirovaných přírodou* je, že při optimalizaci nepotřebují užívat výpočet analytického gradientu a přistupují k funkci pouze jako k *černé skříncе*, která pouze dává informaci o našich kandidátech na řešení.[4, p. 16]

Z metod inspirovaných přírodou byla využita metoda *Covariance Matrix Adaptation Evolution Strategy (CMA-ES)*, která sama využívá evoluční přístup hledání optimálního řešení ve stavovém prostoru kandidátních řešení.

4.3 Návrh

Nejprve budu diskutovat svou volbu návrhu tříd a proměnných pro algoritmus kontinuální evoluce. Při návrhu algoritmu bylo třeba mít na paměti následující dvě hlediska. Algoritmus by měl být implementován dostatečně obecně, aby se dal aplikovat i na jiné problémy než hledání extrémů funkcí, např. na optimalizaci neuronových sítí. Zároveň by ale měl být dostatečně rychlý a přehledný.

Nakonec jsem zvolil následující návrh, který je určitým kompromisem obou hledisek.



Obrázek 4.1: Diagram tříd CEA.

4.3.1 Třída Experiment

Třída Experiment vychází z obdobné třídy poskytnuté v rámci benchmarku BBOB, kde slouží ke spouštění zvolených experimentů (viz 4.1). V našem případě zde dochází především k opakovanému volání kroku evolučního algoritmu.

Nyní uvedu zjednodušený způsob vyvolání algoritmu CEA pro určitou funkci BBOB dané dimenze. *FGeneric* je zde třídou z BBOB starající se o generování funkčních hodnot, *maxFunEvals* je maximální počet žádoucích funkčních ohodnocení algoritmu, po jejichž dosažení algoritmus nejpozději končí, třída *MyFitness* obstarává výpočet *fitness hodnoty*, *CEA* je hlavní třída algoritmu (viz následující sekce), *dim* je dimenze aktuálně vyhodnocované funkce. Algoritmus CEA lze volat následujícím způsobem:

```
MyFitness myFitness = new MyFitness(fgeneric, (int)maxFunEvals);
Cea cea = new Cea(myFitness, dim, fgeneric);
```

Dále stačí opakovaně volat krok evolučního algoritmu:

```
while(! myFitness.noMoreEvaluations()) {
                                    // počet fitness ohodnocení nevyčerpán
  cea.step();                       // krok evolučního algoritmu
}
```

4.3.2 Třída CEA

Třída CEA implementuje hlavní rysy evolučního algoritmu. Reprezentuje globální pohled na algoritmus. V metodě

```
public void step(int generation)
```

se nachází jeden krok již diskutovaného hlavního těla evolučního algoritmu (viz kapitola 3.6). Třída CEA si uchovává svou populaci jedinců (kandidátních řešení). Dále slouží jako určitý rozcestník zanořeným třídám.

4.3.3 Třída Population

Zde se řeší úkoly zahrnující populaci jedinců. Kromě jedinců si populace uchovává také svou velikost a své dosud nejlepší nalezené řešení. Třída zároveň umožňuje jedincům přiřazovat identifikátory, které jsou unikátní v rámci celého běhu evolučního algoritmu, čehož lze využít pro případnou budoucí vizualizaci života jedinců v populaci.

4.3.4 Třída Individual

Každý jedinec je pro přehlednost kódován unikátním identifikátorem. O jedinci je znám jeho věk a současná pravděpodobnost smrti (DP) a reprodukce (RP). Jedinec má určen svůj genom.

Pravděpodobnostní funkce představují klíčový způsob, jak řídit velikost populace. V CEA rozlišujeme dva druhy pravděpodobností: tzv. *hrubé* (raw) a *vyvážené* (balanced). Pro kalkulaci pravděpodobnosti smrti a reprodukce jedince se nejprve spočtou pravděpodonosti hrubé, které se odvozují z jedincova věku a jeho fitness hodnoty normované vůči celé populaci. Vyvážené pravděpodobnosti navíc vezmou v úvahu vliv velikosti populace na její budoucí rozvoj. Využívá se zde opět inspirace přírodním procesem evoluce, kdy velikost větších populací dlouhodobě neroste při omezených životních zdrojích tak rychle, jako malá populace, která má zdrojů dostatek.

Funkce na výpočet hrubých i vyvážených pravděpodobností jsou převzaty z [5, p. 28].

Pro výpočet hrubých pravděpodobností lze tedy použít např. následující funkce:

```
//---------------------------------------------------------------------------
private double calcRawDP(double a, double f) { // age, fitnessValue
  return Math.min(Math.max(f*Math.pow(a, 10) + 3*a*(1-f), 0), 1);
}

//---------------------------------------------------------------------------
private double calcRawRP(double a, double f) { // age, fitnessValue
  return (1-a)*f + a*f/2;
}
```

Hrubá pravděpodobnost reprodukce se s rostoucí fitness hodnotou zvyšuje a věkem snižuje. U hrubé pravděpodobnosti smrti je tomu naopak. To umožňuje jedincům s vyšší fitness hodnotou spíše setrvat v populaci do dalšího kroku algoritmu a podílet se na reprodukci.

Výsledné pravděpodobnosti pak vychází z pravděpodobností hrubých a z velikosti (normované) populace:

```
DP = calcDP(calcRawDP(normAge, normFitness), normSize);
RP = calcRP(calcRawRP(normAge, normFitness), normSize);
```

Konkrétní implementace může vypadat například takto:

```
private double calcDP(double rawDP, double n) { // rawDP, population size
  return Math.min(Math.max(n*rawDP, 0), 1);
}

//---------------------------------------------------------------------------
private double calcRP(double rawRP, double n) { // rawRP, population size
  return Math.min(Math.max((1-n)*(1-n)*rawRP, 0), 1);
}
```

4.3.5 Třída MyGenome

Třída je pro větší pružnost algoritmu navržena jako implementace rozhraní Genome. MyGenome obsahuje konkrétní implementaci genomu jedince. Může obsahovat kupříkladu proměnné pro kódování topologie (struktury) a vrozených a získaných vlastností určité neuronové sítě. Konkrétní implementace je v rukou uživatele. Současná třída je implementována pro vyhledávání extrému funkce a genetická reprezentace jedince je reálná (viz kapitola 2.3.1). Třída MyGenome si také pamatuje již dříve vypočtenou fitness hodnotu pro svůj genom.

4.3.6 Třída MyFitness

Třída MyFitness rovněž ponechává uživateli volnost implementace. Dochází zde k výpočtu fitness hodnoty. Jako fitness hodnotu může například vracet funkční hodnoty pro daný vektor (hledáme-li maximum funkce). Současná implementace před vyhodnocením fitness hodnoty ověřuje, že nebyl dosažen maximální počet fitness ohodnocení. Také implementace využívá jednoduchou cache paměť, aby nedocházelo k opakovanému fitness vyhodnocování dostatečně stejných vektorů (viz dále třída MyCache).

4.3.7 Třída MyCache

Jelikož je hlavním cílem CEA vyvarovat se zbytečných vyhodnocování fitness, nabízí se jako vhodné implementovat cache a použít ji pro již dotazovanou sadu genů. V implementované třídě lze nastavit minimální rozdíl hodnot, pro který jsou například dvě reálná čísla již cache pamětí považována za rozdílná.

4.3.8 Třída MyConstants

Všechny statické konstanty můžeme mít uloženy např. v samostatné třídě, případně souboru.

4.4 Hierarchie složek

Zde je uveden pouye přehled struktury složek elektronické verze bakalářské práce. Elektronická verze obsahuje dvě podsložky:

- *Práce*: obsahuje soubory Latex projektu a výsledný PDF soubor.
- *Program*: obsahuje Java projekt a součásti k jeho překladu a spuštění. Projekt se nachází ve složce *CEA*. Složka *bbob* obsahuje BBOB knihovnu [3], *JCOOL-master* JCOOL knihovnu [6], *CEA2* starší verzi programu funkční k lednu 2015 a složka *maven* obsahuje Maven 2.

Vlastní projekt (viz složka *Program/CEA*) obsahuje následující balíčky, které zahrnují:

- *solution.cea*: zahrnuje vlastní CEA algoritmus.
- *solution.utils*: základní soubory JCOOL knihovny uzpůsobené pro CEA.

- *JCOOL-master*: nalézá se v ní podmnožina souborů JCOOL knihovny pro projekt.
- *methods.cmaes*: CMA-ES optimalizační metodu JCOOL knihovny uzpůsobenou pro CEA projekt (viz kap. 5.3)
- *methods.gradient*: Conjugate Gradient a Quasi Newton metody JCOOL knihovny uzpůsobené pro CEA projekt (viz kap. 5.3).
- *javabbob*: navíc příklady testování pomocí BBOB [10].

4.5 Instalace

Zde se nachází stručný návod k úspěšnému spuštění programu.

Nutné podmínky chodu programu jsou následující [6]:

- Úspěšná instalace Java verze 1.5 a vyšší používající 32-bit JDK verzi. Je třeba také nastavit systémové proměnné prostředí JAVA_HOME a PATH. K ověření funkčnosti stačí provést příkazy:

  ```
  java -version
  javac -version
  ```

- Úspěšná instalace Maven 2. Stačí zkopírovat z CD složku *maven* a nastavit systémovou proměnnou PATH na *maven/bin*. K ověření funkčnosti pak stačí příkaz:

  ```
  mvn --version
  ```

- Instalace BBOB knihovny. Stačí zkopírovat z CD složku *bbob*. Dále je nutné nastavit systémovou proměnnou PATH ve Windows (respektive LD_LIBRARY_PATH v Linuxu, DYLD_LIBRARY_PATH v Mac OS) na cestu do složky *bbob/java*. Poté ve složce *bbob/java/javabbob* zkompilovat zdrojové soubory a ověřit kompilaci např. příkazy:

  ```
  javac javabbob/*.java
  java javabbob.ExampleExperiment
  ```

Program by měl po instalaci těchto komponent fungovat. Ale protože se při překladu stahují některé soubory ze stránek [12], jejichž dostupnost není v čase stabilní, může být potřeba ještě nakompilovat a nainstalovat dvě nejvýše postavené složky JCOOL knihovny *configurations* a *jcool*, nacházející se v *JCOOL-master*, a případně optimalizovat závislosti v projektu. Toho se docílí příkazy [6]:

```
cd configurations
mvn compile
mvn install
cd..
cd jcool
mvn compile
mvn install
```

Kapitola 5

Experimenty

Tato kapitola ukazuje přehled některých experimentů, které byly vykonány pro demonstraci funkčnosti implementace a otestování aplikace CEA.

Pro účely testování byl použit benchmark Black-Box-Optimization-Benchmarkink (BBOB)[3] (viz kap. 4.1), jeho platforma Comparing Continuous Optimizers (COCO), soubor funkcí BBOB [10] a knihovna JCOOL [6] (viz kap. 4.2).

Vizualizace BBOB funkcí je uvedena v příloze D.

Pro potřeby testování jsem provedl například následující experimenty:

- *Porovnání s BBOB*: porovnání implementovaného algoritmu CEA s genetickým algoritmem GA-100, jehož výsledky jsou k dispozici na stránkách BBOB[1]. Porovnání sloužilo především k ověření efektivity implementovaného algoritmu CEA.
- *Analýza konstant*: porovnání výsledků algoritmu CEA za odlišných konfigurací konstant programu.
- *Srovnání CEA a optimalizačních metod*: porovnání algoritmu CEA s několika optimalizačními metodami knihovny JCOOL pro demonstraci funkčnosti implementace algoritmu.
- *Junit testy*: provedení základních junit testů.

5.1 Porovnání s BBOB algoritmem

5.1.1 Problematika

Pro informace o BBOB benchmarking a COCO platformě viz kapitolu 4.1.

Pro otestování efektivity algoritmu CEA jsem se rozhodl porovnat jeho výsledky s výsledky některého z algoritmů, jenž je dostupný na stránkách BBOB[1]. Výhodou tohoto přístupu je, že oba algoritmy svá řešení hledají nad stejnou množinou BBOB funkcí, a jsou tedy pro stejné vstupní funkce přímo porovnatelné. Pro porovnání jsem zvolil genetický algoritmus GA-100 [1] z roku 2013, neboť z algoritmů dostupných na stránkách BBOB vykazuje největší příbuznost s CEA.

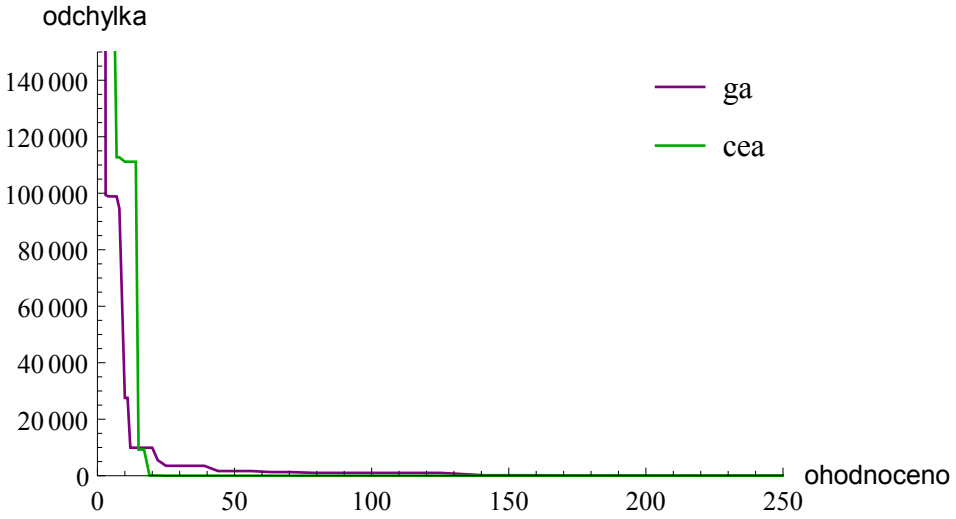
Pro velmi zjednodušené schéma experimentů aplikace na testovacích funkcích BBOB viz kapitolu 4.1

5.1.2 Výsledky

Pro účely testování se uvažovalo všech 24 poskytnutých BBOB funkcí bez šumu (noiseless) pro všechny dimenze z dané množiny dimenzí {2, 3, 5, 10, 20, 40} a všechny instance z dané množiny instancí {1, ..., 5, 41, .., 50}. Mezi funkcemi se vyskytují jak unimodální, tak multimodální funkce, BBOB funkce se snaží reflektovat obtížnější část zastoupení funkcí, které se vyskytují v praxu. Dospěl jsem k následujícím výsledkům pro algoritmus GA-100 a vlastní algoritmus CEA.

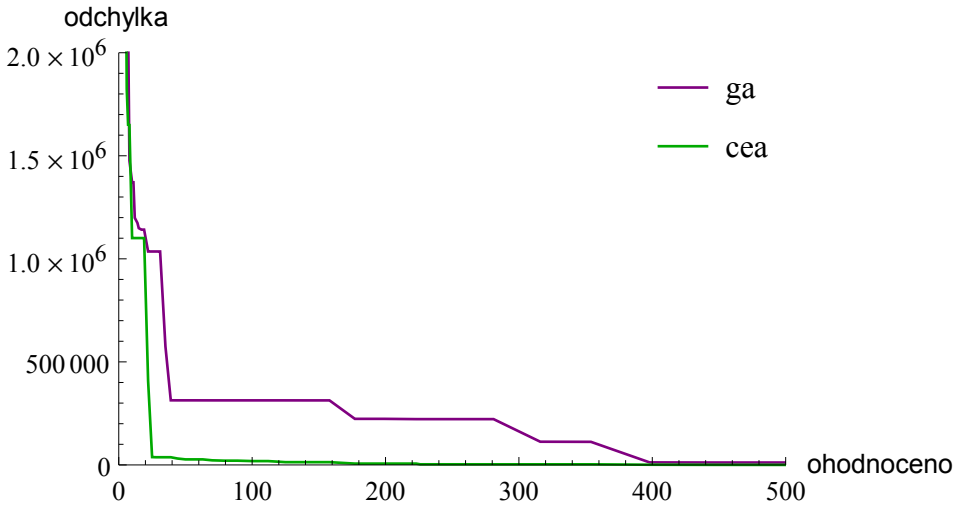
Pro přehlednost zde uvádím pouze některé výsledky, všechny jsou k dispozici v příloze A. Vizualizace BBOB funkcí je uvedena v příloze D.

Pro dimenzi 2 vychází CEA lépe po dosažení určitého minimálního počtu funkčních ohodnocení, které algoritmus provedl.



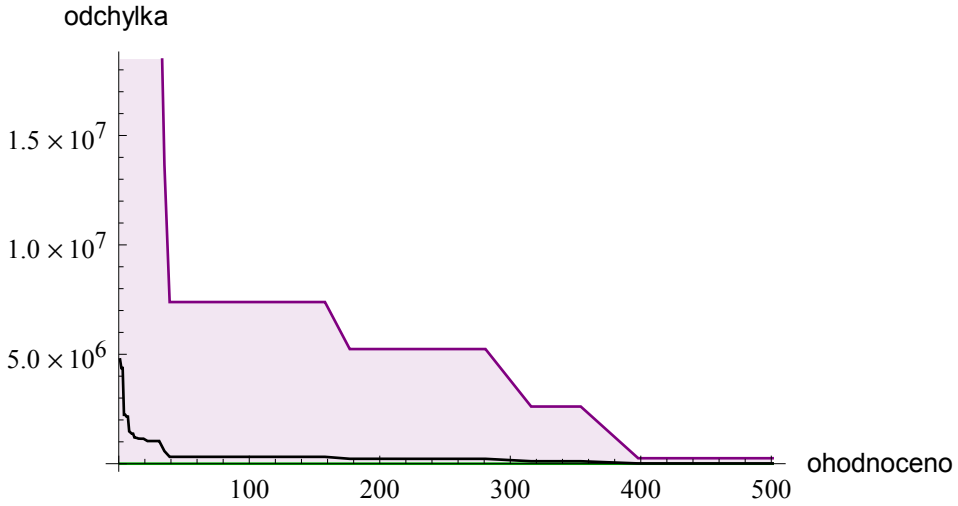
Obrázek 5.1: Závislost odchylky funkční hodnoty průměrného nalezeného řešení od optima (osa Y) na počtu provedených funkčních ohodnocení (osa X) pro všechny funkce dimenze 2.

Pro vyšší dimenzi 5 vychází CEA ve srovnání ještě lépe, přestože vyšší dimenze znamená, že fixní omezení maximálního počtu funkčních ohodnocení je více svazující. Při lokálním prohledávání se totiž uvažuje každá dimenze.

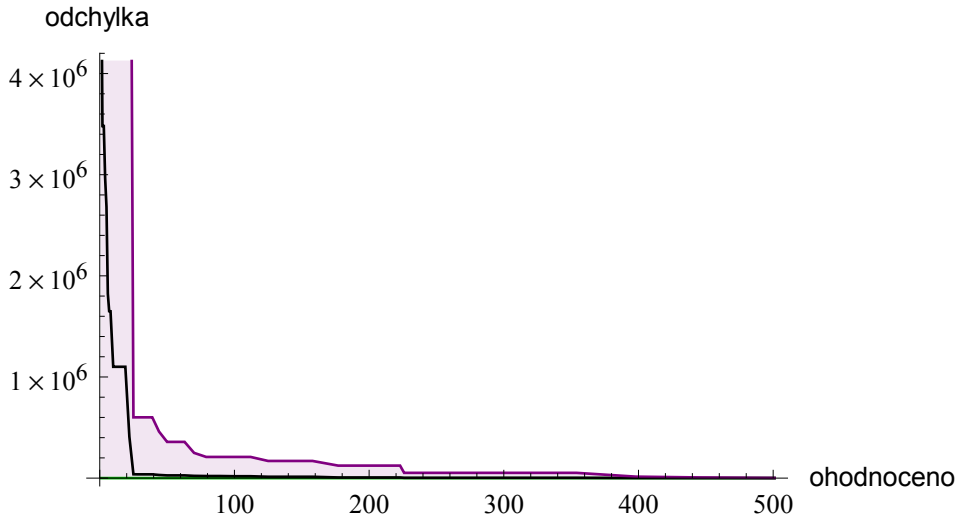


Obrázek 5.2: Závislost odchylky funkční hodnoty průměrného nalezeného řešení od optima (osa Y) na počtu provedených funkčních ohodnocení (osa X) pro všechny funkce dimenze 5.

Ještě zde uvedu grafy se zvýrazněnými minimálními, průměrnými a maximálními odchylkami funkcí dimenze 5 pro algoritmus GA-100 a CEA:



Obrázek 5.3: GA-100 algoritmus. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa Y) na počtu provedených funkčních ohodnocení (osa X) pro všechny funkce dimenze 5. Zvýrazněny hodnoty pro maximální, průměrnou a minimální odchylku.



Obrázek 5.4: CEA algoritmus. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa Y) na počtu provedených funkčních ohodnocení (osa X) pro všechny funkce dimenze 5. Zvýrazněny hodnoty pro maximální, průměrnou a minimální odchylku.

5.2 Analýza konstant

5.2.1 Problematika

Při svém běhu CEA využívá celé řady přednastavených konstant. Za nalezení optimální konfigurace konstant algoritmu pro danou sadu problémů se v případě hledání globálních minim sady funkcí dá považovat takové nastavení, pro které algoritmus nachází ve srovnání se všemi ostatními možnými konfiguracemi ty nejlepší výsledky v souhrnu za všechny funkce BBOB (tj. nalezená řešení dávající nejnižší funkční hodnotu, situaci může případně komplikovat rozptyl hodnot výsledků z více experimentů). Určení optimální konfigurace konstant CEA se dá vlastně samo považovat za optimalizační problém.

Protože CEA používá mnoho konstant, rozhodl jsem se kvůli transparentnosti porovnat pouze ty nejrelevantnější konstanty. Kromě dále uvedených CEA samozřejmě ještě využívá dalších konstant, třeba pravděpodobnosti mutace genomu jedince při rekombinaci nebo pravděpodobnosti reprodukce jedinců. Tyto konstanty byly voleny jako pevné.

Mezi základní konstanty CEA patří například počáteční a maximální velikost populace, maximální věk jedinců a počet iterací lokálního prohledávání pro jedince během jednoho kroku algoritmu.

Při vyhodnocování byl maximální počet funkčních ohodnocení nastaven na pětseti-násobek aktuálně vyhodnocované dimenze funkce.

V kapitole jsou dále užívány následující pojmy:

- *počet iterací adaptace*: je počet optimalizačních kroků lokálního prohledávání každého jedince za jednu generaci,

- *počáteční velikost populace*: je počet jedinců populace generovaných na počátku běhu algoritmu.
- *maximální velikost populace*: je horní limit počtu jedinců v populaci, při jehož dosažení není povolena reprodukce jedinců, dokud není velikost populace snížena eliminací jedinců.
- *maximální věk jedince*: je maximálním počtem kroků algoritmu, po který smí jedinec žít v populaci. Při dosažení tohoto limitu je jedinec z populace odstraněn, pokud se ovšem v souladu s implementovaným elitismem nejedná o řešení v populaci nejlepší.

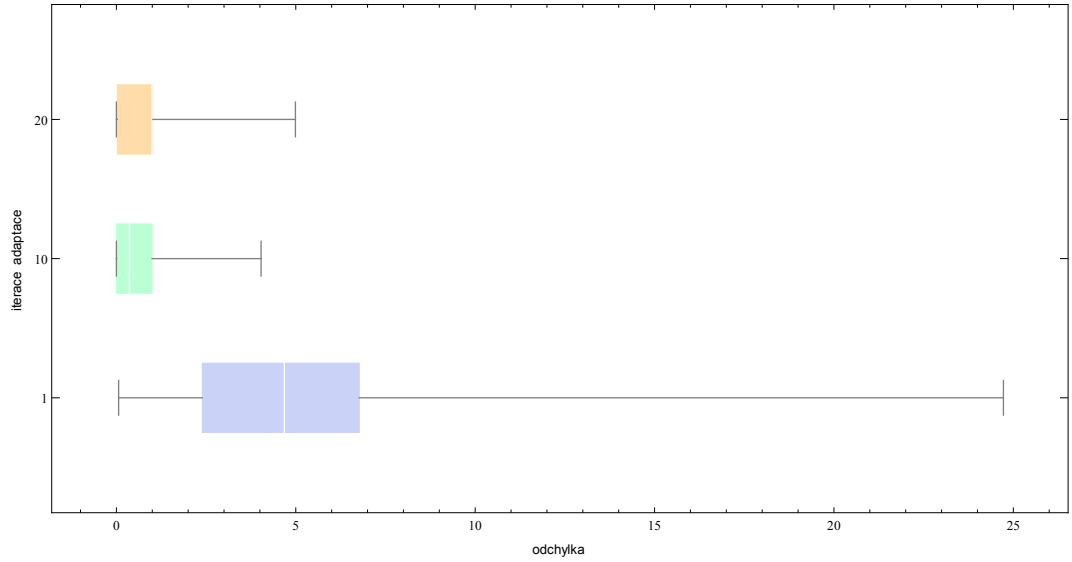
Ostatní konstanty zůstaly během výpočtu neměnné. Pro vyhodnocení byly pro přehlednost uvažovány následující instance konstant:

- *počet iterací adaptace*: 1, 10, 20
- *počáteční velikost populace*: 5, 10, 40
- *maximální velikost populace*: byla vždy dvojnásobkem počáteční, tj. 10, 20, 80
- *maximální věk jedince*: 4, 8, 16

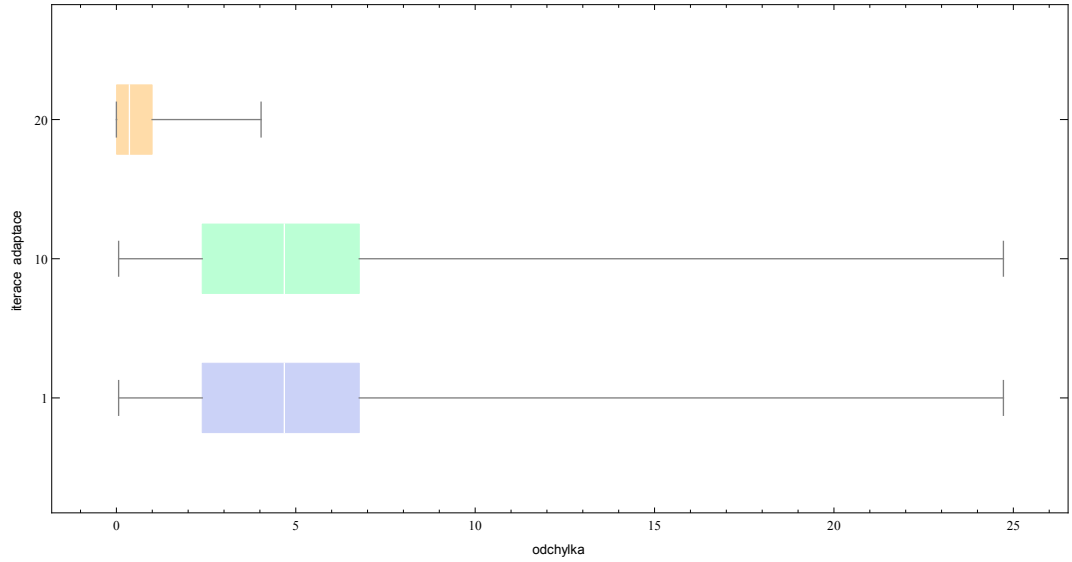
5.2.2 Výsledky

Experimenty ukázaly, že celkově nejvýraznější vliv na kvalitu získaných výsledků měl zřejmě počet kroků lokálního prohledávání (počet iterací adaptace). Uvádím zde pro srovnání grafy závislosti odchylky nalezeného řešení od optimálního na počtu iterací lokálního prohledávání pro dvě specifické funkce, jednu unimodální (má jen jeden lokální extrém) a druhou vysoce multimodální (má mnoho lokálních extrémů). Pro grafy byly použity hodnoty všech kombinací uvedených instancí konstant *maximální velikost populace* a *maximální věk jedince* a všech 15 instancí u každé funkce.

Vizualizace BBOB funkcí je uvedena v příloze D.



Obrázek 5.5: Multimodální funkce f_3. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na počtu iterací lokálního prohledávání (osa Y) po stejném počtu provedených funkčních ohodnocení (500-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.

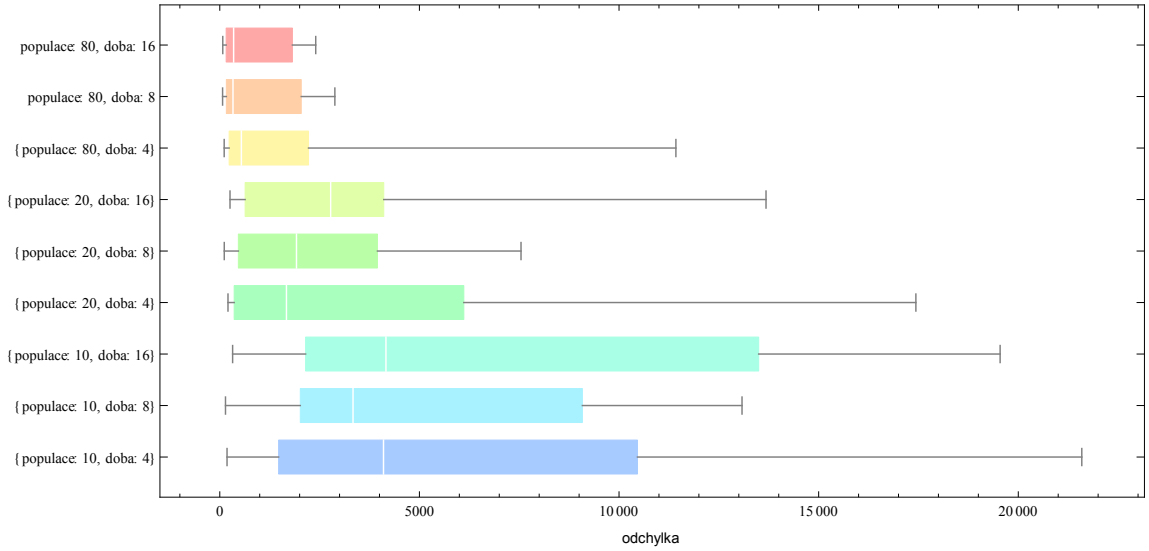


Obrázek 5.6: Unimodální funkce f_2. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na počtu iterací lokálního prohledávání (osa Y) po stejném počtu provedených funkčních ohodnocení (500-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.

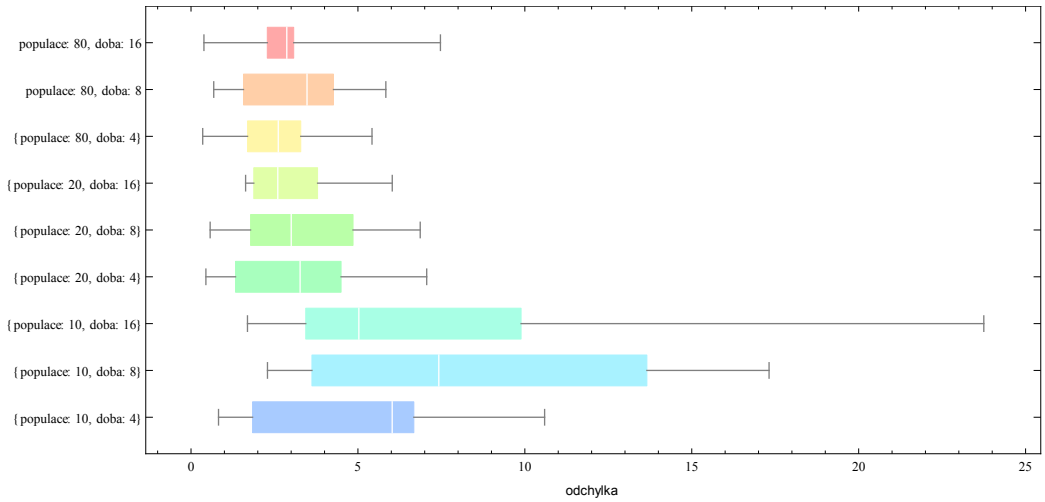
Je vidět, že lokální prohledávání (adaptace) je v tomto případě důležitější u výrazně multimodální funkce. Důvodem je skutečnost, že u multimodální funkce je při lokálním prohledávání vyšší pravděpodobnost, že dojde ke zlepšení stávajícího řešení nalezením lokálního minima než u unimodální, která má lokální minimum jediné.

V následujících grafech je patrné, že čím menší je počet iterací lokálního prohledávání

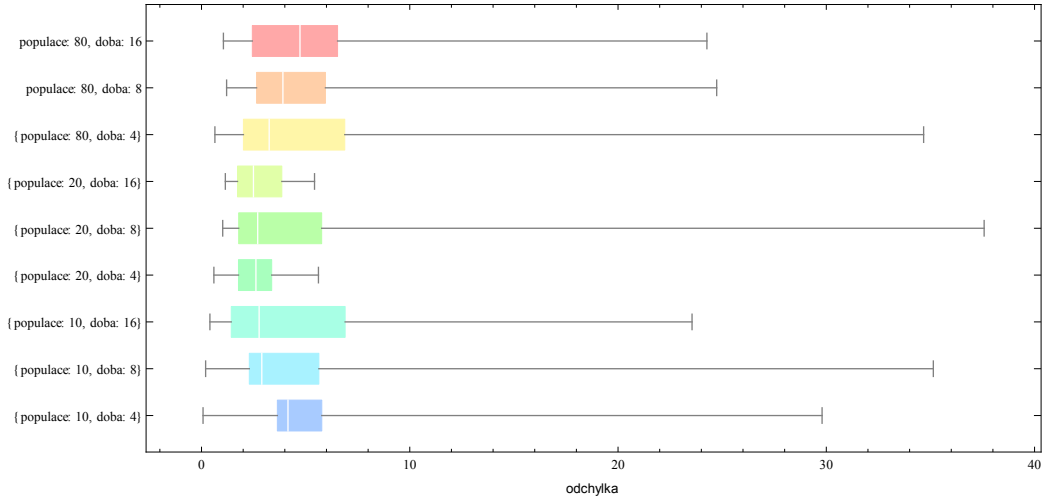
za jeden krok algoritmu, tím větší roli na kvalitu nalezeného řešení má počet jedinců v populaci. Důvodem je skutečnost, že při malém počtu iterací lokálního prohledávání za krok algoritmu se algoritmus velmi podobá nehybridizovanému evolučnímu algoritmu (viz kap. 3.2), u kterého může být větší počet jedinců v populaci výhodou kvůli většímu pokrytí oblasti, v níž se minimum hledá.



Obrázek 5.7: Počet iterací lokálního prohledávání: 1. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na maximálním počtu jedinců populace a maximální době dožití (osa Y) po stejném počtu provedených funkčních ohodnocení (500-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.



Obrázek 5.8: Počet iterací lokálního prohledávání: 10. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na maximálním počtu jedinců populace a maximální době dožití (osa Y) po stejném počtu provedených funkčních ohodnocení (500-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.



Obrázek 5.9: Počet iterací lokálního prohledávání: 20. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na maximálním počtu jedinců populace a maximální době dožití (osa Y) po stejném počtu provedených funkčních ohodnocení (500-krát dimenze funkce). Bráno souhrnně přes dimenze 2, 3, 5, 10, 20, 40.

5.3 Srovnání CEA a optimalizačních metod

5.3.1 Problematika

Pro demonstraci funkčnosti implementace jsem provedl také srovnání CEA, kombinujícího evoluční přístup s lokálním prohledáváním, a jiných optimalizačních metod z poskytnuté knihovny JCOOL [6], které neuplatňují přístup kontinuální evoluce, pro hledání minim BBOB funkcí. K porovnání jsem zvolil následující algoritmy (viz kap. 4.2):

- *CG*: metoda Conjugate Gradient (numerická metoda)[11]
- *QN*: metoda Quasi-Newton (numerická metoda)[4, p. 10]
- *CMA-ES*: metoda Covariance Matrix Adaptation Evolution Strategy (metoda inspirovaná přírodou, používá sama o sobě evoluční přístup)[4, p. 14]
- *CEA + QN*: CEA používající metodu Quasi Newton pro lokální prohledávání

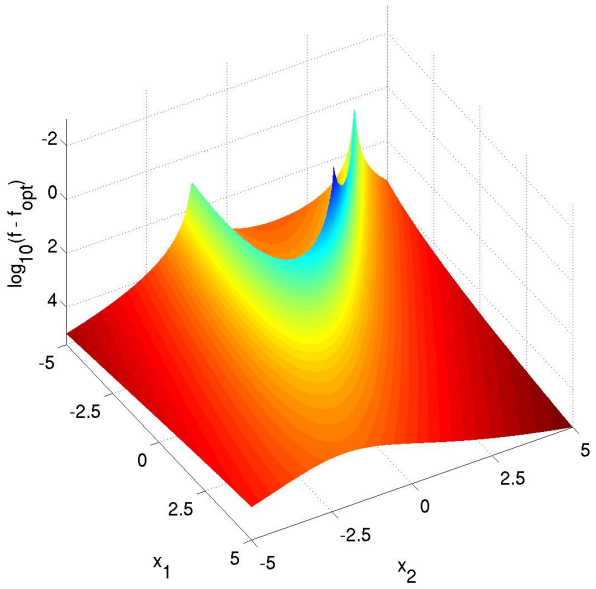
Při vyhodnocování byl maximální počet funkčních ohodnocení nastaven na padesátinásobek aktuálně vyhodnocované dimenze funkce.

5.3.2 Výsledky

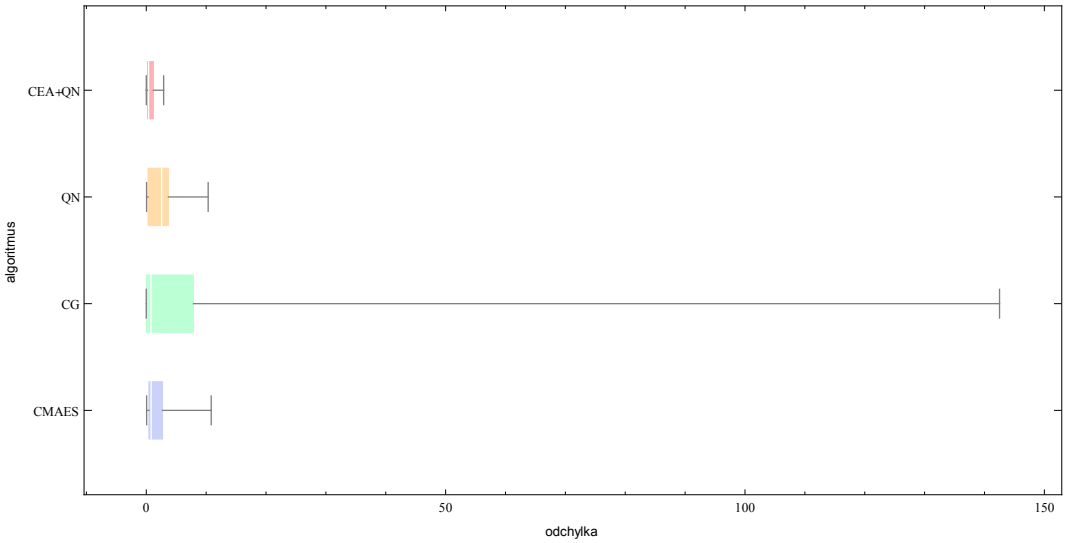
Žádný algoritmus se přirozeně neukázal být univerzálně nejlepším pro všechny funkce. Vlastnosti funkce, především míra, do jaké je multimodální, můžou značně ovlivnit, který z uvedených algoritmů je nejvhodnější použít.

Pro názornost uvádím výsledky pro dvě specifické funkce, Rosenbrockovu a Katsuurovu (viz dále). Výsledky použitých algoritmů pro všechny funkce jsou v příloze C. Pro vizualizaci všech BBOB funkcí viz přílohu D.

Pro funkce, které mají lokálních extrémů málo, jakou je např. Rosenbrockova funkce, se CEA ukazovalo jako lepší než algoritmus používající pouze lokální prohledávání.

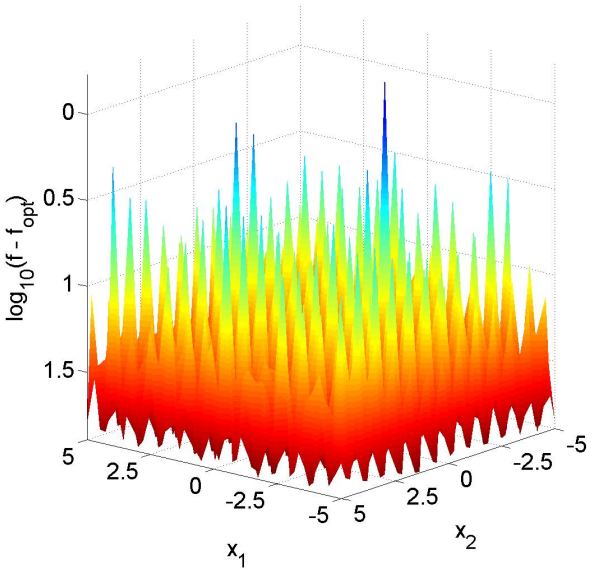


Obrázek 5.10: Rosenbrockova funkce f_9.

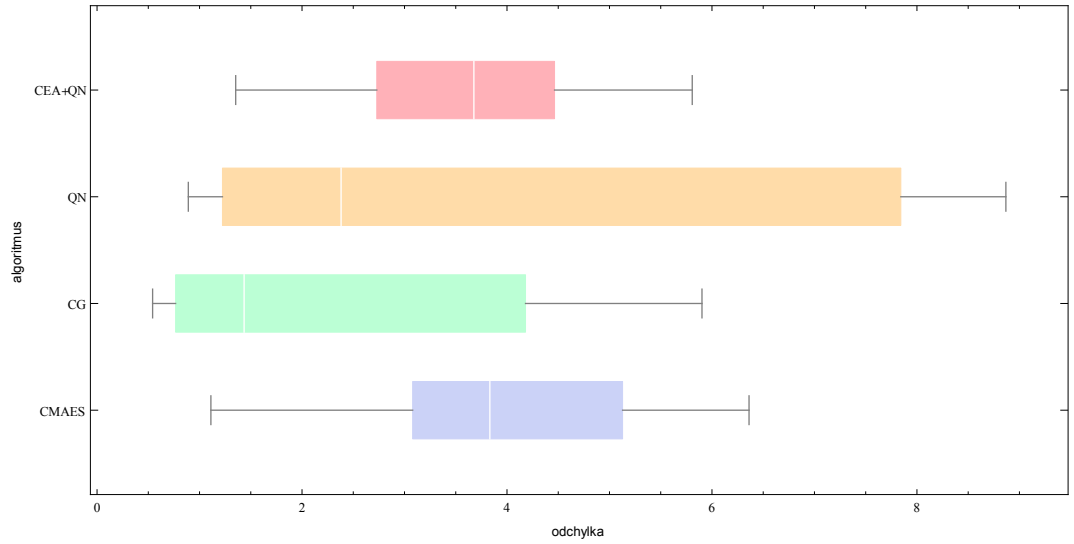


Obrázek 5.11: Porovnání výsledků algoritmů pro téměř unimodální funkci f_9. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na zvoleném algoritmu (osa Y) po stejném počtu provedených funkčních ohodnocení (50-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.

Naproti tomu Katsuurova funkce patřila k ojedinělému případu, kdy se lokální prohledávání vyplatilo více než evoluční algoritmus. Možným důvodem je skutečnost, že Katsuurova funkce je tak výrazně multimodální, že rekombinace nebo mutace jedinců může způsobit, že se výsledný jedinec ocitne mimo lokální minima, ke kterým se rodičovští jedinci blížili.

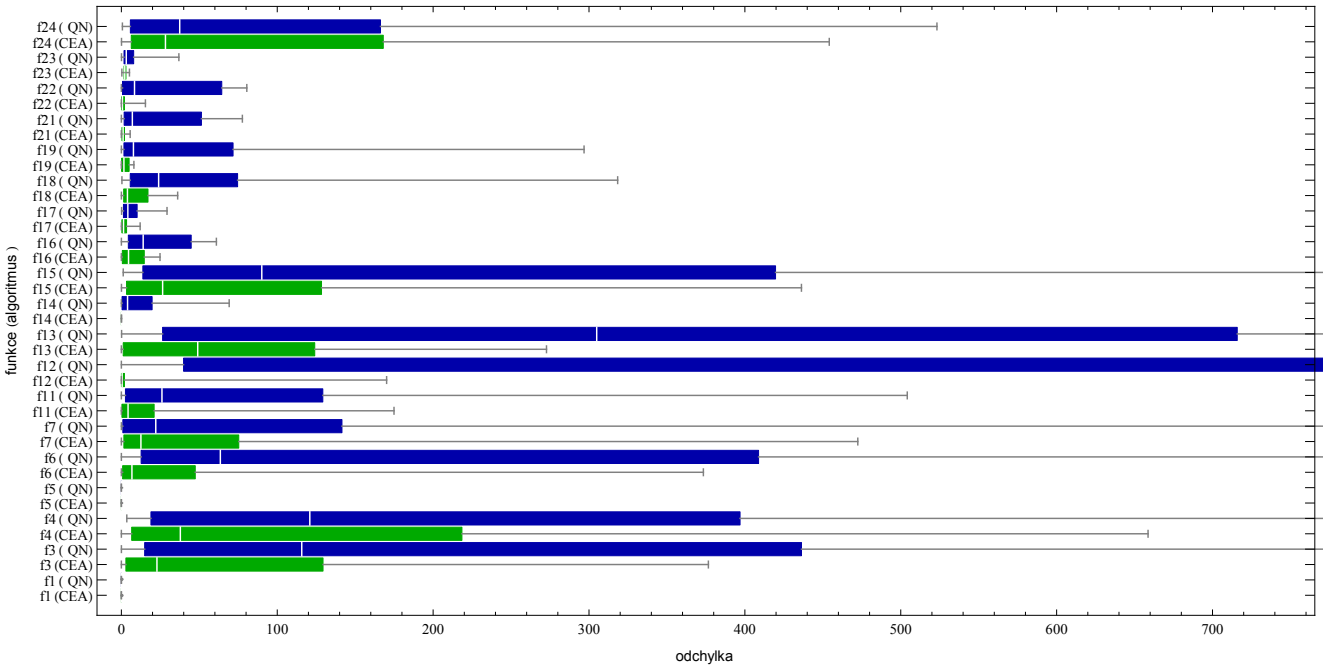


Obrázek 5.12: Katsuurova funkce f_{23}.

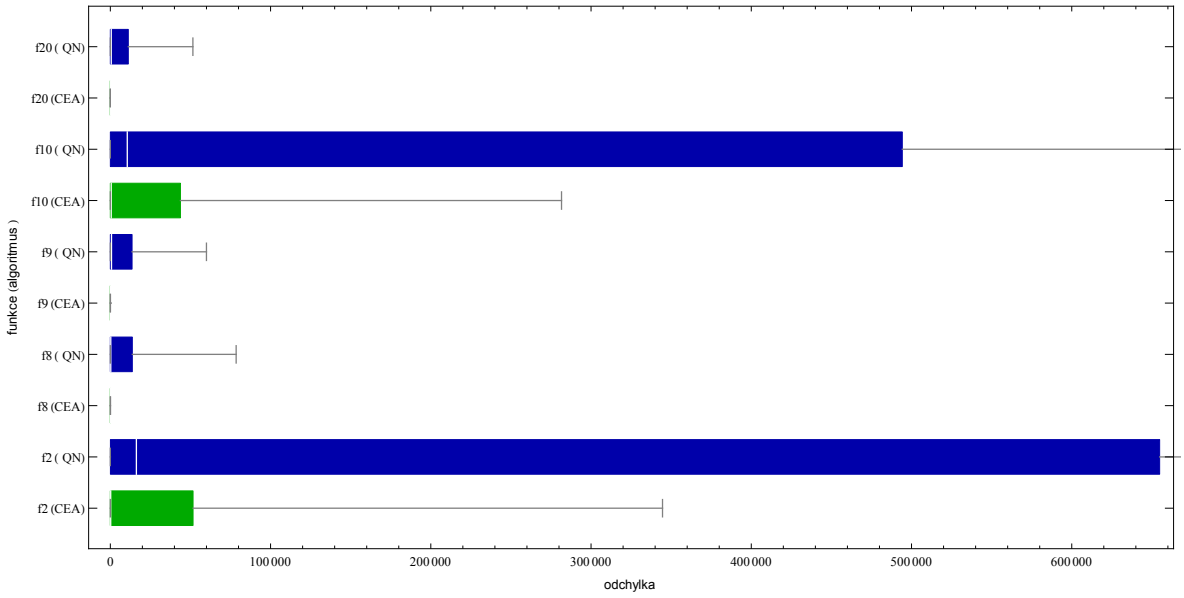


Obrázek 5.13: Porovnání výsledků algoritmů pro multimodální funkci f_{23}. Závislost odchylky funkční hodnoty nalezeného řešení od optima (osa X) na zvoleném algoritmu (osa Y) po stejném počtu provedených funkčních ohodnocení (50-krát dimenze funkce). Bráno souhrnně pro dimenze 2, 3, 5, 10, 20, 40.

Závěrem uvedu ještě úplné srovnání CEA používajícího lokální prohledávací metodu Quasi-Newton a algoritmu, používajícího pouze tuto metodu bez evolučního přístupu:



Obrázek 5.14: Porovnání CEA s lokálním prohledáváním QN metodou a QN metody pro 19 funkcí. Na ose X je odchylka funkční hodnoty řešení od optima po počtu funkčních ohodnocení 50-krát dimenze.



Obrázek 5.15: Porovnání CEA s lokálním prohledáváním QN metodou a QN metody pro zbývajících 5 funkcí. Na ose X je odchylka funkční hodnoty řešení od optima po počtu funkčních ohodnocení 50-krát dimenze.

Kapitola 6

Závěr

V práci jsem prezentoval obecný úvod do evolučních algoritmů. Také jsem prostudoval algoritmus kontinuální evoluce (CEA), jenž patří mezi třídu hybridních evolučních metod, a specifikoval jsem jeho klíčové principy.

Dále jsem implementoval CEA do jazyka Java. Tato implementace je dostatečně obecná, aby její podstata mohla být aplikována i na jiné problémy než je pouze hledání globálních extrémů funkcí.

Provedl jsem porovnání mnou implementovaného algoritmu CEA s evolučním algoritmem, jehož výsledky jsou dostupné z oficiálních webových stránek pro testovací benchmark BBOB (Black-Box Optimization Benchmarking), a došel jsem ke srovnatelným, někdy i lepším, výsledkům.

Funkcionalitu implementace jsem dále demonstroval také za rozdílných konfigurací konstant a v porovnání s různými metodami prostředí JCOOL (Java COntinuos Optimization Library) na funkcích BBOB. Demonstroval jsem, jak CEA svou kombinací evolučního přístupu s lokální prohledávací metodou často poskytuje lepší řešení pro dané problémy než jiné algoritmy, které evoluční přístup s lokálním prohledáváním nekombinují, jako je tomu u algoritmů Quasi-Newton nebo Conjugate Gradient. Implementaci jsem testoval junit testy.

Cíle práce byly splněny.

Literatura

[1] Black-box algorithms. http://coco.gforge.inria.fr/doku.php?id=bbob-2013-algorithms, 2013. [Online].

[2] Black-box experiment. http://coco.lri.fr/downloads/download15.02/bbobdocexperiment.pdf, 2013. [Online].

[3] Black-box testing. http://coco.gforge.inria.fr/doku.php?id=start, 2013. [Online].

[4] V. Bičík. Continuous optimization algorithms. Master's thesis, České vysoké učení technické v Praze, 2010.

[5] Z. Buk. *Continual Evolution Algorithm.* PhD thesis, České vysoké učení technické v Praze, 2012.

[6] J. Drchal. Java COntinuous Optimization Library. https://github.com/dhonza/JCOOL, 2012. [Online].

[7] A. E. Eiben and J. E. Smith. *An Introduction to Evolutionary Computing.* Springer, 2003.

[8] D. Floreano and C. Mattiussi. *Bio-Inspired Artificial Intelligence - Theories, Methods, and Technologies.* MIT Press, Cambridge, MA, USA, 2008.

[9] K. A. D. Jong. *Evolutionary Computation - A Unified Approach.* MIT Press, Cambridge, MA, USA, 2006.

[10] R. R. "S. Finck, N. Hansen and A. Auger". "real-parameter black-box optimization benchmarking 2010: Presentation of the noiseless functions.". http://coco.lri.fr/downloads/download13.09/bbobdocfunctions.pdf, 2013. "Working Paper 2009/20, Compiled April 13, 2013".

[11] J. R. Shewchuk. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Carnegie Mellon University, 1994.

[12] ČVUT. neuron.felk. http://neuron.felk.cvut.cz/, 2012. [Online].

[13] T. Werner. Optimalizace. https://cw.fel.cvut.cz/wiki/_media/courses/a4b33opt/opt.pdf, 2014. [Online].

[14] J. E. S. William E. Hart, N Krasnogor. *Recent Advances in Memetic Algorithms*. Springer-Verlag Berlin Heidelberg, 2005.