Одеський національний університет імені І. І. Мечникова

(повне найменування вищого навчального закладу)

Інститут математики, економіки і механіки

(повне найменування інституту/факультету)

Кафедра оптимальн=ого керування та економічної кібернетики

(повна назва кафедри)

Д и п л о м н а р о б о т а

бакалавра

(освітньо-кваліфікаційний рівень)

на тему: «Стратегії оптимального вибору параметрів для методу диференціальної еволюції»

«Optimal strategies for parameters choosing in differential evolution»

Виконав: студент денної форми навчання
напряму підготовки 6.040301 Прикладна математика
Берков Костянтин Євгенійович

(прізвище, ім'я, по-батькові)

Керівник Професор, доктор ф.-м. наук, Плотніков А.В.

(науковий ступінь, вчене звання, прізвище та ініціали, підпис)

Рецензент Доцент, кандидат ф.-м. наук, Єфімова Г.А.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№_____ від _____ р.

Захищено на засіданні ЕК №_____
протокол №____ від _____ р.
Оцінка_____/___ ___/_____
(за національною шкалою, шкалою ECTS, бали)

Завідувач кафедри

_____ _____
(підпис) (прізвище, ініціали)

Голова ЕК

_____ _____
(підпис) (прізвище, ініціали)

Одеса – 2017

ЗМІСТ

ВСТУП

У прикладних задачах часто виникає необхідність вирішення завдання глобальної оптимізації в безперервному просторі. У загальному випадку, завдання полягає в оптимізації певних властивостей системи вибираючи відповідні параметри системи, які зазвичай висловлюють через вектор. Стандартний підхід до вирішення цього завдання - побудова деякої функції, яка дозволяє змоделювати цю проблему і накласти певні обмеження. Ця функція називається цільовою.

У разі нелінійної або не диференційованої цільової функції, потрібно використовувати прямі методи оптимізації. Кращі з відомих прямих алгоритмів: метод Нелдера-Міда, метод Хука-Дживса, генетичні алгоритми, еволюційні стратегії. Головне у всіх цих прямих методах це стратегія генерації векторів параметрів. Щоразу після генерації нових векторів приймається рішення о їх використанні. Найчастіше це вирішується значенням цільової функції. Але хоча ця «жадібна» стратегія сходиться досить швидко, існує ризик потрапити в локальний мінімум. Цього можна уникнути проводячи паралельні розрахунки різних наборів векторів, тоді якщо одна з груп потрапить в локальний мінімум, можна продовжити обробку тих що залишилися. Таким чином вимоги до техніки мінімізації такі:

1) Працювати з негладкими, нелінійними, мультимодальними функціями;
2) Можливість розпаралелювання розрахунків;
3) Мала кількість параметрів, які легко вибирати і змінювати;
4) Послідовна збіжність до глобального мінімуму в незалежних випробуваннях.

ВИСНОВКИ

Алгоритм DE представляє собою потужний і простий у використанні інструмент для розв’язку задач глобальної оптимізації. На більшості тестових функцій DE справляється краще, у плані кількості розрахунків цільової функції, ніж інші пригідні для розв’язку цієї задачі алгоритми.

Завдяки своїй простоті реалізації алгоритм представляє багато можливостей для модифікацій. Також алгоритм можна прискорити якщо використати можливості розпаралелювання розрахунків. У додатку А записані тестові функції, у додатку Б подано результати застосування базового алгоритму і двох вище наведених модифікацій до тестових функцій.

Використання мета-оптимізації може значно покращити результати роботи алгоритму, так як будуть використовуватися оптимальні значення для керуючих параметрів.

Для багатовимірних задач розпаралелювання розрахунків принесе результат швидше, але буде використано більше ресурсів.

СПИСОК ЛІТЕРАТУРИ

1) Storn R. Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces / Storn R., Price K. Journal of Global Optimization, Kluwer Academic Publishers, 1997, Vol. 11, 341 - 359
2) Hvass Pedersen M. E. / Good Parameters for Differential Evolution / Hvass Laboratories Technical Report no. HL1002 2010 1, - 3
3) Fleetwood K. An Introduction to Differential Evolution / http://www.maths.uq.edu.au/MASCOS/Multi-Agent04/Fleetwood.pdf, 1- 20
4) Mezura-Montes E. Parameter Control in Differential Evolution for Constrained Optimization / Mezura-Montes E., Palomeque-Ortiz A. G. / 2009 IEEE Congress on Evolutionary Computation, 1 - 7
5) Brest J. CONTROL PARAMETERS IN SELF-ADAPTIVE DIFFERENTIAL EVOLUTION / Brest J., Zumer V., Maucec M. S., Faculty of Electrical Engineering and Computer Science University of Maribor, Slovenia, 35 – 39
6) Zielinski K. Examination of Stopping Criteria for Differential Evolution based on a Power Allocation Problem / Zielinski K., Weitkemper P., Laur R., Kammeyer K.-D., Institute for Electromagnetic Theory and Microelectronics (ITEM), University of Bremen, Germany, Department of Communications Engineering, University of Bremen, Germany, 1 - 5

ДОДАТОК А

Тестові функції

N	Формула	F_{min}	X_{min}	S
F1	$\sum_{i=1}^{n} x_i^2$	0	$(0)^n$	$[-100;100]^n$
F2	$\sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	0	$(0)^n$	$[-100;100]^n$
F3	$\sum_{i=1}^{n} \left(\sum_{j=1}^{i} \lvert x_j \rvert \right)$	0	$(0)^n$	$[-10;10]^n$
F4	$\max_{1 \le i \le n} \lvert x_i \rvert$	0	$(0)^n$	$[-100;100]^n$
F5	$\sum_{i=1}^{n-1} 100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2$	0	$(1)^n$	$[-30;30]^n$
F6	$\sum_{i=1}^{n} i x_i^4$	0	$(0)^n$	$[-1.28;1.28]^n$
F7	$\sum_{i=1}^{n} x_i \sin\sqrt{\lvert x_i \rvert}$	-418.98*n	$(-420.96)^n$	$[-500;500]^n$
F8	$\sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	0	$(0)^n$	$[-5.12;5.12]^n$
F9	$e + 20 - 20e^{-0.2\sqrt{F1}} - e^{\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n}}$	0	$(0)^n$	$[-32;32]^n$
F10	$\sum_{i=1}^{n} x_i^2 + \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	0	$(0)^n$	$[-600;600]^n$
F11	$\frac{\pi}{n}\left[10\sin^2 \pi x_1 + \sum_{1}^{n-1} (x_i - 1)^2 (1 + 10\sin^2 \pi x_{i+1}) + (x_n - 1)^2\right] + \sum_{1}^{n} u(x_i, 10, 100, 4)$	0	$(1)^n$	$[-50;50]^n$
F12	$\sum_{i=1}^{n} \sum_{k=0}^{k_{max}} \left[a^k \cos\left(2\pi b^k \left(x_i + \frac{1}{2}\right)\right)\right] - n\sum_{i=1}^{n} a^k \cos(\pi b^k)$	0	$(0)^n$	$[-0.5;0.5]^n$

$$a = \frac{1}{2}, b = 3, k_{max} = 25$$

$$u(z, a, k, m) = \begin{cases} k(z-a)^m, a < z \\ 0, -z \le a \le z \\ k(-z-a)^m, z < -a \end{cases}$$

ДОДАТОК Б

Розрахунки для DE з параметрами F = 0.9, CR = 0.5, для D = 10, NP = 100

Функція	Best value	Iterations	Time
F1	0.000015	518	2.197526
F2	0.000081	734	2.240165
F3	0.000023	697	2.231585
F4	1.414717	403	2.1113
F5	231.995377	261	2.070192
F6	0	227	2.059419
F7	-2975.677246	88	2.06188
F8	29.963766	140	2.09729
F9	20.000002	41	2.032342
F10	0.606623	313	2.216047
F11	0.000013	630	2.450329
F12	-0.000702	823	25.888463

Розрахунки для PDE з параметрами F = 0.9, CR = 0.5, для D = 10, NP = 100

Функція	Best value	Iterations	Time
F1	0.000001	5	1.068041
F2	0.000004	3	1.40708
F3	0.000004	4	1.24311
F4	0.000009	5	2.025747
F5	0.000159	9	4.06734
F6	0	3	0.651421
F7	-4189.828125	3	1.022684
F8	0	5	0.954157
F9	0.000002	6	1.276624
F10	0	5	1.456601
F11	0.000001	3	1.000619
F12	-0.000702	3	8.648963

Код програми

```
package me.berkow.diffeval

import me.berkow.diffeval.message.SubTask
import me.berkow.diffeval.problem.bestValue
import me.berkow.diffeval.problem.createRandomPopulation
import java.nio.file.Files
import java.nio.file.Paths
import java.nio.file.StandardOpenOption
import java.text.NumberFormat
import java.util.*

fun main(args: Array<String>) {
   val map = args.toMap()

   val amplification = map.getAsFloat("-amplification", 0.6F)
   val crossover = map.getAsFloat("-crossover", 0.9F)

   val populationSize = map.getAsInt("-populationSize", 100)
   val randomSeed = map.getAsLong("-randomSeed", 1612)
   val precision = map.getAsInt("-precision", 6)

   val benchName = map.getOrDefault("-name", "noname" + System.currentTimeMillis())

   val size = map.getAsInt("-size", 10)

   val maxIterations = map.getAsInt("-maxIterations", 10000)

   val format = NumberFormat.getInstance()
   format.maximumFractionDigits = precision

   val problem100LowerConstraints = FloatArray(size) { -100F }
   val problem100UpperConstraints = FloatArray(size) { 100F }

   val problem10LowerConstraints = FloatArray(size) { -10F }
   val problem10UpperConstraints = FloatArray(size) { 10F }

   val problem30LowerConstraints = FloatArray(size) { -30F }
   val problem30UpperConstraints = FloatArray(size) { 30F }

   val problem128LowerConstraints = FloatArray(size) { -1.28F }
   val problem128UpperConstraints = FloatArray(size) { 1.28F }

   val problem500LowerConstraints = FloatArray(size) { -500F }
   val problem500UpperConstraints = FloatArray(size) { 500F }
```

```
  val problem512LowerConstraints = FloatArray(size) { -5.12F }
  val problem512UpperConstraints = FloatArray(size) { 5.12F }

  val problem32LowerConstraints = FloatArray(size) { -32F }
  val problem32UpperConstraints = FloatArray(size) { 32F }

  val problem600LowerConstraints = FloatArray(size) { -600F }
  val problem600UpperConstraints = FloatArray(size) { 600F }

  val problem50LowerConstraints = FloatArray(size) { -50F }
  val problem50UpperConstraints = FloatArray(size) { 50F }

  val problem05LowerConstraints = FloatArray(size) { -0.5F }
  val problem05UpperConstraints = FloatArray(size) { 0.5F }

  val file = Paths.get(benchName + ".csv")
  if (Files.exists(file)) {
     Files.delete(file)
  }
  Files.createFile(file)

  listOf<Problem>(
        Problems.createProblemWithConstraints(1,                    problem100LowerConstraints,
problem100UpperConstraints),
        Problems.createProblemWithConstraints(2,                    problem100LowerConstraints,
problem100UpperConstraints),
        Problems.createProblemWithConstraints(3,                     problem10LowerConstraints,
problem10UpperConstraints),
        Problems.createProblemWithConstraints(4,                    problem100LowerConstraints,
problem100UpperConstraints),
        Problems.createProblemWithConstraints(5,                     problem30LowerConstraints,
problem30UpperConstraints),
        Problems.createProblemWithConstraints(6,                    problem128LowerConstraints,
problem128UpperConstraints),
        Problems.createProblemWithConstraints(7,                    problem500LowerConstraints,
problem500UpperConstraints),
        Problems.createProblemWithConstraints(8,                    problem512LowerConstraints,
problem512UpperConstraints),
        Problems.createProblemWithConstraints(9,                     problem32LowerConstraints,
problem32UpperConstraints),
        Problems.createProblemWithConstraints(10,                   problem600LowerConstraints,
problem600UpperConstraints),
        Problems.createProblemWithConstraints(11,                    problem50LowerConstraints,
problem50UpperConstraints),
        Problems.createProblemWithConstraints(12,                    problem05LowerConstraints,
problem05UpperConstraints)
  )
```

```
        .map { createTask(maxIterations, populationSize, randomSeed, amplification, crossover, it,
precision) }
        .map { task -> doAndMeasure({ Algorithms.standardDE(task, Random(randomSeed)) }) }
        .forEach { (result, time) ->
            val bestValue = result.bestValue()
            val iterations = result.iterationsCount
            val seconds = time / 1000000000.0

            val formatedValue = format.format(bestValue).replace(",", "")

            Files.write(file, "${result.problem.id} , $formatedValue , $iterations ,
${format.format(seconds)}\n".toByteArray(),
                    StandardOpenOption.APPEND)
        }
}

fun createTask(maxIterations: Int, populationSize: Int, randomSeed: Long, amplification: Float,
crossover: Float,
           problem: Problem, precision: Int): SubTask {
   val population = problem.createRandomPopulation(populationSize, Random(randomSeed))

   return SubTask(maxIterations, population, amplification, crossover, problem, precision)
}

package me.berkow.diffeval;

import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Props;
import akka.dispatch.OnComplete;
import akka.event.Logging;
import akka.event.LoggingAdapter;
import akka.pattern.Patterns;
import com.typesafe.config.Config;
import com.typesafe.config.ConfigFactory;
import me.berkow.diffeval.actor.TaskActor;
import me.berkow.diffeval.message.MainResult;
import me.berkow.diffeval.message.MainTask;
import me.berkow.diffeval.problem.Population;
import me.berkow.diffeval.problem.ProblemsKt;
import me.berkow.diffeval.util.Util;

import java.io.Console;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.text.NumberFormat;
import java.util.*;
```

```
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class DEFrontendMain {

   private static volatile boolean sCanProcessInput = true;

   public static void main(String[] args) {
      final Map<String, String> argsMap = new HashMap<>();
      for (int i = 0; i < args.length; i += 2) {
         argsMap.put(args[i], args[i + 1]);
      }

      final String port = argsMap.getOrDefault("-port", "0");

      Config config = ConfigFactory.parseString("akka.remote.netty.tcp.port=" + port);

      try {
         InetAddress localHost = InetAddress.getLocalHost();

         String hostAddress = localHost.getHostAddress();

         Map<String, Object> map = new HashMap<>();

         map.put("akka.remote.netty.tcp.hostname", hostAddress);
         map.put("akka.cluster.seed-nodes", Arrays.asList(
               "akka.tcp://DifferentialEvolution@" + hostAddress + ":2552",
               "akka.tcp://DifferentialEvolution@" + hostAddress + ":2553"
         ));

         config = config.withFallback(ConfigFactory.parseMap(map));

      } catch (UnknownHostException e) {
         e.printStackTrace();
      }

      config = config
            .withFallback(ConfigFactory.parseString("akka.cluster.roles = [frontend]"))
            .withFallback(ConfigFactory.load());

      final ActorSystem system = ActorSystem.create("DifferentialEvolution", config);

      final Props taskActorProps = Props.create(TaskActor.class, port);

      final ActorRef taskActorRef = system.actorOf(taskActorProps, "frontend");
```

```
        startReadingInput(system, taskActorRef);
    }

    private static void startReadingInput(final ActorSystem system, ActorRef taskActorRef) {
        final LoggingAdapter logger = Logging.getLogger(system, "Frontend");
        final Console console = System.console();

        boolean working = true;
        logger.info("Input your task's parameters:");
        while (working) {
            final String input = console.readLine();
            if ("stop".equals(input)) {
                working = false;
            } else if (sCanProcessInput) {
                try {
                    processInput(system, taskActorRef, input, logger);
                    sCanProcessInput = false;
                } catch (Exception e) {
                    logger.error("Failed to process your input: {} due: {}", input, e);
                }
            } else {
                logger.error("Please wait for previous task!");
            }
        }

        System.exit(0);
    }

    private static void processInput(final ActorSystem system, ActorRef taskActorRef, String input,
final LoggingAdapter logger) {
        final List<String> splits = new ArrayList<>();
        final Pattern regex = Pattern.compile("[^\\s\"']+|\"([^\"]*)\"|'([^']*)'");
        final Matcher regexMatcher = regex.matcher(input);
        while (regexMatcher.find()) {
            if (regexMatcher.group(1) != null) {
                // Add double-quoted string without the quotes
                splits.add(regexMatcher.group(1));
            } else if (regexMatcher.group(2) != null) {
                // Add single-quoted string without the quotes
                splits.add(regexMatcher.group(2));
            } else {
                // Add unquoted word
                splits.add(regexMatcher.group());
            }
        }

        final Map<String, String> argsMap = new HashMap<>();
```

```
for (int i = 0; i < splits.size(); i += 2) {
   argsMap.put(splits.get(i), splits.get(i + 1));
}

final int maxIterations = Util.getIntOrDefault(argsMap, "-maxIterations", 100);
final int problemId = Util.getIntOrDefault(argsMap, "-problemId", 5);
final int populationSize = Util.getIntOrDefault(argsMap, "-populationSize", 100);
final int splitCount = Util.getIntOrDefault(argsMap, "-splitCount", 10);
final long randomSeed = Util.getLongOrDefault(argsMap, "-randomSeed", -1);

float amplification = Util.getFloatOrDefault(argsMap, "-amplification", 0.9F);
amplification = Math.max(0, Math.min(amplification, 2));

float crossoverProbability = Util.getFloatOrDefault(argsMap, "-crossover", 0.5F);
crossoverProbability = Math.max(0, Math.min(crossoverProbability, 1));

final float[] lowerBounds = Util.getFloatArrayOrThrow(argsMap, "-lowerBounds", "Supply
lower bounds!");
final float[] upperBounds = Util.getFloatArrayOrThrow(argsMap, "-upperBounds", "Supply
upper bounds!");

final int precision = Util.getIntOrDefault(argsMap, "-precision", 6);

final Problem problem = Problems.createProblemWithConstraints(problemId, lowerBounds,
upperBounds);

final Random random = randomSeed == -1 ? new Random() : new Random(randomSeed);

final Population population = ProblemsKt.createRandomPopulation(problem, populationSize,
random);

final MainTask task = new MainTask(maxIterations, population,
      amplification, crossoverProbability, splitCount, problem, precision);

final long nanoTime = System.nanoTime();

Patterns.ask(taskActorRef, task, 10000).transform(
      value -> (MainResult) value,
      error -> error, system.dispatcher()).onComplete(new OnComplete<MainResult>() {
   @Override
   public void onComplete(Throwable failure, MainResult success) throws Throwable {
      sCanProcessInput = true;
      if (failure == null) {
         onCompleted(system, problemId, success, logger, task, System.nanoTime() - nanoTime);
      } else {
         onFailure(system, failure);
      }
```

```
        }
      }, system.dispatcher());
    }

    private static void onCompleted(ActorSystem system, int taskId, MainResult result,
LoggingAdapter logger,
                                    MainTask task, long timeConsumed) {
      final NumberFormat format = NumberFormat.getInstance();
      format.setMaximumFractionDigits(task.getPrecision());

      final String formattedValue = format.format(ProblemsKt.bestValue(result)).replace(",", "");

      logger.info("{}  ,  {}  ,  {}\n",formattedValue,  result.getIterationsCount(),
format.format(timeConsumed / 1000000000.0));
    }

    private static void onFailure(ActorSystem system, Throwable failure) {
      system.log().error(failure, "Failed due {}", failure);
    }
}

package me.berkow.diffeval;

import akka.actor.ActorSystem;
import akka.actor.Props;
import com.typesafe.config.Config;
import com.typesafe.config.ConfigFactory;
import me.berkow.diffeval.actor.WorkerActor;

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;


public class DEBackendMain {

    public static void main(String[] args) {
      // Override the configuration of the port and ip when specified as program argument
      final String remoteHostname = args.length > 0 ? args[0] : null;
      final String port = args.length > 1 ? args[1] : "0";

      Map<String, Object> map = new HashMap<>();
      if (remoteHostname != null) {
        map.put("akka.cluster.seed-nodes", Arrays.asList(
            "akka.tcp://DifferentialEvolution@" + remoteHostname + ":2552",
```

```
                "akka.tcp://DifferentialEvolution@" + remoteHostname + ":2553"
            ));
        }
        map.put("akka.remote.netty.tcp.port", port);

        try {
            InetAddress localHost = InetAddress.getLocalHost();

            String hostAddress = localHost.getHostAddress();

            map.put("akka.remote.netty.tcp.hostname", hostAddress);

        } catch (UnknownHostException e) {
            e.printStackTrace();
        }

        Config config = ConfigFactory.parseMap(map)
                .withFallback(ConfigFactory.parseString("akka.cluster.roles = [backend]"))
                .withFallback(ConfigFactory.load());

        ActorSystem system = ActorSystem.create("DifferentialEvolution", config);

        system.actorOf(Props.create(WorkerActor.class, port), "backend");
    }
}
```

```
package me.berkow.diffeval;

import me.berkow.diffeval.problem.Member;

/**
 * Created by konstantinberkow on 5/11/17.
 */
@SuppressWarnings("SameParameterValue")
public final class Problems {
    private static final int K_MAX = 26;
    private static final float[] AK = new float[K_MAX];
    private static final float[] BK = new float[K_MAX];

    static {
        AK[0] = 1;
        BK[0] = (float) Math.PI;
        for (int i = 1; i < K_MAX; i++) {
            AK[i] = AK[i - 1] * .5F;
            BK[i] = BK[i - 1] * 3;
        }
    }
```

```
//Sphere
public static float calculateProblem1(float[] vector) {
   float sum = 0;

   for (float x : vector) {
      sum += x * x;
   }

   return sum;
}

public static float calculateProblem2(float[] vector) {
   float sum = 0;
   float product = 1;

   for (float x : vector) {
      float absX = Math.abs(x);
      sum += absX;
      product *= absX;
   }

   return sum + product;
}

//looks like problem in definition, {0...0} won't be it's minimum maybe abs needed
public static float calculateProblem3(float[] vector) {
   float outerSum = 0;

   for (int i = 0; i < vector.length; i++) {
      float innerSum = 0;
      for (int j = 0; j <= i; j++) {
         innerSum += Math.abs(vector[j]);
      }
      outerSum += innerSum;
   }

   return outerSum;
}

public static float calculateProblem4(float[] vector) {
   float max = Math.abs(vector[0]);
   for (int i = 1; i < vector.length; i++) {
      final float x = Math.abs(vector[i]);
      max = x > max ? x : max;
   }
   return max;
```

```
}

// Rosenbrock
public static float calculateProblem5(float[] vector) {
   float sum = 0;

   for (int i = 0; i < vector.length - 1; i++) {
      final float x = vector[i];
      sum += 100 * (vector[i + 1] - x * x) * (vector[i + 1] - x * x) + (x - 1) * (x - 1);
   }

   return sum;
}

public static float calculateProblem6(float[] vector) {
   float sum = 0;

   for (int i = 0; i < vector.length; i++) {
      float x = vector[i];
      sum += (i + 1) * x * x * x * x;
   }

   return sum;
}

//looks like min is {-418.9829, ..., -418.9828} not {418.9828, ..., 418.9828}
public static float calculateProblem7(float[] vector) {
   float sum = 0;

   for (int i = 0; i < vector.length; i++) {
      float x = vector[i];
      sum += x * Math.sin(Math.sqrt(Math.abs(x)));
   }

   return sum;
}

//Rastrigin
public static float calculateProblem8(float[] vector) {
   float sum = 0;

   for (int i = 0; i < vector.length; i++) {
      float x = vector[i];
      sum += (x * x - 10 * Math.cos(2 * Math.PI * x) + 10);
   }

   return sum;
```

```
}

//Ackley's
public static float calculateProblem9(float[] vector) {
   float tmp1 = calculateProblem1(vector);
   float part1 = (float) (20 * Math.exp(-0.2 * Math.sqrt(tmp1)));

   float tmp2 = 0;
   for (float x : vector) {
      tmp2 += Math.cos(2 * Math.PI * x);
   }
   float part2 = (float) Math.exp(tmp2 / vector.length);

   return (float) (-part1 - part2 + 20 + Math.E);
}

//Griewangk
public static float calculateProblem10(float[] vector) {
   float f1Result = calculateProblem1(vector);

   float product = 1;
   for (int i = 0; i < vector.length; i++) {
      product *= Math.cos(vector[i] / Math.sqrt(i + 1));
   }

   return f1Result / 4000 - product + 1;
}

//Goldstein3 with penalty
public static float calculateProblem11(float[] vector) {
   float penalty = 0;

   for (int i = 0; i < vector.length; i++) {
      penalty += penalty(vector[i], 10, 100, 4);
   }

   return goldstein3(vector) + penalty;
}

//Goldstein3
public static float goldstein3(float[] array) {
   final int size = array.length;

   float result = 0;

   {
      final double tmp = Math.sin(Math.PI * array[0]);
```

```
      result += 10 * tmp * tmp;
    }

    for (int i = 0; i < size - 1; i++) {
      final float tmp1 = array[i] - 1;
      final double tmp2 = Math.sin(Math.PI * array[i + 1]);

      result += tmp1 * tmp1 * (1 + 10 * tmp2 * tmp2);
    }

    {
      final double tmp = array[size - 1] - 1;
      result += tmp * tmp;
    }

    return (float) (Math.PI * result / size);
  }

  public static float penalty(float z, float a, float k, float m) {
    if (z > a) {
      return (float) (k * Math.pow(z - a, m));
    } else if (z < -a) {
      return (float) (k * Math.pow(-z - a, m));
    } else {
      return 0;
    }
  }

  public static float calculateProblem12(float[] vector) {
    float result = 0;
    final int n = vector.length;

    for (int i = 0; i < n; i++) {
      for (int k = 0; k < 26; k++) {
        result += AK[k] * Math.cos(2 * BK[k] * (vector[i] + .5F));
      }
    }

    float minus = 0;
    for (int k = 0; k < 26; k++) {
      minus += AK[k] * Math.cos(BK[k]);
    }

    return result - n * minus;
  }
```

```
  public static Problem createProblemWithConstraints(int problemId, float[] lowerBounds, float[]
upperBounds) {
    switch (problemId) {
      case 1:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem1(vector.toArray());
          }
        };
      case 2:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem2(vector.toArray());
          }
        };
      case 3:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem3(vector.toArray());
          }
        };

      case 4:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem4(vector.toArray());
          }
        };
      case 5:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem5(vector.toArray());
          }
        };
      case 6:
        return new Problem(problemId, lowerBounds, upperBounds) {
          @Override
          public float calculate(Member vector) {
            return calculateProblem6(vector.toArray());
          }
        };
      case 7:
```

```
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem7(vector.toArray());
                }
            };
        case 8:
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem8(vector.toArray());
                }
            };
        case 9:
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem9(vector.toArray());
                }
            };
        case 10:
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem10(vector.toArray());
                }
            };
        case 11:
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem11(vector.toArray());
                }
            };
        case 12:
            return new Problem(problemId, lowerBounds, upperBounds) {
                @Override
                public float calculate(Member vector) {
                    return calculateProblem12(vector.toArray());
                }
            };
        default:
            throw new IllegalArgumentException("Unknown problem id: " + problemId);
    }
  }
}
```

```
package me.berkow.diffeval.actor;

import akka.actor.AbstractActor;
import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Terminated;
import akka.dispatch.Futures;
import akka.event.Logging;
import akka.event.LoggingAdapter;
import akka.japi.Pair;
import akka.pattern.Patterns;
import me.berkow.diffeval.UtilKt;
import me.berkow.diffeval.message.*;
import me.berkow.diffeval.problem.Population;
import me.berkow.diffeval.Problem;
import me.berkow.diffeval.problem.ProblemsKt;
import scala.concurrent.ExecutionContextExecutor;
import scala.concurrent.Future;

import java.text.NumberFormat;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Random;

/**
 * Created by konstantinberkow on 5/8/17.
 */
public class TaskActor extends AbstractActor {
   static final String BACKEND_REGISTRATION = "register";

   private final String port;
   private final List<ActorRef> backends;
   private final Random random;
   private final LoggingAdapter logger = Logging.getLogger(this);
   private final NumberFormat format = NumberFormat.getInstance();

   private int currentIterationCount = 0;
   private MainTask currentTask = null;
   private float previousValue;

   public TaskActor(String port) {
      this.port = port;
      backends = new ArrayList<>();
      random = new Random();
   }
```

```
  private static SubTask createTask(MainTask task) {
    return new SubTask(task.getMaxIterationsCount(), task.getPopulation(),
        task.getAmplification(), task.getCrossoverProbability(), task.getProblem(),
task.getPrecision());
  }

  private static SubTask createTask(MainTask task, Random random) {
    final float f0 = task.getAmplification();
    final float c0 = task.getCrossoverProbability();

    float newF = f0;
    while (newF == f0) {
      newF = UtilKt.nextFloat(random, 0, 2);
    }

    float newC = c0;
    while (newC == c0) {
      newC = UtilKt.nextFloat(random, 0, 1);
    }

    return new SubTask(task.getMaxIterationsCount(), task.getPopulation(), newF, newC,
task.getProblem(), task.getPrecision());
  }

  private static SubResult selectResult(Iterable<SubResult> results) {
    final Iterator<SubResult> iterator = results.iterator();
    float previousValue = Float.MAX_VALUE;
    SubResult bestResult = null;
    while (iterator.hasNext()) {
      final SubResult result = iterator.next();
      final float value = result.getValue();

      if (bestResult == null) {
        previousValue = value;
        bestResult = result;
      } else if (value < previousValue) {
        previousValue = value;
        bestResult = result;
      }
    }

    return bestResult;
  }

  @Override
  public void preStart() throws Exception {
    logger.debug("{} pre start!", this);
```

```
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(MainTask.class, task -> backends.isEmpty(), task -> {
                    TaskFailedMsg message = new TaskFailedMsg("Service unavailable, try again later",
task);
                    final ActorRef sender = getSender();
                    sender.tell(message, sender);
                })
                .match(MainTask.class, task -> {
                    currentIterationCount = 0;
                    previousValue = ProblemsKt.averageValue(task.getPopulation(), task.getProblem());

                    currentTask = task;
                    format.setMaximumFractionDigits(task.getPrecision());

                    calculate(task, getSender());
                })
                .match(Pair.class, pair -> {
                    final SubResult result = (SubResult) pair.first();
                    final ActorRef originalSender = (ActorRef) pair.second();

                    proceedResults(result, originalSender);
                })
                .matchEquals(BACKEND_REGISTRATION, $ -> {
                    final ActorRef sender = getSender();
                    getContext().watch(sender);
                    backends.add(sender);
                })
                .match(Terminated.class, terminated -> backends.remove(terminated.getActor()))
                .build();
    }

    private void proceedResults(SubResult result, ActorRef originalSender) {
        currentIterationCount++;

        final int maxIterationsCount = currentTask.getMaxIterationsCount();
        final float amplification = result.getAmplification();
        final float crossoverProbability = result.getCrossoverProbability();
        final Population population = result.getPopulation();
        final Problem problem = result.getProblem();
        final int precision = currentTask.getPrecision();
        final float newValue = result.getValue();

        logger.info("iteration: {}", currentIterationCount);
```

```
        logger.info("new amplification: {}", result.getAmplification());
        logger.info("new crossover probability: {}", result.getCrossoverProbability());
        logger.info("new population value: {}", result.getValue());
        logger.info("diff: {}", Math.abs(newValue - previousValue));


        if (Math.abs(newValue - previousValue) < 1.0 / Math.pow(10, precision)) {
          onCompleted(result, "converged_population", originalSender);
          return;
        }

        if (currentIterationCount >= maxIterationsCount) {
          onCompleted(result, "max_iterations", originalSender);
          return;
        }

        final MainTask newTask = new MainTask(maxIterationsCount, population, amplification,
crossoverProbability,
            currentTask.getSplitSize(), problem, precision);

        previousValue = newValue;

        calculate(newTask, originalSender);
      }

      private void onCompleted(SubResult result, String type, ActorRef originalSender) {
        final MainResult mainDEResult = new MainResult(result, type, currentIterationCount);
        originalSender.tell(mainDEResult, getSelf());
      }

      @Override
      public void postStop() throws Exception {
        logger.debug("{} postStop", this);
        backends.clear(); //huh?
      }

      private void calculate(MainTask task, final ActorRef originalSender) {
        final ActorSystem system = getContext().getSystem();

        logger.debug("Calculate from: {}, by: {}", task.getPopulation(), this);

        final int splitSize = task.getSplitSize();

        final ExecutionContextExecutor dispatcher = system.dispatcher();

        final List<SubTask> tasks = new ArrayList<>(splitSize);
        tasks.add(createTask(task));
```

```
      for (int i = 1; i < splitSize; i++) {
         tasks.add(createTask(task, random));
      }

      final List<Future<SubResult>> futures = new ArrayList<>(splitSize);
      for (int i = 0; i < tasks.size(); i++) {
         final SubTask splitedTask = tasks.get(i);

         logger.debug("new control values F: {}, CR: {}", splitedTask.getAmplification(),
splitedTask.getCrossoverProbability());

         final Future<SubResult> future = Patterns.ask(backends.get(i % backends.size()), splitedTask,
10000)
               .transform(result -> (SubResult) result, error -> error, dispatcher);

         futures.add(future);
      }

      final Future<Iterable<SubResult>> resultsFuture = Futures.sequence(futures, dispatcher);

      final Future<Pair<SubResult, ActorRef>> resultFuture = resultsFuture
            .transform(results -> new Pair<>(selectResult(results), originalSender), error -> error,
dispatcher);

      final ActorRef self = getSelf();

      Patterns.pipe(resultFuture, dispatcher).to(self, self);
   }

   @Override
   public String toString() {
      return "TaskActor{" +
            "port='" + port + '\'' +
            ", backends=" + backends +
            '}';
   }
}

package me.berkow.diffeval.actor;

import akka.actor.AbstractActor;
import akka.actor.ActorSystem;
import akka.cluster.Cluster;
import akka.cluster.ClusterEvent;
import akka.cluster.Member;
import akka.cluster.MemberStatus;
import akka.dispatch.Futures;
```

```
import akka.pattern.Patterns;
import me.berkow.diffeval.Algorithms;
import me.berkow.diffeval.message.SubResult;
import me.berkow.diffeval.message.SubTask;
import scala.concurrent.ExecutionContextExecutor;
import scala.concurrent.Future;

import java.util.Random;
import java.util.concurrent.Callable;
import java.util.stream.StreamSupport;


public class WorkerActor extends AbstractActor {

   private final String port;
   private final Random random = new Random();
   private Cluster cluster;

   public WorkerActor(String port) {
      this.port = port;
   }

   @Override
   public void preStart() throws Exception {
      final ActorSystem system = getContext().getSystem();

      cluster = Cluster.get(system);
      cluster.subscribe(getSelf(), ClusterEvent.MemberUp.class);
   }

   @Override
   public Receive createReceive() {
      return receiveBuilder()
            .match(SubTask.class, task -> {
               final ExecutionContextExecutor dispatcher = getContext().getSystem().dispatcher();
               final Future<SubResult> result = Futures.future(createCalculationCallable(task),
dispatcher);

               Patterns.pipe(result, dispatcher).to(getSender(), getSelf());
            })
            .match(ClusterEvent.CurrentClusterState.class, state -> {
               StreamSupport.stream(state.getMembers().spliterator(), false)
                     .filter(member -> member.status().equals(MemberStatus.up()))
                     .forEach(this::register);
            })
            .match(ClusterEvent.MemberUp.class, mUp -> register(mUp.member()))
            .build();
```

```
    }

    @Override
    public void postStop() {
        cluster.unsubscribe(getSelf());
    }

    private void register(Member member) {
        if (member.hasRole("frontend")) {
            getContext().actorSelection(member.address() + "/user/frontend")
                    .tell(TaskActor.BACKEND_REGISTRATION, getSelf());
        }
    }

    private Callable<SubResult> createCalculationCallable(final SubTask task) {
        return () -> Algorithms.standardDE(task, random);
    }

    @Override
    public String toString() {
        return "WorkerActor{" +
                "port='" + port + '\'' +
                ", cluster=" + cluster +
                '}';
    }
}
```

```
package me.berkow.diffeval;

import me.berkow.diffeval.message.SubResult;
import me.berkow.diffeval.message.SubTask;
import me.berkow.diffeval.problem.Member;
import me.berkow.diffeval.problem.Population;
import me.berkow.diffeval.problem.ProblemsKt;
import me.berkow.diffeval.util.Util;

import java.text.NumberFormat;
import java.util.*;


public class Algorithms {

    public static SubResult standardDE(SubTask task, Random random) {
        return standardDE(task, random, false);
    }

    public static SubResult standardDE(SubTask task, Random random, boolean debug) {
```

```
    final int maxIterationsCount = task.getMaxIterationsCount();
    final float amplification = task.getAmplification();
    final float crossoverProbability = task.getCrossoverProbability();
    final Problem problem = task.getProblem();
    final double precision = 1.0 / Math.pow(10, task.getPrecision());

    Population population = task.getInitialPopulation();
    float previousValue = ProblemsKt.averageValue(population, problem);
    for (int i = 0; i < maxIterationsCount; i++) {
      population = createNewGeneration(population, task, random);

      final float newValue = ProblemsKt.averageValue(population, problem);
      if (debug) {
        System.out.println("new value: " + newValue);
        System.out.println("prev value: " + previousValue);
      }
      if (Math.abs(newValue - previousValue) < precision) {
        return new SubResult(population, amplification, crossoverProbability, problem,
"converged population", i, newValue);
      }

      previousValue = newValue;
    }

    return new SubResult(population, amplification, crossoverProbability, problem,
"max_iterations", maxIterationsCount, previousValue);
  }

  public static Population createNewGeneration(final Population previousGeneration, SubTask
task, Random random) {
    final int populationSize = task.getPopulationSize();
    final List<Member> newVectors = new ArrayList<>(populationSize);

    final Problem problem = task.getProblem();
    final int size = problem.getSize();

    final float crossoverProbability = task.getCrossoverProbability();
    final float amplification = task.getAmplification();

    final Member[] members = previousGeneration.members().toArray(new
Member[populationSize]);

    final float[] lowerConstraints = problem.getLowerConstraints();
    final float[] upperConstraints = problem.getUpperConstraints();

    for (int i = 0; i < populationSize; i++) {
      final Member oldVector = members[i];
```

```
        final int[] indexes = Util.selectIndexes(0, populationSize, i, 3, random);
        final int mandatoryIndex = random.nextInt(size);
        final float[] newVector = new float[size];

        for (int j = 0; j < size; j++) {
          if (mandatoryIndex == j || random.nextDouble() < crossoverProbability) {
            newVector[j] = members[indexes[0]].get(j) + amplification * (members[indexes[1]].get(j)
- members[indexes[2]].get(j));
          } else {
            newVector[j] = oldVector.get(j);
          }

          final float min = lowerConstraints[j];
          final float max = upperConstraints[j];

          if (newVector[j] < min) {
            newVector[j] = min;
          } else if (max < newVector[j]) {
            newVector[j] = max;
          }
        }

        final Member member = new Member(newVector);

        if (problem.calculate(member) < problem.calculate(oldVector)) {
          newVectors.add(member);
        } else {
          newVectors.add(oldVector);
        }
      }

      return new Population(newVectors.toArray(new Member[0]));
    }

    public static void main(String[] args) {
      final Map<String, String> argsMap = new HashMap<>();
      for (int i = 0; i < args.length; i += 2) {
        argsMap.put(args[i], args[i + 1]);
      }

      final int maxIterations = Util.getIntOrDefault(argsMap, "-maxIterations", 1000);
      final int problemId = Util.getIntOrDefault(argsMap, "-problemId", 5);
      final int populationSize = Util.getIntOrDefault(argsMap, "-populationSize", 100);
      final long randomSeed = Util.getLongOrDefault(argsMap, "-randomSeed", -1);

      float amplification = Util.getFloatOrDefault(argsMap, "-amplification", 0.9F);
      amplification = Math.max(0, Math.min(amplification, 2));
```

```
        float crossoverProbability = Util.getFloatOrDefault(argsMap, "-crossover", 0.5F);
        crossoverProbability = Math.max(0, Math.min(crossoverProbability, 1));

        final float[] lowerBounds = Util.getFloatArrayOrThrow(argsMap, "-lowerBounds", "Supply lower bounds!");
        final float[] upperBounds = Util.getFloatArrayOrThrow(argsMap, "-upperBounds", "Supply upper bounds!");

        final int precision = Util.getIntOrDefault(argsMap, "-precision", 6);

        final boolean debug = Boolean.parseBoolean(argsMap.getOrDefault("debug", "false"));

        final NumberFormat format = NumberFormat.getInstance();
        format.setMaximumFractionDigits(precision);

        final Problem problem = Problems.createProblemWithConstraints(problemId, lowerBounds, upperBounds);

        final Random random = randomSeed == -1 ? new Random() : new Random(randomSeed);

        final Population population = ProblemsKt.createRandomPopulation(problem, populationSize, random);

        final SubTask task = new SubTask(maxIterations, population, amplification, crossoverProbability, problem, precision);

        final long nanoTime = System.nanoTime();

        final SubResult result = standardDE(task, random, debug);
//      System.out.printf("Result population: %s\n", result.getPopulation());
        System.out.printf("Result type: %s\n", result.getType());
        System.out.printf("Result iterations: %d\n", result.getIterationsCount());

        final float bestValue = ProblemsKt.bestValue(result.getPopulation(), problem);
//                                  System.out.printf("Result average member: %s\n", Util.prettyFloatArray(averageMember.toArray(), format));
        System.out.printf("Value: %s\n", Util.prettyNumber(bestValue, format));
        System.out.printf("Time consumed: %s\n", Util.prettyNumber((System.nanoTime() - nanoTime) / 1000000000.0, format));
    }
}
```