GP 2	Generative Programming	ST 17, Exercise 4

Deadline: 19.05.2017, 10:30

[✓ **Gr. 1,** **J. Karder, MSc.**]
☐ **Gr. 2,** **P. Fleck, MSc.**

Name Thomas Herzog **Effort in h** 10

Points ______ **Lecturer** ______

1. Tracing (5 Points)

Implement a simple class with at least one data field and two methods; one of these methods is to be called by the other one. In the test program create an object of this class, and call the methods and access the data fields.

Furthermore, implement an aspect *Tracing* that protocols each method call as well as each access to one of the data fields. Please distinguish between the regular ending of a method and its abortion by an exception. The so generated output could for example look like this:

```
Entering addPositiveValue
Accessing last
Accessed last
Entering setPositiveValue
Accessing positiveValues
Accessed positiveValues
Exiting setPositiveValue
Accessing last
Accessed last
Accessing last
Accessed last
Exiting addPositiveValue
Entering setPositiveValue
Exiting setPositiveValue ERROR: value is not positive (java.lang.IllegalArgumentException)
Exiting addPositiveValue ERROR: value is not positive (java.lang.IllegalArgumentException)
```

As we now want to indent outputs according to their interlacing in order to increase the lucidity of the outputs, implement indentation without changing the original *Tracing* aspect.

2. Caching (5 Points)

Develop a class *BinomialCoefficient* that offers the static method *Calculate* for calculating the binomial coefficient. This calculation is defined as follows:

$$bc(n,0) = 1$$
$$bc(n,n) = 1$$
$$bc(n,m) = bc(n-1,m-1) + bc(n-1,m)$$

Implement an aspect *LogRecursiveCalls* that counts the number of recursive method calls and writes it to the console as soon as the execution of the first call of *Calculate* is finished.

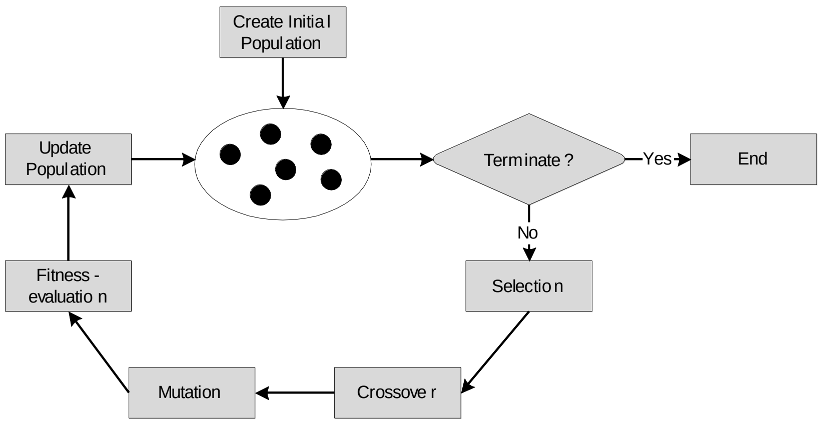
Furthermore, implement another aspect *BinomialCache* that caches calculated (intermediate) results and calls *Calculate* only if the required result is not known yet.

Finally, add an aspect *RuntimeMeasurement* that can be used for measuring and displaying the runtime consumed by a method.

Test all aspects extensively and document, whether the use of *BinomialCache* is reasonable or not.

3. Aspect-Oriented TSP Solver (3 + 3 + 2 + 6 Points)

On elearning you can find a simple program *TSPSolver* that solves the traveling salesman problem (TSP) with genetic algorithms (GAs). GAs are evolutionary algorithms that simulate natural evolution using the adaption of species as optimization technique. The basic workflow of a GA looks like this:



A standard GA with tournament selection (tournament group size $k = 2$), order crossover, inversion mutation and generational replacement is already implemented. Path encoding is used for representing solution candidates, i.e., each roundtrip is encoded as a permutation; e.g., [1 3 2 5 4] represents the roundtrip from city 1 to 3 to … to 4 und back to city 1.

Furthermore, we here already have several aspects that modify the behavior of the program: The aspect *MeasureRuntime* is used for measuring the runtime of an execution of the algorithm, the aspect *RandomSelection* replaces the originally used tournament selection, and the aspects *CyclicCrossover* and *MaximalPreservativeCrossover* replace the respective originally used crossover operator.

Thus, we here see that aspect oriented programming is ideal to enhance (and hopefully improve) the TSPSolver with new concepts. Your task is now to implement the following additional aspects for TSPSolver:

a) Implement an aspect *CountEvaluatedSolutions* that calculates the number of evaluated solutions and eventually writes it to the console as soon as the algorithm has terminated. Based on this implement another aspect *LimitEvaluatedSolutions* that ensures that no next iteration of the algorithm is executed as soon as a predefined maximum number of evaluations is reached.

b) Develop an aspect *Elitism* that adds 1-elitism to the already implemented replacement strategy (generational replacement). I.e., at each generation step the best individual of the parents generation survives and replaces the worst of the new children. Using this aspect we get a monotonous improvement of the quality of the population’s best individual.

c) Add an aspect *ProtocolProgress* to TSPSolver that stores best, worst and average qualities of the population and writes it to the console after the algorithm has finished.

d) Unfortunately TSPSolver does not have any graphical visualization. Since we already have a framework for generating SVGs, your final task is now to use this framework for plotting the progress of the best, average and worst quality as well as the best roundtrip found by the algorithm.

1 Tracing

Dieser Abschnitt beschäftigt sich mit der Dokumentation der Aufgabenstellung *Tracing*.

1.1 Lösungsidee

Für das Testen des *Tracings* wurde die Klasse *application.PositiveValueStore* implementiert, die mehrere verschachtelte Aufrufe sowie das Auslösen einer Ausnahme implementiert. Die Klasse *application.Main* wurde von den Aspekten ausgenommen, damit die implementierten Testmethoden nicht in den *Logs* aufscheinen.

Der Aspekt *TracingAspect logged* alle Aufrufe von Konstruktoren, Methoden und Zugriffe auf *Properties* von Klassen und zwar vor und nach dem Aufruf. Es wurden die *PointCuts methodCall, fieldAccess, newObject* implementiert, für welche die *before, after Advices* definiert wurden. Die *Advices loggen* die Zugriffe, wobei die *Logs* immer mit den Wörtern '*[Before|After]*' beginnt.

Der Aspekt *IndentionLogTrace* implementiert das Einrücken der *Logs*, um die Schachtelung der Methodenaufrufe zu verdeutlichen. Es wird ein *PointCut* auf alle Methoden der Schnittstelle *org.slf4j.Logger* definiert und ein *around Advice* implementiert, der den übergebenen Text formatiert, je nachdem ob die erwartende Zeichenkette (*[Before|After]*) am Anfang des Textes gefunden wurde. Wird eine solche Zeichenkette am Anfang des Textes gefunden, wird bei *Before* eine definierte Anzahl von Leerzeichen am Anfang des Textes eingefügt und beim Finden des Textes *After* einmalig die definierte Anzahl von Leerzeichen am Anfang des Textes entfernt.

1.2 Quelltexte

Folgender Abschnitt enthält die implementierten Klassen, Aspekte und das implementierte Testprogramm.

Listing 1: PositiveValueStore.java

```java
package application;

/**
 * This class represents a positive value store
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public class PositiveValueStore {

    private int[] positiveValues;
    private int size;
    private int last = 0;

    public PositiveValueStore(int size) {
        this.positiveValues = new int[size];
        this.size = size;
    }

    public void addPositiveValue(int value) {
        setPositiveValues(last, value);
    }

    public void setPositiveValues(int idx,
                                  int value) {
        checkIdx(idx);
        checkValue(value);
        getPositiveValues()[idx] = value;
        last++;
    }

    public int[] getPositiveValues() {
        return positiveValues;
    }

    private void checkValue(final int value) {
        if (value < 0) {
            throwExceptionIfIdxInvalid();
        }
    }

    private void checkIdx(final int idx) {
        if (idx >= size) {
            throwExceptionIfValueInvalid();
        }
    }

    private void throwExceptionIfIdxInvalid() {
        throw new ArrayIndexOutOfBoundsException("Index exceeds size");
    }

    private void throwExceptionIfValueInvalid() {
        throw new IllegalArgumentException("Only positive values are supported");
    }
}
```

Listing 2: TracingAspect.aj

```
package aspects;

import application.Main;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * The aspect for tracing the chained method calls.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public aspect TracingAspect {

    private static Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);

    pointcut methodCall():
            call(* application.*.*(..))
                    && !within(application.Main);

    pointcut fieldAccess():
            (get(* application..*.*) || set(* application..*.*))
                    && !within(application.Main);

    pointcut newObject():
            call(application.*.new(..));

    before(): methodCall(){
        log.info("Before method '{}#{}'",
 ↪   thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
                thisJoinPointStaticPart.getSignature().getName());
    }

    after() returning: methodCall(){
        log.info("After method '{}#{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
                thisJoinPointStaticPart.getSignature().getName());
    }

    after() throwing(Throwable t): methodCall(){
        log.info("After method '{}#{}' / {}#'{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
                thisJoinPointStaticPart.getSignature().getName(),
                t.getClass().getSimpleName(),
                t.getMessage());
    }

    before(): fieldAccess() {
        log.info("Before field '{}#{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
                thisJoinPointStaticPart.getSignature().getName());
    }

    after(): fieldAccess() {
        log.info("After field '{}#{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType().getSimpleName(),
                thisJoinPointStaticPart.getSignature().getName());
    }

    before():newObject() {
        log.info("Before constructor '{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType());
    }

    after():newObject() {
        log.info("After constructor '{}'",
 ↪  thisJoinPointStaticPart.getSignature().getDeclaringType());
    }
}
```

Listing 3: IndentionLogTrace.aj

```
package aspects;

/**
 * This class intercepts the log calls within the TRacingAspect for log message indention.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/12/17
 */
public aspect IndentionLogTrace {

    private String currentIndent = "";
    private static final String INDENT = "        ";
    private static final int MAX_INDENT_IDX = INDENT.length() - 1;

    pointcut logCall(String msg):
            if(application.Main.logIndentionEnabled)
                    && call(void org.slf4j.Logger.* (String, ..)) && !within(IndentionLogTrace)
                    && args(msg, ..)
                    && within(TracingAspect);

    void around(String msg): logCall(msg){
```

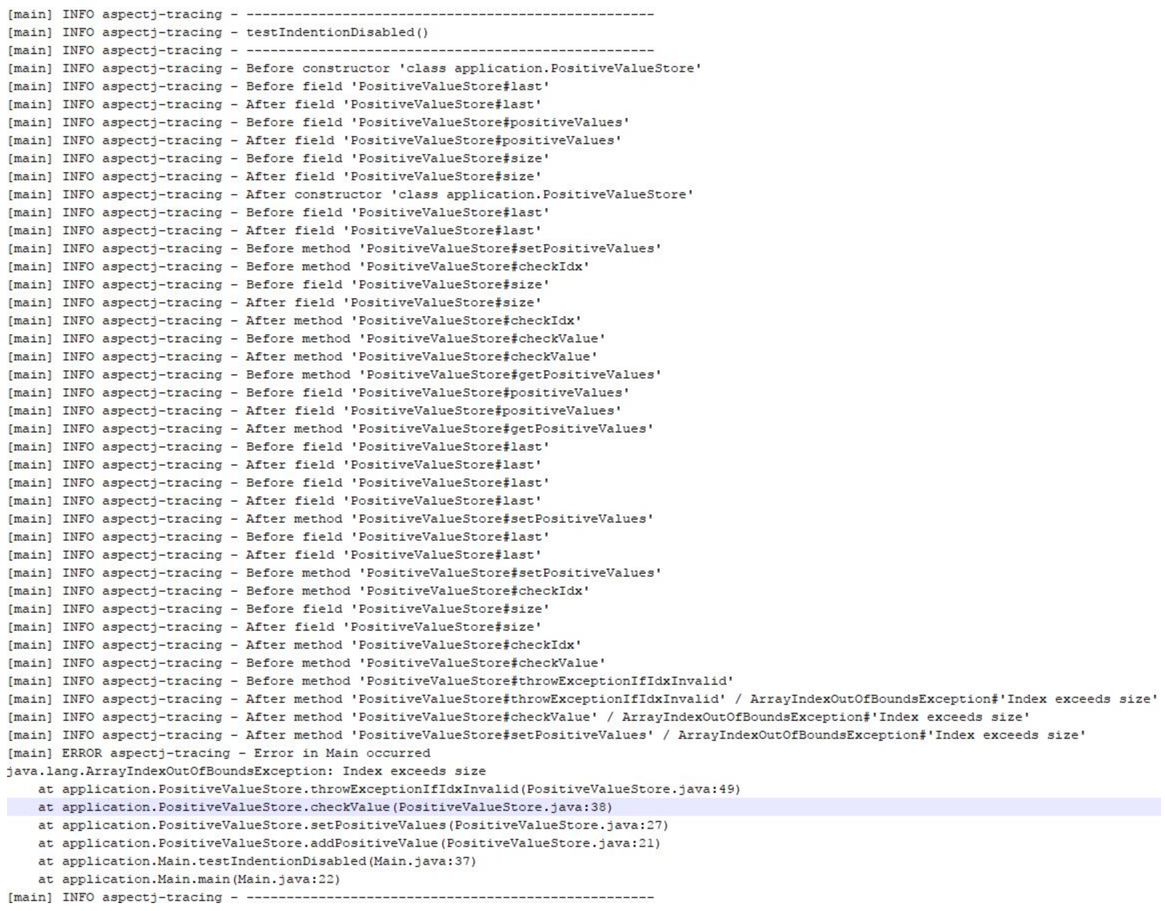
```java
        if ((msg.startsWith("After")) && (currentIndent.length() >= MAX_INDENT_IDX)) {
            currentIndent = currentIndent.substring(MAX_INDENT_IDX, currentIndent.length() - 1);
        }
        proceed(currentIndent + msg);
        if (msg.startsWith("Before")) {
            currentIndent = INDENT + currentIndent;
        }
    }
}
```

Listing 4: Main.java

```java
package application;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Main class for testing the implemented aspects.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public class Main {

    public static final String LOGGER_NAME = "aspectj-tracing";
    private static final Logger log = LoggerFactory.getLogger(LOGGER_NAME);
    public static boolean logIndentionEnabled = false;

    public static void main(String args[]) {
        log.info("---------------------------------------------------");
        log.info("testIndentionDisabled()");
        log.info("---------------------------------------------------");
        testIndentionDisabled();
        log.info("---------------------------------------------------");
        log.info("");
        log.info("---------------------------------------------------");
        log.info("testIndentionEnabled()");
        log.info("---------------------------------------------------");
        testIndentionEnabled();
        log.info("---------------------------------------------------");
    }

    private static void testIndentionDisabled(){
        logIndentionEnabled = false;
        PositiveValueStore value = new PositiveValueStore(10);
        try {
            value.addPositiveValue(1);
            value.addPositiveValue(-1);
        } catch (Throwable e) {
            log.error("Error in Main occurred", e);
        }
    }

    private static void testIndentionEnabled(){
        logIndentionEnabled = true;
        PositiveValueStore value = new PositiveValueStore(10);
        try {
            value.addPositiveValue(1);
            value.addPositiveValue(-1);
        } catch (Throwable e) {
            log.error("Error in Main occurred", e);
            }
        }
}
```

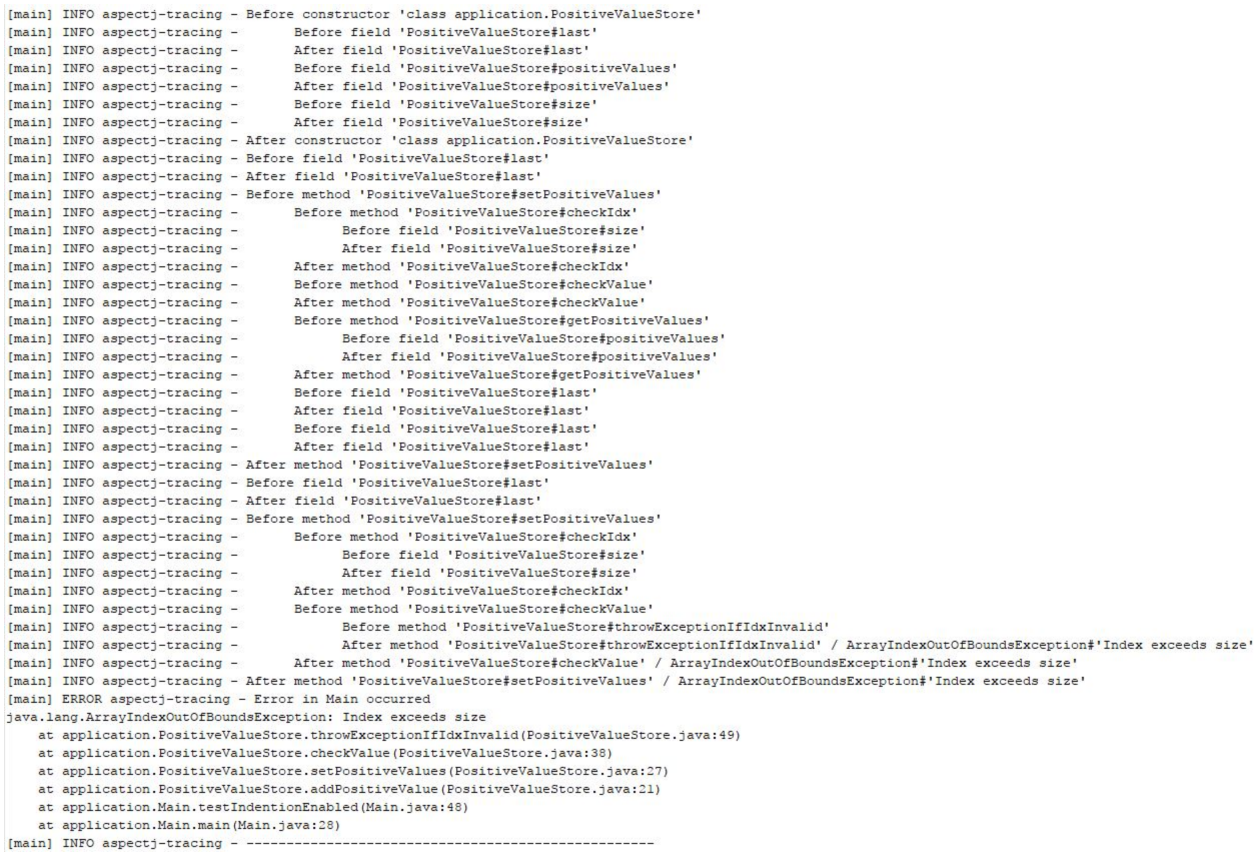
1.3 Tests

Folgender Abschnitt enthält die Tests der Aufgabenstellung in Form der generierten *logs*.



```
[main] INFO aspectj-tracing - --------------------------------------------------
[main] INFO aspectj-tracing - testIndentionDisabled()
[main] INFO aspectj-tracing - --------------------------------------------------
[main] INFO aspectj-tracing - Before constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#throwExceptionIfIdxInvalid'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#throwExceptionIfIdxInvalid' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#checkValue' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] ERROR aspectj-tracing - Error in Main occurred
java.lang.ArrayIndexOutOfBoundsException: Index exceeds size
    at application.PositiveValueStore.throwExceptionIfIdxInvalid(PositiveValueStore.java:49)
    at application.PositiveValueStore.checkValue(PositiveValueStore.java:38)
    at application.PositiveValueStore.setPositiveValues(PositiveValueStore.java:27)
    at application.PositiveValueStore.addPositiveValue(PositiveValueStore.java:21)
    at application.Main.testIndentionDisabled(Main.java:37)
    at application.Main.main(Main.java:22)
[main] INFO aspectj-tracing - --------------------------------------------------
```

Abbildung 1: Nicht eingerückter *log*



```
[main] INFO aspectj-tracing - Before constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing - After constructor 'class application.PositiveValueStore'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#positiveValues'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#getPositiveValues'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing -       After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing - Before field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - After field 'PositiveValueStore#last'
[main] INFO aspectj-tracing - Before method 'PositiveValueStore#setPositiveValues'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -             Before field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -             After field 'PositiveValueStore#size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkIdx'
[main] INFO aspectj-tracing -       Before method 'PositiveValueStore#checkValue'
[main] INFO aspectj-tracing -             Before method 'PositiveValueStore#throwExceptionIfIdxInvalid'
[main] INFO aspectj-tracing -             After method 'PositiveValueStore#throwExceptionIfIdxInvalid' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing -       After method 'PositiveValueStore#checkValue' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] INFO aspectj-tracing - After method 'PositiveValueStore#setPositiveValues' / ArrayIndexOutOfBoundsException#'Index exceeds size'
[main] ERROR aspectj-tracing - Error in Main occurred
java.lang.ArrayIndexOutOfBoundsException: Index exceeds size
    at application.PositiveValueStore.throwExceptionIfIdxInvalid(PositiveValueStore.java:49)
    at application.PositiveValueStore.checkValue(PositiveValueStore.java:38)
    at application.PositiveValueStore.setPositiveValues(PositiveValueStore.java:27)
    at application.PositiveValueStore.addPositiveValue(PositiveValueStore.java:21)
    at application.Main.testIndentionEnabled(Main.java:48)
    at application.Main.main(Main.java:28)
[main] INFO aspectj-tracing - ---------------------------------------------------
```

Abbildung 2: Eingerückter *log*

2 Caching

Dieser Abschnitt beschäftigt sich mit der Dokumentation der Aufgabenstellung *Caching*.

2.1 Lösungsidee

Der Algorithmus für die Berechnung des Binominialkoeffizienten wird in der Klasse *BinomialCoefficient* als statische Methode *calculate* implementiert.

Es wird ein abstrakter Aspekt *AbstractAspect* implementiert, der die *Pointcut firstCall, allCallsWithArgs* und *innerCalls* definiert, sowie *before, after Advices* für *firstCall* definiert, die vordefinierte implementierte Methoden aufrufen, die von den abgeleiteten Aspekten überschrieben werden können. Dies wird so strukturiert, damit alle Aspekte auf den ersten Aufruf der Berechnungsmethode reagieren können, um ihre Zustände zu initialisieren und zurückzusetzen. Dazu werden zwei Methoden *beforeFristCall* und *afterFirstCall* zur Verfügung gestellt, die von den konkreten Aspekten überschrieben werden können. Die Methoden *beforeFristCall* und *afterFirstCall* sind mit leeren Methodenrumpf in der Klasse *AbstractAspect* implementiert.

Es wird der Aspekt *LogRecursiveCallsAspect* implementiert, der einen *after Advice* definiert, der nur dann ausgeführt wird wenn die boolesche Variable *Main.LoggingEnabled* auf den Wert *true* gesetzt ist und die Bedingungen definiert im *PointCut innerCalls()* erfüllt sind. Beim ersten Aufruf der Berechnungsmethode wird vor dem Aufruf der Methode der Zähler initialisiert und nach dem Aufruf das Resultat über den *Logger* in die *Logs* geschrieben und der Zähler zurückgesetzt.

Es wird der Aspekt *BinomialCacheAspect* implementiert, der einen *around Advice* definiert, der nur ausgeführt wird wenn die boolesche Variable *Main.CachingEnabled* auf den Wert *true* gesetzt ist und die Bedingungen definiert im *PointCut allCallsWithArgs(n,m)* erfüllt sind. Der *around Advice* speichert die Berechnungsergebnisse, um sie bei einem erneuten Auftreten der Variablen *n, m* zurückzuliefern, um so einen weiteren rekursiven Abstiegt zu verhindern. Beim ersten Aufruf der Berechnungsmethode wird vor dem Aufruf der Methode der *Cache* initialisiert und nach dem Aufruf der *Cache* geleert. Es wird die Klasse *BinomMapKey* implementiert, die als Schlüssel in einer *java.util.HashMap* fungiert, welche die berechneten Werte speichert.

Es wird der Aspekt *RuntimeMeasurementAspect* implementiert, der die Dauer der gesamten Berechnung misst und über den *Logger* in die *Logs* schreibt. Dieser Aspekt überschreibt die beiden Methoden *beforeFristCall* und *afterFirstCall*, die von der abgeleiteten Klasse *AbstraktAspect* zur Verfügung gestellt werden. Dieser Aspekt wird nur dann ausgeführt, wenn die boolesche Variable *Main.RuntimeMeasurementEnabled* auf den Wert *true* gesetzt ist

2.2 Quelltexte

Folgender Abschnitt enthält die implementierten Klassen, Aspekte und das implementierte Testprogramm.

Listing 5: AbstractAspect.aj

```
package aspects;

/**
 * This is the base class for providing advice for the first calls and defines all of the used
 ↪  point cut.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/17/17
 */
public abstract aspect AbstractAspect {

    public pointcut firstCall():
            call(long application.BinomialCoefficient.calculate(..))
                    && !within(application.BinomialCoefficient)
                    && !within(aspects.*);

    public pointcut allCallsWithArgs(int n,
                                     int m):
            call(long application.BinomialCoefficient.calculate(int,int))
                    && args(n,m)
                    &&!within(aspects.*);

    public pointcut innerCalls():call(long application.BinomialCoefficient.calculate(..))
            && within(application.BinomialCoefficient)
            && !within(aspects.*);

    before(): firstCall() {
        beforeFirstCall();
    }

    after(): firstCall() {
        afterFirstCall();
    }

    protected void beforeFirstCall() {
        // default does nothing
    }

    protected void afterFirstCall() {
        // default does nothing
    }
}
```

Listing 6: LogRecursiveCallsAspect.aj

```
package aspects;

import application.Main;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * This
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public aspect LogRecursiveCallsAspect extends AbstractAspect {

    private int callCount;

    private static final Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);

    @Override
    protected void beforeFirstCall() {
        if (Main.LoggingEnabled) {
            callCount = 0;
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.LoggingEnabled) {
            log.info("Recursive calls: {}", callCount);
            callCount = 0;
        }
    }

    after(): if(application.Main.LoggingEnabled)
            && innerCalls() {
        callCount++;
    }
}
```

Listing 7: BinomialCacheAspect.aj

```
package aspects;

import application.Main;
import model.BinomMapKey;

import java.util.HashMap;
import java.util.Map;

/**
 * This is the caching aspect which caches the calculated value for n,m and returns the cached
 ↪  value if already calculated,
 * otherwise calculates it and caches it for the occurrence of n,m.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public aspect BinomialCacheAspect extends AbstractAspect {

    private Map<BinomMapKey, Long> cache = new HashMap<>(500);

    @Override
    protected void beforeFirstCall() {
        if (Main.CachingEnabled) {
            cache = new HashMap<>(500);
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.CachingEnabled) {
            cache = null;
        }
    }

    long around(int n,
                int m): if(application.Main.CachingEnabled) && allCallsWithArgs(n,m ) {
        final BinomMapKey key = new BinomMapKey(n, m);
        Long value;
        if ((value = cache.get(key)) == null) {
            value = proceed(n, m);
            // Will be null after last call, no need to cache anymore
            if (cache != null) {
                cache.put(key, value);
            }
        }

        return value;
    }
}
```

Listing 8: RuntimeMeasureAspect.aj

```
package aspects;

import application.Main;
import org.apache.commons.lang3.time.StopWatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/12/17
 */
public aspect RuntimeMeasureAspect extends AbstractAspect {

    private static final Logger log = LoggerFactory.getLogger(Main.LOGGER_NAME);
    private static StopWatch watch;

    @Override
    protected void beforeFirstCall() {
        if (Main.RuntimeMeasurementEnabled) {
            watch = new StopWatch();
            watch.start();
        }
    }

    @Override
    protected void afterFirstCall() {
        if (Main.RuntimeMeasurementEnabled) {
            watch.stop();
            log.info("Calculation duration: millis={}", watch.getTime());
            watch = null;
        }
    }
}
```

Listing 9: BinomMapKey.java

```
package model;

/**
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/12/17
 */
public class BinomMapKey {

    private final int n;
    private final int m;

    public BinomMapKey(int n,
                       int m) {
        this.n = n;
        this.m = m;
    }

    @Override
```

```java
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        BinomMapKey that = (BinomMapKey) o;

        if (n != that.n) return false;
        return m == that.m;
    }

    @Override
    public int hashCode() {
        int result = n;
        result = 31 * result + m;
        return result;
    }
}
```

Listing 10: BinomialCoefficient.java

```java
package application;

/**
 * This class calculates the binomial coefficient for n and m.
 *
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public class BinomialCoefficient {

    public static long calculate(int n,
                                 int m) {
        return (m == 0 || m == n)
                ? 1L
                : (calculate(n - 1, m - 1) + calculate(n - 1, m));
    }
}
```

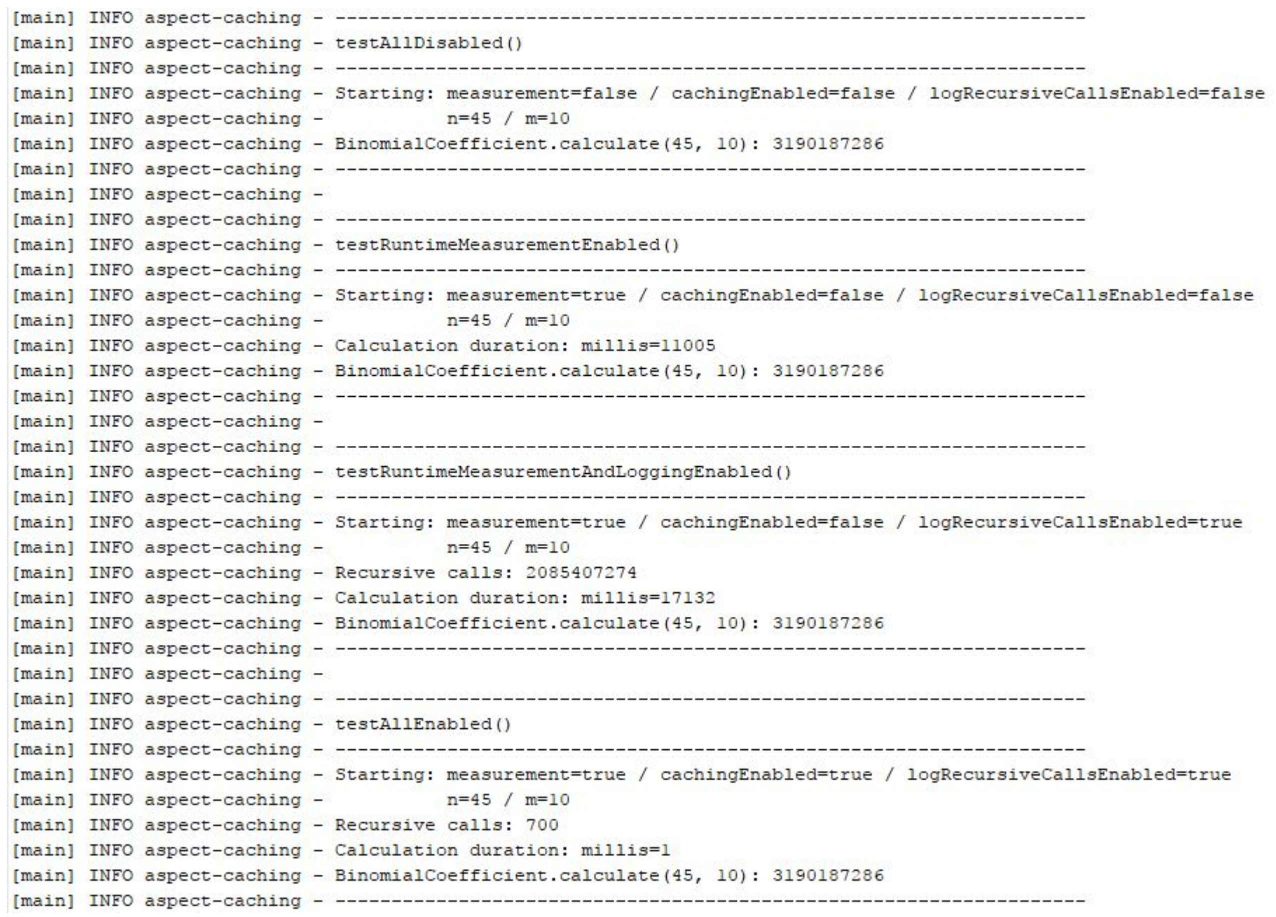
Listing 11: Main.java

```java
package application;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * @author Thomas Herzog <herzog.thomas81@gmail.com>
 * @since 05/05/17
 */
public class Main {

    public static boolean LoggingEnabled = false;
    public static boolean CachingEnabled = false;
    public static boolean RuntimeMeasurementEnabled = false;
    public static final String LOGGER_NAME = "aspect-caching";

    private static final Logger log = LoggerFactory.getLogger(LOGGER_NAME);

    public static void main(String args[]) {
        final int n = 45;
        final int m = 10;
        log.info("------------------------------------------------------------------");
        log.info("testAllDisabled()");
        log.info("------------------------------------------------------------------");
        testAllDisabled(n, m);
        log.info("------------------------------------------------------------------");
        log.info("");
        log.info("------------------------------------------------------------------");
        log.info("testRuntimeMeasurementEnabled()");
        log.info("------------------------------------------------------------------");
        testRuntimeMeasurementEnabled(n, m);
        log.info("------------------------------------------------------------------");
        log.info("");
        log.info("------------------------------------------------------------------");
        log.info("testRuntimeMeasurementAndLoggingEnabled()");
        log.info("------------------------------------------------------------------");
        testRuntimeMeasurementAndLoggingEnabled(n, m);
        log.info("------------------------------------------------------------------");
        log.info("");
        log.info("------------------------------------------------------------------");
        log.info("testAllEnabled()");
        log.info("------------------------------------------------------------------");
        testAllEnabled(n, m);
        log.info("------------------------------------------------------------------");
    }

    private static void testAllDisabled(final int n,
                                        final int m) {
        LoggingEnabled = false;
        CachingEnabled = false;
        RuntimeMeasurementEnabled = false;

        log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}", RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
        log.info("          n={} / m={}", n,m);
        log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n, m));
    }

    private static void testRuntimeMeasurementEnabled(final int n,
                                                      final int m) {
        LoggingEnabled = false;
        CachingEnabled = false;
        RuntimeMeasurementEnabled = true;

        log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}", RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
        log.info("          n={} / m={}", n,m);
        log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n, m));
    }

    private static void testRuntimeMeasurementAndLoggingEnabled(final int n,
                                                                final int m) {
        CachingEnabled = false;
        LoggingEnabled = true;
        RuntimeMeasurementEnabled = true;

        log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}", RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
        log.info("          n={} / m={}", n,m);
        log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n, m));
    }

    private static void testAllEnabled(final int n,
                                       final int m) {
        LoggingEnabled = true;
        CachingEnabled = true;
        RuntimeMeasurementEnabled = true;

        log.info("Starting: measurement={} / cachingEnabled={} / logRecursiveCallsEnabled={}", RuntimeMeasurementEnabled, CachingEnabled, LoggingEnabled);
        log.info("          n={} / m={}", n,m);
        log.info("BinomialCoefficient.calculate(45, 10): {}", BinomialCoefficient.calculate(n, m));
    }
}
```

2.3 Tests

Folgender Abschnitt enthält die Tests der Aufgabenstellung in Form der generierten *logs*.

Während der Tests hat sich gezeigt, dass mit aktiviertem *Caching* sich das Laufzeitverhalten deutlich verbessert hat, da die rekursiven Aufrufe deutlich weniger geworden sind und daher auch die Anzahl der Berechnungen sich deutlich verringert hat.



```
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - testAllDisabled()
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=false / cachingEnabled=false / logRecursiveCallsEnabled=false
[main] INFO aspect-caching -          n=45 / m=10
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - testRuntimeMeasurementEnabled()
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=false / logRecursiveCallsEnabled=false
[main] INFO aspect-caching -          n=45 / m=10
[main] INFO aspect-caching - Calculation duration: millis=11005
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - testRuntimeMeasurementAndLoggingEnabled()
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=false / logRecursiveCallsEnabled=true
[main] INFO aspect-caching -          n=45 / m=10
[main] INFO aspect-caching - Recursive calls: 2085407274
[main] INFO aspect-caching - Calculation duration: millis=17132
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching -
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - testAllEnabled()
[main] INFO aspect-caching - -------------------------------------------------------------------
[main] INFO aspect-caching - Starting: measurement=true / cachingEnabled=true / logRecursiveCallsEnabled=true
[main] INFO aspect-caching -          n=45 / m=10
[main] INFO aspect-caching - Recursive calls: 700
[main] INFO aspect-caching - Calculation duration: millis=1
[main] INFO aspect-caching - BinomialCoefficient.calculate(45, 10): 3190187286
[main] INFO aspect-caching - -------------------------------------------------------------------
```

Abbildung 3: *Caching* Test *Logs*

3 Asepct-Oriented TSP Solver

Dieser Abschnitt beschäftigt sich mit der Dokumentation der Aufgabenstellung *Asepct-Oriented TSP Solver*.

3.1 Lösungsidee

Die Projektstruktur wurde dahingehend verändert, dass die Schnittstellen und die *Exceptions* in eigene Pakete ausgelagert wurden. Die Konfiguration der Aspekte wurde in der Klasse *util.AspectjConfig* zusammengeführt, die jetzt alle booleschen Variablen enthält die von den Aspekten verwendet werden, um zu entscheiden, ob sie aktiviert sind oder nicht. Die nötigen Änderungen in den bestehenden Aspekten wurden vorgenommen, damit die Aspekte mit der geänderten Projektkonfiguration arbeiten können.

Es wird der Aspekt *CountEvaluatedSolutionsAspect* implementiert, der die Anzahl der evaluierten Lösungen zählt. Es wird ein *PointCut executeCall* definiert, für den die beiden *before, after Advices* definiert werden, wobei der *before Advice* vor dem Methodenaufruf der Methode *Algorithm.execute* den Zähler initialisiert der *after Advice* nach dem Methodenaufruf der Methode *Algorithm.execute* das Resultat über einen *Logger* in die *Logs* schreibt und den Zähler zurücksetzt. Ein weiterer *after Advice* erhöht den Zähler nach dem Methodenaufruf der Methode *Solution.evaluate*. Dieser Aspekt arbeitet nur gegen die Schnittstellen *Algorithm* und *Solution* und ist daher auf alle Implementierungen dieser Schnittstellen anwendbar. Dieser Asüect wird als abstrakt ausgewiesen, da der zu implementierende Aspekt *LimitEvaluatedSolutions*, von diesem Aspekt ableiten soll. Dieser Aspekt greift nur wenn die boolesche Variable *AspectjConfig.countSolutionsEnabled* auf den Wert *true* gesetzt ist.

Es wird der Aspekt *GAElitismAspect* implementiert, der die schlechteste Lösung der neu erstellten Kinder durch die beste Lösung des vorherigen Durchlaufs ersetzt. Es wird ein *around Advice* für die Methode *GA.createChildren* implementiert, der das zurückgelieferte *Array* der Kinder manipuliert. Wird ein Kind ausgetauscht, so wird eine Meldung über einen *Logger* in die *Logs* geschrieben. Dieser Aspekt ist abhängig von der Klasse *GA*, da die Schnittstelle *Algorithm* die Methode *createChildren* nicht definiert. Dieser Aspekt greift nur wenn die boolesche Variable *AspectjConfig.elitismEnabled* auf den Wert *true* gesetzt ist.

Es wird der Aspekt *LimitEvaluatedSolutions* implementiert, der die Iteration beim Erreichen einer vorgegebenen Anzahl von evaluierten Lösungen abbricht. Es wird hierbei das Ausführen der nächsten Iteration verhindert, wodurch in der aktuellen Iteration trotzdem mehr als die vorgegebenen Lösungen evaluiert werden können, daher muss die vorgegebene Anzahl der Iterationen ein Vielfaches der evaluierten Lösungen/Iteration sein, damit genau bei der vorgegebenen Anzahl abgebrochen wird. Dieser Aspekt erbt von dem implementierten abstrakten Aspekt *CountEvaluatedSolutionsAspect* und verwendet dessen Zähler. Ein *around advice* für die Methode *Algorithm.iterate* verhindert das Ausführen der Iteration und setzt eine boolesche Variable auf *true*, die beim *around advice* für die Methode *Algorithm.isTerminated true* als Resultat liefert, wenn die Iteration abgebrochen werden soll. Diese Aspekt ist anwendbar auf alle Implementierungen der Schnittstelle *Algorithm*.

Es wird der Aspekt *GAProtocolProgressAspect* implementiert, der die beste Lösung, schlechteste Lösung und den Durchschnitt der Lösungen einer Population einer Iteration speichert und nach dem Ausführen des Algorithmus über einen *Logger* in die *Logs* schreibt und ein SVG-Diagramm mit der Bibliothek *gp2.svg-generator* erstellt. Zusätzlich wird ein SVG-Diagramm mit dem besten *roundtrip*, der besten Lösung, die vom Algorithmus gefunden wurde erstellt. Es wird der *PointCut firstExecuteCall* definiert, für den die beiden *before, after Advices* definiert werden, wobei der *before Advice* die Zustände des Aspekts initialisiert und der *after Advice* die Zustände des Aspekts zurücksetzt. Es werden zwei *after Advices* definiert, wobei ein *after Advice* nach der Ausführung der Methode *Algorithm.initialize* und der andere *after Advice* nach der Ausführung der Methode *Algorithm.iterate* ausgeführt werden. Der *after Advice* für die Methode *Algorithm.initialize* ist notwendig, weil dort die erste Population erstellt wird. Dieser Aspekt greift nur wenn die Variable *AspectjConfig.reportAlgorithmEnabled* auf den Wert *true* gesetzt ist.

Es wird eine Klasse *AspectReport* implementiert, welche die gesammelten Daten über einen *Logger* in die *Logs* schreibt, ein SVG-Diagramm für *best, worst, average* erstellt und ein SVG-Diagramm für den besten *rountrip* der besten vom Algorithmus gefundenen Lösung erstellt. Um das SVG-Diagramm zu zeichnen, wurde der in der vorherigen Übung implementierte SVG-Generator verwendet. Da in der letzten Übung kein *PolylineShape* implementiert wurde, wird der Verlauf mi einem *LineShape* gezeichnet, wobei die *Destination* der vorherigen Iteration als *Origin* der aktuellen Iteration verwendet wird. Eine *Polyline* wäre zwar effizienter und das SVG-Dokument würde weitaus kleiner ausfallen, jedoch stand mir kein implementiertes *PolylineShape* zur Verfügung.

3.2 Quelltexte

Folgender Abschnitt enthält die implementierten Klassen, Aspekte und das implementierte Testprogramm.

Listing 12: CountEvaluatedSolutionsAspect.aj

```
package aspects;

import aspects.util.AspectjConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * This aspect counts the solution evaluation within the {@link tsp.api.Algorithm}
 ↪   implementations.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/13/17
 */
public abstract aspect CountEvaluatedSolutionsAspect {

    long solutionCount = 0;

    private static final Logger log = LoggerFactory.getLogger(AspectjConfig.LOGGER_NAME);

    pointcut executeCall():
            if(aspects.util.AspectjConfig.countSolutionsEnabled)
                    && call(* *.*.Algorithm.execute(..))
                    && !within(*.*.Algorithm+);

    before(): executeCall() {
        solutionCount = 0;
    }

    after(): executeCall() {
        log.info("Evaluation count: '{}'", solutionCount);
        solutionCount = 0;
    }

    after(): if(aspects.util.AspectjConfig.countSolutionsEnabled)
            &&call(* *.*.Solution.evaluate(..))
            && within(*.*.Algorithm+) {
        solutionCount++;
    }
}
```

Listing 13: GAElitismAspect.aj

```
package aspects;

import aspects.util.AspectjConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import tsp.GA;
import tsp.api.Solution;

import java.util.Arrays;

/**
 * This aspects realizes the 1-elitism mechanism by replacing the worst child with the best parent
 ↪   of the former run.<br>
 * This aspect is for the implemented {@link GA} algorithm.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/13/17
 */
public privileged aspect GAElitismAspect {

    private Solution bestParent;

    private static final Logger log = LoggerFactory.getLogger(AspectjConfig.LOGGER_NAME);

    Solution[] around(): if(aspects.util.AspectjConfig.elitismEnabled)
            && call(Solution[] *.GA.createChildren(..))
            && withincode(* *.GA.iterate(..)) {
        bestParent = ((GA) thisJoinPoint.getTarget()).best;

        final Solution[] children = proceed();

        if (bestParent != null) {
            Arrays.sort(children);
            final Solution worstChild = children[children.length - 1];
            children[children.length - 1] = bestParent;
            //log.info("Replaced worst child with best of former run. worstChild={} /
 ↪  bestParent={}", worstChild.getQuality(), bestParent.getQuality());
        }

        return children;
    }
}
```

Listing 14: GAProtocolProgressAspect.aj

```
package aspects;

import aspects.util.AspectReport;
import aspects.util.AspectjConfig;
import at.fh.ooe.gp2.template.api.Coordinate;
import tsp.GA;
import tsp.PathSolution;
import tsp.api.Solution;

import java.util.Arrays;

```

```
/**
 * This aspect protocols the best and worst found solution during the algorithm execution.
 * This aspect is for the implemented {@link GA} algorithm.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/14/17
 */
public privileged aspect GAProtocolProgressAspect {

    private AspectReport report;
    pointcut firstExecuteCall():
            if(aspects.util.AspectjConfig.reportAlgorithmEnabled)
                    && call(* *.*.Algorithm.execute(..))
                    && !within(*.*.Algorithm+);

    // Init
    before(): firstExecuteCall() {
        report = new AspectReport(AspectjConfig.reportFileName + "-chart-",
                                  AspectjConfig.reportFileName + "-path-");
    }

    // Report and Cleanup
    after(): firstExecuteCall() {
        final Solution solution = (((GA) thisJoinPoint.getTarget())).best;
        if (solution instanceof PathSolution) {
            final int[] bestTour = ((PathSolution) solution).tour;
            final double[][] vertices = ((PathSolution) solution).tsp.vertices;
            Arrays.stream(bestTour).mapToObj(i -> new Coordinate(vertices[i][0],
↪  vertices[i][1])).forEach(report::addPathValue);
        }
        report.generateAllReports();
        report = null;
    }

    // First population
    after(): if(aspects.util.AspectjConfig.reportAlgorithmEnabled)
            && call(* *.*.Algorithm.initialize(..))
            && withincode(* *.*.Algorithm.execute(..)) {
        final GA target = ((GA) thisJoinPoint.getTarget());
        handleBestAndWorstAndAverage(target.population);
    }

    // All other populations
    after(): if(aspects.util.AspectjConfig.reportAlgorithmEnabled)
            && call(* *.*.Algorithm.iterate(..))
            && withincode(* *.*.Algorithm.execute(..)) {
        final GA target = ((GA) thisJoinPoint.getTarget());
        handleBestAndWorstAndAverage(target.population);
    }

    private void handleBestAndWorstAndAverage(final Solution[] population) {
        // calculate average of population
        double average = 0.0;
        double best = 0.0;
        double worst = 0.0;

        if (population.length > 0) {
            for (final Solution solution : population) {
                average += solution.getQuality();
            }
            average = (average / population.length);
            best = population[0].getQuality();
            worst = population[population.length - 1].getQuality();
```

```java
        }

        // Set calculated run results on report context
        report.addRunValue(best, worst, average);
    }
}
```

Listing 15: AspectjConfig.java

```java
package aspects.util;

/**
 * This class holds the global configuration for the aspects.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/14/17
 */
public class AspectjConfig {

    public static boolean measureRuntime = true;
    public static boolean randomSelection = false;
    public static boolean cyclicCrossover = false;
    public static boolean maximalPreservativeCrossover = false;

    public static boolean elitismEnabled = false;
    public static boolean countSolutionsEnabled = false;
    public static boolean limitIterationsActive = false;
    public static boolean reportAlgorithmEnabled = false;
    public static long maxSolutions = 100;
    public static String reportFileName = "tsp-solver";

    public static final String LOGGER_NAME = "aspectj-tsp-solver";
}
```

Listing 16: AspectReport.java

```java
package aspects.util;

import at.fh.ooe.gp2.template.api.Coordinate;
import at.fh.ooe.gp2.template.api.shape.Diagram;
import at.fh.ooe.gp2.template.api.shape.LineShape;
import at.fh.ooe.gp2.template.api.shape.PointShape;
import at.fh.ooe.gp2.template.api.shape.TextShape;
import at.fh.ooe.gp2.template.impl.generator.FreemarkerGenerators;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.awt.*;
import java.io.File;
import java.io.FileWriter;
import java.io.Writer;
import java.math.BigDecimal;
import java.util.HashSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;

/**
 * This class represents a report context for the evaluated solutions evaluated during an
 ↪   algorithm execution.
 *
 * @author Thomas Herzog <t.herzog@curecomp.com>
 * @since 05/14/17
 */
public class AspectReport {

    /**
     * Represents a y value of a run report
     */
    private static final class YValue {
        public final double best;
        public final double worst;
        public final double average;

        public YValue(double best,
                      double worst,
                      double average) {
            this.best = best;
            this.worst = worst;
            this.average = average;
        }

        public double getBest() {
            return best;
        }

        public double getWorst() {
            return worst;
        }

        public double getAverage() {
            return average;
        }
    }

    private final int height;
    private final int width;
    private final double stokeWidth;
    private final double pathStokeWidth;
    private final Color bestStrokeColor;
    private final Color worstStrokeColor;
    private final Color avgStrokeColor;
    private final String chartFilename;
    private final String pathFilename;
    private final List<YValue> yRunValues;
    private final List<Coordinate> pathValues;

    private static final Logger log = LoggerFactory.getLogger(AspectjConfig.LOGGER_NAME);
    private static final double DEFAULT_STROKE_WIDTH = 0.8;
    private static final double DEFAULT_PATH_STROKE_WIDTH = 0.5;
    private static final Color DEFAULT_BEST_COLOR = Color.GREEN;
    private static final Color DEFAULT_WORST_COLOR = Color.RED;
    private static final Color DEFAULT_AVG_COLOR = Color.ORANGE;

    /**
     * @param chartFilename the filename of the run report
     * @param pathFilename  the filename of the path report
     */
    public AspectReport(final String chartFilename,
                        final String pathFilename) {
        this.height = 700;
        this.width = 900;
        stokeWidth = DEFAULT_STROKE_WIDTH;
        bestStrokeColor = DEFAULT_BEST_COLOR;
        worstStrokeColor = DEFAULT_WORST_COLOR;
        avgStrokeColor = DEFAULT_AVG_COLOR;
        pathStokeWidth = DEFAULT_PATH_STROKE_WIDTH;
        this.chartFilename = chartFilename;
        this.pathFilename = pathFilename;

        pathValues = new LinkedList<>();
        yRunValues = new LinkedList<>();
    }

    /**
     * Adds a run y value
     *
     * @param best    the best of the run
     * @param worst   the worst of the run
     * @param average the average of the run
     */
    public void addRunValue(final double best,
                            final double worst,
                            final double average) {
        yRunValues.add(new YValue(best, worst, average));
    }

    /**
     * Adds a path coordinate.
     *
     * @param coordinate the coordinate of the path
     */
    public void addPathValue(final Coordinate coordinate) {
        pathValues.add(coordinate);
    }

    /**
     * Generates all supported reports.
     */
    public void generateAllReports() {
        generateConsoleReport();
        generateRunSvgReport();
        generatePathSvgReport();
    }

    /**
     * Generates the console run report.
     */
    public void generateConsoleReport() {
        int run = 0;
        for (final YValue item : yRunValues) {
            log.info("run={}: best={} / worst={} / average={}", run, item.best, item.worst,
 ↪  item.average);
            run++;
        }
    }

    /**
     * Generates the run svg report.
     */
    public void generateRunSvgReport() {
        if(yRunValues.isEmpty()) {
            log.warn("Cannot create run report because no run data available");
            return;
        }
        try {
            // get max value for normalization over all values of all captured types
            final Set<Double> allValues = new HashSet<Double>() {{
                addAll(yRunValues.stream().map(YValue::getBest).collect(Collectors.toList()));
                addAll(yRunValues.stream().map(YValue::getAverage).collect(Collectors.toList()));
                addAll(yRunValues.stream().map(YValue::getWorst).collect(Collectors.toList()));
            }};

            // Lower bound is 0.0 if value not smaller than 0
            double minValue = allValues.stream().min(Double::compare).orElse(0.0);
            minValue = (minValue >= 0) ? 0.0 : minValue;

            // get maximum value which will be the upper bound
            double maxValue = allValues.stream().max(Double::compare).orElse(0.0);

            // freemarker generators
            final FreemarkerGenerators.DiagramGenerator diagramGenerator = new
    ↪   FreemarkerGenerators.DiagramGenerator();
            final FreemarkerGenerators.LineGenerator lineGenerator = new
    ↪   FreemarkerGenerators.LineGenerator();
            final Diagram diagram = new Diagram(diagramGenerator, width, height, 0.0, (double)
    ↪   width, 0.0, (double) height, false);

            // chart margins and dimensions
            final double widthMargin = 50.0;
            final double heightMargin = 25.0;
            final double chartWidth = width - (widthMargin * 2);
            final double chartHeight = height - (heightMargin * 2);
            final double chartStep = chartWidth / yRunValues.size();

            generatePlotAxis(diagram, 0.0, yRunValues.size(), minValue, maxValue, widthMargin,
    ↪   heightMargin);

            double currentStep = 0.0 + widthMargin;
            // remember origin for next value
            Coordinate origBest, origWorst, origAvg;
            origBest = origWorst = origAvg = null;
            for (final YValue value : yRunValues) {
                final double yBest = normalizeValue(minValue, maxValue, 0.0, chartHeight,
    ↪   value.best);
                final double yWorst = normalizeValue(minValue, maxValue, 0.0, chartHeight,
    ↪   value.worst);
                final double yAvg = normalizeValue(minValue, maxValue, 0.0, chartHeight,
    ↪   value.average);

                // coordinates with inverted y position
                final Coordinate destBest = new Coordinate(currentStep, (chartHeight - yBest +
    ↪   heightMargin));
                final Coordinate destWorst = new Coordinate(currentStep, (chartHeight - yWorst +
    ↪   heightMargin));
                final Coordinate destAvg = new Coordinate(currentStep, (chartHeight - yAvg +
    ↪   heightMargin));

                diagram.addShape(new LineShape(diagram, lineGenerator, (origBest != null) ?
    ↪   origBest : destBest, destBest, bestStrokeColor, stokeWidth));
                diagram.addShape(new LineShape(diagram, lineGenerator, (origWorst != null) ?
    ↪   origWorst : destWorst, destWorst, worstStrokeColor, stokeWidth));
                diagram.addShape(new LineShape(diagram, lineGenerator, (origAvg != null) ? origAvg
    ↪   : destAvg, destAvg, avgStrokeColor, stokeWidth));

                origBest = destBest;
                origWorst = destWorst;
                origAvg = destAvg;
                currentStep += chartStep;
```

```java
            }

            // Generate svg files
            generateFile(chartFilename, diagram, diagramGenerator);
        } catch (Throwable e) {
            log.error("Svg report could not be generated", e);
        }
    }


    /**
     * Generates the path svg report.
     */
    public void generatePathSvgReport() {
        if(pathValues.isEmpty()) {
            log.warn("Cannot create path report because no path data available");
            return;
        }
        try {
            double minXValue =
↪   pathValues.stream().map(Coordinate::getX).min(Double::compare).orElse(0.0);
            double maxXValue =
↪   pathValues.stream().map(Coordinate::getX).max(Double::compare).orElse(0.0);
            double minYValue =
↪   pathValues.stream().map(Coordinate::getY).min(Double::compare).orElse(0.0);
            double maxYValue =
↪   pathValues.stream().map(Coordinate::getY).max(Double::compare).orElse(0.0);

            // freemarker generators
            final FreemarkerGenerators.DiagramGenerator diagramGenerator = new
↪   FreemarkerGenerators.DiagramGenerator();
            final FreemarkerGenerators.LineGenerator lineGenerator = new
↪   FreemarkerGenerators.LineGenerator();
            final FreemarkerGenerators.TextGenerator textGenerator = new
↪   FreemarkerGenerators.TextGenerator();
            final FreemarkerGenerators.PointGenerator pointGenerator = new
↪   FreemarkerGenerators.PointGenerator();
            final Diagram diagram = new Diagram(diagramGenerator, width, height, 0.0, (double)
↪   width, 0.0, (double) height, false);

            // chart margins and dimensions
            final double widthMargin = 50.0;
            final double heightMargin = 25.0;
            final double chartWidth = width - (widthMargin * 2);
            final double chartHeight = height - (heightMargin * 2);

            generatePlotAxis(diagram, minXValue, maxXValue, minYValue, maxYValue, widthMargin,
↪   heightMargin);

            // remember origin for next value
            Coordinate origBest = null;
            int counter = 1;
            for (final Coordinate coordinate : pathValues) {
                final double x = normalizeValue(minXValue, maxXValue, 0.0, chartWidth,
↪   coordinate.getX());
                final double y = normalizeValue(minYValue, maxYValue, 0.0, chartHeight,
↪   coordinate.getY());

                // coordinates with inverted y position
                final Coordinate destBest = new Coordinate(x + widthMargin, (chartHeight - y +
↪   heightMargin));
                diagram.addShape(new LineShape(diagram, lineGenerator, (origBest != null) ?
↪   origBest : destBest, destBest, bestStrokeColor, pathStokeWidth));
```

```
                double radius = 1.0;
                if ((counter == 1) || (counter == pathValues.size())) {
                    radius = 2.5;
                }
                diagram.addShape(new PointShape(diagram, pointGenerator, destBest, Color.RED,
↪   Color.RED, radius, 1.0));
                if ((counter == 1) || (counter == pathValues.size())) {
                    diagram.addShape(new TextShape(diagram, textGenerator, destBest, Color.BLACK,
↪   null, String.valueOf(counter), "Arial", 9.0, 0.5));
                }

                origBest = destBest;
                counter++;
            }
            generateFile(pathFilename, diagram, diagramGenerator);
        } catch (Throwable e) {
            log.error("Svg report could not be generated", e);
        }
    }

    /**
     * Generates the plot axis with markers.
     *
     * @param diagram      the diagram to add plot to
     * @param minX         the minimum x value for axis markers
     * @param maxX         the maximum x value for axis markers
     * @param minY         the minimum y value for axis markers
     * @param maxY         the maximum y value for axis markers
     * @param widthMargin  the margin to the left and right
     * @param heightMargin the margin to the top and bottom
     * @throws Exception if the generation fails for any reason
     */
    private void generatePlotAxis(final Diagram diagram,
                                  final double minX,
                                  final double maxX,
                                  final double minY,
                                  final double maxY,
                                  final double widthMargin,
                                  final double heightMargin) throws Exception {
        // freemarker generators
        final FreemarkerGenerators.LineGenerator lineGenerator = new
↪   FreemarkerGenerators.LineGenerator();
        final FreemarkerGenerators.TextGenerator textGenerator = new
↪   FreemarkerGenerators.TextGenerator();

        // chart dimensions
        final double chartWidth = width - (widthMargin * 2);
        final double chartHeight = height - (heightMargin * 2);

        // Coordinate lines
        final LineShape xAxis = new LineShape(diagram, lineGenerator, new Coordinate(0.0, (height
↪   - heightMargin)), new Coordinate(width - widthMargin, (height - heightMargin)), Color.BLACK,
↪   1.0);
        final LineShape yAxis = new LineShape(diagram, lineGenerator, new Coordinate(widthMargin,
↪   height), new Coordinate(widthMargin, heightMargin), Color.BLACK, 1.0);
        diagram.addShape(xAxis);
        diagram.addShape(yAxis);

        // Add xAxis markers
        final int markerStep = 25;
        for (int i = 1; i <= markerStep; i++) {
            // x, y position for x,y marker
            final double xPos = widthMargin + ((chartWidth / markerStep) * i);
            final double yPos = (height - heightMargin) - ((chartHeight / markerStep) * i);

            // Calculate marker values
            final String yMarkerValue = (i == markerStep)
                    ? BigDecimal.valueOf(maxY).setScale(2, BigDecimal.ROUND_HALF_DOWN).toString()
                    : BigDecimal.valueOf(maxY - minY)
                                .setScale(2, BigDecimal.ROUND_DOWN)
                                .divide(BigDecimal.valueOf(markerStep),
    ↪  BigDecimal.ROUND_HALF_EVEN)
                                .multiply(BigDecimal.valueOf(i))
                                .toString();
            final String xMarkerValue = (i == markerStep)
                    ? BigDecimal.valueOf(maxX).setScale(2, BigDecimal.ROUND_HALF_DOWN).toString()
                    : BigDecimal.valueOf(maxX - minX)
                                .setScale(2, BigDecimal.ROUND_DOWN)
                                .divide(BigDecimal.valueOf(markerStep),
    ↪  BigDecimal.ROUND_HALF_EVEN)
                                .multiply(BigDecimal.valueOf(i))
                                .toString();

            // Add axis markers
            diagram.addShape(new LineShape(diagram,
                                           lineGenerator,
                                           new Coordinate(widthMargin - 5, yPos),
                                           new Coordinate(widthMargin + 5, yPos),
                                           Color.BLACK,
                                           1.0));
            diagram.addShape(new LineShape(diagram,
                                           lineGenerator,
                                           new Coordinate(xPos, (height - heightMargin - 5)),
                                           new Coordinate(xPos, (height - heightMargin + 5)),
                                           Color.BLACK,
                                           1.0));
            diagram.addShape(new LineShape(diagram,
                                           lineGenerator,
                                           new Coordinate(widthMargin, yPos),
                                           new Coordinate(width - widthMargin, yPos),
                                           Color.DARK_GRAY,
                                           0.1));

            // Adda xis marker texts
            final int markerValueYOffset = ((i % 2) != 0) ? 0 : 9;
            diagram.addShape(new TextShape(diagram,
                                           textGenerator,
                                           new Coordinate(xPos - 15, height - heightMargin + 15 +
    ↪  markerValueYOffset),
                                           Color.BLACK,
                                           null,
                                           xMarkerValue,
                                           "Arial",
                                           8.5,
                                           0.25));
            diagram.addShape(new TextShape(diagram, textGenerator, new Coordinate(5, yPos - 5),
    ↪  Color.BLACK, null, yMarkerValue, "Arial", 8.5, 0.25));
        }
    }

    /**
     * Generate the svg diagram file.
     *
     * @param filename         the filename of the svg diagram
     * @param diagram          the diagram to be saved
     * @param diagramGenerator the generator for the diagram
     * @throws Exception if the generation fails for any reason
     */
    private void generateFile(final String filename,
                              final Diagram diagram,
                              final FreemarkerGenerators.DiagramGenerator diagramGenerator) throws
    ↪  Exception {
        // Generate svg files
        String svgContent = diagramGenerator.generate(diagram);
        File svgFile = File.createTempFile(filename, ".svg");
        try (final Writer fileWriter = new FileWriter(svgFile)) {
            fileWriter.write(svgContent);
        }
        log.info("SVG file location: {}", svgFile.getAbsolutePath());
    }

    /**
     * Normalizes the values in the new range
     *
     * @param oldMin the old minimum
     * @param oldMax the old maximum
     * @param newMin the new minimum
     * @param newMax the new maximum
     * @param value  the value to normalize
     * @return the normalized value
     */
    private double normalizeValue(final double oldMin,
                                  final double oldMax,
                                  final double newMin,
                                  final double newMax,
                                  final double value) {
        return (((newMin + (value - oldMin)) * (newMax - newMin)) / (oldMax - oldMin));
    }
}
```

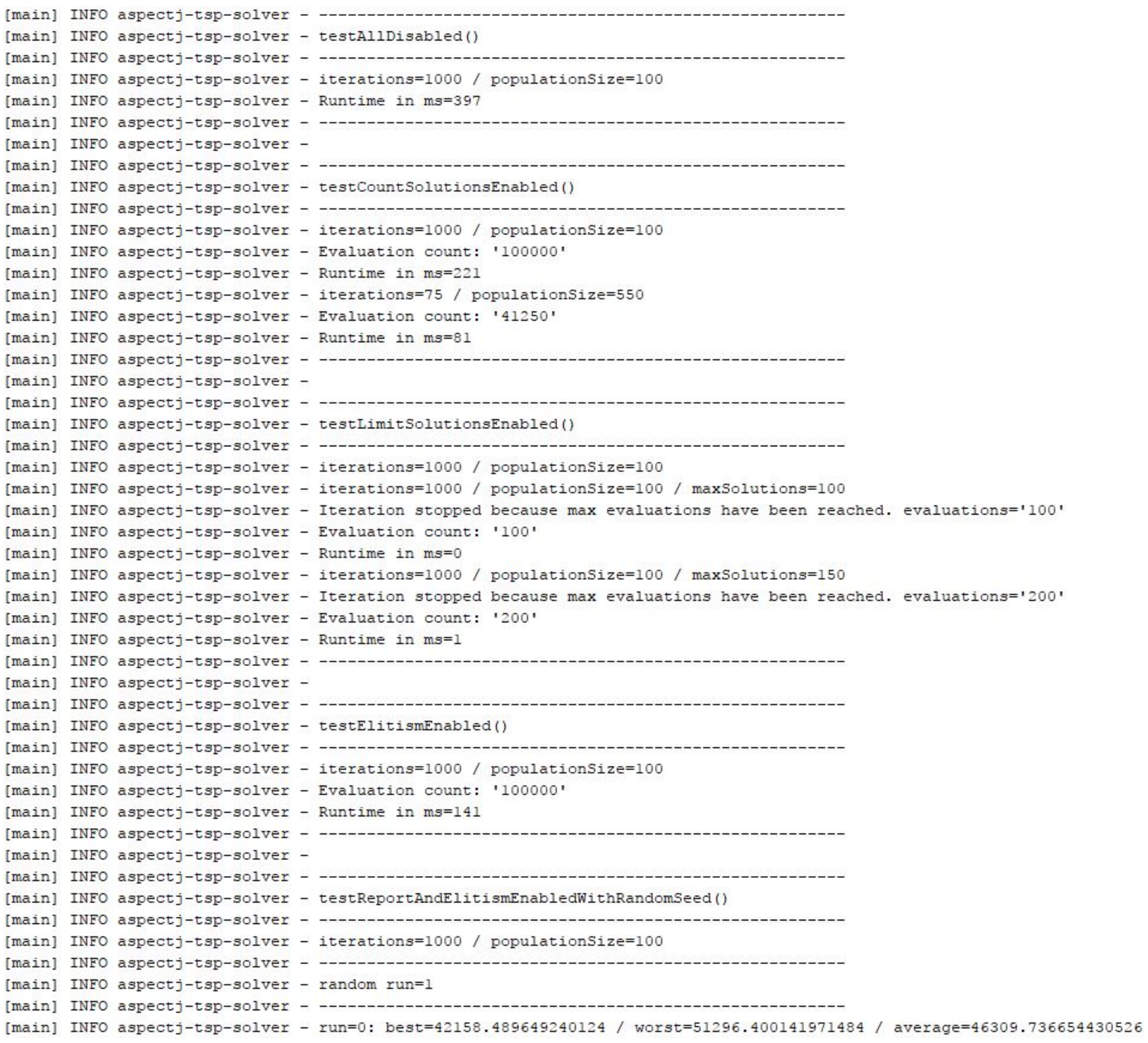
Listing 17: Main.java

```
package tsp;

import aspects.util.AspectjConfig;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import tsp.api.Problem;
import tsp.config.AlgorithmConfig;

import java.util.concurrent.ThreadLocalRandom;

public class Main {

    private static final int ITERATIONS = 1000;
    private static final int POPULATION_SIZE = 100;
    private static final int RANDOM_RUN = 3;
    private static final Logger log = LoggerFactory.getLogger(AspectjConfig.LOGGER_NAME);

    public static void main(String[] args) {
        // Always enabled
        AspectjConfig.measureRuntime = true;
        try {
            // Ensure same results with same seed
            AlgorithmConfig.init();
            log.info("-------------------------------------------------------");
            log.info("testAllDisabled()");
            log.info("-------------------------------------------------------");
```

```java
            testAllDisabled();
            log.info("-------------------------------------------------------");
            log.info("");
            log.info("-------------------------------------------------------");
            log.info("testCountSolutionsEnabled()");
            log.info("-------------------------------------------------------");
            testCountSolutionsEnabled();
            log.info("-------------------------------------------------------");
            log.info("");
            log.info("-------------------------------------------------------");
            log.info("testLimitSolutionsEnabled()");
            log.info("-------------------------------------------------------");
            testLimitSolutionsEnabled();
            log.info("-------------------------------------------------------");
            log.info("");
            log.info("-------------------------------------------------------");
            log.info("testElitismEnabled()");
            log.info("-------------------------------------------------------");
            testElitismEnabled();
            log.info("-------------------------------------------------------");
            log.info("");
            log.info("-------------------------------------------------------");
            log.info("testReportAndElitismEnabledWithRandomSeed()");
            log.info("-------------------------------------------------------");
            testReportAndElitismEnabledWithRandomSeed();
            log.info("-------------------------------------------------------");
            log.info("");
            log.info("-------------------------------------------------------");
            log.info("testReportEnabledWithRandomSeed()");
            log.info("-------------------------------------------------------");
            testReportEnabledWithRandomSeed();
            log.info("-------------------------------------------------------");
        } catch (Exception e) {
            System.err.println(e.getMessage());
        }
    }

    private static void testAllDisabled() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE);

        AspectjConfig.countSolutionsEnabled = false;
        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.elitismEnabled = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.reportAlgorithmEnabled = false;
        AspectjConfig.limitIterationsActive = false;
        AspectjConfig.randomSelection = false;

        createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
    }

    private static void testCountSolutionsEnabled() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE);

        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.elitismEnabled = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.reportAlgorithmEnabled = false;
        AspectjConfig.limitIterationsActive = false;
        AspectjConfig.randomSelection = false;

        AspectjConfig.countSolutionsEnabled = true;

        createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
        log.info("iterations={} / populationSize={}", 75, 550);
        createAlgorithm(75, 550).execute();
    }

    private static void testLimitSolutionsEnabled() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE);

        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.elitismEnabled = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.reportAlgorithmEnabled = false;
        AspectjConfig.randomSelection = false;

        AspectjConfig.countSolutionsEnabled = true;
        AspectjConfig.limitIterationsActive = true;

        AspectjConfig.maxSolutions = 100;
        log.info("iterations={} / populationSize={} / maxSolutions={}", ITERATIONS,
 ↪  POPULATION_SIZE, AspectjConfig.maxSolutions);
        createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();

        AspectjConfig.maxSolutions = 150;
        log.info("iterations={} / populationSize={} / maxSolutions={}", ITERATIONS,
 ↪  POPULATION_SIZE, AspectjConfig.maxSolutions);
        createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
    }

    private static void testElitismEnabled() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE,
 ↪  AspectjConfig.maxSolutions);


        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.reportAlgorithmEnabled = false;
        AspectjConfig.randomSelection = false;
        AspectjConfig.limitIterationsActive = false;

        AspectjConfig.elitismEnabled = true;
        AspectjConfig.countSolutionsEnabled = true;

        createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
    }

    private static void testReportAndElitismEnabledWithRandomSeed() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE,
 ↪  AspectjConfig.maxSolutions);


        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.randomSelection = false;
        AspectjConfig.limitIterationsActive = false;
        AspectjConfig.countSolutionsEnabled = false;

        AspectjConfig.elitismEnabled = true;
        AspectjConfig.reportAlgorithmEnabled = true;

        for (int i = 1; i <= RANDOM_RUN; i++) {
            AlgorithmConfig.init(ThreadLocalRandom.current().nextInt() + 1);
            log.info("-------------------------------------------------------");
            log.info("random run={}", i);
            log.info("-------------------------------------------------------");
            AspectjConfig.reportFileName = "tsp-solver-elitism-random-" + i + "--";

            createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
        }
    }

    private static void testReportEnabledWithRandomSeed() throws Exception {
        log.info("iterations={} / populationSize={}", ITERATIONS, POPULATION_SIZE, AspectjConfig.maxSolutions);

        AspectjConfig.cyclicCrossover = false;
        AspectjConfig.maximalPreservativeCrossover = false;
        AspectjConfig.randomSelection = false;
        AspectjConfig.limitIterationsActive = false;
        AspectjConfig.elitismEnabled = false;
        AspectjConfig.countSolutionsEnabled = false;

        AspectjConfig.reportAlgorithmEnabled = true;

        for (int i = 1; i <= RANDOM_RUN; i++) {
            AlgorithmConfig.init(ThreadLocalRandom.current().nextInt() + 1);
            log.info("-------------------------------------------------------");
            log.info("random run={}", i);
            log.info("-------------------------------------------------------");
            AspectjConfig.reportFileName = "tsp-solver-random-" + i + "--";

            createAlgorithm(ITERATIONS, POPULATION_SIZE).execute();
        }
    }

    private static GA createAlgorithm(final int iterations,
                                      final int populationSize) throws Exception {
        Problem problem = new TSP("/ch130.tsp", 6110);
        return new GA(problem, iterations, populationSize, 0.05);
    }
}
```

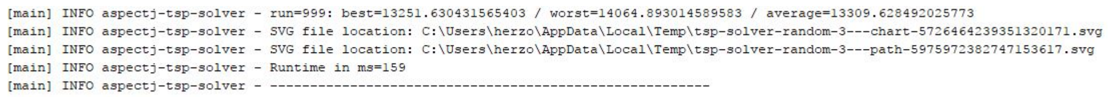
3.3 Tests

Folgender Abschnitt enthält die Tests der Aufgabenstellung in Form der generierten *logs* und der generierten SVG-Diagramme.



```
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - testAllDisabled()
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100
[main] INFO aspectj-tsp-solver - Runtime in ms=397
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver -
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - testCountSolutionsEnabled()
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100
[main] INFO aspectj-tsp-solver - Evaluation count: '100000'
[main] INFO aspectj-tsp-solver - Runtime in ms=221
[main] INFO aspectj-tsp-solver - iterations=75 / populationSize=550
[main] INFO aspectj-tsp-solver - Evaluation count: '41250'
[main] INFO aspectj-tsp-solver - Runtime in ms=81
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver -
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - testLimitSolutionsEnabled()
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100 / maxSolutions=100
[main] INFO aspectj-tsp-solver - Iteration stopped because max evaluations have been reached. evaluations='100'
[main] INFO aspectj-tsp-solver - Evaluation count: '100'
[main] INFO aspectj-tsp-solver - Runtime in ms=0
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100 / maxSolutions=150
[main] INFO aspectj-tsp-solver - Iteration stopped because max evaluations have been reached. evaluations='200'
[main] INFO aspectj-tsp-solver - Evaluation count: '200'
[main] INFO aspectj-tsp-solver - Runtime in ms=1
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver -
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - testElitismEnabled()
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100
[main] INFO aspectj-tsp-solver - Evaluation count: '100000'
[main] INFO aspectj-tsp-solver - Runtime in ms=141
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver -
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - testReportAndElitismEnabledWithRandomSeed()
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - iterations=1000 / populationSize=100
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - random run=1
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
[main] INFO aspectj-tsp-solver - run=0: best=42158.489649240124 / worst=51296.400141971484 / average=46309.736654430526
```

Abbildung 4: *TSP-Solver Logs* Teil 1



```
[main] INFO aspectj-tsp-solver - run=999: best=13251.630431565403 / worst=14064.893014589583 / average=13309.628492025773
[main] INFO aspectj-tsp-solver - SVG file location: C:\Users\herzo\AppData\Local\Temp\tsp-solver-random-3---chart-5726464239351320171.svg
[main] INFO aspectj-tsp-solver - SVG file location: C:\Users\herzo\AppData\Local\Temp\tsp-solver-random-3---path-5975972382747153617.svg
[main] INFO aspectj-tsp-solver - Runtime in ms=159
[main] INFO aspectj-tsp-solver - -------------------------------------------------------
```

Abbildung 5: *TSP-Solver Logs* Teil 2

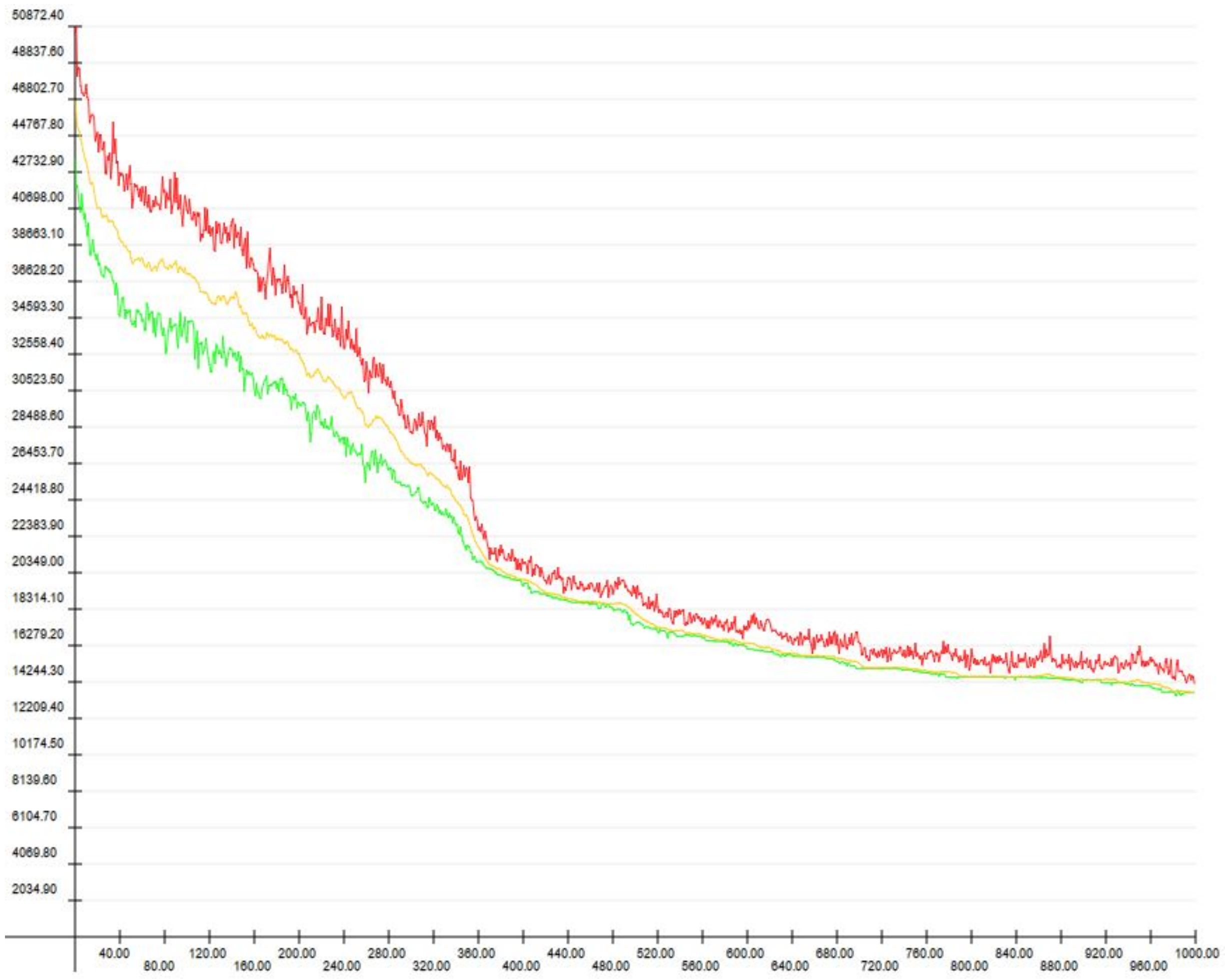


Abbildung 6: *TSP-Solver, Random Seed* erster Durchlauf

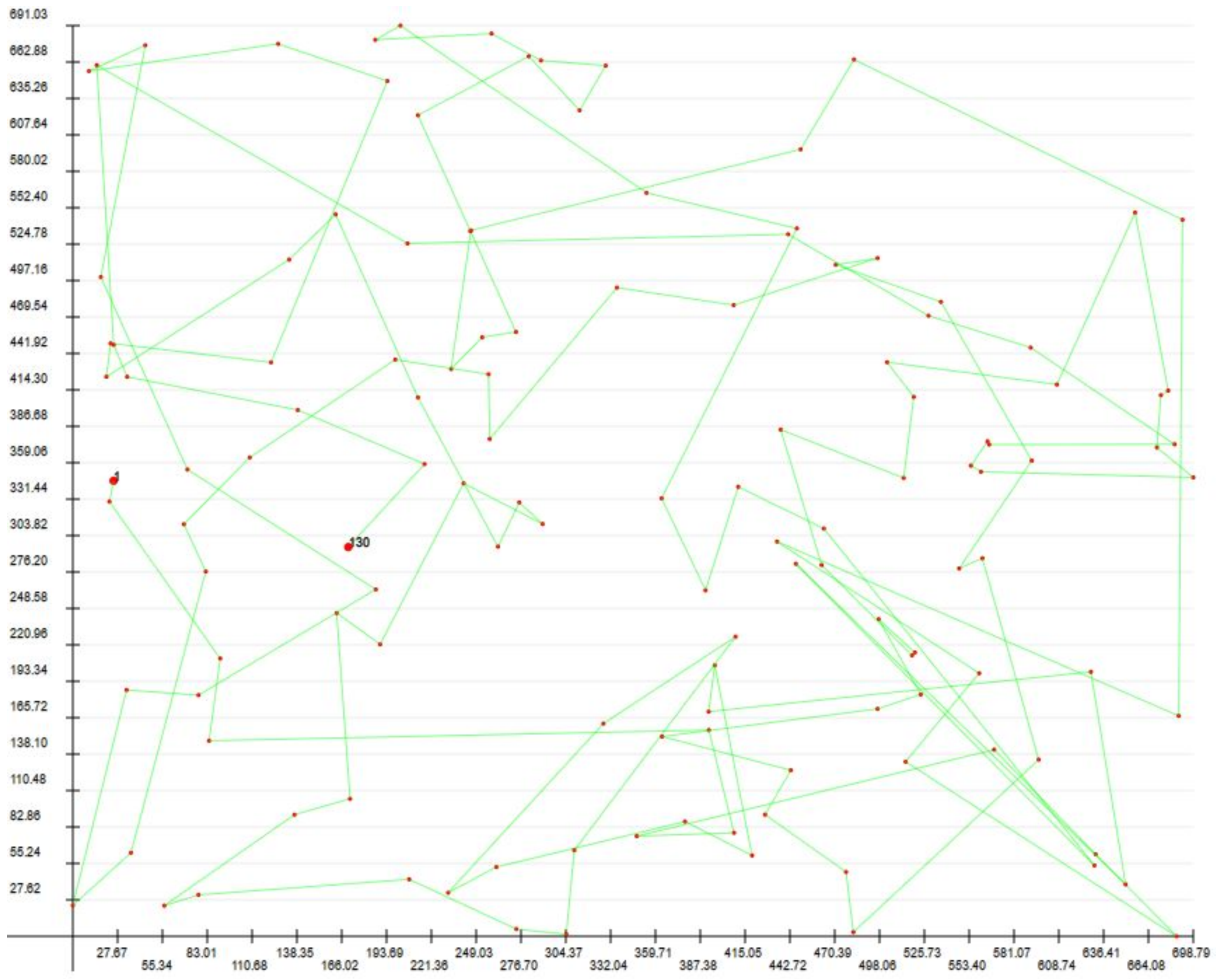


Abbildung 7: *TSP-Solver, Random Seed* erster Durchlauf *roundtrip*

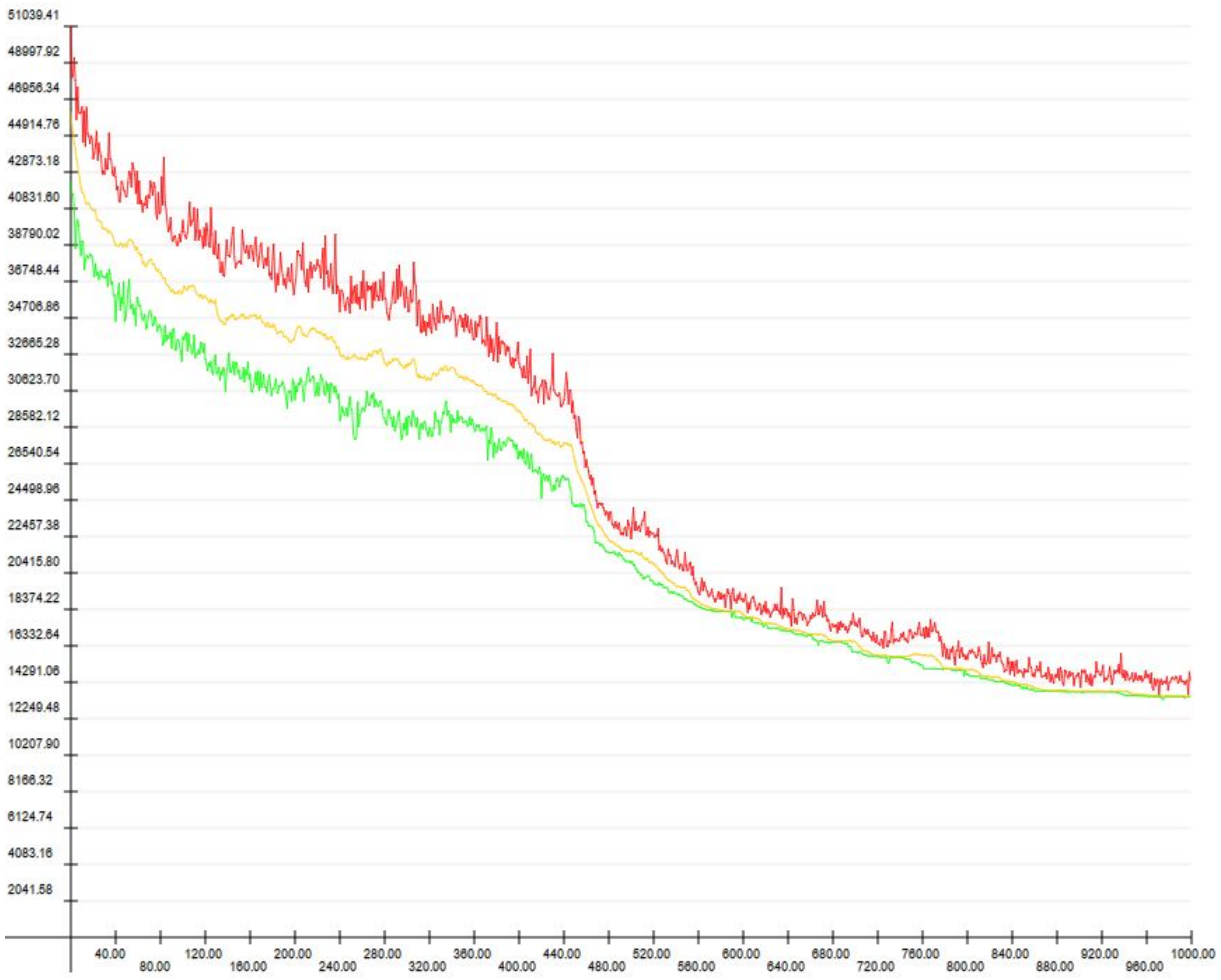


Abbildung 8: *TSP-Solver, Random Seed* zweiter Durchlauf

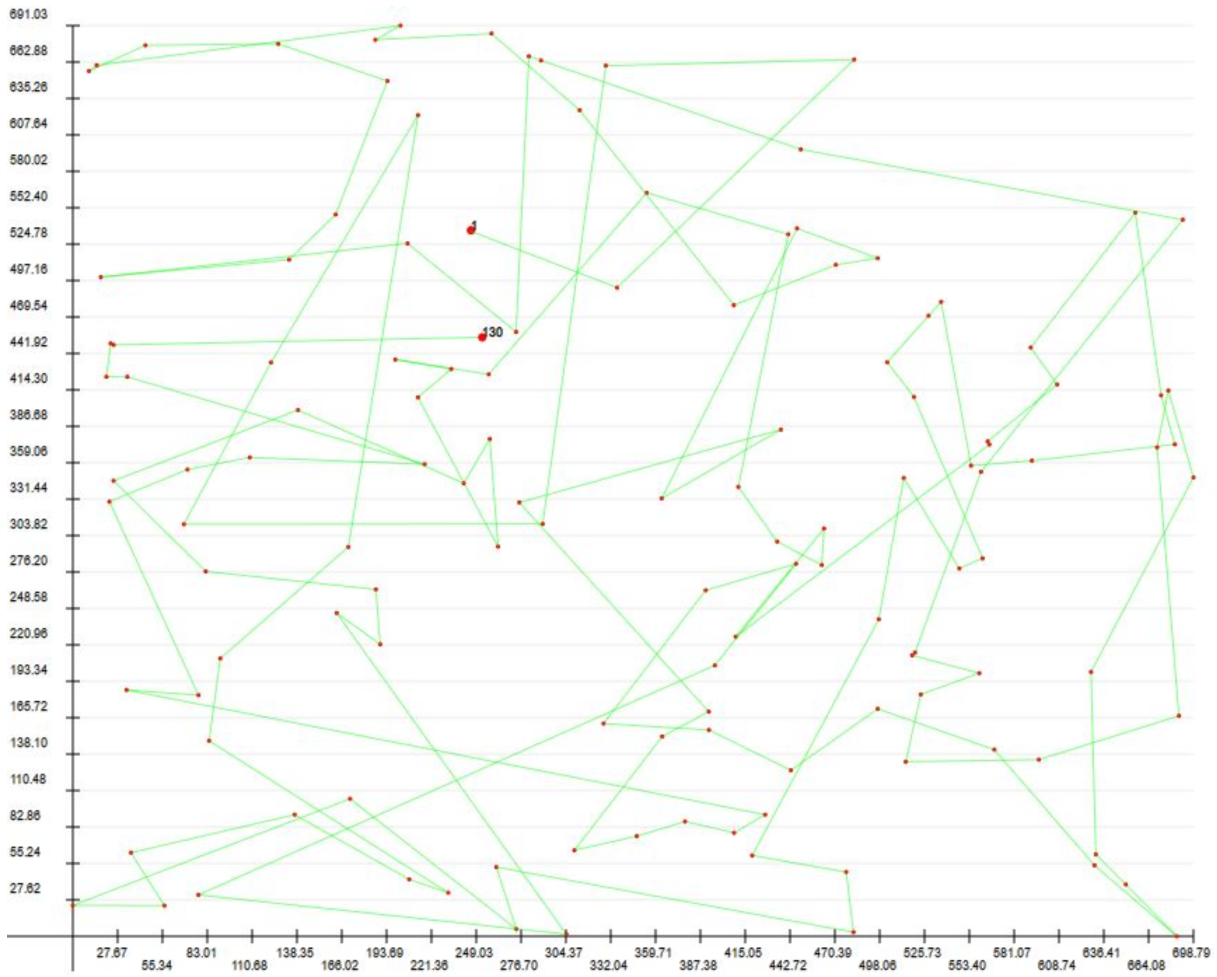


Abbildung 9: *TSP-Solver, Random Seed* zweiter Durchlauf *roundtrip*

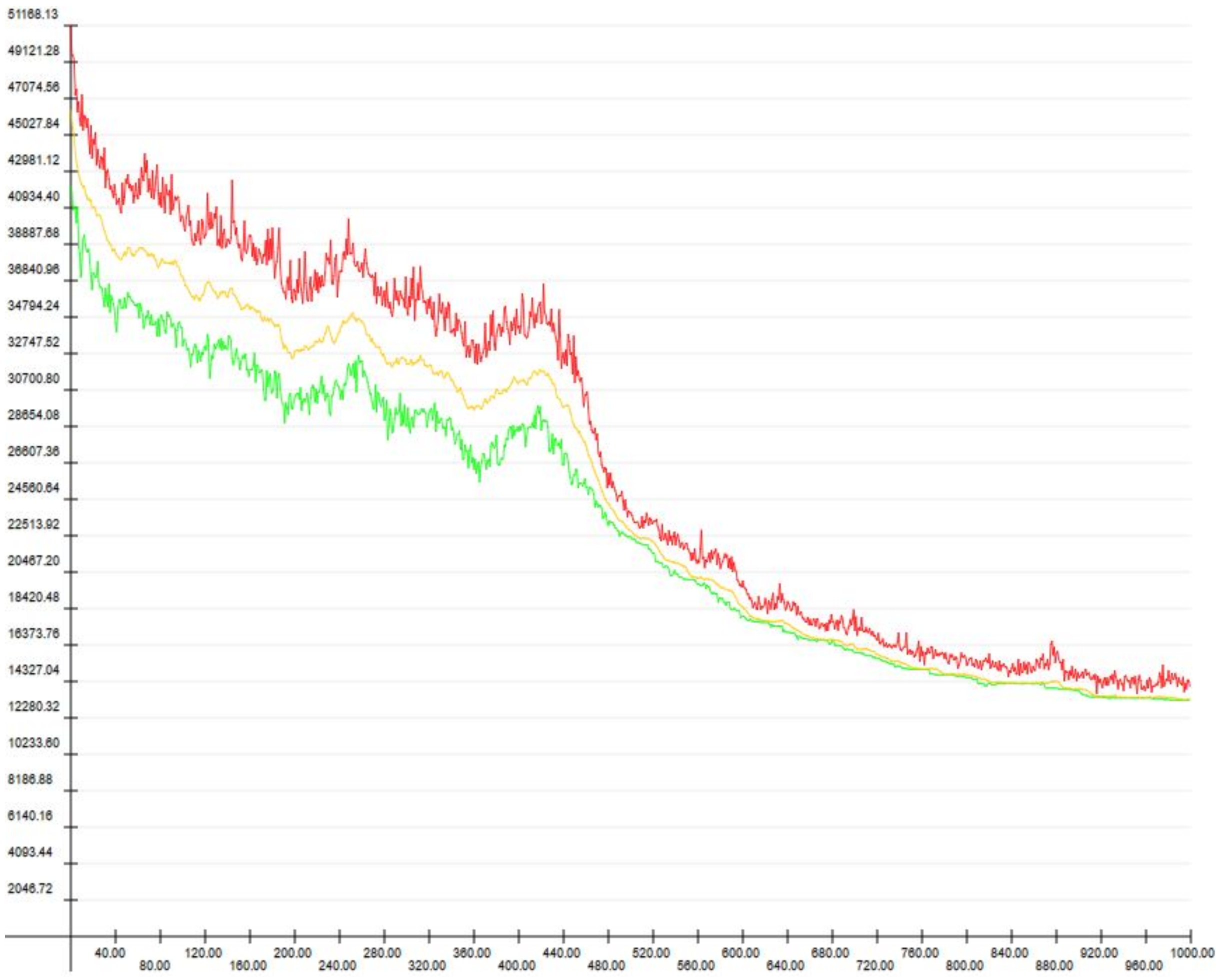


Abbildung 10: *TSP-Solver, Random Seed* dritterDurchlauf

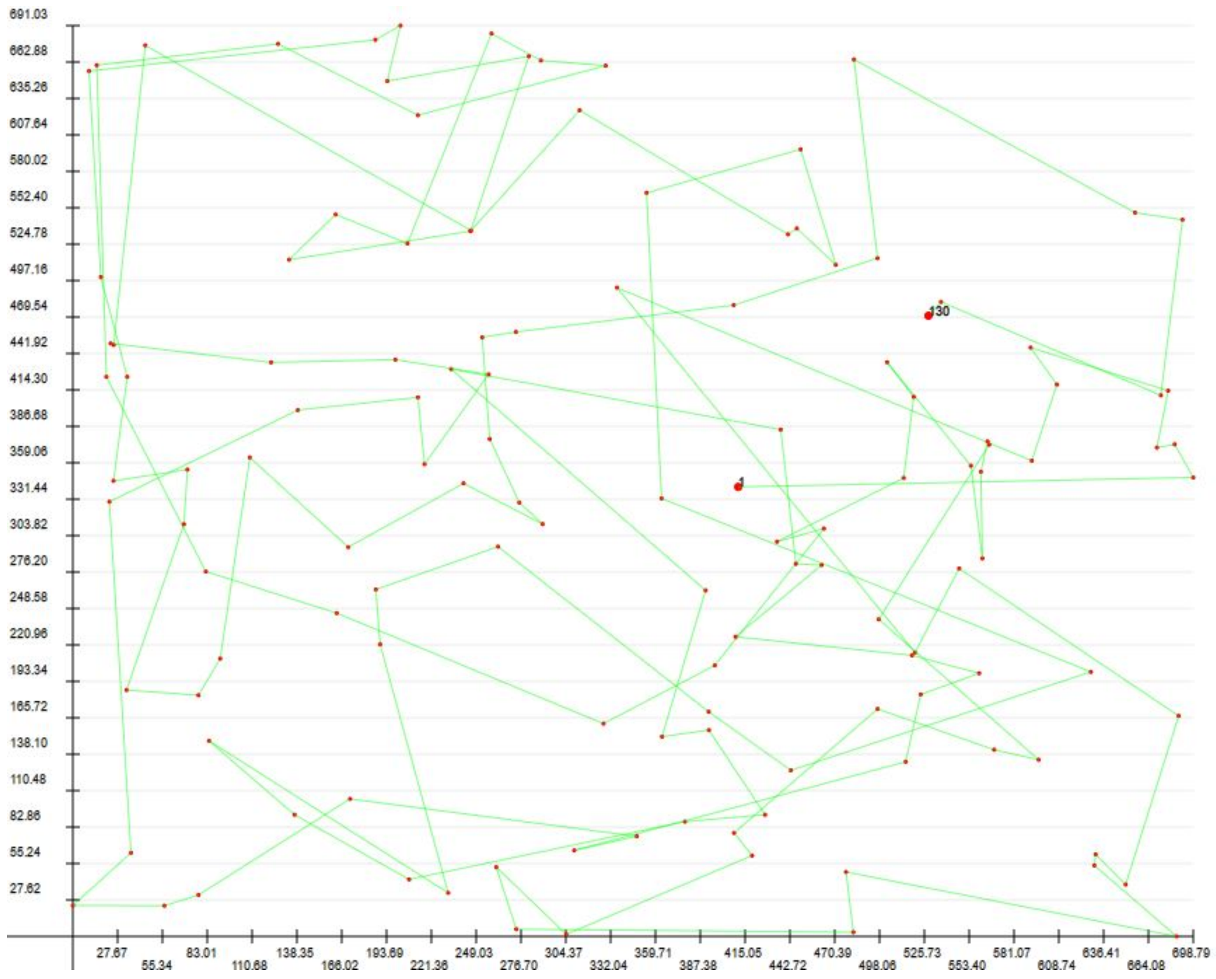


Abbildung 11: *TSP-Solver, Random Seed* dritter Durchlauf *roundtrip*

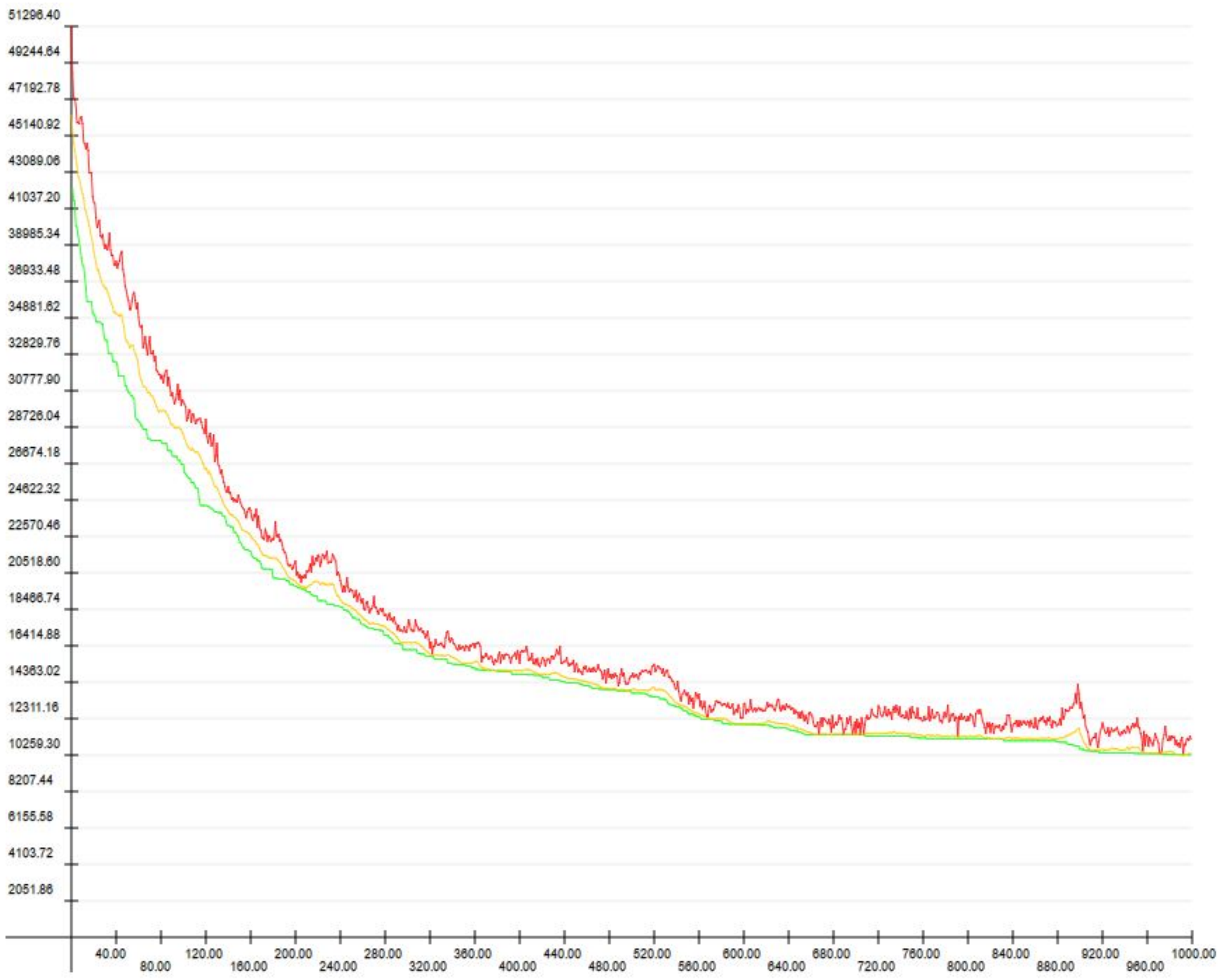


Abbildung 12: *TSP-Solver, Random Seed, Elitism* erster Durchlauf

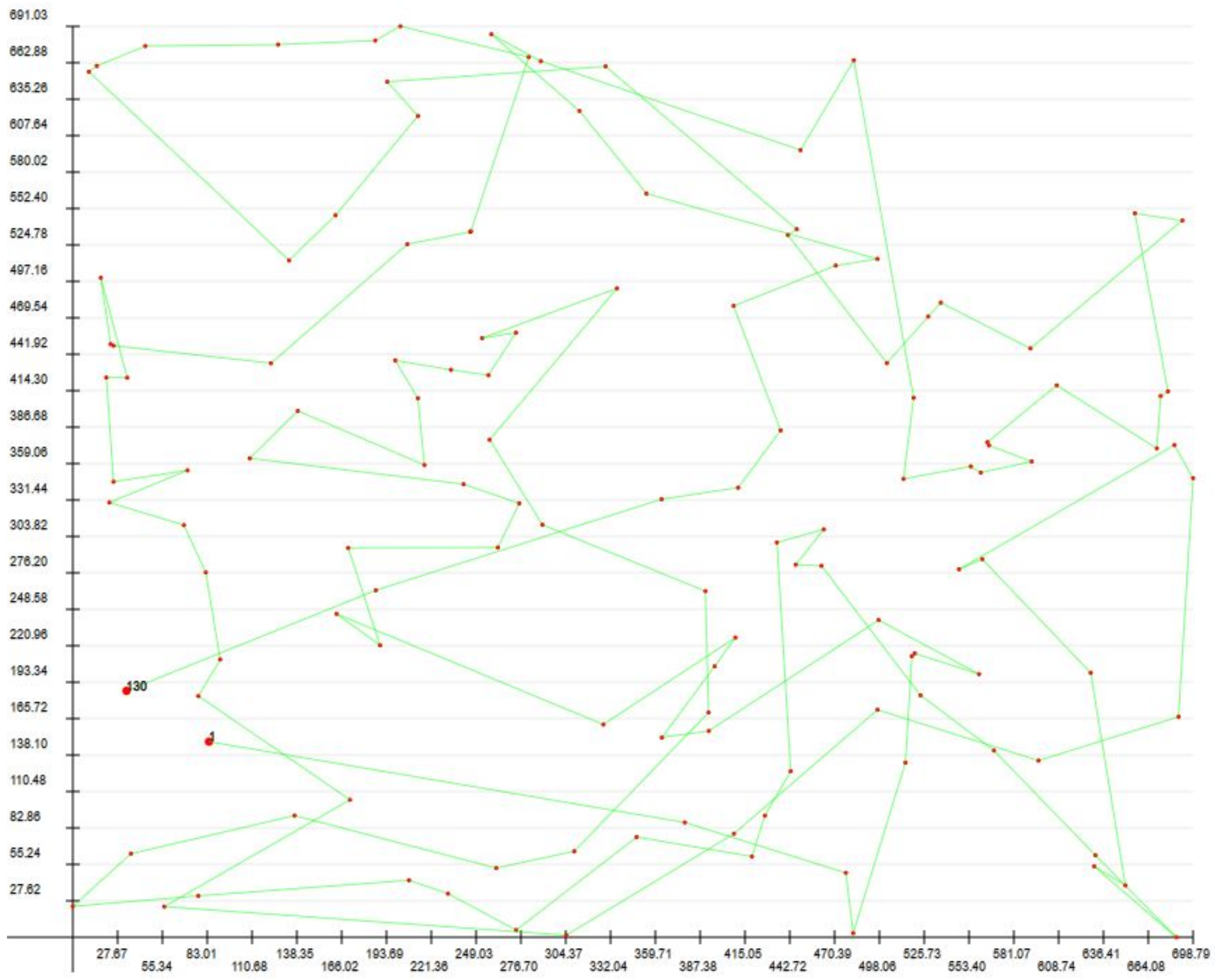


Abbildung 13: *TSP-Solver, Random Seed, Elitism* erster Durchlauf *roundtrip*

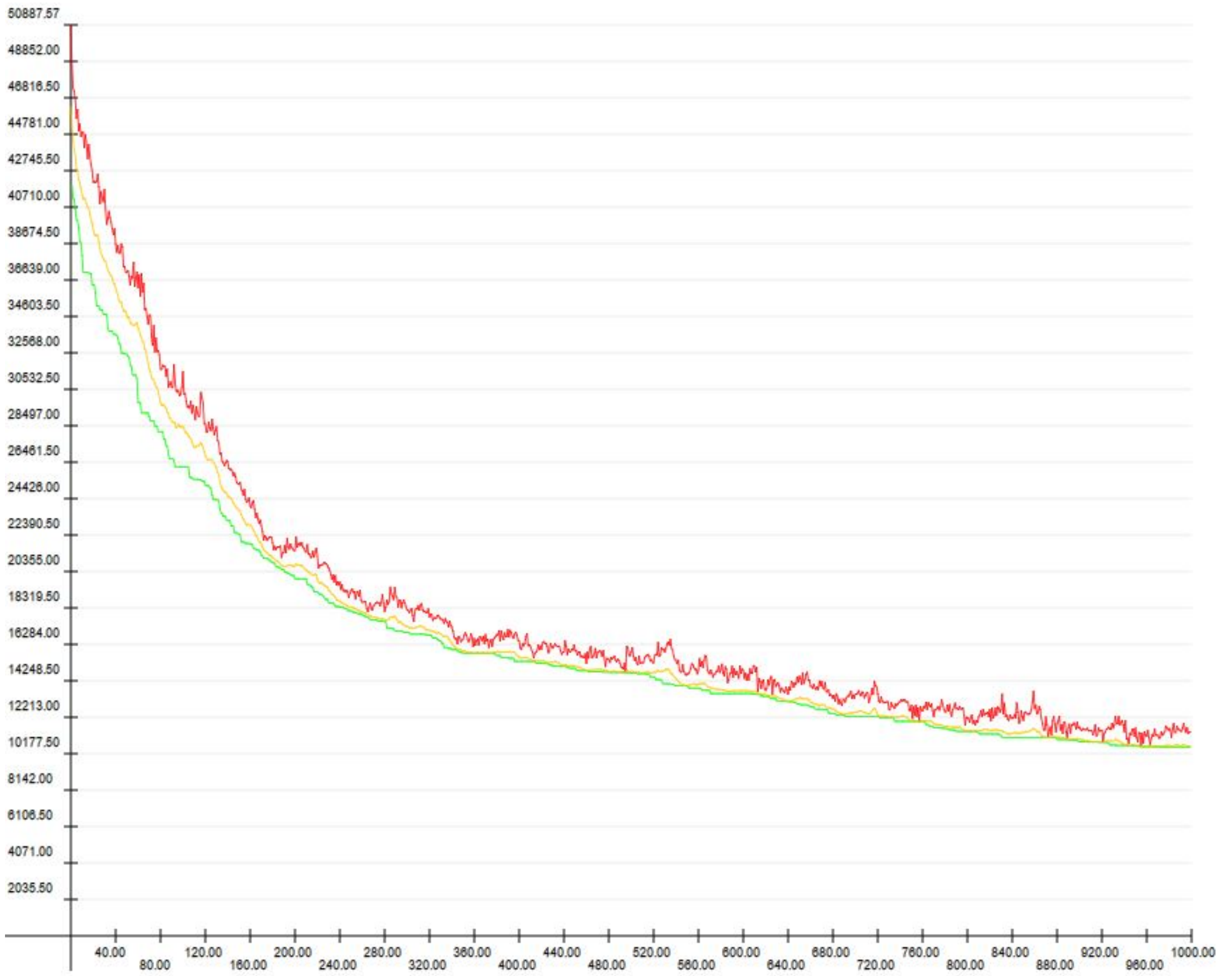


Abbildung 14: *TSP-Solver, Random Seed, Elitism* zweiter Durchlauf

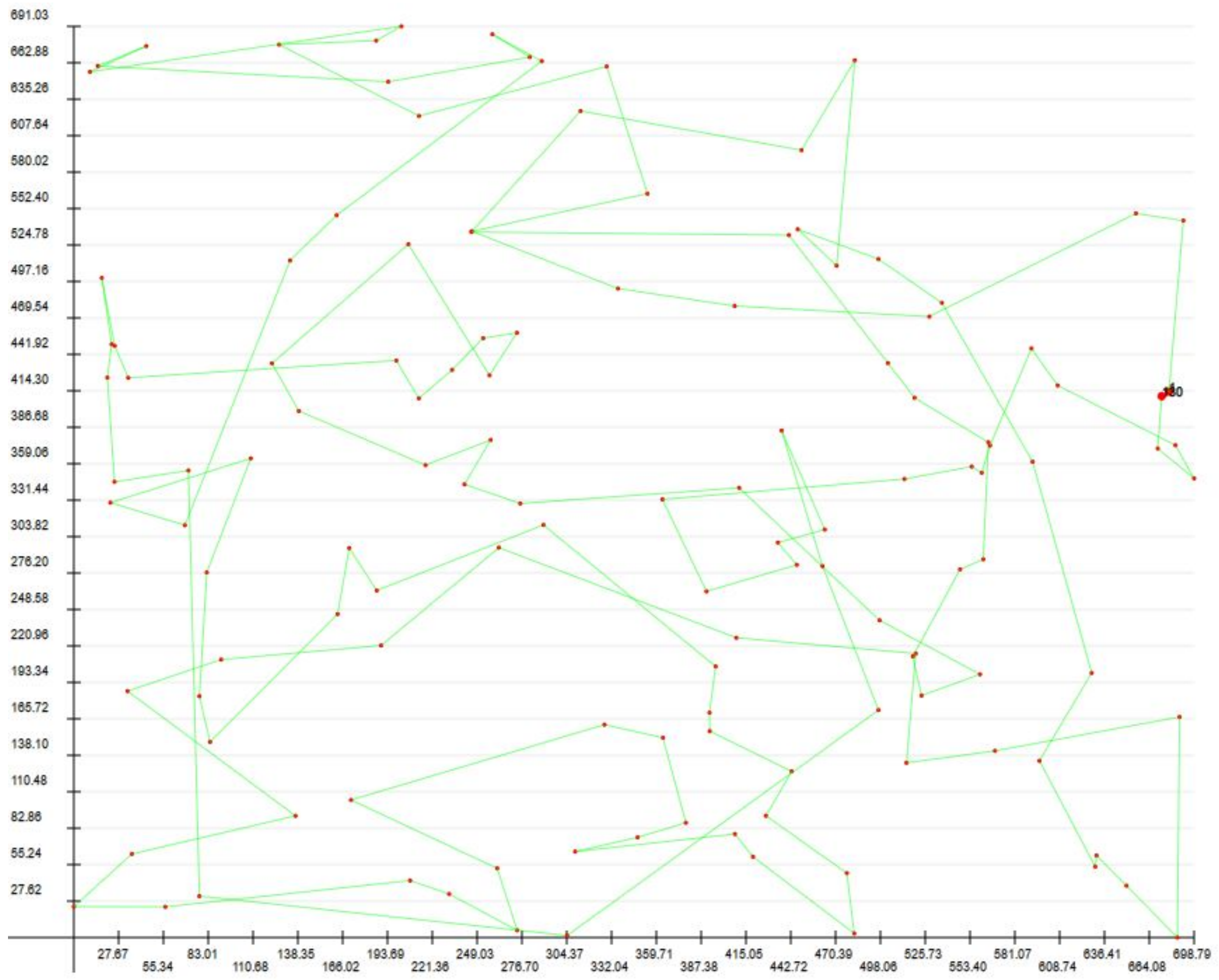


Abbildung 15: *TSP-Solver, Random Seed, Elitism* zweiter Durchlauf *roundtrip*

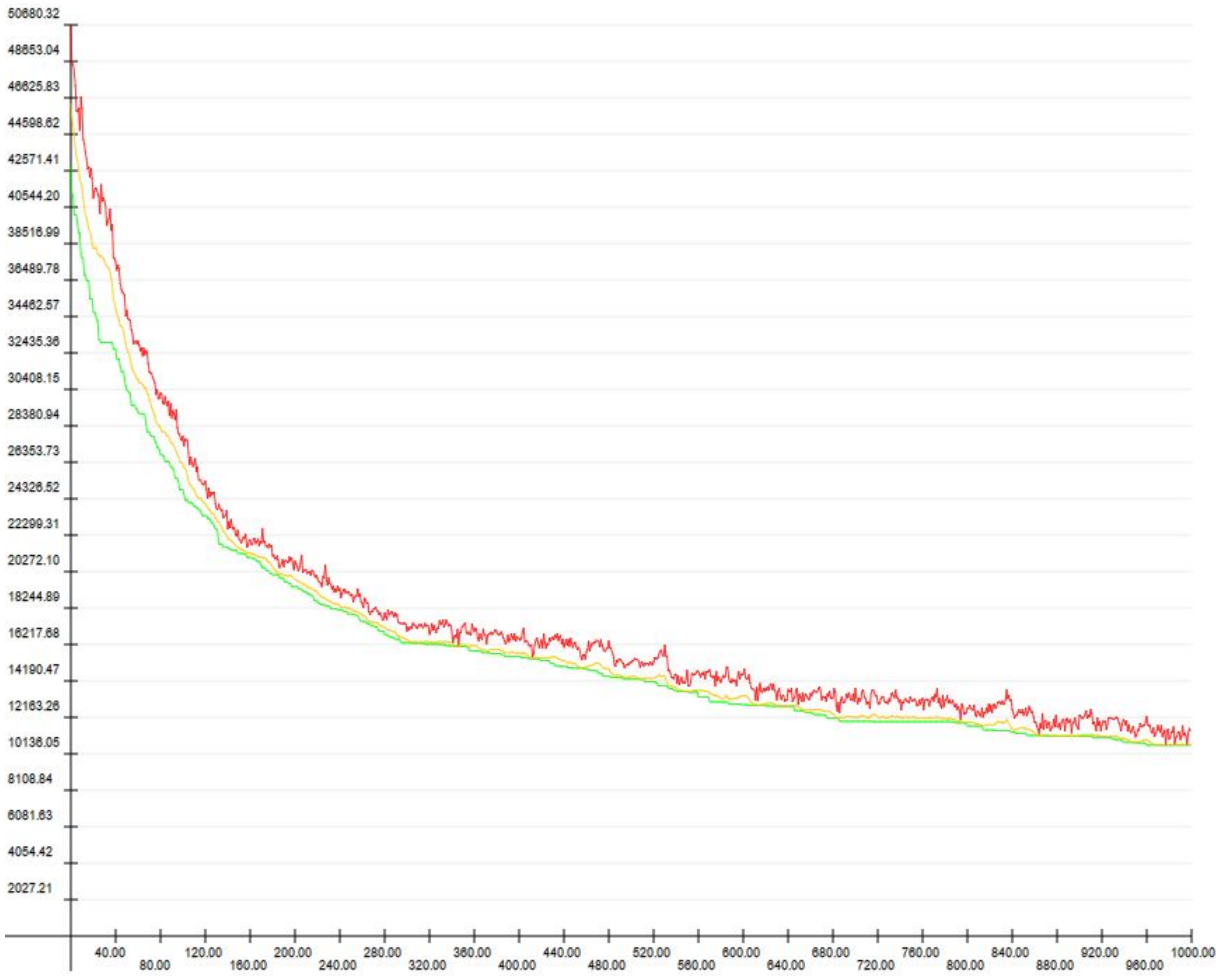


Abbildung 16: *TSP-Solver, Random Seed, Elitism* dritter Durchlauf

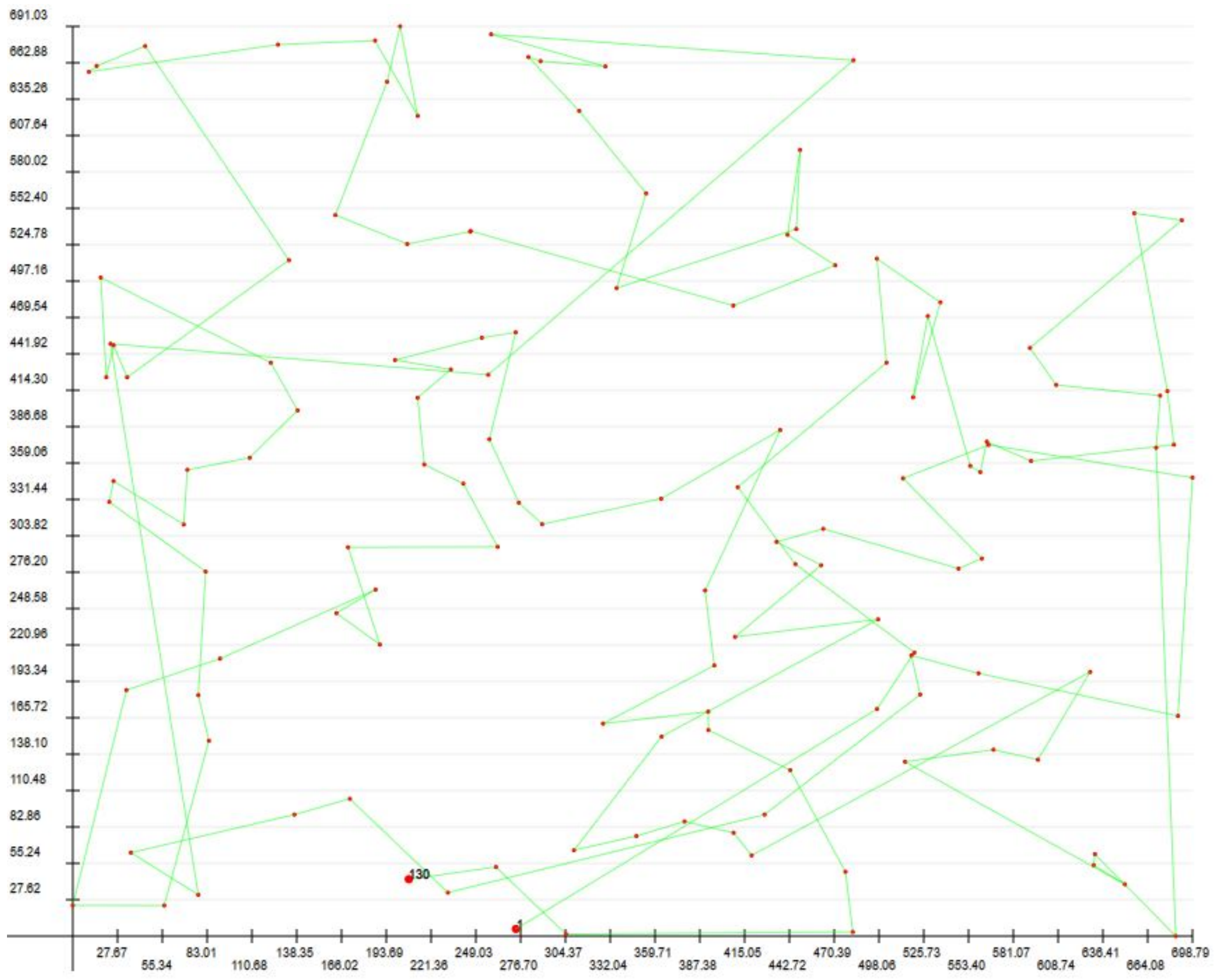


Abbildung 17: *TSP-Solver, Random Seed, Elitism* dritter Durchlauf *roundtrip*