SPW 4	Test Automation & Continuous Delivery	ST 22, Exercise 1

Deadline: 30.05.2022 14:40

Name Angelos Angelis

Points _____ **Effort in hours** 6

1. TSP Solver (5 + 5 + 5 + 5 + 4 Points)

The *Traveling Salesperson Problem (TSP)* is a combinatorial optimization problem, where you have to find the shortest roundtrip to visit *n* locations (cities) exactly once and then return back to the starting point. It belongs to the class of NP-hard problems, which means that all known algorithms for solving the TSP have exponential asymptotic runtime complexity. The following Figure 1 shows the TSP problem instance *ch130* and its optimal solution from the TSPLib[1] benchmark library.

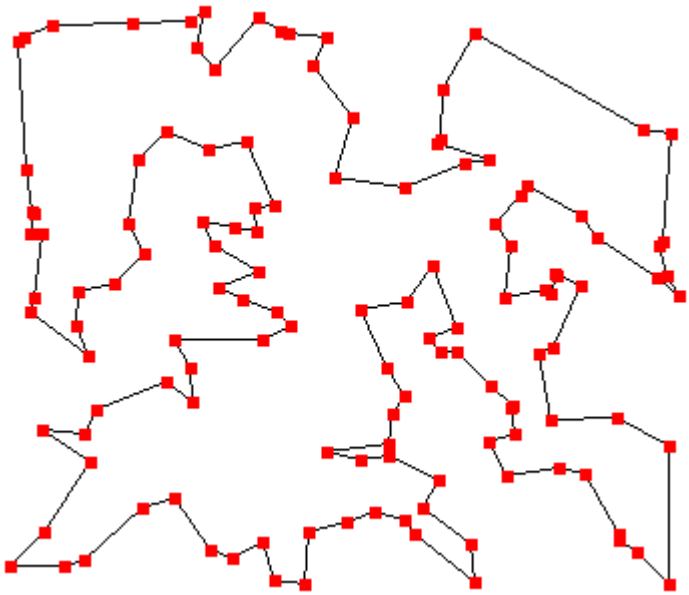


Figure 1: TSP instance *ch130* and its optimal solution

One approach to deal with hard optimization problems as the TSP is to use metaheuristics. Metaheuristic algorithms, such as Evolutionary Algorithms, Simulated Annealing, or Tabu Search, are generic solvers (often inspired by nature) which try to improve a solution iteratively. Thereby they cannot guarantee that a global optimal solution is ever found, but they often can find sufficiently good solutions in acceptable time.

Evolutionary Algorithms are inspired by the natural evolution of species and try to improve a population of solutions in an artificial evolutionary process. *Genetic Algorithms (GAs)* are one kind of Evolutionary Algorithms which start with a population of randomly generated solutions. In each iteration children are created by selecting parent solutions of above-average quality (selection), combining these two parent solutions in order to cross a new child (crossover), mutating this child with a small probability (mutation), and replacing the old generation with the new one (replacement). Figure 2 shows this basic cycle of each Genetic Algorithm. In this way the quality of the solutions in the population is increased over time and the algorithm usually terminates after a given number of generations.

[1] http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95

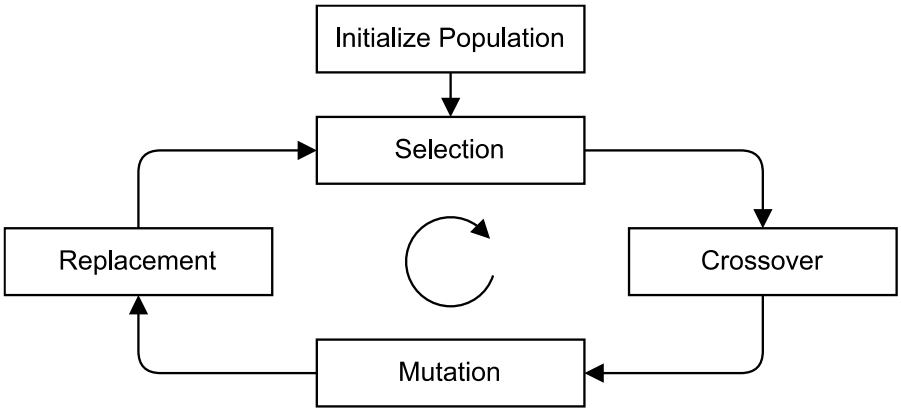


Figure 2: Basic cycle of a Genetic Algorithm

To encode a valid roundtrip in each individual of the population, an array can be used to store a permutation. This form of solution encoding is called path encoding, where each number in the permutation represents a city. An example for a small TSP of 6 cities is shown in Figure 3. For crossing and mutating permutations specific operators are required, which assure that only valid permutations are created in order to avoid invalid individuals (e.g., OrderCrossover, CyclicCrossover, InversionMutator, etc.).

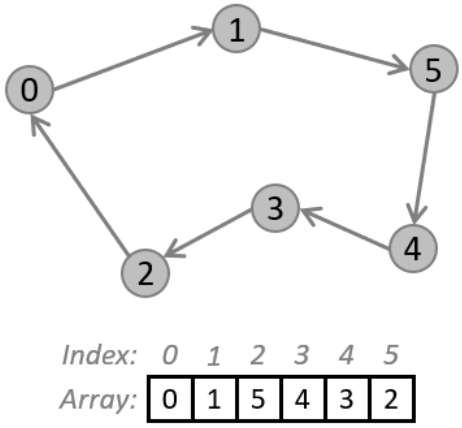


Figure 3: Path encoding of a TSP solution

In Moodle you find the source code of a small class library for solving TSPs with GAs. A class diagram of this library is shown in Figure 4.

This class library does not provide any unit tests yet, which has to be changed immediately. Study the provided code carefully and implement unit tests for the following classes (tasks (a) to (d) which are worth 5 points per task):

- (a) Permutation
- (b) Solution
- (c) RandomSelector
- (d) InversionMutator

Your goal is to achieve 100% branch coverage of these classes, which you should also evaluate and document (task (e) which is worth 4 points).

As discussed in the lecture, remember that test code is first-class code. Be careful and implement clean and readable tests.

Figure 4: Class diagram of the TSPSolver class library

Concluding Remark:

If you like Evolutionary Algorithms and metaheuristics in general, have a look at the work of our research group *Heuristic and Evolutionary Algorithms Laboratory (HEAL)*[2] of the Software Engineering Department and at our open-source optimization environment *HeuristicLab*[3]. :-)

[2] https://heal.heuristiclab.com
[3] https://dev.heuristiclab.com

Quellcode:

PermutationTests:

```
package spw4.tsp;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.NullAndEmptySource;
import org.junit.jupiter.params.provider.ValueSource;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.IntStream;
import java.util.stream.Stream;

import static org.assertj.core.api.AssertionsForClassTypes.assertThat;
import static org.junit.jupiter.api.Assertions.*;

public class PermutationTests {
    @DisplayName("Permutation.ctor when lenght > 1 returns valid
permutation")
    @ParameterizedTest(name = "length = {0}")
    @ValueSource(ints = {2,3,7})
    void ctorWhenLengthIsValidReturnsPermutation(int length){
        int[] expected = IntStream.range(0,length).toArray();
        int[] actual = new Permutation(length).getValues();
        assertThat(actual).containsExactlyInAnyOrder(expected);
    }

    @DisplayName("Permutation.ctor when length < 2 throws
IllegalArgumentException")
    @ParameterizedTest(name = "length = {0}")
    @ValueSource(ints = {-5,0,1})
    void ctorWhenLengthIsInvalidThrowsException(int length){
        assertThrows(IllegalArgumentException.class, () -> {
            new Permutation(length);
        });
    }

    @DisplayName("Permutation.ctor when at least 2 permutation values are
valid returns permutation")
    @ParameterizedTest(name = "values = {0}")
    @MethodSource("generateValidPermutationValues")
    void ctorWhenValuesAreValidReturnsPermutation(int[] values){
        int[] result = new Permutation(values).getValues();
        assertThat(result).containsExactly(values);
    }

    static Stream<Arguments> generateValidPermutationValues(){
        return Stream.of(
                Arguments.of(new int [] {0,1}),
```

```
                Arguments.of(new int [] {0,1}),
                Arguments.of(new int [] {0,1}),
                Arguments.of(new int [] {0,1})
        );
    }

    @DisplayName("Permutation.ctor when permutation values are invalid throws
IllegalArgumentException")
    @ParameterizedTest(name = "invalid values = {0}")
    @NullAndEmptySource
    @MethodSource("generateInvalidPermutationValues")
    void ctorWhenValuesAreInvalidThrowsException(int[] values) {
        assertThrows(IllegalArgumentException.class, () -> {
            new Permutation(values);
        });
    }
    static Stream<Arguments> generateInvalidPermutationValues() {
        return Stream.of(
                Arguments.of(new int[] { 0 }),
                Arguments.of(new int[] { 0, 0 }),
                Arguments.of(new int[] { 0, 1, -1 }),
                Arguments.of(new int[] { 7, 2, 3 })
        );
    }

    @DisplayName("Permutation.getValues returns a clone of the permutation")
    @Test
    void getValuesReturnsClone() {
        int[] values = IntStream.range(0, 50).toArray();
        int[] result = new Permutation(values).getValues();
        assertNotSame(values, result);
    }

    @DisplayName("Permutation.createRandom when length > 1 returns valid
random permutation")
    @Test
    void createRandomWhenLengthIsValidReturnsRandomPermutation() {
        TSPSolver.random = new Random(919);
        int[] result = Permutation.createRandom(5).getValues();
        assertEquals(5, Arrays.stream(result).count());
    }

    @DisplayName("Permutation.createRandom when length < 2 throws
IllegalArgumentException")
    @ParameterizedTest(name = "invalid values = {0}")
    @ValueSource(ints = {1, 0, -1})
    void createRandomWhenLengthIsInvalidThrowsIllegalArgumentException(int
length) {
        assertThrows(IllegalArgumentException.class, () -> {
            Permutation.createRandom(length);
        });
    }

    @DisplayName("Permutation.toString returns correct String")
    @ParameterizedTest(name = "array range = {0}")
    @ValueSource(ints = {2, 4, 6})
    void toStringReturnsCorrectString(int length) {
```

```
        int[] expected = IntStream.range(0, length).toArray();
        Permutation permutation = new Permutation(length);
        String result = permutation.toString();
        assertEquals(Arrays.toString(expected), result);
    }
}
```

SolutionTests:

```
package spw4.tsp;

import org.hamcrest.CoreMatchers;
import org.junit.jupiter.api.*;
import org.junit.jupiter.params.*;
import org.junit.jupiter.params.provider.*;

import java.util.stream.*;

import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.containsString;
import static org.junit.jupiter.api.Assertions.*;

public class SolutionTests {
    @DisplayName("Solution.ctor when solutionData null throws Illegal
Argument Exception")
    @Test
    void ctorWhenSolutionDataNullThrowsIllegalArgumentException() {
        assertThrows(IllegalArgumentException.class, () -> {
            new Solution<>(null, 0);
        });
    }

    @DisplayName("Solution.ctor when solutionData is valid")
    @ParameterizedTest(name = "solutionData = '{0}'")
    @ValueSource(strings = {"123", "333", "1", "09", ""})
    void ctorWhenSolutionDataIsValid(String solutionData) {
        String expected = solutionData;
        double quality = 0;
        Solution<String> solution = new Solution<>(solutionData, quality);
        assertEquals(expected, solution.getSolutionData());
    }

    @DisplayName("Solution.compareTo different qualities")
    @ParameterizedTest(name = "quality1 = {0}, quality2 = {1}, expected =
{2}")
    @MethodSource("compareToReturnsFalseWhenDifferentQualities")
    void compareToReturnsFalseWhenDifferentQualities(double quality1, double
quality2, int expected) {
        Object solutionData = new Object();
        Solution<Object> solution1 = new Solution<>(solutionData, quality1);
        Solution<Object> solution2 = new Solution<>(solutionData, quality2);
        int result = solution1.compareTo(solution2);
        assertEquals(expected, result);
    }

    private static Stream<Arguments>
```

```
compareToReturnsFalseWhenDifferentQualities() {
        return Stream.of(
                Arguments.of(0, 11, -1),
                Arguments.of(11, 0, 1),
                Arguments.of(-11, 11, -1),
                Arguments.of(11, -11, 1),
                Arguments.of(0.1, 0.11, -1),
                Arguments.of(0.11, 0.1, 1)
        );
    }

    @DisplayName("Solution.toString returns correct format")
    @ParameterizedTest(name = "quality = {0}, solutionData = {1}")
    @MethodSource("toStringReturnsCorrectFormat")
    void toStringReturnsCorrectFormat(double quality, String qualityString,
String solutionData) {
        Solution<String> solution = new Solution(solutionData, quality);
        String result = solution.toString();
        assertThat(result, CoreMatchers.allOf(
                containsString(qualityString),
                containsString(solutionData.toString())));
    }

    private static Stream<Arguments> toStringReturnsCorrectFormat() {
        return Stream.of(
                Arguments.of(0.1, "0,12", "Testing"),
                Arguments.of(1, "12", "Testö"),
                Arguments.of(-10, "-102", "Testä"),
                Arguments.of(0, "0", "Test"),
                Arguments.of(0, "0", "\n")
        );
    }
}
```

RandomSelectorTests:

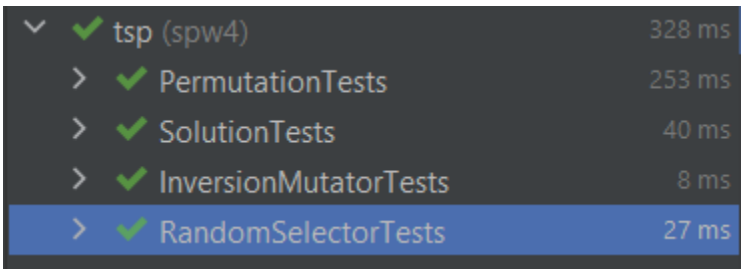
```
package spw4.tsp;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;
import org.junit.jupiter.params.provider.NullAndEmptySource;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.Stream;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.*;

public class RandomSelectorTests {

    @DisplayName("RandomSelector.select when solutions null or empty, throws
```

```
Illegal Argument Exception")
    @ParameterizedTest(name = "solutions = {0}")
    @NullAndEmptySource
    void
selectWhenSolutionsNullOrEmptyThrowsIllegalArgumentException(List<Solution<In
teger>> solutions) {
        RandomSelector<Integer> selector = new RandomSelector<>();
        TSPSolver.random = new Random();
        assertThrows(IllegalArgumentException.class, () -> {
            selector.select(solutions, 0);
        });
    }

    @DisplayName("RandomSelector.select when parents smaller 0, throws
Illegal Argument Exception")
    @Test()
    void selectWhenParentsSmaller0ThrowsIllegalArgumentException() {
        RandomSelector<Integer> selector = new RandomSelector<>();
        TSPSolver.random = new Random();
        assertThrows(IllegalArgumentException.class, () -> {
            selector.select(List.of(new Solution<>(1, 1)), -1);
        });
    }


    @DisplayName("RandomSelector.select returns correct number of selected
solutions")
    @ParameterizedTest(name = "solutions = {0}, parents = {1}")
    @MethodSource("selectReturnsCorrectNumberOfSelectedSolutions")
    void
selectReturnsCorrectNumberOfSelectedSolutions(List<Solution<Integer>>
solutions, int parents) {
        RandomSelector<Integer> selector = new RandomSelector<>();
        TSPSolver.random = new Random();
        List<Solution<Integer>> result = selector.select(solutions, parents);
        assertEquals(parents, result.size());
    }

    private static Stream<Arguments>
selectReturnsCorrectNumberOfSelectedSolutions() {
        List<Solution<Integer>> solutions = List.of(
                new Solution<>(1, 1),
                new Solution<>(11, 0),
                new Solution<>(111, 1),
                new Solution<>(1111, 0),
                new Solution<>(11111, 1)
        );
        return Stream.of(
                Arguments.of(solutions, 1),
                Arguments.of(solutions, 2),
                Arguments.of(solutions, 0),
                Arguments.of(solutions, 5),
                Arguments.of(solutions, 20),
                Arguments.of(solutions, 100)
        );
    }
}
```

InversionMutatorTests:

```
package spw4.tsp;

import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.Arguments;
import org.junit.jupiter.params.provider.MethodSource;

import java.util.Arrays;
import java.util.List;
import java.util.Random;
import java.util.stream.Stream;

import static org.assertj.core.api.Assertions.assertThat;
import static org.junit.jupiter.api.Assertions.assertDoesNotThrow;
import static org.junit.jupiter.api.Assertions.assertThrows;

public class InversionMutatorTests {

    @DisplayName("InversionMutator.mutate when solution null throws Illegal
Argmuent Exception")
    @Test
    void mutateWhenSolutionNullThrowsIllegalArgumentException() {
        InversionMutator mutator = new InversionMutator();
        assertThrows(IllegalArgumentException.class, () -> {
            mutator.mutate(null);
        });
    }

    @DisplayName("InversionMutator.mutate when permutation small does not
throw")
    @Test
    void mutateWhenPermutationSmallDoesNotThrow() {
        TSPSolver.random = new Random();
        int[] data = new int[]{1, 0};
        Permutation permutation = new Permutation(data);
        InversionMutator mutator = new InversionMutator();
        assertDoesNotThrow(() -> mutator.mutate(permutation));
    }


}
```

Results:



Element	Class, %	Method, %	Line, %
Algorithm	100% (0/0)	100% (0/0)	100% (0/0)
Crossover	100% (0/0)	100% (0/0)	100% (0/0)
CyclicCrossover	0% (0/1)	0% (0/1)	0% (0/19)
GA	0% (0/1)	0% (0/11)	0% (0/52)
InversionMutator	100% (1/1)	100% (1/1)	100% (9/9)
Mutator	100% (0/0)	100% (0/0)	100% (0/0)
OrderCrossover	0% (0/1)	0% (0/1)	0% (0/20)
Permutation	100% (1/1)	100% (6/6)	100% (24/24)
Problem	100% (0/0)	100% (0/0)	100% (0/0)
RandomSelector	100% (1/1)	100% (1/1)	100% (7/7)
Selector	100% (0/0)	100% (0/0)	100% (0/0)
Solution	100% (1/1)	100% (5/5)	92% (12/13)
TournamentSelector	0% (0/1)	0% (0/1)	0% (0/10)
TSP	0% (0/1)	0% (0/6)	0% (0/27)
TSPLibFormatException	0% (0/1)	0% (0/1)	0% (0/1)
TSPLibParser	0% (0/1)	0% (0/12)	0% (0/123)
TSPSolver	0% (0/1)	0% (0/2)	0% (0/18)