Aufgabe 1: Weniger krumme Touren

Teilnahme-ID: 65433

Bearbeiter/-in dieser Aufgabe:
Paul Franosch

17. April 2023

Inhaltsverzeichnis

1 Lösungsidee

Das Problem der Aufgabenstellung beschreibt eine Variante des NP-vollständigen Traveling-Salesman Problems. Es ist somit nicht möglich, optimale Lösungen in polynomieller Zeit für beliebige Graphen zu finden. Für die Graphen aus den Beispieldateien ist dies aufgrund ihrer Größe auch nicht möglich. Folglich müssen zur Bewältigung der Aufgabe Heuristiken herangezogen werden.
Zunächst wird eine initiale Lösung mittels der Nearest-Neighbour Heuristik ermittelt. Diese wird danach iterativ durch die TwoOpt-Postoptimierung [1] verbessert.

1.1 Nearest-Neighbour Heuristik

Die Eröffnungsheuristik ist vergleichsweise einfach. Es wird ausgehend von einem Startknoten der Knoten gewählt, der die geringste Distanz zum aktuellen Knoten hat und noch nicht besucht wurde. Anschließend wird der aktuelle Knoten zum Startknoten. Dieses Verfahren wird so lange wiederholt, bis alle Knoten besucht wurden. Es genauer formuliert:
1.
Ausgehend vom Startknoten, wird der Knoten v mit der geringsten Distanz gewählt. Der Startknoten

[1] Vergleiche https://wwwmayr.in.tum.de/lehre/2011SS/optprak/uebung/tutorial5.pdf

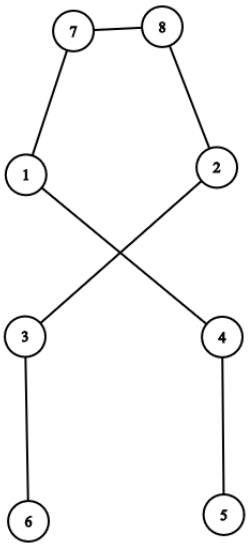
und v werden als besucht markiert. Die Kante e zwischen diesen beiden Knoten wird für die Suche des nächsten Knotens genutzt.

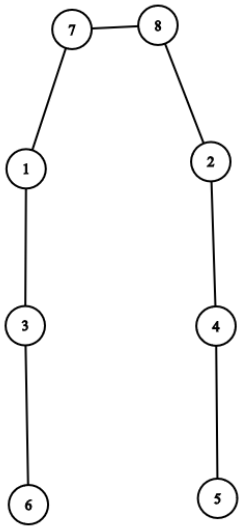
2.

Solange es noch unbesuchte Knoten gibt, wird folgendes Verfahren wiederholt: Betrachte in aufsteigender Distanz zu v alle Knoten die noch nicht besucht wurden. Wenn die Kante, die den aktuell betrachteten Knoten und v verbindet, e derart schneidet, dass die Winkelbedingung erfüllt ist, dann wähle diesen Knoten als nächsten v. Markiere v als besucht.
Sollte kein solcher Knoten ermittelt werden können, gehe einen Schritt zurück und wähle den nächstbesten Knoten (Backtracking).

1.2 TwoOpt Postoptimierung

Zugrundelegende Idee der Optimierung ist die Dreiecksungleichung. Diese besagt, dass eine Dreiecksseite höchstens so lang wie die Summe der beiden anderen Seiten ist. Höchstens ist dabei der Spezialfall, dass es sich um eine Gerade handelt. Konkret lässt sich daraus folgern, dass ein direkter Weg immer kürzer ist als über einen Umweg. Diese Eigenschaft wird sich in der Aufgabe zunutze gemacht, indem versucht wird die Reihenfolge der besuchten Knoten eines bereits gegeben Pfades derart zu verändern, dass Überkreuzungen reduziert werden. Die linke Graphik zeigt eine überkreuzte Tour. Die Kante 1–4 schneidet





die Kante 3–2 im Punkt P. In der linken Abbildung werden je zwei Seiten der Dreiecke $\triangle 3P1$ und $\triangle 42P$ passiert. Dies kann unter Berücksichtigung der Dreiecksungleichung optimiert werden, indem jeweils nur eine der Seiten traversiert wird. Die rechte Graphik zeigt wie die Tour dafür verändert werden muss. Die ursprüngliche Reihenfolge 6–3–2–8–7–1–4–5 wird zu 6–3–1–7–8–2–4–5. Durch Beibehalten der Knoten und Umkehrung der Reihenfolge einer Teilroute, im Beispiel bleibt 6–3 und 4–5 erhalten und 2–8–7–1 wird zu 1–7–8–2, kann die Strecke der Gesamttour reduziert werden.
Idealerweise werden alle Überkreuzungen innerhalb der Tour durch geschicktes Vertauschen entfernt. Dieser Ansatz findet allerdings nur ein lokales Minimum bezüglich der Strecke der Gesamttour. Möglicherweise muss zuerst eine Vertauschung vorgenommen werden, die die Gesamtstrecke erhöht, um danach in einen besseren Zustand als zuvor zu gelangen. Um das globale Minimum (oder wenigstens ein besonders gutes lokales Minimum) zu finden, wird der stochastische Optimierungsalgorithmus des Simulated Annealings verwendet.

1.3 Simulated Annealing

Der Algorithmus ist von dem natürlichen Prozess des Abschreckens von Metallen inspiriert, genannt “annealing” auf Englisch, bei dem Metalle durch Erhitzen und langsames Abkühlen verfestigt werden. Es wird eine simulierte Temperatur T eingeführt. Zunächst werden zwei Knoten der Tour zufällig gewählt. Sofern dabei nicht die Winkelbedingung verletzt wird, wird die Tour errechnet, welche entstehen würde, wenn die beiden Knoten vertauscht werden. Das heißt, die Teiltour, welche durch die beiden gewählten Knoten begrenzt wird, wird umgekehrt. Ist die resultierende Tour besser als die bisherige, so werden

die Knoten vertauscht. Ist die Tour schlechter als die bisherige, so wird diese, sowohl abhängig von der aktuellen Temperatur T, als auch der absoluten Änderung der Länge, sowie abhängig eines Zufallswertes entweder gewählt oder verworfen. Die Anzahl der erfolgten verbessernden oder verschlechternden Vertauschungen werden gezählt. Nach einer gewissen Zahl solcher Operationen wird T abgekühlt (reduziert). Grundsätzlich gilt, eine höhere Temperatur erlaubt ungünstige Vertauschungen wahrscheinlicher als eine niedrige Temperatur. Diese Vorgehensweise erlaubt große Schwankungen am Anfang des Verfahrens, welche sich aber im Optimalfall um ein gutes Minimum gegen Ende einpendeln. Ausschlaggebend für die Ergebnisqualität sind die gewählten Parameter. Diese sind die Temperatur T, die Abkühlrate, die Anzahl der verbessernden Operationen bis zur Abkühlung sowie die Gesamtzahl an Operationen bis zur Abkühlung.

2 Umsetzung

Die Implementierung erfolgt in Java 17.
Die Beispieldateien enthalten Punkte mit einer X- und Y-Koordinate. Anhand der Punkte wird ein vollständiger, ungerichteter, gewichteter Graph erstellt. Zur Berechnung der Kantengewichte muss die Distanz der Punkte zueinander ausgerechnet werden. Dies erfolgt, indem die Ortsvektoren der einzelnen Punkte gebildet und voneinander abgezogen werden, um die Verbindungsvektoren zu erhalten und anschließend den Betrag dieser zu errechnen.

```
private Double length(Point a, Point b) {
    return Math.sqrt((a.x() - b.x()) * (a.x() - b.x()) + (a.y() - b.y()) * (a.y() - b.y()));
}
```

Um den "Abbiege-Winkel" an einem Knoten auszurechnen, wird der Schnittwinkel der beiden Richtungsvektoren der betrachteten anliegenden Kanten ausgerechnet. Ist dieser mindestens 90 Grad groß, so ist die Winkelbedingung erfüllt.

```
public static boolean matchesAngleCriteria(Vector first, Vector second) {
    double degree = VectorCalculator.calcDegree(first, second);
    return degree >= 90;
}

public static Double calcDegree(Vector a, Vector b) {
    double dotProduct = a.x() * b.x() + a.y() * b.y();
    double lengthA = length(a);
    double lengthB = length(b);
    return Math.toDegrees(Math.acos(dotProduct / (lengthA * lengthB)));
}

public static Double length(Vector vector) {
    return Math.sqrt(vector.x() * vector.x() + vector.y() * vector.y());
}
```

2.1 Simulated Annealing

Wie bereits beschrieben, hängt die Entscheidung, ob eine Vertauschung im betrachteten Pfad von verschiedenen Parametern ab. Konkret wird ein Zug welcher die Gesamtlänge der Tour um d erhöht genau dann erlaubt, wenn $e^{-d/T}$ größer ist als ein Zufallswert zwischen 0 und 1. Zum Vergleich, für $T = 500$ verhält sich die Funktion wie folgt:

X	50	200	350	500	650	800	950
Y	0,905	0,670	0,497	0,368	0,273	0,202	0,150

Tabelle 1: Temperatur 500

Neben der Temperatur gibt es noch weitere Parameter. Diese beschreiben, wie und wie oft sich die Temperatur abkühlt. Die Funktion, welche das Abkühlverhalten der Temperatur beschreibt, ist in diesem

Fall linear, könnte aber auch beliebig komplex sein.

```
private void reduceTemperatur() {
    this.temperatur = temperaturModifier * temperatur;
}
```

Die Abkühlrate liegt typischerweise zwischen 0.9 und $0,99$. Die letzten beiden Parameter bestimmen darüber, nach wie vielen Operationen die Temperatur abkühlt. a meint die Anzahl der verbessernden Iterationen und b meint die Anzahl der Iterationen insgesamt.
Im Allgemeinen haben sich die Parameter $Starttemperatur = 100$, $Abkühlrate = 0,9$, $a = 50$, $b = 100$ als gute Konfiguration erwiesen, die vergleichsweise gute Ergebnisse für beliebige Eingabe liefert.
Da die Simulated Annealing Methode kein klar definiertes Ziel bzw. Ende besitzt wird das Verfahren über die Zeitdauer limitiert. Um das Problem, dass es sich um ein stochastisches Verfahren handelt und damit die Qualität der Lösung schwanken kann, zu entgehen, wird das Verfahren öfters für die gleiche Ausgangstour wiederholt. Um die Wartezeit zu reduzieren, wird der Prozess mittels der Reactive Streams Library[2] parallelisiert.

2.2 Komplexitätsbetrachtung

Eine optimale Lösung zu finden, würde bedeuten, alle Permutationen bestehend aus allen Knoten auszuprobieren. Für den ersten Knoten gibt es n Möglichkeiten, für den zweiten $n-1$ etc. Damit ergibt sich die Laufzeit $n!$. Im schlechtesten Fall muss die Nearest-Neighbour Heuristik, wie oben beschrieben, auch an jedem Knoten jeden noch nicht besuchten Knoten ausprobieren. Damit folgt für die Nearest-Neighbour Heuristik die Worst-Case-Laufzeit von $n!$. Tatsächlich findet der Algorithmus nahezu instantan eine Lösung. Eine Ausnahme hierbei stellt das Beispiel 5 dar. Für genaueres siehe Kapitel 3.5.
Um auch Beispiele wie Beispiel 5 lösen zu können, werden für jeden Knoten des Graphens die Anzahl der möglichen Vorgänger- und Nachfolgerkanten ermittelt, um diese im Zweifelsfall in die Auswahl des nächsten Knotens in der Nearest Neighbour Heuristik einfließen zu lassen. Das hat eine Laufzeit von n^2
Die Postoptimierung vertauscht Knoten in einer linearen Laufzeit. Die Laufzeit der 2opt-Optimierung ist unabhängig von der Größe des Graphens, lediglich die Parameter entscheiden über die Anzahl der Operationen.

2.3 Kann das Programm immer eine Lösung finden?

Bevor nach einer Route gesucht wird, werden alle Knoten dahingehend geprüft, ob es mindestens eine valide Kombination an anliegenden Kanten gibt, über welche der Knoten traversiert werden kann. Es darf maximal zwei Knoten geben, die diese Eigenschaft nicht besitzen. Diese zwei Knoten müssen Start- und Endknoten sein. Gibt es mehr als zwei solcher Knoten, ist es nicht möglich alle diese Knoten zu erreichen, und somit auch nicht möglich eine Lösung zu ermitteln. Ein Beispiel für einen solchen nicht lösbaren Graphen wäre Graph in welchem die drei äußersten Punkte ein gleichseitiges Dreieck bilden.

3 Beispiele

Die Beispiele entsprechen den Beispieldateien. Beispiel 1 bis 3 lassen sich derart gut durch die Nearest-Neighbour Heuristik lösen, dass die Postoptimierung keine Verbesserungen erzielt. Die Graphiken zeigen jeweils wie die ermittelte Tour verläuft und wie lang diese ist. Gegebenenfalls sind zudem auch die Parameter der Postoptimierung angegeben. Die vollständige Programmausgabe ist im Anhang zu finden.

3.1 Beispiel 1

[2] https://github.com/reactive-streams/reactive-streams-jvm

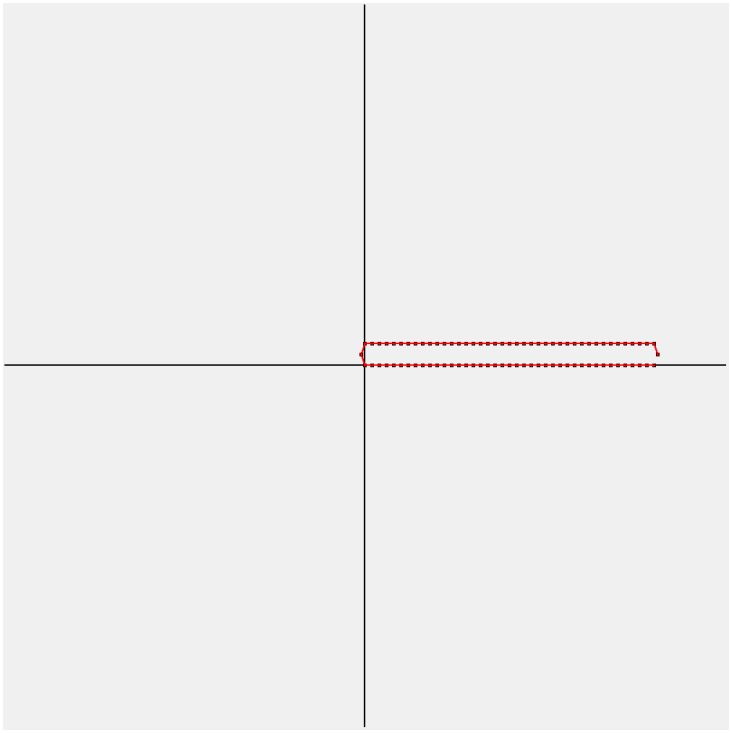


Abbildung 1: Naive Lösung Länge 847.4341649025257

3.2 Beispiel 2

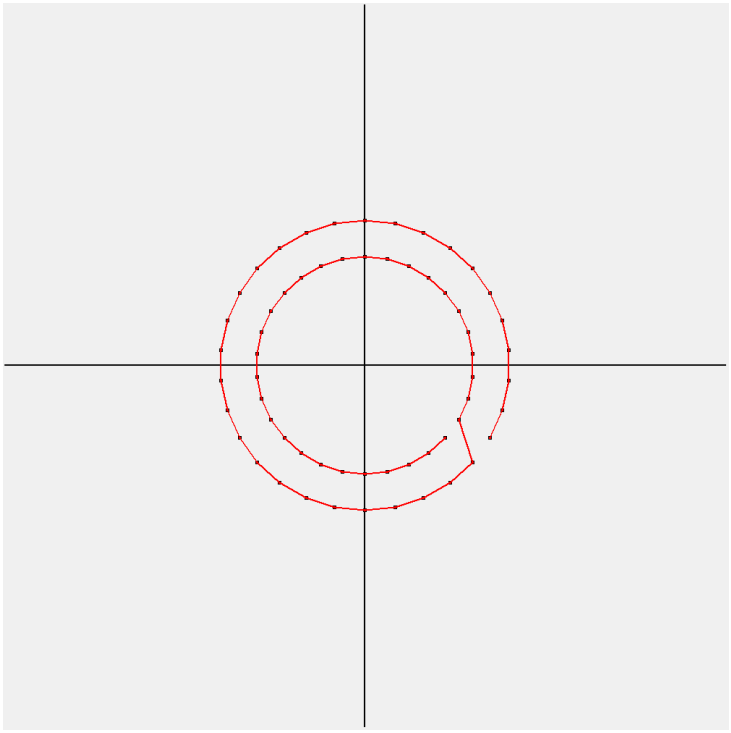


Abbildung 2: Naive Lösung Länge 2183.6622668054965

3.3 Beispiel 3

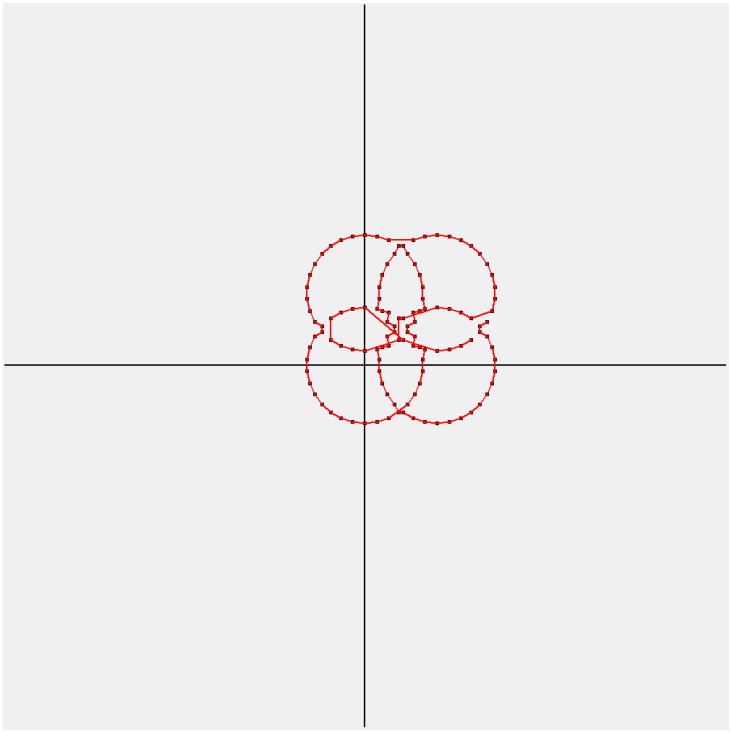


Abbildung 3: Naive Lösung Länge 2001.318764396512

3.4 Beispiel 4

Parameter	Wert
Starttemperatur	1300
Temperaturänderung	0,985
Verbessernde Iterationen bis zur Abkühlung	10
Iterationen bis zur Abkühlung	16

Tabelle 2: Parameterkonfiguration Beispiel 5

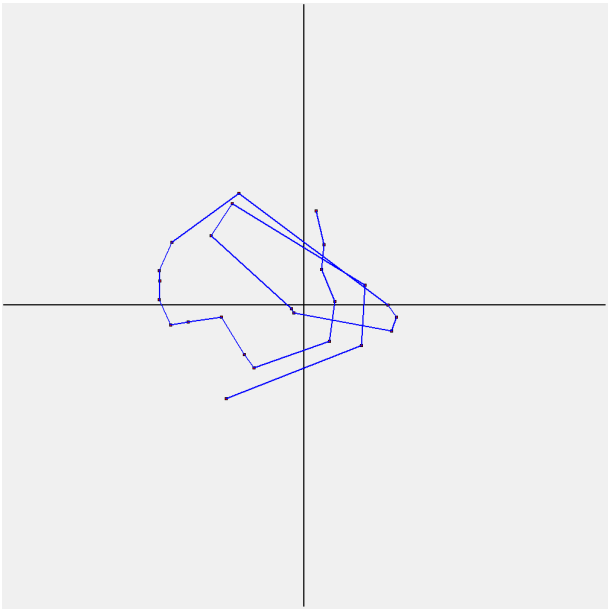


Abbildung 4: Naive Lösung Länge 2197.965650598992

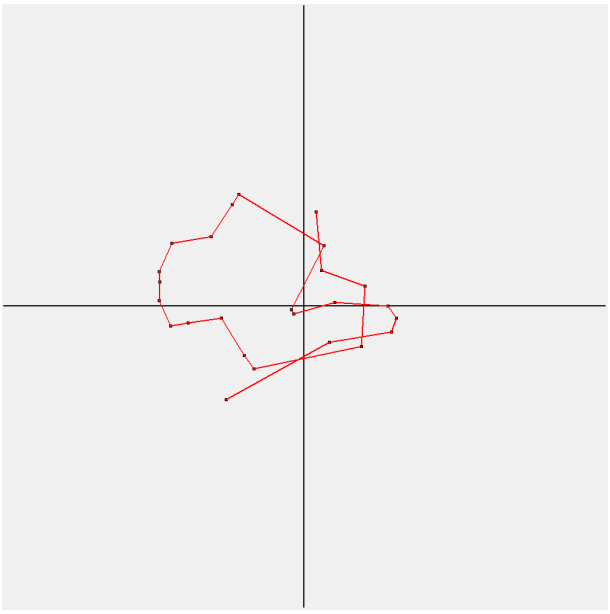


Abbildung 5: Optimierte Lösung Länge 1736.980701656645

3.5 Beispiel 5

Zunächst wurde versucht mittels Nearest-Neighbour Heuristik eine initiale Lösung zu finden. Dies führte auch nach über einer Stunde Laufzeit nicht zu einem Ergebnis. Aufgrund dessen wurde die Nearest-Neighbour Heuristik derart modifiziert, dass nicht nur der Abstand eines Knotens zum aktuellen Knoten in betracht gezogen wird, sondern auch die Anzahl der möglichen Pfade die über den Knoten führen. Je weniger solcher Pfade es gibt, desto höhere Priorität hat der Knoten in der Suche des nächsten Knotens. Dieses Vorgehen erklärt die vergleichsweise schlechte initiale Lösung. Um so interessanter ist es, dass die Postoptimierung dennoch zu so einem guten Ergebnis führt.

Parameter	Wert
Starttemperatur	1300
Temperaturänderung	0,985
Verbessernde Iterationen bis zur Abkühlung	10
Iterationen bis zur Abkühlung	16

Tabelle 3: Parameterkonfiguration Beispiel 5

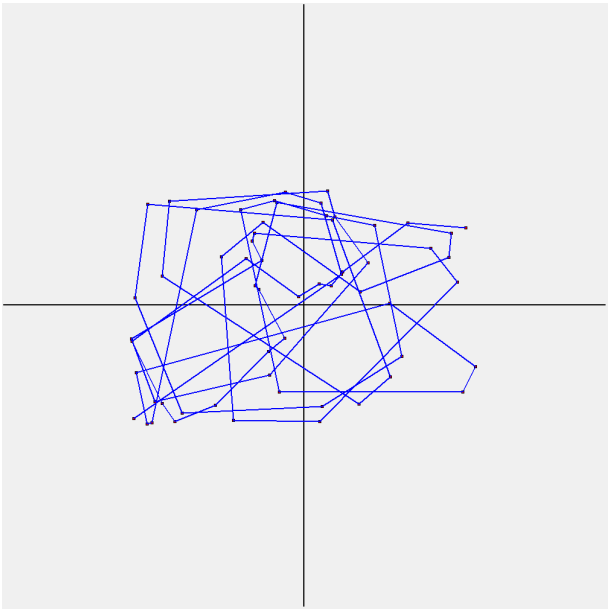


Abbildung 6: Naive Lösung Länge 9277.77985102936

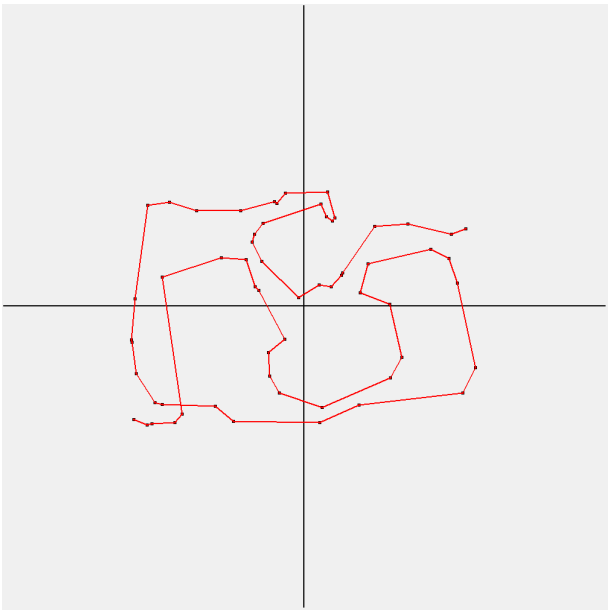


Abbildung 7: Optimierte Lösung Länge 3381.157719454162

3.6 Beispiel 6

Parameter	Wert
Starttemperatur	1300
Temperaturänderung	0,985
Verbessernde Iterationen bis zur Abkühlung	10
Iterationen bis zur Abkühlung	16

Tabelle 4: Parameterkonfiguration Beispiel 6

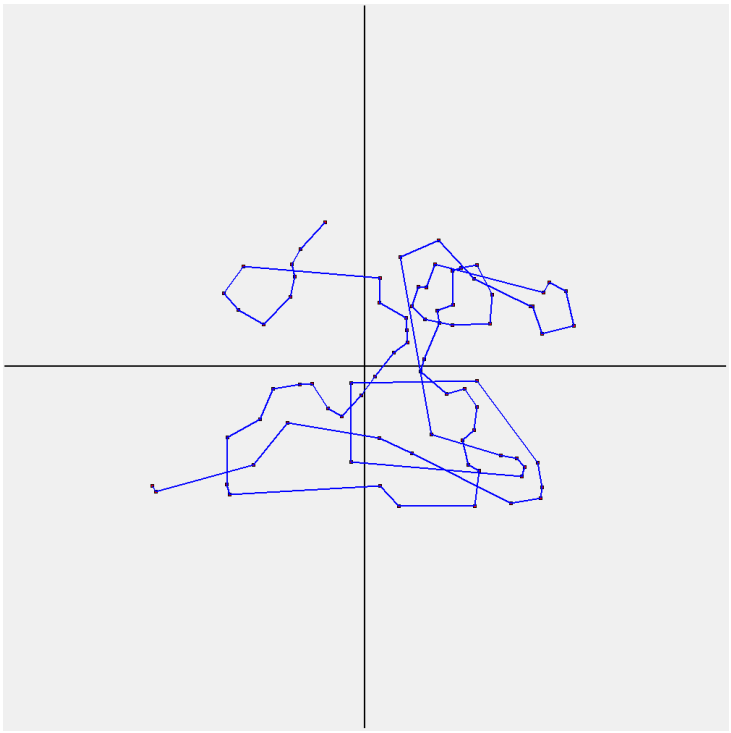


Abbildung 8: Naive Lösung Länge 4362.616434557695

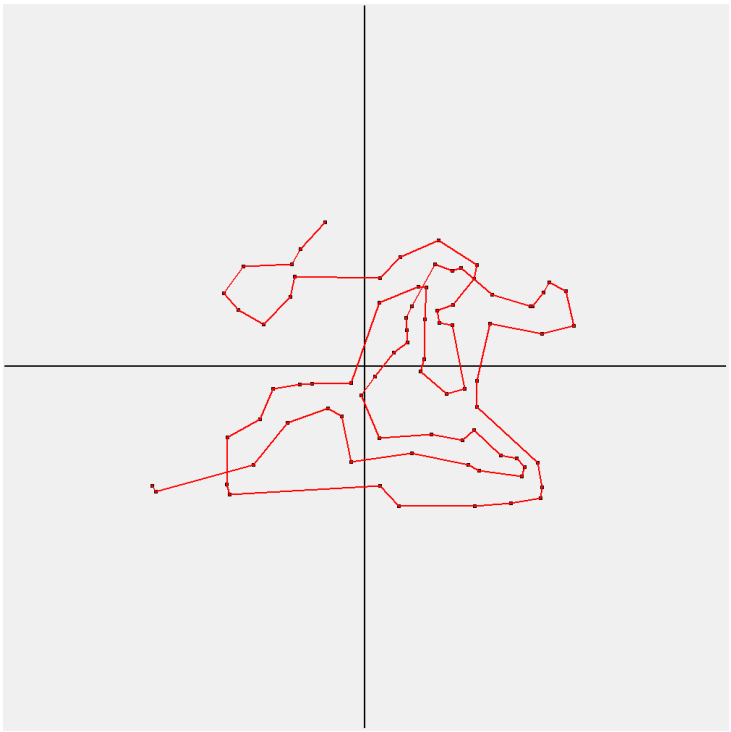


Abbildung 9: Optimierte Lösung Länge 3740.816749985773

3.7 Beispiel 7

Parameter	Wert
Starttemperatur	1450
Temperaturänderung	0,975
Verbessernde Iterationen bis zur Abkühlung	10
Iterationen bis zur Abkühlung	19

Tabelle 5: Parameterkonfiguration Beispiel 7

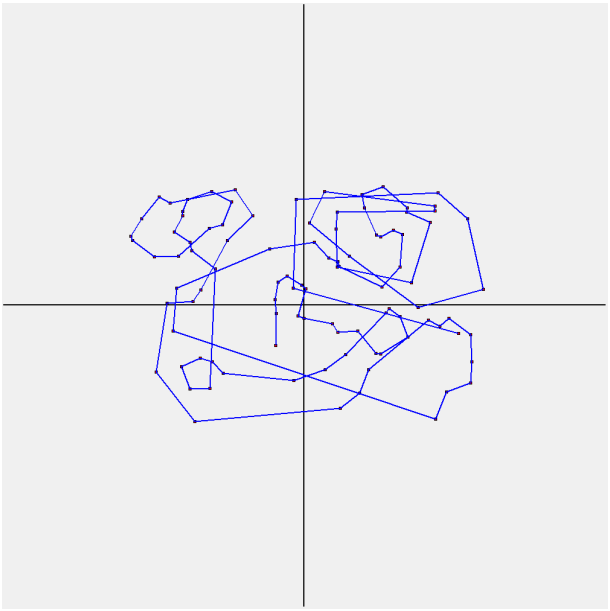


Abbildung 10: Naive Lösung Länge 6218.113121079488

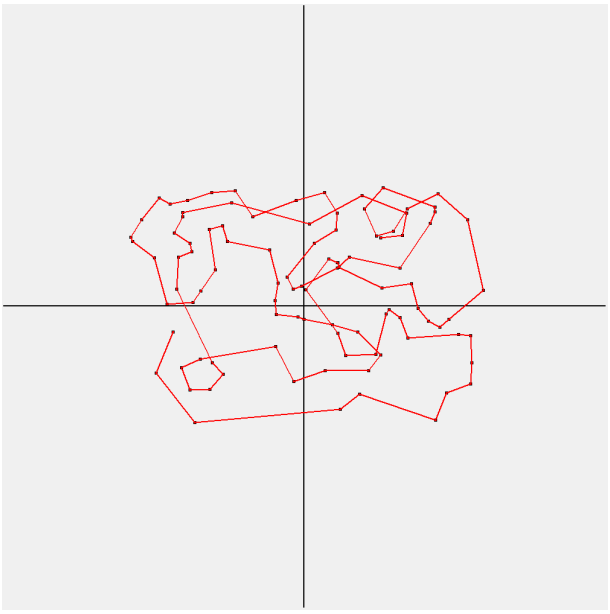


Abbildung 11: Optimierte Lösung Länge 4451.743885428826

4 Quellcode

```
public class VectorCalculator {

    public static boolean matchesAngleCriteria(Vector first, Vector second) {
        double degree = VectorCalculator.calcDegree(first, second);
        return degree >= 90;
    }

    public static Double calcDegree(Vector a, Vector b) {
        double dotProduct = a.x() * b.x() + a.y() * b.y();
        double lengthA = length(a);
        double lengthB = length(b);
        return Math.toDegrees(Math.acos(dotProduct / (lengthA * lengthB)));
    }

    public static Double length(Vector vector) {
        return Math.sqrt(vector.x() * vector.x() + vector.y() * vector.y());
    }
}
```

```
    public ParameterConfiguration findGoodParameter(Graph graph, List<Node> path) {

        SolutionEvaluator solutionEvaluator = new SolutionEvaluator();
        System.out.println("naiv solution " + solutionEvaluator.evaluate(graph, path));
        PathPrinter pathPrinter = new PathPrinter();
        pathPrinter.print(path);
        pathPrinter.printPoints(path);
        Set<ParameterConfiguration> parameterConfigurations =
            this.generateAllParameterConfigurations();

        final int iterations = 5;
        final long timeLimit = 10000;

        System.out.println("iterations " + iterations);
        System.out.println("length " + parameterConfigurations.size());
        System.out.println("iterations * length " + iterations * parameterConfigurations.size());

        final int cores = Runtime.getRuntime().availableProcessors();

        System.out.println("available cores " + cores);

        double seconds = iterations * parameterConfigurations.size() * timeLimit / 1000.0;
        System.out.println("estimated time in seconds without multithreading " + seconds);
        System.out.println("estimated time in minutes without multithreading " + seconds / 60.0);
        System.out.println("estimated time in hours without multithreading " + seconds / 3600.0);

        seconds = seconds / cores;

        System.out.println("estimated time in seconds using multithreading " + seconds);
        System.out.println("estimated time in minutes using multithreading " + seconds / 60.0);
        System.out.println("estimated time in hours using multithreading " + seconds / 3600.0);


        final Map<ParameterConfiguration, ParameterTestResult> results =
            Flux.fromIterable(parameterConfigurations)
               .parallel()
               .flatMap(parameterConfiguration -> Mono.fromCallable(() -> {
                   // System.out.println("now testing " + parameterConfiguration);
                   ParameterTestResult result = test(parameterConfiguration, solutionEvaluator,
                       graph, path, iterations, timeLimit);
                   // System.out.println("finished testing " + parameterConfiguration);
                   return Tuples.of(parameterConfiguration, result);
```

```
                }).subscribeOn(Schedulers.parallel()))
                .sequential()
                .collectMap(Tuple2::getT1, Tuple2::getT2)
                .block();

        String date = Instant.now().toString();

        this.saveBestResults(results, graph, date);

        return this.findBestByAverage(results,
            1).stream().findAny().map(Map.Entry::getKey).orElseThrow();
    }

        @Override
    public List<Node> solve(final Graph graph, Node start) {
        traverse(graph, start);

        return solution;
    }

    @Override
    public SolvingStrategy getSolvingStrategy() {
        return SolvingStrategy.NEAREST_NEIGHBOUR_HEURISTIC;
    }


    private void traverse(Graph graph, Node start) {
        this.traverse(graph, new HashSet<>(graph.getNodes()), start, null, new ArrayList<>());
    }

    private void traverse(Graph graph, Set<Node> open, Node current, Edge last, List<Node>
         path) {
        if (solutionFound) {
            return;
        }
        path.add(current);
        open.remove(current);
        if (open.isEmpty()) {
            solutionFound = true;
            solution = path;
            return;
        }

        List<Edge> possibleNext = this.weightedNextEdgeProvider.calcPossibleNext(graph, current,
             last, open);

        for (final Edge edge : possibleNext) {
            Node next = edge.flip(current);
            traverse(graph, new HashSet<>(open), next, edge, new ArrayList<>(path));
        }
    }

public class AngleCriteriaNextEdgeProvider {

    public Set<Edge> getNext(Graph graph, Node current, Edge last) {
        final Set<Edge> set = new HashSet<>();
        for (final Edge next : graph.getNeighbouringEdges().get(current)) {
            if (last.equals(next)) {
                continue;
            }
            if (VectorCalculator.matchesAngleCriteria(last.vector(current),
                 next.vector(current))) {
                set.add(next);
```

```
            }
        }
        return set;
    }

}


@RequiredArgsConstructor
public class WeightedNextEdgeProvider {

    private final AngleCriteriaNextEdgeProvider nextEdgeProvider;

    public List<Edge> calcPossibleNext(Graph graph, Node current, Edge last, Set<Node> open) {
        List<Edge> possibleNext;

        if (last == null) {
            possibleNext = new ArrayList<>(graph.getNeighbouringEdges().get(current));
        } else {
            possibleNext = new ArrayList<>(this.nextEdgeProvider.getNext(graph, current, last));
        }

        possibleNext.sort(Comparator.comparingDouble(Edge::weight));

        List<Edge> result = new ArrayList<>();

        for (final Edge edge : possibleNext) {
            Node next = edge.flip(current);
            if (open.contains(next)) {
                result.add(edge);
            }
        }

        return result;

    }


}


@RequiredArgsConstructor
public class NearestNeighbourHeuristic implements Solver {


    private boolean solutionFound = false;

    private List<Node> solution = null;

    private final WeightedNextEdgeProvider weightedNextEdgeProvider;

    @Override
    public List<Node> solve(final Graph graph, Node start) {
        traverse(graph, start);

        return solution;
    }

    @Override
    public SolvingStrategy getSolvingStrategy() {
        return SolvingStrategy.NEAREST_NEIGHBOUR_HEURISTIC;
    }
```

```
    private void traverse(Graph graph, Node start) {
        this.traverse(graph, new HashSet<>(graph.getNodes()), start, null, new ArrayList<>());
    }

    private void traverse(Graph graph, Set<Node> open, Node current, Edge last, List<Node>
         path) {
        if (solutionFound) {
            return;
        }
        path.add(current);
        open.remove(current);
        if (open.isEmpty()) {
            solutionFound = true;
            solution = path;
            return;
        }

        List<Edge> possibleNext = this.weightedNextEdgeProvider.calcPossibleNext(graph, current,
             last, open);

        for (final Edge edge : possibleNext) {
            Node next = edge.flip(current);
            traverse(graph, new HashSet<>(open), next, edge, new ArrayList<>(path));
        }
    }


}


/**
 * swap cities using simulated annealing to allow worsening swaps
 * this hopefully evades local minima
 */
@RequiredArgsConstructor
public class TwoOptPostOptimization implements PostOptimization {

    private final static ParameterConfiguration DEFAULT_CONFIGURATION =
            new ParameterConfiguration(100.0,
                    0.9,
                    50,
                    100);

    private final SolutionEvaluator solutionEvaluator;
    private final double temperaturModifier;
    private final int improvingIterationsUntilCooling;
    private final int iterationsUntilCooling;
    private double temperatur;


    public TwoOptPostOptimization(SolutionEvaluator solutionEvaluator) {
        this(solutionEvaluator, DEFAULT_CONFIGURATION);
    }

    public TwoOptPostOptimization(SolutionEvaluator solutionEvaluator,
                                  ParameterConfiguration configuration) {
        this.solutionEvaluator = solutionEvaluator;
        this.temperatur = configuration.startingTemperature();
        this.temperaturModifier = configuration.temperaturModifier();
        this.improvingIterationsUntilCooling = configuration.improvingIterationsUntilCooling();
        this.iterationsUntilCooling = configuration.iterationsUntilCooling();
    }
```

```
@Override
public List<Node> optimize(final Graph graph, final List<Node> tour, long timeLimit) {
    // TODO: 29.03.2023 optimize data structure of tour

    List<Node> current = new ArrayList<>(tour);

    double currentLength = solutionEvaluator.evaluate(graph, current);

    int improvingIterations = 0;
    int iterations = 0;
    long stopAfterTimeLimit = System.currentTimeMillis() + timeLimit;

    while (System.currentTimeMillis() < stopAfterTimeLimit) {
        ResultType improved = this.improve(graph, current, stopAfterTimeLimit);
        // double nextLength = solutionEvaluator.evaluate(graph, current);
        // System.out.println("improvement of " + (currentLength - nextLength));
        // currentLength = nextLength;
        if (improvingIterations == improvingIterationsUntilCooling) {
            improvingIterations = 0;
            iterations = 0;
            this.reduceTemperatur();
        }
        if (iterations == iterationsUntilCooling) {
            improvingIterations = 0;
            iterations = 0;
            this.reduceTemperatur();
        }
        // System.out.println("t " + temperatur);
        if (improved.equals(ResultType.ZERO)) {
            continue;
        }
        iterations++;
        if (improved.equals(ResultType.BETTER)) {
            improvingIterations++;
        }
    }
    return current;
}

private ResultType improve(Graph graph, List<Node> current, long timeLimit) {
    while (System.currentTimeMillis() < timeLimit) {

        int first = this.randomInRange(0, current.size() - 4);
        int second = this.randomInRange(first + 2, current.size() - 1);

        Node prevFirstNode = first == 0 ? null : current.get(first - 1);
        Node firstNode = current.get(first);
        Node postFirstNode = current.get(first + 1);
        Node postPostFirstNode = current.get(first + 2);

        Node prevSecondNode = current.get(second - 1);
        Node secondNode = current.get(second);
        Node postSecondNode = second == current.size() - 1 ? null : current.get(second + 1);
        Node postPostSecondNode = second >= current.size() - 2 ? null : current.get(second +
            2);

        Edge a = prevFirstNode == null ? null : graph.getEdge(prevFirstNode, firstNode);
        Edge insertedEdgeOne = graph.getEdge(firstNode, secondNode);
        Edge insertedEdgeTwo = postSecondNode == null ? null : graph.getEdge(postFirstNode,
            postSecondNode);
        Edge d = graph.getEdge(prevSecondNode, secondNode);
        Edge e = graph.getEdge(postFirstNode, postPostFirstNode);
```

```
        Edge f = postPostSecondNode == null ? null : graph.getEdge(postSecondNode,
            postPostSecondNode);

        if (doesNotMatchAngleCriteria(a, insertedEdgeOne)) {
            continue;
        }

        if (doesNotMatchAngleCriteria(insertedEdgeOne, d)) {
            continue;
        }

        if (doesNotMatchAngleCriteria(e, insertedEdgeTwo)) {
            continue;
        }

        if (doesNotMatchAngleCriteria(insertedEdgeTwo, f)) {
            continue;
        }

        Edge b = graph.getEdge(secondNode, postSecondNode);
        Edge c = graph.getEdge(firstNode, postFirstNode);

        double difference = -c.weight() - b.weight() + insertedEdgeOne.weight() +
            insertedEdgeTwo.weight();

        if (difference < 0) {
            twoOptSwap(current, first, second);
            return ResultType.BETTER;
        }

        if (difference == 0) {
            continue;
        }

        if (allowWorseningSwap(difference)) {
            twoOptSwap(current, first, second);
            return ResultType.WORSE;
        }
    }
    return ResultType.ZERO;
}

private int randomInRange(int from, int to) {

    if (from > to) {
        throw new IllegalArgumentException("from " + from + " > to " + to);
    }

    if (from == to) {
        return from;
    }

    int out = (int) (Math.random() * (to - from));
    out += from;
    // System.out.println("out " + (out) + " in range " + from + " to " + to);
    return out;
}

private boolean doesNotMatchAngleCriteria(Edge a, Edge b) {
    if (a == null || b == null) {
        return false;
    }
    if (a.from().equals(b.from())) {
```

```
            return !VectorCalculator.matchesAngleCriteria(a.vector(a.from()),
                b.vector(a.from()));
        }
        if (a.to().equals(b.to())) {
            return !VectorCalculator.matchesAngleCriteria(a.vector(a.to()), b.vector(a.to()));
        }
        if (a.to().equals(b.from())) {
            return !VectorCalculator.matchesAngleCriteria(a.vector(a.to()), b.vector(a.to()));
        }
        if (a.from().equals(b.to())) {
            return !VectorCalculator.matchesAngleCriteria(a.vector(a.from()),
                b.vector(a.from()));
        }
        throw new IllegalArgumentException();
    }

    /**
     * only call if swap would make the tour longer
     *
     * @param difference current length - prev length
     * @return true if swap is allowed event though it worsens the length of the tour
     */
    private boolean allowWorseningSwap(double difference) {
        double random = Math.random();
        double exp = Math.exp((-difference) / temperatur);

        return exp > random;
    }


    private void twoOptSwap(List<Node> tour, int first, int second) {
        List<Node> firstToSecond = tour.subList(first + 1, second + 1);

        Collections.reverse(firstToSecond);
    }

    private enum ResultType {
        ZERO,
        BETTER,
        WORSE
    }

    private void reduceTemperatur() {
        this.temperatur = temperaturModifier * temperatur;
    }

}


public record Vector(double x, double y) {

    public static Vector of(Edge edge) {
        Point from = edge.from().point();
        Point to = edge.to().point();
        double x = to.x() - from.x();
        double y = to.y() - from.y();
        return new Vector(x, y);
    }

    public static Vector inverse(Vector vector) {
        return new Vector(-vector.x(), -vector.y());
    }
```

```
}


public record Point(Double x, Double y) {
}


public record Node(int id, Point point) {
}

public final class Edge {
    private final Node from;
    private final Node to;
    private final Double weight;
    private final Vector vectorFrom;
    private final Vector vectorTo;

    public Edge(Node from, Node to, Double weight) {
        this.from = from;
        this.to = to;
        this.weight = weight;
        this.vectorFrom = Vector.of(this);
        this.vectorTo = Vector.inverse(this.vectorFrom);
    }

    public Node from() {
        return from;
    }

    public Node to() {
        return to;
    }

    public Double weight() {
        return weight;
    }

    public Node flip(Node current) {
        if (from.equals(current)) return to;
        if (to.equals(current)) return from;
        throw new IllegalArgumentException("Node " + current.id() + " is not part of edge " +
            this + " and can therefor not be used to flip!");
    }

    public Vector vector(Node base) {
        if (base.equals(to())) {
            return this.vectorTo;
        }
        if (base.equals(from())) {
            return this.vectorFrom;
        }
        throw new IllegalArgumentException();
    }
}


@Getter
public class Graph {

    private final Set<Node> nodes;
    private final Set<Edge> edges;
    private final Map<Node, Set<Node>> neighbours;
    private final Map<Node, Set<Edge>> neighbouringEdges;
```

```
    private final Map<Node, Integer> numberOfRoutes;

    public Graph(final Set<Node> nodes, final Set<Edge> edges) {
        this.nodes = nodes;
        this.edges = edges;
        this.neighbours = this.calcNeighbours(nodes, edges);
        this.neighbouringEdges = this.calcNeighbouringEdges(nodes, edges);
        this.numberOfRoutes = this.calcNumberOfRoutes();
    }


    private Map<Node, Set<Edge>> calcNeighbouringEdges(final Set<Node> nodes, final Set<Edge>
         edges) {
        Map<Node, Set<Edge>> map = new HashMap<>();
        for (final Node node : nodes) {
            Set<Edge> set = new HashSet<>();
            for (final Edge edge : edges) {
                if (edge.from().equals(node) || edge.to().equals(node)) {
                    set.add(edge);
                }
            }
            map.put(node, set);
        }
        return map;
    }

    private Map<Node, Set<Node>> calcNeighbours(Set<Node> nodes, Set<Edge> edges) {
        Map<Node, Set<Node>> map = new HashMap<>();
        for (Node node : nodes) {
            Set<Node> set = new HashSet<>();
            for (Edge edge : edges) {
                if (edge.from().equals(node)) {
                    set.add(edge.to());
                }
            }
            map.put(node, set);
        }
        return map;
    }

    public Edge getEdge(Node from, Node to) {
        for (final Edge edge : this.getNeighbouringEdges().get(from)) {
            if (edge.from().equals(from) && edge.to().equals(to) || edge.to().equals(from) &&
                 edge.from().equals(to)) {
                return edge;
            }
        }
        System.out.println("smth went wrong finding an edge between node " + from + " and " +
             to);
        return null;
    }

    public Node getNodeById(int id) {
        return this.nodes.stream().filter(node -> node.id() == id).findAny().orElseThrow();
    }

    private Map<Node, Integer> calcNumberOfRoutes() {
        Map<Node, Integer> result = new HashMap<>();
        for (final Node node : this.nodes) {
            int i = this.countValidConnectionsOverNode(node);
            result.put(node, i);
        }
        return result;
```

```
    }

    private int countValidConnectionsOverNode(Node node) {
        int counter = 0;

        for (final Node from :
             this.getNodes().stream().sorted(Comparator.comparingInt(Node::id)).toList()) {
            if (from.equals(node)) {
                continue;
            }

            Edge edge = this.getEdge(from, node);

            int size = this.getNextSize(node, edge);
            counter += size;
        }
        return counter;
    }

    private int getNextSize(Node current, Edge last) {
        int counter = 0;
        for (final Edge next : this.getNeighbouringEdges().get(current)) {
            if (next.equals(last)) {
                continue;
            }
            if (VectorCalculator.matchesAngleCriteria(last.vector(current),
                 next.vector(current))) {
                counter += 1;
            }
        }
        return counter;
    }

    public Node findStartNode(){
        return this.getNodes().stream().filter(node -> node.id() == 0).findAny().orElseThrow();

    }


}
```

WENIGERKRUMM 1
nodes 84 edges 3486
all starting node nearest neighbour heuristic found best tour with length 847.4341649025257 for starting node Node[id=26, point=Point[x=400.0, y=0.0]]
time to find init solution 0 seconds
found init solution
Path
400.0 0.0 390.0 0.0
390.0 0.0 380.0 0.0
380.0 0.0 370.0 0.0
370.0 0.0 360.0 0.0
360.0 0.0 350.0 0.0
350.0 0.0 340.0 0.0
340.0 0.0 330.0 0.0
330.0 0.0 320.0 0.0
320.0 0.0 310.0 0.0
310.0 0.0 300.0 0.0
300.0 0.0 290.0 0.0
290.0 0.0 280.0 0.0
280.0 0.0 270.0 0.0
270.0 0.0 260.0 0.0
260.0 0.0 250.0 0.0
250.0 0.0 240.0 0.0
240.0 0.0 230.0 0.0
230.0 0.0 220.0 0.0
220.0 0.0 210.0 0.0
210.0 0.0 200.0 0.0
200.0 0.0 190.0 0.0
190.0 0.0 180.0 0.0
180.0 0.0 170.0 0.0
170.0 0.0 160.0 0.0
160.0 0.0 150.0 0.0
150.0 0.0 140.0 0.0
140.0 0.0 130.0 0.0
130.0 0.0 120.0 0.0
120.0 0.0 110.0 0.0
110.0 0.0 100.0 0.0
100.0 0.0 90.0 0.0
90.0 0.0 80.0 0.0
80.0 0.0 70.0 0.0
70.0 0.0 60.0 0.0
60.0 0.0 50.0 0.0
50.0 0.0 40.0 0.0
40.0 0.0 30.0 0.0
30.0 0.0 20.0 0.0
20.0 0.0 10.0 0.0
10.0 0.0 0.0 0.0
0.0 0.0 -5.0 15.0
-5.0 15.0 0.0 30.0
0.0 30.0 10.0 30.0
10.0 30.0 20.0 30.0
20.0 30.0 30.0 30.0
30.0 30.0 40.0 30.0
40.0 30.0 50.0 30.0
50.0 30.0 60.0 30.0
60.0 30.0 70.0 30.0
70.0 30.0 80.0 30.0
80.0 30.0 90.0 30.0
90.0 30.0 100.0 30.0

```
100.0 30.0 110.0 30.0
110.0 30.0 120.0 30.0
120.0 30.0 130.0 30.0
130.0 30.0 140.0 30.0
140.0 30.0 150.0 30.0
150.0 30.0 160.0 30.0
160.0 30.0 170.0 30.0
170.0 30.0 180.0 30.0
180.0 30.0 190.0 30.0
190.0 30.0 200.0 30.0
200.0 30.0 210.0 30.0
210.0 30.0 220.0 30.0
220.0 30.0 230.0 30.0
230.0 30.0 240.0 30.0
240.0 30.0 250.0 30.0
250.0 30.0 260.0 30.0
260.0 30.0 270.0 30.0
270.0 30.0 280.0 30.0
280.0 30.0 290.0 30.0
290.0 30.0 300.0 30.0
300.0 30.0 310.0 30.0
310.0 30.0 320.0 30.0
320.0 30.0 330.0 30.0
330.0 30.0 340.0 30.0
340.0 30.0 350.0 30.0
350.0 30.0 360.0 30.0
360.0 30.0 370.0 30.0
370.0 30.0 380.0 30.0
380.0 30.0 390.0 30.0
390.0 30.0 400.0 30.0
400.0 30.0 405.0 15.0

eval naive 847.4341649025257
eval optimized 1627.4341649025255
improvement achieved by post optimization algorithm -779.9999999999998
Path 26-18-33-70-5-56-9-34-63-55-53-39-54-15-28-51-35-79-38-37-0-4-25-13-
66-1-21-83-10-74-6-40-27-72-43-81-14-36-62-75-20-47-31-19-42-73-7-16-44-
46-3-30-24-12-65-64-45-41-78-76-58-67-11-2-22-29-32-49-17-23-57-48-77-60-
80-68-82-69-61-8-71-59-52-5
Path
400.0 0.0 390.0 0.0
390.0 0.0 380.0 0.0
380.0 0.0 370.0 0.0
370.0 0.0 360.0 0.0
360.0 0.0 350.0 0.0
350.0 0.0 340.0 0.0
340.0 0.0 330.0 0.0
330.0 0.0 320.0 0.0
320.0 0.0 310.0 0.0
310.0 0.0 300.0 0.0
300.0 0.0 290.0 0.0
290.0 0.0 280.0 0.0
280.0 0.0 270.0 0.0
270.0 0.0 260.0 0.0
260.0 0.0 250.0 0.0
250.0 0.0 240.0 0.0
240.0 0.0 230.0 0.0
230.0 0.0 220.0 0.0
220.0 0.0 210.0 0.0
210.0 0.0 200.0 0.0
```

200.0 0.0 190.0 0.0
190.0 0.0 180.0 0.0
180.0 0.0 170.0 0.0
170.0 0.0 160.0 0.0
160.0 0.0 150.0 0.0
150.0 0.0 140.0 0.0
140.0 0.0 130.0 0.0
130.0 0.0 120.0 0.0
120.0 0.0 110.0 0.0
110.0 0.0 100.0 0.0
100.0 0.0 90.0 0.0
90.0 0.0 80.0 0.0
80.0 0.0 70.0 0.0
70.0 0.0 60.0 0.0
60.0 0.0 50.0 0.0
50.0 0.0 40.0 0.0
40.0 0.0 30.0 0.0
30.0 0.0 20.0 0.0
20.0 0.0 10.0 0.0
10.0 0.0 0.0 0.0
0.0 0.0 -5.0 15.0
-5.0 15.0 0.0 30.0
0.0 30.0 10.0 30.0
10.0 30.0 20.0 30.0
20.0 30.0 30.0 30.0
30.0 30.0 40.0 30.0
40.0 30.0 50.0 30.0
50.0 30.0 60.0 30.0
60.0 30.0 70.0 30.0
70.0 30.0 80.0 30.0
80.0 30.0 90.0 30.0
90.0 30.0 100.0 30.0
100.0 30.0 110.0 30.0
110.0 30.0 120.0 30.0
120.0 30.0 130.0 30.0
130.0 30.0 140.0 30.0
140.0 30.0 150.0 30.0
150.0 30.0 160.0 30.0
160.0 30.0 170.0 30.0
170.0 30.0 180.0 30.0
180.0 30.0 190.0 30.0
190.0 30.0 200.0 30.0
200.0 30.0 210.0 30.0
210.0 30.0 220.0 30.0
220.0 30.0 230.0 30.0
230.0 30.0 240.0 30.0
240.0 30.0 250.0 30.0
250.0 30.0 260.0 30.0
260.0 30.0 270.0 30.0
270.0 30.0 280.0 30.0
280.0 30.0 290.0 30.0
290.0 30.0 300.0 30.0
300.0 30.0 310.0 30.0
310.0 30.0 320.0 30.0
320.0 30.0 330.0 30.0
330.0 30.0 340.0 30.0
340.0 30.0 350.0 30.0
350.0 30.0 360.0 30.0
360.0 30.0 370.0 30.0
370.0 30.0 380.0 30.0

380.0 30.0 390.0 30.0
390.0 30.0 400.0 30.0
400.0 30.0 405.0 15.0

WENIGERKRUMM 2
nodes 60 edges 1770
all starting node nearest neighbour heuristic found best tour with length 2183.6622668054965 for starting node Node[id=24, point=Point[x=173.205081, y=-100.0]]
time to find init solution 0 seconds
found init solution
Path
173.205081 -100.0 190.211303 -61.803399
190.211303 -61.803399 198.904379 -20.905693
198.904379 -20.905693 198.904379 20.905693
198.904379 20.905693 190.211303 61.803399
190.211303 61.803399 173.205081 100.0
173.205081 100.0 148.628965 133.826121
148.628965 133.826121 117.55705 161.803399
117.55705 161.803399 81.347329 182.709092
81.347329 182.709092 41.582338 195.62952
41.582338 195.62952 0.0 200.0
0.0 200.0 -41.582338 195.62952
-41.582338 195.62952 -81.347329 182.709092
-81.347329 182.709092 -117.55705 161.803399
-117.55705 161.803399 -148.628965 133.826121
-148.628965 133.826121 -173.205081 100.0
-173.205081 100.0 -190.211303 61.803399
-190.211303 61.803399 -198.904379 20.905693
-198.904379 20.905693 -198.904379 -20.905693
-198.904379 -20.905693 -190.211303 -61.803399
-190.211303 -61.803399 -173.205081 -100.0
-173.205081 -100.0 -148.628965 -133.826121
-148.628965 -133.826121 -117.55705 -161.803399
-117.55705 -161.803399 -81.347329 -182.709092
-81.347329 -182.709092 -41.582338 -195.62952
-41.582338 -195.62952 0.0 -200.0
0.0 -200.0 41.582338 -195.62952
41.582338 -195.62952 81.347329 -182.709092
81.347329 -182.709092 117.55705 -161.803399
117.55705 -161.803399 148.628965 -133.826121
148.628965 -133.826121 129.903811 -75.0
129.903811 -75.0 142.658477 -46.352549
142.658477 -46.352549 149.178284 -15.679269
149.178284 -15.679269 149.178284 15.679269
149.178284 15.679269 142.658477 46.352549
142.658477 46.352549 129.903811 75.0
129.903811 75.0 111.471724 100.369591
111.471724 100.369591 88.167788 121.352549
88.167788 121.352549 61.010496 137.031819
61.010496 137.031819 31.186754 146.72214
31.186754 146.72214 0.0 150.0
0.0 150.0 -31.186754 146.72214
-31.186754 146.72214 -61.010496 137.031819
-61.010496 137.031819 -88.167788 121.352549
-88.167788 121.352549 -111.471724 100.369591
-111.471724 100.369591 -129.903811 75.0
-129.903811 75.0 -142.658477 46.352549
-142.658477 46.352549 -149.178284 15.679269
-149.178284 15.679269 -149.178284 -15.679269
-149.178284 -15.679269 -142.658477 -46.352549
-142.658477 -46.352549 -129.903811 -75.0
-129.903811 -75.0 -111.471724 -100.369591
-111.471724 -100.369591 -88.167788 -121.352549

-88.167788 -121.352549 -61.010496 -137.031819
-61.010496 -137.031819 -31.186754 -146.72214
-31.186754 -146.72214 0.0 -150.0
0.0 -150.0 31.186754 -146.72214
31.186754 -146.72214 61.010496 -137.031819
61.010496 -137.031819 88.167788 -121.352549
88.167788 -121.352549 111.471724 -100.369591

eval naive 2183.6622668054965
eval optimized 2183.6622668054965
improvement achieved by post optimization algorithm 0.0
Path 24-55-59-15-9-46-10-41-0-57-37-11-22-12-19-45-48-54-25-51-7-21-3-35-4-13-44-1-49-47-56-40-5-36-29-18-38-34-30-31-16-17-43-52-14-23-27-20-42-39-8-6-26-50-53-58-33-32-2-2
Path
173.205081 -100.0 190.211303 -61.803399
190.211303 -61.803399 198.904379 -20.905693
198.904379 -20.905693 198.904379 20.905693
198.904379 20.905693 190.211303 61.803399
190.211303 61.803399 173.205081 100.0
173.205081 100.0 148.628965 133.826121
148.628965 133.826121 117.55705 161.803399
117.55705 161.803399 81.347329 182.709092
81.347329 182.709092 41.582338 195.62952
41.582338 195.62952 0.0 200.0
0.0 200.0 -41.582338 195.62952
-41.582338 195.62952 -81.347329 182.709092
-81.347329 182.709092 -117.55705 161.803399
-117.55705 161.803399 -148.628965 133.826121
-148.628965 133.826121 -173.205081 100.0
-173.205081 100.0 -190.211303 61.803399
-190.211303 61.803399 -198.904379 20.905693
-198.904379 20.905693 -198.904379 -20.905693
-198.904379 -20.905693 -190.211303 -61.803399
-190.211303 -61.803399 -173.205081 -100.0
-173.205081 -100.0 -148.628965 -133.826121
-148.628965 -133.826121 -117.55705 -161.803399
-117.55705 -161.803399 -81.347329 -182.709092
-81.347329 -182.709092 -41.582338 -195.62952
-41.582338 -195.62952 0.0 -200.0
0.0 -200.0 41.582338 -195.62952
41.582338 -195.62952 81.347329 -182.709092
81.347329 -182.709092 117.55705 -161.803399
117.55705 -161.803399 148.628965 -133.826121
148.628965 -133.826121 129.903811 -75.0
129.903811 -75.0 142.658477 -46.352549
142.658477 -46.352549 149.178284 -15.679269
149.178284 -15.679269 149.178284 15.679269
149.178284 15.679269 142.658477 46.352549
142.658477 46.352549 129.903811 75.0
129.903811 75.0 111.471724 100.369591
111.471724 100.369591 88.167788 121.352549
88.167788 121.352549 61.010496 137.031819
61.010496 137.031819 31.186754 146.72214
31.186754 146.72214 0.0 150.0
0.0 150.0 -31.186754 146.72214
-31.186754 146.72214 -61.010496 137.031819
-61.010496 137.031819 -88.167788 121.352549
-88.167788 121.352549 -111.471724 100.369591
-111.471724 100.369591 -129.903811 75.0

-129.903811 75.0 -142.658477 46.352549
-142.658477 46.352549 -149.178284 15.679269
-149.178284 15.679269 -149.178284 -15.679269
-149.178284 -15.679269 -142.658477 -46.352549
-142.658477 -46.352549 -129.903811 -75.0
-129.903811 -75.0 -111.471724 -100.369591
-111.471724 -100.369591 -88.167788 -121.352549
-88.167788 -121.352549 -61.010496 -137.031819
-61.010496 -137.031819 -31.186754 -146.72214
-31.186754 -146.72214 0.0 -150.0
0.0 -150.0 31.186754 -146.72214
31.186754 -146.72214 61.010496 -137.031819
61.010496 -137.031819 88.167788 -121.352549
88.167788 -121.352549 111.471724 -100.369591

WENIGERKRUMM 3
nodes 120 edges 7140
all starting node nearest neighbour heuristic found best tour with length 2001.318764396512 for starting node Node[id=78, point=Point[x=169.282032, y=60.0]]
time to find init solution 0 seconds
found init solution
Path
169.282032 60.0 159.451586 53.530449
159.451586 53.530449 159.451586 46.469551
159.451586 46.469551 169.282032 40.0
169.282032 40.0 176.084521 24.72136
176.084521 24.72136 179.561752 8.362277
179.561752 8.362277 179.561752 -8.362277
179.561752 -8.362277 176.084521 -24.72136
176.084521 -24.72136 169.282032 -40.0
169.282032 -40.0 159.451586 -53.530449
159.451586 -53.530449 147.02282 -64.72136
147.02282 -64.72136 132.538931 -73.083637
132.538931 -73.083637 116.632935 -78.251808
116.632935 -78.251808 100.0 -80.0
100.0 -80.0 83.367065 -78.251808
83.367065 -78.251808 67.461069 -73.083637
67.461069 -73.083637 52.97718 -64.72136
52.97718 -64.72136 47.02282 -64.72136
47.02282 -64.72136 40.548414 -53.530449
40.548414 -53.530449 30.717968 -40.0
30.717968 -40.0 23.915479 -24.72136
23.915479 -24.72136 20.438248 -8.362277
20.438248 -8.362277 20.438248 8.362277
20.438248 8.362277 16.632935 21.748192
16.632935 21.748192 23.915479 24.72136
23.915479 24.72136 32.538931 26.916363
32.538931 26.916363 30.717968 40.0
30.717968 40.0 40.548414 46.469551
40.548414 46.469551 40.548414 53.530449
40.548414 53.530449 30.717968 60.0
30.717968 60.0 32.538931 73.083637
32.538931 73.083637 23.915479 75.27864
23.915479 75.27864 16.632935 78.251808
16.632935 78.251808 20.438248 91.637723
20.438248 91.637723 20.438248 108.362277
20.438248 108.362277 23.915479 124.72136
23.915479 124.72136 30.717968 140.0
30.717968 140.0 40.548414 153.530449
40.548414 153.530449 47.02282 164.72136
47.02282 164.72136 52.97718 164.72136
52.97718 164.72136 59.451586 153.530449
59.451586 153.530449 69.282032 140.0
69.282032 140.0 76.084521 124.72136
76.084521 124.72136 79.561752 108.362277
79.561752 108.362277 79.561752 91.637723
79.561752 91.637723 83.367065 78.251808
83.367065 78.251808 76.084521 75.27864
76.084521 75.27864 67.461069 73.083637
67.461069 73.083637 69.282032 60.0
69.282032 60.0 59.451586 53.530449
59.451586 53.530449 59.451586 46.469551
59.451586 46.469551 69.282032 40.0
69.282032 40.0 67.461069 26.916363

67.461069 26.916363 76.084521 24.72136
76.084521 24.72136 83.367065 21.748192
83.367065 21.748192 79.561752 8.362277
79.561752 8.362277 79.561752 -8.362277
79.561752 -8.362277 76.084521 -24.72136
76.084521 -24.72136 69.282032 -40.0
69.282032 -40.0 59.451586 -53.530449
59.451586 -53.530449 32.538931 -73.083637
32.538931 -73.083637 16.632935 -78.251808
16.632935 -78.251808 0.0 -80.0
0.0 -80.0 -16.632935 -78.251808
-16.632935 -78.251808 -32.538931 -73.083637
-32.538931 -73.083637 -47.02282 -64.72136
-47.02282 -64.72136 -59.451586 -53.530449
-59.451586 -53.530449 -69.282032 -40.0
-69.282032 -40.0 -76.084521 -24.72136
-76.084521 -24.72136 -79.561752 -8.362277
-79.561752 -8.362277 -79.561752 8.362277
-79.561752 8.362277 -76.084521 24.72136
-76.084521 24.72136 -69.282032 40.0
-69.282032 40.0 -59.451586 46.469551
-59.451586 46.469551 -59.451586 53.530449
-59.451586 53.530449 -69.282032 60.0
-69.282032 60.0 -76.084521 75.27864
-76.084521 75.27864 -79.561752 91.637723
-79.561752 91.637723 -79.561752 108.362277
-79.561752 108.362277 -76.084521 124.72136
-76.084521 124.72136 -69.282032 140.0
-69.282032 140.0 -59.451586 153.530449
-59.451586 153.530449 -47.02282 164.72136
-47.02282 164.72136 -32.538931 173.083637
-32.538931 173.083637 -16.632935 178.251808
-16.632935 178.251808 0.0 180.0
0.0 180.0 16.632935 178.251808
16.632935 178.251808 32.538931 173.083637
32.538931 173.083637 67.461069 173.083637
67.461069 173.083637 83.367065 178.251808
83.367065 178.251808 100.0 180.0
100.0 180.0 116.632935 178.251808
116.632935 178.251808 132.538931 173.083637
132.538931 173.083637 147.02282 164.72136
147.02282 164.72136 159.451586 153.530449
159.451586 153.530449 169.282032 140.0
169.282032 140.0 176.084521 124.72136
176.084521 124.72136 179.561752 108.362277
179.561752 108.362277 179.561752 91.637723
179.561752 91.637723 176.084521 75.27864
176.084521 75.27864 147.02282 64.72136
147.02282 64.72136 132.538931 73.083637
132.538931 73.083637 116.632935 78.251808
116.632935 78.251808 100.0 80.0
100.0 80.0 52.97718 64.72136
52.97718 64.72136 47.02282 64.72136
47.02282 64.72136 47.02282 35.27864
47.02282 35.27864 0.0 20.0
0.0 20.0 -16.632935 21.748192
-16.632935 21.748192 -32.538931 26.916363
-32.538931 26.916363 -47.02282 35.27864
-47.02282 35.27864 -47.02282 64.72136
-47.02282 64.72136 -32.538931 73.083637

-32.538931 73.083637 -16.632935 78.251808
-16.632935 78.251808 0.0 80.0
0.0 80.0 52.97718 35.27864
52.97718 35.27864 100.0 20.0
100.0 20.0 116.632935 21.748192
116.632935 21.748192 132.538931 26.916363
132.538931 26.916363 147.02282 35.27864

eval naive 2001.318764396512
eval optimized 2269.2195922254655
improvement achieved by post optimization algorithm -267.9008278289534
Path 78-27-102-42-14-110-1-97-53-46-58-59-108-107-22-39-91-79-99-119-12-40-68-23-16-73-103-71-11-52-43-76-94-29-55-62-77-69-111-47-84-88-116-112-28-13-5-61-83-44-10-21-60-26-51-9-65-115-75-41-63-2-33-37-45-96-117-49-74-25-20-109-105-35-17-93-98-101-7-89-80-30-66-104-90-18-106-6-50-87-56-24-118-57-34-0-3-85-82-114-70-8-64-95-54-100-86-36-72-48-19-92-4-81-31-38-67-15-113-3
Path
169.282032 60.0 159.451586 53.530449
159.451586 53.530449 159.451586 46.469551
159.451586 46.469551 169.282032 40.0
169.282032 40.0 176.084521 24.72136
176.084521 24.72136 179.561752 8.362277
179.561752 8.362277 179.561752 -8.362277
179.561752 -8.362277 176.084521 -24.72136
176.084521 -24.72136 169.282032 -40.0
169.282032 -40.0 159.451586 -53.530449
159.451586 -53.530449 147.02282 -64.72136
147.02282 -64.72136 132.538931 -73.083637
132.538931 -73.083637 116.632935 -78.251808
116.632935 -78.251808 100.0 -80.0
100.0 -80.0 83.367065 -78.251808
83.367065 -78.251808 67.461069 -73.083637
67.461069 -73.083637 52.97718 -64.72136
52.97718 -64.72136 47.02282 -64.72136
47.02282 -64.72136 40.548414 -53.530449
40.548414 -53.530449 30.717968 -40.0
30.717968 -40.0 23.915479 -24.72136
23.915479 -24.72136 20.438248 -8.362277
20.438248 -8.362277 20.438248 8.362277
20.438248 8.362277 16.632935 21.748192
16.632935 21.748192 23.915479 24.72136
23.915479 24.72136 32.538931 26.916363
32.538931 26.916363 30.717968 40.0
30.717968 40.0 40.548414 46.469551
40.548414 46.469551 40.548414 53.530449
40.548414 53.530449 30.717968 60.0
30.717968 60.0 32.538931 73.083637
32.538931 73.083637 23.915479 75.27864
23.915479 75.27864 16.632935 78.251808
16.632935 78.251808 20.438248 91.637723
20.438248 91.637723 20.438248 108.362277
20.438248 108.362277 23.915479 124.72136
23.915479 124.72136 30.717968 140.0
30.717968 140.0 40.548414 153.530449
40.548414 153.530449 47.02282 164.72136
47.02282 164.72136 52.97718 164.72136
52.97718 164.72136 59.451586 153.530449
59.451586 153.530449 69.282032 140.0
69.282032 140.0 76.084521 124.72136

76.084521 124.72136 79.561752 108.362277
79.561752 108.362277 79.561752 91.637723
79.561752 91.637723 83.367065 78.251808
83.367065 78.251808 76.084521 75.27864
76.084521 75.27864 67.461069 73.083637
67.461069 73.083637 69.282032 60.0
69.282032 60.0 59.451586 53.530449
59.451586 53.530449 59.451586 46.469551
59.451586 46.469551 69.282032 40.0
69.282032 40.0 67.461069 26.916363
67.461069 26.916363 76.084521 24.72136
76.084521 24.72136 83.367065 21.748192
83.367065 21.748192 79.561752 8.362277
79.561752 8.362277 79.561752 -8.362277
79.561752 -8.362277 76.084521 -24.72136
76.084521 -24.72136 69.282032 -40.0
69.282032 -40.0 59.451586 -53.530449
59.451586 -53.530449 32.538931 -73.083637
32.538931 -73.083637 16.632935 -78.251808
16.632935 -78.251808 0.0 -80.0
0.0 -80.0 -16.632935 -78.251808
-16.632935 -78.251808 -32.538931 -73.083637
-32.538931 -73.083637 -47.02282 -64.72136
-47.02282 -64.72136 -59.451586 -53.530449
-59.451586 -53.530449 -69.282032 -40.0
-69.282032 -40.0 -76.084521 -24.72136
-76.084521 -24.72136 -79.561752 -8.362277
-79.561752 -8.362277 -79.561752 8.362277
-79.561752 8.362277 -76.084521 24.72136
-76.084521 24.72136 -69.282032 40.0
-69.282032 40.0 -59.451586 46.469551
-59.451586 46.469551 -59.451586 53.530449
-59.451586 53.530449 -69.282032 60.0
-69.282032 60.0 -76.084521 75.27864
-76.084521 75.27864 -79.561752 91.637723
-79.561752 91.637723 -79.561752 108.362277
-79.561752 108.362277 -76.084521 124.72136
-76.084521 124.72136 -69.282032 140.0
-69.282032 140.0 -59.451586 153.530449
-59.451586 153.530449 -47.02282 164.72136
-47.02282 164.72136 -32.538931 173.083637
-32.538931 173.083637 -16.632935 178.251808
-16.632935 178.251808 0.0 180.0
0.0 180.0 16.632935 178.251808
16.632935 178.251808 32.538931 173.083637
32.538931 173.083637 67.461069 173.083637
67.461069 173.083637 83.367065 178.251808
83.367065 178.251808 100.0 180.0
100.0 180.0 116.632935 178.251808
116.632935 178.251808 132.538931 173.083637
132.538931 173.083637 147.02282 164.72136
147.02282 164.72136 159.451586 153.530449
159.451586 153.530449 169.282032 140.0
169.282032 140.0 176.084521 124.72136
176.084521 124.72136 179.561752 108.362277
179.561752 108.362277 179.561752 91.637723
179.561752 91.637723 176.084521 75.27864
176.084521 75.27864 147.02282 64.72136
147.02282 64.72136 132.538931 73.083637
132.538931 73.083637 116.632935 78.251808

116.632935 78.251808 100.0 80.0
100.0 80.0 52.97718 64.72136
52.97718 64.72136 47.02282 64.72136
47.02282 64.72136 47.02282 35.27864
47.02282 35.27864 0.0 20.0
0.0 20.0 -16.632935 21.748192
-16.632935 21.748192 -32.538931 26.916363
-32.538931 26.916363 -47.02282 35.27864
-47.02282 35.27864 -47.02282 64.72136
-47.02282 64.72136 -32.538931 73.083637
-32.538931 73.083637 -16.632935 78.251808
-16.632935 78.251808 0.0 80.0
0.0 80.0 52.97718 35.27864
52.97718 35.27864 100.0 20.0
100.0 20.0 116.632935 21.748192
116.632935 21.748192 132.538931 26.916363
132.538931 26.916363 147.02282 35.27864

```
WENIGERKRUMM 4
nodes 25 edges 300
time to find init solution 0 seconds
found init solution
Path
20.212169 156.013261 33.379688 100.161238
33.379688 100.161238 28.913721 58.69988
28.913721 58.69988 51.00814 5.769601
51.00814 5.769601 42.137753 -60.319863
42.137753 -60.319863 -82.864121 -104.1736
-82.864121 -104.1736 -98.760442 -81.770618
-98.760442 -81.770618 -137.317503 -20.146939
-137.317503 -20.146939 -191.716829 -28.360492
-191.716829 -28.360492 -221.149792 -32.862538
-221.149792 -32.862538 -239.848226 8.671399
-239.848226 8.671399 -239.414022 40.427118
-239.414022 40.427118 -240.369194 57.426131
-240.369194 57.426131 -219.148505 103.685337
-219.148505 103.685337 -107.988514 185.173669
-107.988514 185.173669 139.446709 0.233238
139.446709 0.233238 153.130159 -20.36091
153.130159 -20.36091 144.832862 -43.476284
144.832862 -43.476284 -16.72313 -12.689542
-16.72313 -12.689542 -20.971208 -5.637107
-20.971208 -5.637107 -154.088455 115.022553
-154.088455 115.022553 -119.026308 168.453598
-119.026308 168.453598 101.498782 33.484198
101.498782 33.484198 94.789917 -67.087689
94.789917 -67.087689 -129.104485 -155.04164

eval naive 2197.965650598992
eval optimized 1736.980701656645
improvement achieved by post optimization algorithm 460.98494894234705
Path 0-13-4-15-1-18-14-12-23-6-11-22-17-7-19-16-24-10-5-20-8-2-9-21-
Path
20.212169 156.013261 28.913721 58.69988
28.913721 58.69988 101.498782 33.484198
101.498782 33.484198 94.789917 -67.087689
94.789917 -67.087689 -82.864121 -104.1736
-82.864121 -104.1736 -98.760442 -81.770618
-98.760442 -81.770618 -137.317503 -20.146939
-137.317503 -20.146939 -191.716829 -28.360492
-191.716829 -28.360492 -221.149792 -32.862538
-221.149792 -32.862538 -239.848226 8.671399
-239.848226 8.671399 -239.414022 40.427118
-239.414022 40.427118 -240.369194 57.426131
-240.369194 57.426131 -219.148505 103.685337
-219.148505 103.685337 -154.088455 115.022553
-154.088455 115.022553 -119.026308 168.453598
-119.026308 168.453598 -107.988514 185.173669
-107.988514 185.173669 33.379688 100.161238
33.379688 100.161238 -20.971208 -5.637107
-20.971208 -5.637107 -16.72313 -12.689542
-16.72313 -12.689542 51.00814 5.769601
51.00814 5.769601 139.446709 0.233238
139.446709 0.233238 153.130159 -20.36091
153.130159 -20.36091 144.832862 -43.476284
144.832862 -43.476284 42.137753 -60.319863
42.137753 -60.319863 -129.104485 -155.04164
```

WENIGERKRUMM 5
nodes 60 edges 1770
time to find init solution 61 seconds
naiv path
Path 0-7-38-33-47-45-48-20-21-55-16-59-37-22-58-32-9-29-35-4-2-28-14-18-
46-36-53-15-27-40-56-41-34-19-24-11-49-51-44-25-23-43-1-54-50-3-13-6-12-
10-42-30-5-8-17-31-52-26-39-5
Path
-281.67899 -187.717923 62.366656 50.713913
62.366656 50.713913 27.706327 169.284192
27.706327 169.284192 -30.991436 186.807059
-30.991436 186.807059 -177.685937 158.265884
-177.685937 158.265884 -251.656688 -195.189953
-251.656688 -195.189953 -260.477802 -196.955535
-260.477802 -196.955535 -278.409792 -111.914073
-278.409792 -111.914073 141.513053 2.821137
141.513053 2.821137 283.989938 -101.866465
283.989938 -101.866465 263.236651 -144.293091
263.236651 -144.293091 -41.263039 -144.118212
-41.263039 -144.118212 -104.781549 158.212048
-104.781549 158.212048 -49.447381 173.210759
-49.447381 173.210759 36.599805 147.88535
36.599805 147.88535 116.702667 132.021991
116.702667 132.021991 162.493244 -84.574019
162.493244 -84.574019 30.366828 -167.573232
30.366828 -167.573232 -202.218443 -178.735864
-202.218443 -178.735864 -280.008136 11.657786
-280.008136 11.657786 -258.868593 166.669198
-258.868593 166.669198 47.040512 141.206562
47.040512 141.206562 142.765554 -118.682439
142.765554 -118.682439 90.584569 -164.218416
90.584569 -164.218416 -235.099412 47.810306
-235.099412 47.810306 -223.039999 171.558368
-223.039999 171.558368 38.65473 188.608557
38.65473 188.608557 51.417675 146.417721
51.417675 146.417721 106.03343 69.754891
106.03343 69.754891 -57.266232 -115.737582
-57.266232 -115.737582 -247.341131 -160.277639
-247.341131 -160.277639 -286.024059 -55.955204
-286.024059 -55.955204 -70.183535 73.738342
-70.183535 73.738342 -86.45758 105.836348
-86.45758 105.836348 -82.17351 119.465553
-82.17351 119.465553 209.544977 94.267052
209.544977 94.267052 253.534863 38.014987
253.534863 38.014987 26.451074 -192.813352
26.451074 -192.813352 -116.831788 -191.380552
-116.831788 -191.380552 -136.787038 79.501703
-136.787038 79.501703 -68.446198 137.178953
-68.446198 137.178953 92.639946 22.21603
92.639946 22.21603 239.63955 79.491132
239.63955 79.491132 244.228552 119.192512
244.228552 119.192512 -44.669924 170.088013
-44.669924 170.088013 -81.384378 32.368323
-81.384378 32.368323 -74.639411 25.542881
-74.639411 25.542881 -31.548604 -55.223912
-31.548604 -55.223912 -58.684205 -76.988884
-58.684205 -76.988884 -147.023475 -166.22013
-147.023475 -166.22013 -214.362324 -193.26519
-214.362324 -193.26519 -234.711279 -162.774591
-234.711279 -162.774591 -284.547616 -60.154961

-284.547616 -60.154961 -95.621797 77.468533
-95.621797 77.468533 -8.936916 13.543851
-8.936916 13.543851 25.098172 35.205544
25.098172 35.205544 45.123674 31.740242
45.123674 31.740242 63.541591 55.140221
63.541591 55.140221 171.595574 135.520994
171.595574 135.520994 267.845908 127.627482

eval naive 9277.77985102936
eval optimized 3381.157719454162
improvement achieved by post optimization algorithm 5896.622131575197
Path 0-48-45-42-35-46-25-8-3-13-6-12-40-59-29-14-9-21-43-27-11-1-49-55-
16-18-51-44-10-30-56-20-5-41-4-2-36-47-37-22-50-33-53-15-28-58-38-23-24-
19-34-17-31-52-7-26-32-39-54-5
Path
-281.67899 -187.717923 -260.477802 -196.955535
-260.477802 -196.955535 -251.656688 -195.189953
-251.656688 -195.189953 -214.362324 -193.26519
-214.362324 -193.26519 -202.218443 -178.735864
-202.218443 -178.735864 -235.099412 47.810306
-235.099412 47.810306 -136.787038 79.501703
-136.787038 79.501703 -95.621797 77.468533
-95.621797 77.468533 -81.384378 32.368323
-81.384378 32.368323 -74.639411 25.542881
-74.639411 25.542881 -31.548604 -55.223912
-31.548604 -55.223912 -58.684205 -76.988884
-58.684205 -76.988884 -57.266232 -115.737582
-57.266232 -115.737582 -41.263039 -144.118212
-41.263039 -144.118212 30.366828 -167.573232
30.366828 -167.573232 142.765554 -118.682439
142.765554 -118.682439 162.493244 -84.574019
162.493244 -84.574019 141.513053 2.821137
141.513053 2.821137 92.639946 22.21603
92.639946 22.21603 106.03343 69.754891
106.03343 69.754891 209.544977 94.267052
209.544977 94.267052 239.63955 79.491132
239.63955 79.491132 253.534863 38.014987
253.534863 38.014987 283.989938 -101.866465
283.989938 -101.866465 263.236651 -144.293091
263.236651 -144.293091 90.584569 -164.218416
90.584569 -164.218416 26.451074 -192.813352
26.451074 -192.813352 -116.831788 -191.380552
-116.831788 -191.380552 -147.023475 -166.22013
-147.023475 -166.22013 -234.711279 -162.774591
-234.711279 -162.774591 -247.341131 -160.277639
-247.341131 -160.277639 -278.409792 -111.914073
-278.409792 -111.914073 -284.547616 -60.154961
-284.547616 -60.154961 -286.024059 -55.955204
-286.024059 -55.955204 -280.008136 11.657786
-280.008136 11.657786 -258.868593 166.669198
-258.868593 166.669198 -223.039999 171.558368
-223.039999 171.558368 -177.685937 158.265884
-177.685937 158.265884 -104.781549 158.212048
-104.781549 158.212048 -49.447381 173.210759
-49.447381 173.210759 -44.669924 170.088013
-44.669924 170.088013 -30.991436 186.807059
-30.991436 186.807059 38.65473 188.608557
38.65473 188.608557 51.417675 146.417721
51.417675 146.417721 47.040512 141.206562
47.040512 141.206562 36.599805 147.88535

36.599805 147.88535 27.706327 169.284192
27.706327 169.284192 -68.446198 137.178953
-68.446198 137.178953 -82.17351 119.465553
-82.17351 119.465553 -86.45758 105.836348
-86.45758 105.836348 -70.183535 73.738342
-70.183535 73.738342 -8.936916 13.543851
-8.936916 13.543851 25.098172 35.205544
25.098172 35.205544 45.123674 31.740242
45.123674 31.740242 62.366656 50.713913
62.366656 50.713913 63.541591 55.140221
63.541591 55.140221 116.702667 132.021991
116.702667 132.021991 171.595574 135.520994
171.595574 135.520994 244.228552 119.192512
244.228552 119.192512 267.845908 127.627482

WENIGERKRUMM 6
nodes 80 edges 3160
all starting node nearest neighbour heuristic found best tour with length 4362.616434557695 for starting node Node[id=10, point=Point[x=-55.091518, y=198.826966]]
time to find init solution 2 seconds
found init solution
Path
-55.091518 198.826966 -89.453831 162.237392
-89.453831 162.237392 -100.569041 140.808607
-100.569041 140.808607 -97.391662 124.120512
-97.391662 124.120512 -102.699992 95.632069
-102.699992 95.632069 -139.74158 57.93668
-139.74158 57.93668 -175.118239 77.842636
-175.118239 77.842636 -194.986965 101.363745
-194.986965 101.363745 -167.994506 138.195365
-167.994506 138.195365 21.067444 122.164599
21.067444 122.164599 20.21829 88.031173
20.21829 88.031173 56.716498 66.959624
56.716498 66.959624 58.019653 49.937906
58.019653 49.937906 58.71662 32.83593
58.71662 32.83593 40.327635 19.216022
40.327635 19.216022 14.005617 -14.015334
14.005617 -14.015334 -4.590656 -40.067226
-4.590656 -40.067226 -31.745416 -69.20796
-31.745416 -69.20796 -51.343758 -57.654823
-51.343758 -57.654823 -72.565291 -24.28182
-72.565291 -24.28182 -90.16019 -25.200829
-90.16019 -25.200829 -126.569816 -30.645224
-126.569816 -30.645224 -144.887799 -73.49541
-144.887799 -73.49541 -189.988471 -98.043874
-189.988471 -98.043874 -191.216327 -162.689024
-191.216327 -162.689024 -187.485329 -177.031237
-187.485329 -177.031237 21.176627 -165.421555
21.176627 -165.421555 46.674278 -193.090008
46.674278 -193.090008 152.102728 -193.381252
152.102728 -193.381252 157.588994 -144.200765
157.588994 -144.200765 143.114152 -135.866707
143.114152 -135.866707 134.692592 -102.152826
134.692592 -102.152826 150.526118 -88.230057
150.526118 -88.230057 155.405344 -56.437901
155.405344 -56.437901 138.136997 -31.348808
138.136997 -31.348808 112.833346 -38.057607
112.833346 -38.057607 76.647124 -7.289705
76.647124 -7.289705 81.740403 10.276251
81.740403 10.276251 102.909291 60.107868
102.909291 60.107868 100.006932 76.579303
100.006932 76.579303 121.661135 85.267672
121.661135 85.267672 120.906436 131.79881
120.906436 131.79881 132.794476 135.681392
132.794476 135.681392 154.870684 140.32766
154.870684 140.32766 175.677917 98.929343
175.677917 98.929343 172.836936 59.184233
172.836936 59.184233 121.392544 56.694081
121.392544 56.694081 83.005905 64.646774
83.005905 64.646774 64.559003 82.567627
64.559003 82.567627 73.689751 110.224271
73.689751 110.224271 85.04383 108.946389
85.04383 108.946389 96.781707 141.370805
96.781707 141.370805 246.621634 101.705861

246.621634 101.705861 255.134924 115.594915
255.134924 115.594915 277.821597 104.262606
277.821597 104.262606 289.298882 56.051342
289.298882 56.051342 245.415055 44.794067
245.415055 44.794067 231.944823 82.961057
231.944823 82.961057 228.929427 82.624982
228.929427 82.624982 151.432196 121.427337
151.432196 121.427337 102.223372 174.201904
102.223372 174.201904 49.091876 150.678826
49.091876 150.678826 92.25582 -93.514104
92.25582 -93.514104 187.669263 -122.655771
187.669263 -122.655771 210.186432 -127.403563
210.186432 -127.403563 221.028639 -139.435079
221.028639 -139.435079 216.82592 -152.024123
216.82592 -152.024123 -19.310012 -131.810965
-19.310012 -131.810965 -18.507391 -22.90527
-18.507391 -22.90527 155.341949 -20.252779
155.341949 -20.252779 238.583388 -133.143524
238.583388 -133.143524 245.020791 -167.448848
245.020791 -167.448848 242.810288 -182.054289
242.810288 -182.054289 202.34698 -189.069699
202.34698 -189.069699 64.943433 -119.784474
64.943433 -119.784474 19.765322 -99.2364
19.765322 -99.2364 -107.196865 -77.792599
-107.196865 -77.792599 -154.225945 -135.522059
-154.225945 -135.522059 -288.744132 -173.349893
-288.744132 -173.349893 -293.833463 -165.440105

eval naive 4362.616434557695
eval optimized 3740.816749985773
improvement achieved by post optimization algorithm 621.7996845719222
Path 10-1-74-58-23-54-28-45-40-20-21-78-13-26-51-18-0-3-16-14-65-59-43-
22-17-5-57-29-75-63-8-53-47-55-73-52-24-6-76-44-25-30-38-36-41-35-32-42-
34-66-11-71-37-19-12-79-56-31-67-62-15-46-33-9-39-64-50-48-27-72-61-7-2-
60-70-49-4-77-68-6
Path
-55.091518 198.826966 -89.453831 162.237392
-89.453831 162.237392 -100.569041 140.808607
-100.569041 140.808607 -167.994506 138.195365
-167.994506 138.195365 -194.986965 101.363745
-194.986965 101.363745 -175.118239 77.842636
-175.118239 77.842636 -139.74158 57.93668
-139.74158 57.93668 -102.699992 95.632069
-102.699992 95.632069 -97.391662 124.120512
-97.391662 124.120512 21.067444 122.164599
21.067444 122.164599 49.091876 150.678826
49.091876 150.678826 102.223372 174.201904
102.223372 174.201904 154.870684 140.32766
154.870684 140.32766 151.432196 121.427337
151.432196 121.427337 121.661135 85.267672
121.661135 85.267672 100.006932 76.579303
100.006932 76.579303 102.909291 60.107868
102.909291 60.107868 121.392544 56.694081
121.392544 56.694081 138.136997 -31.348808
138.136997 -31.348808 112.833346 -38.057607
112.833346 -38.057607 76.647124 -7.289705
76.647124 -7.289705 81.740403 10.276251
81.740403 10.276251 83.005905 64.646774
83.005905 64.646774 85.04383 108.946389
85.04383 108.946389 73.689751 110.224271

73.689751 110.224271 20.21829 88.031173
20.21829 88.031173 -18.507391 -22.90527
-18.507391 -22.90527 -72.565291 -24.28182
-72.565291 -24.28182 -90.16019 -25.200829
-90.16019 -25.200829 -126.569816 -30.645224
-126.569816 -30.645224 -144.887799 -73.49541
-144.887799 -73.49541 -189.988471 -98.043874
-189.988471 -98.043874 -191.216327 -162.689024
-191.216327 -162.689024 -187.485329 -177.031237
-187.485329 -177.031237 21.176627 -165.421555
21.176627 -165.421555 46.674278 -193.090008
46.674278 -193.090008 152.102728 -193.381252
152.102728 -193.381252 202.34698 -189.069699
202.34698 -189.069699 242.810288 -182.054289
242.810288 -182.054289 245.020791 -167.448848
245.020791 -167.448848 238.583388 -133.143524
238.583388 -133.143524 155.405344 -56.437901
155.405344 -56.437901 155.341949 -20.252779
155.341949 -20.252779 172.836936 59.184233
172.836936 59.184233 245.415055 44.794067
245.415055 44.794067 289.298882 56.051342
289.298882 56.051342 277.821597 104.262606
277.821597 104.262606 255.134924 115.594915
255.134924 115.594915 246.621634 101.705861
246.621634 101.705861 231.944823 82.961057
231.944823 82.961057 228.929427 82.624982
228.929427 82.624982 175.677917 98.929343
175.677917 98.929343 132.794476 135.681392
132.794476 135.681392 120.906436 131.79881
120.906436 131.79881 96.781707 141.370805
96.781707 141.370805 64.559003 82.567627
64.559003 82.567627 56.716498 66.959624
56.716498 66.959624 58.019653 49.937906
58.019653 49.937906 58.71662 32.83593
58.71662 32.83593 40.327635 19.216022
40.327635 19.216022 14.005617 -14.015334
14.005617 -14.015334 -4.590656 -40.067226
-4.590656 -40.067226 19.765322 -99.2364
19.765322 -99.2364 92.25582 -93.514104
92.25582 -93.514104 134.692592 -102.152826
134.692592 -102.152826 150.526118 -88.230057
150.526118 -88.230057 187.669263 -122.655771
187.669263 -122.655771 210.186432 -127.403563
210.186432 -127.403563 221.028639 -139.435079
221.028639 -139.435079 216.82592 -152.024123
216.82592 -152.024123 157.588994 -144.200765
157.588994 -144.200765 143.114152 -135.866707
143.114152 -135.866707 64.943433 -119.784474
64.943433 -119.784474 -19.310012 -131.810965
-19.310012 -131.810965 -31.745416 -69.20796
-31.745416 -69.20796 -51.343758 -57.654823
-51.343758 -57.654823 -107.196865 -77.792599
-107.196865 -77.792599 -154.225945 -135.522059
-154.225945 -135.522059 -288.744132 -173.349893
-288.744132 -173.349893 -293.833463 -165.440105

WENIGERKRUMM 7
nodes 100 edges 4950
all starting node nearest neighbour heuristic found best tour with length 5638.012443173076 for starting node Node[id=15, point=Point[x=126.799911, y=112.72728]]
time to find init solution 0 seconds
naiv path
Path 15-39-16-29-91-3-10-27-60-24-74-83-90-47-46-37-68-11-54-55-30-49-0-61-75-57-94-93-80-14-33-28-70-56-7-96-34-17-98-95-92-22-40-63-18-19-86-78-41-1-71-82-20-77-73-36-25-84-72-66-35-88-32-99-89-9-45-26-85-42-51-59-21-62-69-53-64-4-50-23-52-44-43-48-67-31-38-2-65-5-6-12-97-87-81-8-13-79-76-5
Path
126.799911 112.72728 120.375033 115.889661
120.375033 115.889661 100.043893 161.295125
100.043893 161.295125 95.947213 183.278211
95.947213 183.278211 130.854855 195.695082
130.854855 195.695082 170.514252 161.16985
170.514252 161.16985 169.990437 154.260412
169.990437 154.260412 162.923138 117.465744
162.923138 117.465744 158.742184 62.618834
158.742184 62.618834 178.19836 37.031984
178.19836 37.031984 189.387028 -4.465225
189.387028 -4.465225 205.717887 -24.976511
205.717887 -24.976511 224.599361 -34.798485
224.599361 -34.798485 239.616628 -21.94416
239.616628 -21.94416 276.276517 -49.448662
276.276517 -49.448662 278.105364 -93.771765
278.105364 -93.771765 275.793495 -129.415477
275.793495 -129.415477 235.827007 -143.838844
235.827007 -143.838844 217.599278 -189.258062
217.599278 -189.258062 92.29804 -146.169487
92.29804 -146.169487 59.8272 -170.713714
59.8272 -170.713714 -17.356579 -125.254131
-17.356579 -125.254131 -47.266557 -66.984045
-47.266557 -66.984045 -46.403062 -13.755804
-46.403062 -13.755804 -48.354421 9.091412
-48.354421 9.091412 -42.704691 37.679514
-42.704691 37.679514 -27.911955 48.326745
-27.911955 48.326745 -4.434919 33.164884
-4.434919 33.164884 3.152113 27.10389
3.152113 27.10389 -9.580869 -17.516639
-9.580869 -17.516639 -0.200936 -21.927663
-0.200936 -21.927663 47.011363 -30.88718
47.011363 -30.88718 55.550895 -45.089968
55.550895 -45.089968 88.818853 -42.834512
88.818853 -42.834512 118.989764 -80.203583
118.989764 -80.203583 126.904044 -80.733297
126.904044 -80.733297 172.389228 -53.13327
172.389228 -53.13327 158.552316 -19.254407
158.552316 -19.254407 141.433472 -6.023095
141.433472 -6.023095 135.781192 -13.05344
135.781192 -13.05344 68.910854 -82.123346
68.910854 -82.123346 34.959132 -106.842499
34.959132 -106.842499 -133.730932 -113.306155
-133.730932 -113.306155 -152.130365 -93.844349
-152.130365 -93.844349 -172.378071 -88.298187
-172.378071 -88.298187 -202.828627 -101.70005
-202.828627 -101.70005 -189.135201 -139.078513
-189.135201 -139.078513 -155.651746 -138.151811

-155.651746 -138.151811 -147.363185 59.608175
-147.363185 59.608175 -185.649161 90.144456
-185.649161 90.144456 -189.062172 104.285631
-189.062172 104.285631 -215.113949 120.740679
-215.113949 120.740679 -200.771246 147.741054
-200.771246 147.741054 -201.485143 155.27483
-201.485143 155.27483 -192.681053 174.522947
-192.681053 174.522947 -153.13014 187.817274
-153.13014 187.817274 -120.386562 170.589454
-120.386562 170.589454 -134.985023 132.944989
-134.985023 132.944989 -157.423365 126.800331
-157.423365 126.800331 -207.665172 81.410371
-207.665172 81.410371 -248.169463 80.132237
-248.169463 80.132237 -284.129027 107.252583
-284.129027 107.252583 -287.058297 113.599823
-287.058297 113.599823 -268.739142 143.276483
-268.739142 143.276483 -240.249363 179.334919
-240.249363 179.334919 -222.492322 169.033315
-222.492322 169.033315 -114.146166 190.615321
-114.146166 190.615321 -84.6269 148.216494
-84.6269 148.216494 -126.568914 106.964962
-126.568914 106.964962 -171.354954 25.463068
-171.354954 25.463068 -184.0927 5.737284
-184.0927 5.737284 -226.787625 2.658862
-226.787625 2.658862 -244.959501 -111.046573
-244.959501 -111.046573 -181.208895 -192.622935
-181.208895 -192.622935 106.599423 -107.433987
106.599423 -107.433987 129.024315 29.701695
129.024315 29.701695 74.8875 80.586458
74.8875 80.586458 56.389778 71.618509
56.389778 71.618509 54.551865 71.567133
54.551865 71.567133 40.897139 78.152317
40.897139 78.152317 16.573231 104.020979
16.573231 104.020979 8.64366 135.90743
8.64366 135.90743 54.766523 154.053847
54.766523 154.053847 148.108328 123.558283
148.108328 123.558283 208.592696 136.61846
208.592696 136.61846 216.691 156.31437
216.691 156.31437 217.218893 164.294928
217.218893 164.294928 34.079032 187.731112
34.079032 187.731112 -13.030259 174.698005
-13.030259 174.698005 -56.914543 92.501249
-56.914543 92.501249 -18.316063 27.75586
-18.316063 27.75586 55.434792 62.72916
55.434792 62.72916 57.555364 64.417343
57.555364 64.417343 256.475967 -46.591418
256.475967 -46.591418 296.911892 25.811569
296.911892 25.811569 271.301094 142.524086
271.301094 142.524086 221.808162 186.241012
221.808162 186.241012 53.436521 125.683201
53.436521 125.683201 -211.429137 27.770425
-211.429137 27.770425 -217.28247 -43.316616

eval naive 5638.012443173076
path length optimized 4451.743885428826
improvment of 1186.2685577442498

15-27-10-29-44-25-77-20-82-71-1-66-76-63-40-78-86-19-18-0-49-22-69-96-56-
28-33-14-61-75-57-5-85-84-72-41-42-51-59-35-88-32-99-89-9-73-36-45-26-65-

2-43-79-52-94-6-93-12-64-60-67-31-38-91-16-39-48-3-13-8-81-47-90-83-74-
24-53-97-4-50-23-80-70-92-7-95-98-17-34-87-46-37-68-11-54-55-30-62-21-5

126.799911 112.72728 162.923138 117.465744
162.923138 117.465744 169.990437 154.260412
169.990437 154.260412 95.947213 183.278211
95.947213 183.278211 8.64366 135.90743
8.64366 135.90743 -120.386562 170.589454
-120.386562 170.589454 -201.485143 155.27483
-201.485143 155.27483 -200.771246 147.741054
-200.771246 147.741054 -215.113949 120.740679
-215.113949 120.740679 -189.062172 104.285631
-189.062172 104.285631 -185.649161 90.144456
-185.649161 90.144456 -207.665172 81.410371
-207.665172 81.410371 -211.429137 27.770425
-211.429137 27.770425 -152.130365 -93.844349
-152.130365 -93.844349 -133.730932 -113.306155
-133.730932 -113.306155 -155.651746 -138.151811
-155.651746 -138.151811 -189.135201 -139.078513
-189.135201 -139.078513 -202.828627 -101.70005
-202.828627 -101.70005 -172.378071 -88.298187
-172.378071 -88.298187 -47.266557 -66.984045
-47.266557 -66.984045 -17.356579 -125.254131
-17.356579 -125.254131 34.959132 -106.842499
34.959132 -106.842499 106.599423 -107.433987
106.599423 -107.433987 126.904044 -80.733297
126.904044 -80.733297 88.818853 -42.834512
88.818853 -42.834512 47.011363 -30.88718
47.011363 -30.88718 -0.200936 -21.927663
-0.200936 -21.927663 -9.580869 -17.516639
-9.580869 -17.516639 -46.403062 -13.755804
-46.403062 -13.755804 -48.354421 9.091412
-48.354421 9.091412 -42.704691 37.679514
-42.704691 37.679514 -56.914543 92.501249
-56.914543 92.501249 -126.568914 106.964962
-126.568914 106.964962 -134.985023 132.944989
-134.985023 132.944989 -157.423365 126.800331
-157.423365 126.800331 -147.363185 59.608175
-147.363185 59.608175 -171.354954 25.463068
-171.354954 25.463068 -184.0927 5.737284
-184.0927 5.737284 -226.787625 2.658862
-226.787625 2.658862 -248.169463 80.132237
-248.169463 80.132237 -284.129027 107.252583
-284.129027 107.252583 -287.058297 113.599823
-287.058297 113.599823 -268.739142 143.276483
-268.739142 143.276483 -240.249363 179.334919
-240.249363 179.334919 -222.492322 169.033315
-222.492322 169.033315 -192.681053 174.522947
-192.681053 174.522947 -153.13014 187.817274
-153.13014 187.817274 -114.146166 190.615321
-114.146166 190.615321 -84.6269 148.216494
-84.6269 148.216494 -13.030259 174.698005
-13.030259 174.698005 34.079032 187.731112
34.079032 187.731112 54.766523 154.053847
54.766523 154.053847 53.436521 125.683201
53.436521 125.683201 16.573231 104.020979
16.573231 104.020979 -27.911955 48.326745
-27.911955 48.326745 -18.316063 27.75586
-18.316063 27.75586 -4.434919 33.164884
-4.434919 33.164884 55.434792 62.72916

55.434792 62.72916 74.8875 80.586458
74.8875 80.586458 158.742184 62.618834
158.742184 62.618834 208.592696 136.61846
208.592696 136.61846 216.691 156.31437
216.691 156.31437 217.218893 164.294928
217.218893 164.294928 130.854855 195.695082
130.854855 195.695082 100.043893 161.295125
100.043893 161.295125 120.375033 115.889661
120.375033 115.889661 148.108328 123.558283
148.108328 123.558283 170.514252 161.16985
170.514252 161.16985 221.808162 186.241012
221.808162 186.241012 271.301094 142.524086
271.301094 142.524086 296.911892 25.811569
296.911892 25.811569 239.616628 -21.94416
239.616628 -21.94416 224.599361 -34.798485
224.599361 -34.798485 205.717887 -24.976511
205.717887 -24.976511 189.387028 -4.465225
189.387028 -4.465225 178.19836 37.031984
178.19836 37.031984 129.024315 29.701695
129.024315 29.701695 57.555364 64.417343
57.555364 64.417343 56.389778 71.618509
56.389778 71.618509 54.551865 71.567133
54.551865 71.567133 40.897139 78.152317
40.897139 78.152317 3.152113 27.10389
3.152113 27.10389 55.550895 -45.089968
55.550895 -45.089968 68.910854 -82.123346
68.910854 -82.123346 118.989764 -80.203583
118.989764 -80.203583 135.781192 -13.05344
135.781192 -13.05344 141.433472 -6.023095
141.433472 -6.023095 158.552316 -19.254407
158.552316 -19.254407 172.389228 -53.13327
172.389228 -53.13327 256.475967 -46.591418
256.475967 -46.591418 276.276517 -49.448662
276.276517 -49.448662 278.105364 -93.771765
278.105364 -93.771765 275.793495 -129.415477
275.793495 -129.415477 235.827007 -143.838844
235.827007 -143.838844 217.599278 -189.258062
217.599278 -189.258062 92.29804 -146.169487
92.29804 -146.169487 59.8272 -170.713714
59.8272 -170.713714 -181.208895 -192.622935
-181.208895 -192.622935 -244.959501 -111.046573
-244.959501 -111.046573 -217.28247 -43.316616