Machine Learning zur Vorhersage von Kanten in optimalen Lösungen für das Traveling Salesman Problem

Ahmad Kader

Fachbereich Informatik, Hochschule Bonn Rhein Sieg

31. Mai 2023

Abstract

Das Traveling Salesman Problem gehört zu den NP-Vollständigen Problemen und kann daher nicht in effizienter Zeit mit großen Knotenmengen berechnet werden. Deshalb werden Heuristiken verwendet, die nicht die optimale Lösung berechnen, aber dafür eine Lösungsmenge berechnen können, die „gut genug“ ist und in effizienter Zeit berechnet werden kann. In dieser Arbeit wird sich darum befasst, die Kanten in der Lösungsmenge der Heuristiken und der optimalen Lösung zu vergleichen und auf Ähnlichkeit zu überprüfen und unter anderem herauszufinden, wie viele der Kanten, die in der heuristischen Lösung auftauchen, auch in der optimalen Lösungen enthalten sind. Dazu wurden verschiedene Analysen auf den Daten angewendet und betrachtet. Diese Analysen geben einen guten Einblick darauf, wie die verschiedenen Heuristiken mit Blick auf die optimale Lösung abschneiden. Anschließend wurde untersucht, wie man mithilfe von Machine Learning und einem Random-Forest-Tree die Kanten finden kann, die mit hoher Wahrscheinlichkeit Teil der optimalen Lösung sind. Dabei muss der Random-Forest-Tree die Daten, die zum Testen verwendet werden, nicht gesehen haben.

1 Einleitung

In den letzten Jahren wurden Machine Learning (ML) Methoden immer beliebter, um die Berechnung der optimalen Lösung des Traveling Salesman Problem's (TSP) effizienter zu gestalten. Dazu wird mithilfe einer ML-Methode versucht, Kanten der optimalen Lösung vorherzusagen. Die ML-Methode nimmt dazu verschiedene Merkmale als Parameter und berechnet Kanten für ein TSP. Vorher wird die ML-Methode auf einer gewissen Anzahl von Daten trainiert. In der zweiten Phase testet die ML-Methode auf einer anderen Anzahl von Daten diese berechneten Lösungen des TSP.

Die Forschungsfrage dieser Arbeit beinhaltet, ob ML bessere Lösungen berechnen kann als die Lösungen, die heuristische Algorithmen berechnen. Eine weitere Frage ist noch, wie die heuristischen Lösungen untereinander abschneiden.

Ziel dieser Arbeit war es, eine ähnliche Lösung mithilfe von ML zu implementieren und zu testen. Ein weiteres Ziel war es herauszufinden, wie die Lösungen der heuristischen Algorithmen mit den optimalen Lösungen vergleichbar sind. Als ML-Methode wurde dazu das Random-Forest-Tree (RFT) ausgesucht. Das RFT erreicht höhere Genauigkeiten als single decision tree's und ist eine der simpleren ML-Methoden.

Zu Beginn wurden in dieser Arbeit die Daten, auf denen gearbeitet wurde, generiert. Anschließend wurde eine Reihe von Analysen auf den Daten angewendet. Diese Analysen helfen dabei zu verstehen, wie die verschiedenen Heuristiken untereinander und mit Blick auf die optimale Lösung für das jeweilige TSP abschneiden. Anschließend wurden verschiedene Merkmale erstellt, die als Parameter für das RFT dienen. Diese Merkmale helfen dem RFT dabei zu verstehen, welche Kanten häufiger in der optimalen Lösung auftauchen und teilt diesen eine höhere Relevanz zu. Zum Schluss wird das RFT auf der Hälfte der Daten trainiert und auf der anderen Hälfte getestet. Das RFT gibt dann eine Lösung für das TSP. Die Kanten dieser Lösung werden mithilfe der Merkmale erzeugt.

In dieser Arbeit wird zuerst besprochen, was die Recherche zum Stand der Forschung im Themengebiet TSP geliefert hat. Es wird eine kurze Beschreibung des Traveling Salesman Problems gegeben. Daraufhin werden traditionelle- und Machine Learning Lösungsansätze besprochen. Anschließend wird die Methodik und Herangehensweise dieser Arbeit erklärt. Dazu werden die genutzten Daten und die Analysen auf diesen Daten besprochen. Anschließend werden alle genutzten Merkmale vorgestellt. Im nächsten Kapitel werden die Ergebnisse der Analysen vorgestellt und besprochen. Als Letztes wird eine Schlussfolgerung für diese Arbeit gezogen.

2 Stand der Forschung

Das TSP ist ein Graphenproblem $G = (V, E)$ mit der Knotenmenge $V = \{1, ..., n\}$, einer Kantenmenge $E = \{(u, v)|u, v \in V, u \neq v\}$ und einer Gewichtsfunktion $w(e) \rightarrow \mathbb{Z}^+$ [1]. Das Ziel des TSP ist es, die Tour zu finden, die alle Knoten maximal einmal besucht, im Ursprungsknoten endet und die Summe aller Gewichte minimiert. Schon seit der ersten Erwähnung des Traveling Salesman Problem haben sich viele Forscher damit befasst, Verfahren zu entwickeln, um das Problem optimal zu lösen. Das TSP gilt als einer der meist erforschten kombinatorischen Optimierungsprobleme [1]. Aus diesen Forschungen sind viele Lösungsansätze entstanden, die alle ihre Nachteile mitbringen.

2.1 Traditionelle Ansätze zur Lösung des Traveling Salesman Problems

Der Brute-Force Lösungsansatz ist ein exaktes Lösungsverfahren, der die Länge aller möglichen Touren berechnet und den kleinsten auswählt. Dieser Lösungsansatz ist naiv und benötigt selbst für kleinere TSP Instanzen viel Rechenzeit und ist an sich für viele Anwendungen nicht nutzbar. Eine Alternative dazu ist der Branch-and-Cut Lösungsansatz, der zur Lösung vom Traveling Salesman Problem with Profits (TSPP) verwendet wurde und gezeigt hat, dass es effiziente Lösungen berechnen kann [2]. Dort berechnet das Branch-and-Cut Verfahren nicht das gesamte TSP, sondern führt das Verfahren auf Teilgraphen aus. Ein nicht exakter Lösungsansatz sind heuristische Algorithmen, die nicht die optimale Lösung berechnen, sondern eine Lösung, die als befriedigend eingestuft wird. Diese Heuristiken werden größtenteils eingesetzt, um schnell auf brauchbare Lösungen zu kommen. Die Laufzeiten betreffen bei den meisten heuristischen Algorithmen $O(n^2)$. In dieser Arbeit wurden Analysen an heuristischen Algorithmen vorgenommen und unter anderem gezeigt, wie weit die Lösungen der heuristischen Lösungen von den optimalen Lösungen abweichen. Mit dem Nearest Neighbor Algorithmus und dem 2 opt Algorithmus wurde gezeigt, dass die Kombination von heuristischen Algorithmen in der Lage ist, in effizienter Zeit bessere Lösungen zu berechnen als heuristische Algorithmen an sich [3]. Ein weiterer traditioneller Ansatz ist die dynamische Programmierung, in der das TSP in kleinere Teilprobleme aufgeteilt und die optimale Lösung jedes Teilproblems gespeichert wird. Bei diesem Ansatz werden redundante Berechnungen vermieden, wobei es dadurch zu hohen Laufzeit- und Speicheranforderungen für größere Problemgrößen führt. Es wurde gezeigt, dass die Kombination von dynamischer Programmierung und einem TSP mit bidirektionalen Kanten auch zu einer Lösung führen kann [4].

2.2 Machine Learning-Ansätze zur Lösung des Traveling Salesman Problems

In den letzten Jahren hat sich das Interesse an der Anwendung von ML-Methoden auf das TSP verstärkt. Einer dieser ML-Methoden ist das Reinforcement Learning, in dem ein Agent oder Lernalgorithmus trainiert wird, um schrittweise Entscheidungen zur Auswahl des nächsten Knoten's in der Tour zu treffen. Dazu lernt der Agent durch Belohnungen und Bestrafungen, die auf der Gewichtung der Tour basieren. Der Policy Gradient ist ein solches Verfahren und wurde auch genutzt als Lösungsansatz für das TSP und hat Lösungen in effizienter Zeit berechnet [5]. Des Weiteren können auch genetische Algorithmen verwendet werden, um ein TSP zu lösen. Dazu wird eine zufällige Population von Touren erzeugt. Die Tour wird durch den Crossover und der Mutation verbessert. Der Fitnessscore entspricht der Länge der Tour und entscheidet, welche Touren in die nächste Generation weiter verwendet werden. Der genetische Algorithmus wurde verwendet für TSP's mit sehr vielen Knoten und hat gezeigt, dass es effizienter sein kann als vergleichbare Algorithmen [6]. Ein weiterer Lösungsansatz sind neuronale Netze, bei denen eine Funktion trainiert wird, die die optimale Reihenfolge der Knoten vorhersagt. Der Input des neuronalen Netzes sind die Knoten des TSP. Die Ausgabe sind eine Permutation der Knoten, in der die optimale Reihenfolge ausgegeben wird. Eine solche Lösung wurde mit einem neuronalen Netz und dem Monte Carlo Suchbaum erzielt und zeigt das neuronale Netze kürzere Touren berechnen können als andere Lernalgorithmen [7]. Der letzte Lösungsansatz ist die Kombination von ML-Methoden und Metaheuristiken. Ein Beispiel für eine Kombination ist ein genetischer Algorithmus oder ein lokaler Suchalgorithmus, die durch eine maschinelle Lernkomponente erweitert werden. Diese Kombination wäre in der Lage, die Suche in vielversprechende Bereiche des Lösungsraums zu lenken oder effizientere Operationen durchzuführen.

3 Methodik

3.1 Die verwendeten Daten

Die Daten wurden für diese Arbeit von Professor Peter Becker bereitgestellt. Die Daten bestehen aus 1000 Traveling Salesman Problemen. In diesen Daten wurde unter anderem die Dimension und die Koordinaten der Knoten in der 2D Ebene abgespeichert. Außerdem wurden 1000 optimale Lösungen der TSP's bereitgestellt. In den optimalen Lösungen wurden neben den im TSP gespeicherten Daten auch die optimale Tourlänge und die Tour selbst gespeichert. Zum Schluss wurden je 1000 Lösungen für 8 Heuristiken bereitgestellt. Diese Lösungen beinhalten die gleichen Daten wie in der optimalen Lösung mit der Ausnahme, dass die Tour mit der jeweiligen Heuristik berechnet wurde und nicht optimal ist. Die Datenmenge umfasst also 10.000 Einträge im JSON-Datenformat. Die Dimensionen der Probleme liegen zwischen 300 und 500 Knoten.

Verfahren	Typ
Greedy Algorithmus	Einfügeheuristik
Nearest Neighbor	Eröffnungsheuristik
Farthest Insertion	Einfügeheuristik
Random Insertion	Einfügeheuristik
Nearest Insertion	Einfügeheuristik
Cheapest Insertion	Einfügeheuristik
MST Heuristik	Eröffnungsheuristik
Christofides Algorithmus	Eröffnungsheuristik

Tabelle 1: Die genutzten Heuristiken mit ihren Typen

Bei den Typen der Heuristiken handelt es sich um Einfüge- und Eröffnungsheuristiken. Der Hauptunterschied der Typen besteht darin, wie die Lösungen aufgebaut sind. Eine Eröffnungsheuristik beginnt mit einer leeren Lösung und fügt schrittweise Elemente hinzu, um die endgültige Lösung zu erstellen. Eine Einfügeheuristik hingegen beginnt mit einer vorhandenen Teil- oder Anfangslösung und fügt schrittweise weitere Elemente hinzu, um die Lösung zu verbessern.

3.2 Die Analyse der Daten

Alle 8 Heuristiken wurden auf ihre Approximationsgüte und dem Prozentsatz der Kanten in der optimalen Lösung analysiert. Zur Analyse der Approximationsgüte wurde die folgende Funktion für alle Heuristiken benutzt.

$$\frac{heu - opt}{opt} * 100 \tag{1}$$

Wobei *heu* die Tourlänge der Heuristik eines TSP ist und *opt* die optimale Tourlänge eines TSP ist. Für die Analyse aller Heuristiken wurde diese Formel für alle TSP's verwendet, um schließlich die Approximationsgüte aller Heuristiken zu berechnen. Anschließend wurde der Prozentsatz der Kanten, die in der optimalen Lösung enthalten ist, berechnet. Dazu wurden die Tour Daten in die Form (u, v) für die Kanten umgewandelt. Daraufhin wurde gezählt, ob die Kanten, die in den jeweiligen Heuristiken berechnet wurden, in der optimalen Lösung auftauchen. Diese Anzahl wurde anschließend durch die folgende Formel auf den Prozentsatz umgerechnet.

$$\frac{amount}{dimension} * 100 \tag{2}$$

Wobei *amount* die Anzahl der vorkommen, der Kanten in der jeweiligen Heuristik ist und die *dimension* die Dimension des jeweiligen TSP ist. Als weitere Analyse wurden minimum spanning trees (MST) aus den TSP's berechnet und auch für die MST's wurde die Approximationsgüte und der Prozentsatz der Kanten in der optimalen Lösung berechnet. Da die Knoten als Koordinaten gespeichert waren, mussten die Distanzen der Knoten für die Kanten berechnet werden, dazu wurde die folgende Funktion verwendet [8].

$$xd = x_i - x_j \tag{3}$$

$$yd = y_i - y_j \tag{4}$$

$$d_{ij} = \lfloor \sqrt{xd^2 + yd^2} \rceil \tag{5}$$

Anschließend wurden die Distanzen als Parameter in die minimum spanning tree Funktion des Python Packet's scipy übergeben. Zudem wurden alle bisherigen Ergebnisse in einen Scatter Plot übergeben, um die Güte der Approximation der einzelnen Heuristiken im Überblick zu haben. Zum Schluss der Analyse wurde noch für jedes TSP ein k-relative Neighborhood Graph (k-RNG) berechnet. Für die Berechnung der k-RNG wurde mithilfe der Distanzen überprüft, dass $d(u, v) < d(v, w)$ und $d(u, w) < d(v, w)$ gilt, wobei $u, v, w \in V$ und d die berechnete Distanz ist. Es handelt sich um einen direkten Nachbarn, wenn höchstens k - 1 solcher Knoten gefunden wurden. In dieser Arbeit wurde k = 1 berechnet, deshalb dürfen keine Knoten im Schnitt der Kreise liegen. Zur Überprüfung wurden die Graphen mithilfe der NetworkX Bibliothek gezeichnet. Anschließend wurde eine Analyse erstellt, um zu sehen, wie viele der Kanten einer optimalen Tour im k-RNG enthalten sind. Dazu wurde gezählt wie viele der Kanten, die in der optimalen Lösung enthalten sind, auch im k-RNG liegen. Vergleichend dazu wurde eine weitere Analyse erstellt, um zu überprüfen, wie viele Kanten des Greedy Algorithmus im k-RNG enthalten sind. Auch hier wurde gezählt wie viele der Kanten, die im Greedy Algorithmus enthalten sind, auch im k-RNG liegen. Zum Schluss wurde überprüft, wie viele der Kanten einer optimalen Tour im k-RNG sind, wenn das k variabel ist. Dazu wurde gezählt, wie viele der Knoten im Schnitt der beiden Kreise liegen.

3.3 Die ausgewählten Merkmale

Es wurden 25 Merkmale ausgesucht, die als Parameter für das RFT verwendet werden. Dazu werden den Kanten ein binärer Wert gegeben, wenn das jeweilige Merkmal für die Kante zutrifft. Diese binären Werte werden gespeichert und dem RFT übergeben. Die ausgewählten Merkmale geben Information, ob eine Kante in der optimalen Lösung auftaucht. Dazu wurden 15 Merkmale aus den Arbeiten [1] und [9] verwendet. Außerdem wurden aus den 8 Heuristiken, die in 3.1 besprochen wurden, weitere Merkmale erzeugt. Die letzten 2 Merkmale stammen aus dem k-RNG und den berechneten Graden der MST's.

3.3.1 Merkmal aus der linearen Programmierung

Wie in der Arbeit [1] wurde auch hier das TSP als ein lineares Programm definiert.

$$minimize \quad z = \sum_{i,j=1;i\neq j}^{n} A_{ij}x_{ij} \tag{6}$$

$$subject \quad to \qquad \sum_{i;i\neq j}^{n} x_{ij} = 1, \qquad j = 1, ..., n \tag{7}$$

$$\sum_{j;j\neq i}^{n} x_{ij} = 1, \qquad i = 1, ..., n \tag{8}$$

$$\sum_{(i,j)\in W} x_{ij} \leq |W| - 1, \qquad W \subseteq V; \quad |W| \geq 3 \tag{9}$$

$$x_{ij} \in \{0, 1\}, \qquad i, j = 1, ..., m; \quad i \neq j \tag{10}$$

3.3.2 Merkmal aus den Minimum Spanning Tree's

Das Merkmal, welches aus Minimum Spanning Tree' erzeugt wird, wurde der Arbeit [1] entnohmen. Dazu wird ein neuer Graph $G' = (V, \emptyset)$ erstellt, mit der gleichen Knotenmenge wie das des TSP's und einer leeren Kantenmenge. Anschließend wird ein MST $T = (V, E')$ für diesen Graphen berechnet. Die neu berechnete Kantenmenge E' wird aus der eigentlichen Kantenmenge E von der optimalen Lösung $G = (V, E)$ gelöscht und dem Graphen G' hinzugefügt. Dieser Prozess wird k mal durchgeführt, um möglichst viele Kanten aus der optimalen Lösung zu berechnen. In dieser Arbeit wurde k = 10 gesetzt.

3.3.3 Lokale Merkmale

Ein weiteres Merkmal, welches aus der Arbeit [1] stammt, sind eine Menge von lokalen Merkmalen, die wenig rechenintensiv sind. Diese Merkmale vergleichen dazu die Kantengewichte einer Kante (i, j) mit dem Maximum und Minimum der Gewichte aus E. Dazu werden für jedes $(i, j) \in E$ eine Menge von Merkmalen berechnet.

$$q_{ij}^{a} = (1 + A_{ij})\backslash(1 + \max_{(l,k)\in E} A_{lk}) \tag{11}$$

$$q_{ij}^{b} = (1 + A_{ij})\backslash(1 + \max_{l\in V} A_{il}) \tag{12}$$

$$q_{ij}^{c} = (1 + A_{ij})\backslash(1 + \max_{l\in V} A_{lj}) \tag{13}$$

$$q_{ij}^{d} = (1 + \min_{(l,k)\in E} A_{lk})\backslash(1 + A_{ij}) \tag{14}$$

$$q_{ij}^{e} = (1 + \min_{l\in V} A_{il})\backslash(1 + A_{ij}) \tag{15}$$

$$q_{ij}^{f} = (1 + \min_{l\in V} A_{lj})\backslash(1 + A_{ij}) \tag{16}$$

Wie aus [1] beschrieben, sollen die Merkmale 11 und 14 das Gewicht einer Kante im Vergleich zu den Gewichten des gesamten Graphen darstellen. Im Gegensatz dazu beziehen sich die Merkmale 12, 13, 15 und 16 auf das Gewicht einer Kante im Vergleich zu den Gewichten ihrer direkten Nachbarn.

3.3.4 Merkmal aus reduzierten Graphen

Das letzte Merkmal aus der Arbeit [1] berechnet einen reduzierten Graphen $G' = (V, E' \subset E)$, um sicherzustellen das, dass der Graph G' zusammenhängend und hamiltonsch ist, wird überprüft, ob die Kanten in G' zu einer bekannten Lösung gehören. Anschließend können weitere Lösungen berechnet werden, indem eine beliebige Permutation der Knoten verwendet wird. Unter Annahme, dass das Problem metrisch ist, können Heuristiken verwendet werden, um weitere zusammenhängende und hamiltonsche Lösungen zu berechnen.

3.3.5 Merkmale aus Graphen

Das erste Merkmal aus der Arbeit [9] berechnet 4 Merkmale zur Beschreibung der Gewichte C der Kanten $e_{ij}, i, j = 1, ..., n$.

$$f_1(e_{ij}) = \frac{c_{ij} - \min_{k=1,\dots,n} c_{ik}}{\max_{k=1,\dots,n} c_{ik} - \min_{k=1,\dots,n} c_{ik}} \tag{17}$$

$$f_2(e_{ij}) = \frac{c_{ij} - \min_{k=1,\dots,n} c_{kj}}{\max_{k=1,\dots,n} c_{kj} - \min_{k=1,\dots,n} c_{kj}} \tag{18}$$

$$f_3(e_{ij}) = \frac{c_{ij} - \sum_{k=1}^{n} c_{ik}\backslash n}{\max_{k=1,\dots,n} c_{ik} - \min_{k=1,\dots,n} c_{ik}} \tag{19}$$

$$f_4(e_{ij}) = \frac{c_{ij} - \sum_{k=1}^{n} c_{kj}\backslash n}{\max_{k=1,\dots,n} c_{kj} - \min_{k=1,\dots,n} c_{kj}} \tag{20}$$

Wie aus [9] beschrieben, berechnen die Merkmale 17 und 18 den Unterschied zwischen den Kantenkosten einer Kante und den minimalen Kosten, die von einem Knoten ausgehen bzw. in einen Knoten führen. Andererseits basieren 19 und 20 auf den durchschnittlichen Kantenkosten, die mit den Knoten verbunden sind. Diese Merkmale werden durch den Unterschied zwischen den maximalen und minimalen Kantenkosten normalisiert. Sie erfassen lokale Eigenschaften einer Kante.

3.3.6 Statistische Merkmale

Das letzte Merkmal der Arbeit [9] ist das Merkmal, welches aus statistischen Messungen berechnet wird. Ziel dieser statistischen Messungen ist es, Kanten zu finden, die in optimalen Lösungen auftauchen. Dazu werden zusammenhängende und hamiltonsche Touren generiert und mithilfe der folgenden Funktion wird den Touren eine aufsteigende Reihenfolge mit Blick auf ihren Zielwert gegeben.

$$f_r(e_{ij}) = \sum_{k=1}^{m} \frac{x_{ij}^k}{r^k} \tag{21}$$

Wobei x^k einen binären Wert einnimmt, je nachdem, ob die Kante in der optimalen Lösung enthalten ist oder nicht. r^k hingegen gibt die Reihenfolge in der k-ten Tour an. Anschließend werden die Reihenfolgen normalisiert, indem sie durch das Maximum dividiert werden. Diese Normalisierung ist das erste Merkmal der statistischen Messung.

$$f_5(e_{ij}) = \frac{f_r(e_{ij})}{\max_{p,q=1,\dots,n} f_r(e_{pq})} \tag{22}$$

Das 2. Merkmal basiert auf der Pearson Korrelation zwischen den x_{ij} und den Zielwerten der generierten Touren.

$$f_c(e_{ij}) = \frac{\sum_{k=1}^{m}(x_{ij}^k - \bar{x}_{ij})(y^k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{ij}^k - \bar{x}_{ij})^2}\sqrt{\sum_{k=1}^{m}(y^k - \bar{y})^2}} \tag{23}$$

Wobei $\bar{x}_{ij} = \sum_{k=1}^{m} x_{ij}^k \backslash m$ und $\bar{y}_{ij} = \sum_{k=1}^{m} y^k \backslash m$ gilt. Anschließend wird auch dieser Wert normalisiert, um negativ korrelierte Werte nicht in die optimale Lösung zu lassen. Dazu wird der Wert, der aus 23 berechnet wird, durch das Minimum der Korrelation dividiert.

$$f_6(e_{ij}) = \frac{f_c(e_{ij})}{\min_{p,q=1,\dots,n} f_c(e_{pq})} \tag{24}$$

3.3.7 Merkmale aus den Heuristiken

Das erste Merkmal, welches nicht aus anderen Quellen stammt, wird aus den Heuristiken berechnet, die in 3.1 besprochen wurden. Dabei wurde für jede Heuristik ein Merkmal berechnet, was dazu führt, das insgesamt 8 Merkmale berechnet wurden. Dabei wurde für jede Heuristik berechnet, ob die Kanten in der optimalen Lösung auftauchen. Dazu wurde die Reihenfolge der Kantenmenge jeder Heuristik verwendet, um festzustellen, ob eine bestimmte Kante in der Kantenmenge der optimalen Lösung enthalten ist, indem ein binärer Wert x_{ij}^H gespeichert wurde. Wobei i der Index des Problems, j der Index der Kante und H die jeweilige Heuristik darstellt. Falls die Kante nicht in der optimalen Lösung auftaucht, dann wird eine 0 abgespeichert, sonst eine 1. Diese Menge der binären Werte wird als Merkmal benutzt.

Algorithm 1 Kantenvergleich zweier Graphen

```
1:  procedure VergleicheGraphen(G_H, G_O)
2:      M ← {}                                  ▷ Leere Menge für binäre Werte
3:      E_H ← Kanten von G_H
4:      E_0 ← Kanten von G_O
5:      for e ∈ E_H do
6:          if e ∈ E_O then
7:              Füge 1 zu M hinzu
8:          else
9:              Füge 0 zu M hinzu
10:     return M                                ▷ Rückgabe der Menge mit den binären Werten
```

3.3.8 Merkmal mithilfe von k-RNG's

Bei diesem Merkmal wurde berechnet, wie viele Knoten im Schnitt der Kreise innerhalb einer Kante liegen. Dafür wurde in der heuristischen Tour gezählt, wie viele Knoten k gibt es, wobei $d(k,i) < d(i,j)$ und $d(k,j) < d(i,j)$ und $i \neq j$ für eine Kante $\{i,j\}$ gilt. Gleichzeitig bedeutet dies auch das kleinste k, wodurch die Kante $\{i,j\}$ im $(k+1)$-RNG enthalten ist. Der nachfolgende Algorithmus beschreibt das Vorgehen noch mal in Pseudocode.

Algorithm 2 Berechnung der Anzahl der Knoten im Schnitt der Kreise innerhalb einer Kante

```
1: procedure BerechneKnotenSchnitt(G = (V, E))
2:     K ← {}                                          ▷ Leere Menge für k
3:     for {i, j} ∈ Kanten von G do
4:         k ← 0
5:         for Knoten u ∈ G do
6:             if d(u, i) < d(i, j) and d(u, j) < d(i, j) and u ≠ i ≠ j then
7:                 k ← k + 1
8:         Füge k zu K hinzu
9:     return K                                        ▷ Rückgabe der Menge k
```

3.3.9 MST zur Wahrscheinlichkeit des Vorkommens einer Kante

Anstatt zu berechnen, ob eine Kante $\{i, j\}$ in einem MST auftaucht, berechnet das letzte Merkmal, wie genau es auftaucht. Die genaue Ausgestaltung dieses Merkmals ist offen und müsste noch weiter Bedacht werden. Im Nachfolgenden wird ein Prototyp dieses Merkmals vorgestellt. Dazu wird der Grad deg der Knoten für jeden Knoten im MST berechnet. Der Grad in diesem Fall ist, wie viele Kanten von dem Knoten ausgehen bzw. ankommen. Anschließend wird für einer Kante $\{i, j\}$ die Maxima $\max deg(i)$ und $\max deg(j)$ berechnet. Je geringer dieses Maximum ist, desto wahrscheinlicher ist es, dass diese Kante in der optimalen Lösung enthalten ist. Dazu könnten die Kanten, deren Maximum unter einer definierten Schranke k liegen, in einer Menge als 1 abgespeichert werden und als 0, wenn nicht. Ein möglicher Algorithmus wird nachfolgend als Pseudocode dargestellt.

Algorithm 3 Berechnung des genauen Auftretens einer Kante im MST

```
1: procedure BerechneKantenAuftreten(G = (V, E), k)
2:     A ← {}                                          ▷ Leere Menge für das Kantenauftreten
3:     for {i, j} ∈ Kanten von G do
4:         max_deg_i ← max deg(i)
5:         max_deg_j ← max deg(j)
6:         if min(max_deg_i, max_deg_j) < k then
7:             Füge 1 zu A hinzu
8:         else
9:             Füge 0 zu A hinzu
10:    return A                                        ▷ Rückgabe der Menge für das Kantenauftretens
```

Wie schon erwähnt, ist das nur ein erster Prototyp, der noch weiter analysiert und bedacht werden müsste und nicht für die endgültige Nutzung des Merkmals genutzt werden sollte.

3.3.10 Random Forest Tree zur Klassifikation der Graphen

Zur Klassifikation der Graphen wird eine RFT verwendet, der die vorgestellten Merkmale als Parameter bekommt. Der RFT wird dazu genutzt, um Vorhersagen zu treffen, ob eine Kante eines Graphen, welches mithilfe des Greedy Algorithmus berechnet wurde, auch in der optimalen Lösung enthalten sein könnte. Dazu lernt das RFT im 1. Schritt mithilfe 50% der Greedy Touren und der Merkmale, welche Kanten Teil einer optimalen Lösung sein könnten. Im 2. Schritt werden die anderen 50% der Greedy Touren und die optimalen Touren verwendet, um das RFT zu testen. Dabei werden die optimalen Touren zur Evaluierung des RFT und zur Klassifikation der Kanten verwendet. Die Anzahl der Greedy- und optimalen Touren beläuft sich auf 1000 Daten. 500 der Greedy Touren werden für das lernen des RFT und 500 für das Testen des RFT genutzt. Jedes Merkmal berechnet eine Vielzahl von Daten, die es dem RFT leichter machen werden, die korrekte Vorhersage zu machen.

4 Ergebnisse

Aus zeitlichen Gründen konnten die Merkmale und das RFT nicht implementiert, trainiert oder getestet werden. Im Nachfolgenden werden die Ergebnisse der Analysen vorgestellt. Die Analysen wurden in Python implementiert und mithilfe des NetworkX Pakets [10] dargestellt. Zudem wurden die nachfolgenden Diagramme mithilfe der matplotlib Bibliothek dargestellt. Die MST's

wurden durch die Benutzung der mimimum_spanning_tree Funktion innerhalb der scipy Bibliothek berechnet. Die Importierung der JSON Daten wurde mithilfe des JSON Pakets durchgeführt. Alle Analysen wurden auf einem privaten Computer ausgeführt. Der Computer ist mit einem AMD Ryzen 9 5900X CPU mit 12 Kernen und einer RTX 3090 GPU ausgestattet. Außerdem hat der Computer 32 GB Arbeitsspeicher und das genutzte Betriebssystem war Windows 10.

4.1 Analyse der Heuristiken

Wie schon in 3.1 erwähnt, liegen je 1000 Lösungen für die 1000 TSP's vor, die mithilfe der 8 Heuristiken berechnet wurden. Die Ergebnisse der Analysen der Heuristiken wurden mithilfe von Histogrammen und Boxplots grafisch dargestellt. In den Analysen wurden 2 Werte für jede Heuristik untersucht. Erstens wurde analysiert, wie gut die Approximationsgüte der Heuristik ist. Das bedeutet, wie sehr sich die Lösung der Heuristik von der optimalen Lösung unterscheidet. Außerdem wurde untersucht, wie viele Kanten, die in der Heuristik enthalten sind, auch in der optimalen Lösung enthalten sind. Für jede Heuristik wurden beide Werte in Prozent berechnet. Zudem wurden für beide Werte je ein Histogramm und ein Boxplot erstellt.

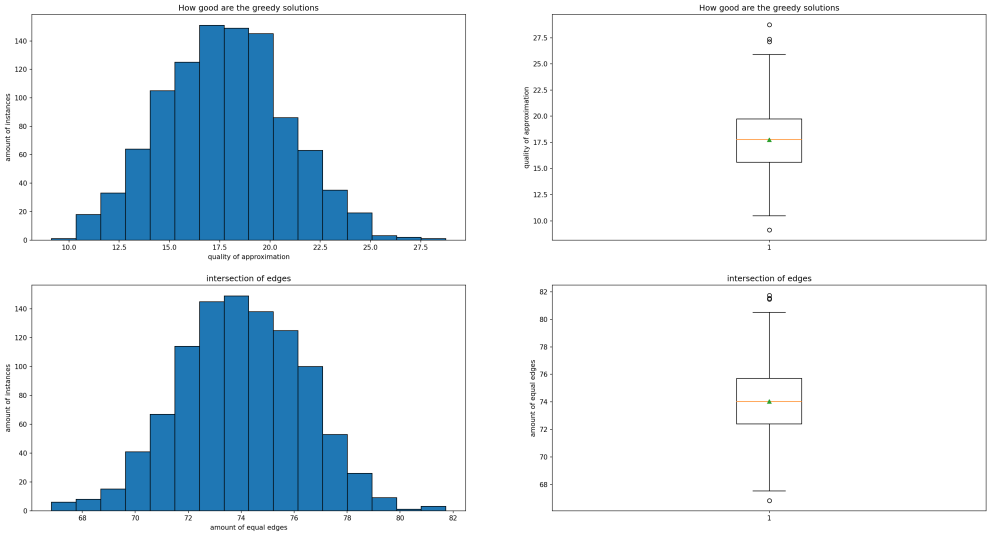

Abbildung 1: Ergebnisse der Analyse des Greedy Algorithmus

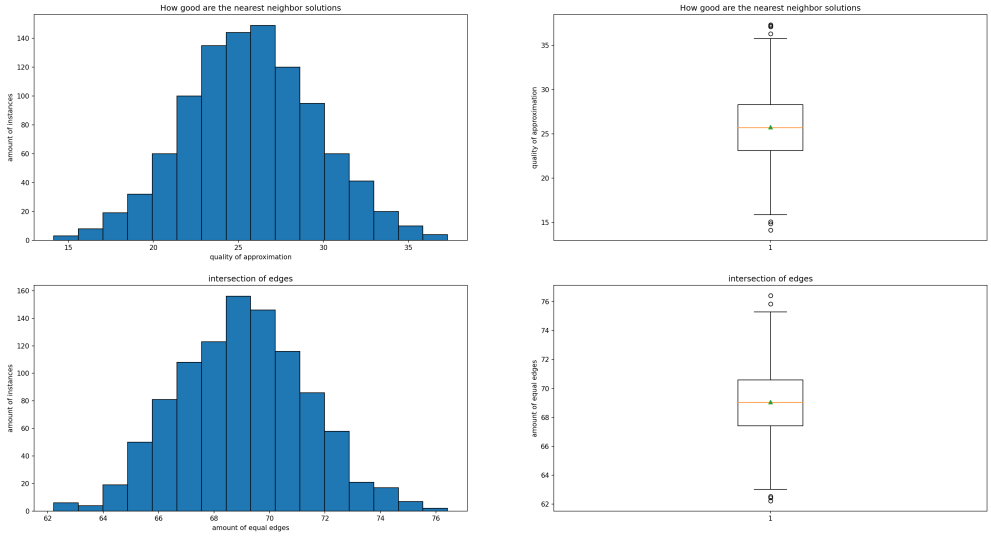

Abbildung 2: Ergebnisse der Analyse der Nearest Neighbor Heuristik

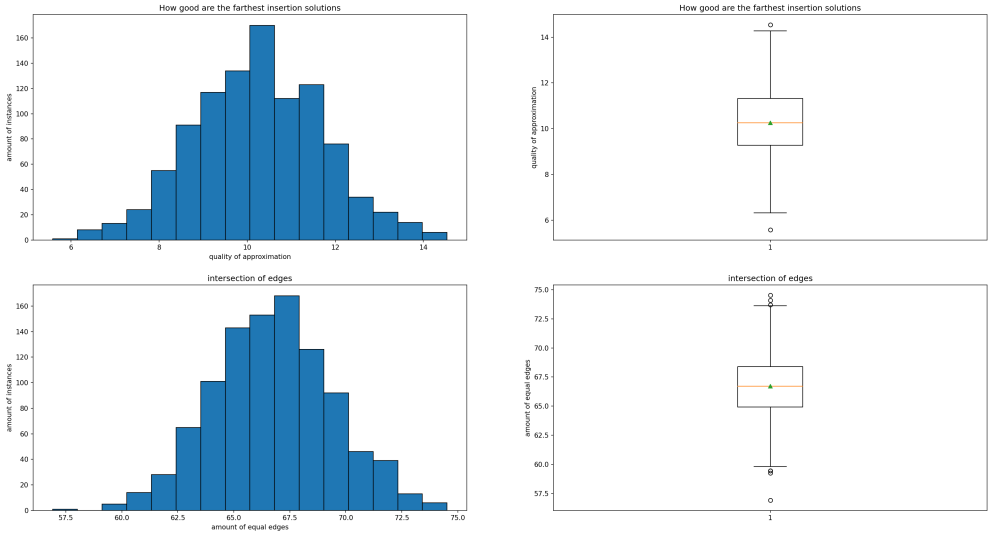

Abbildung 3: Ergebnisse der Analyse der Farthest Insertion Heuristik

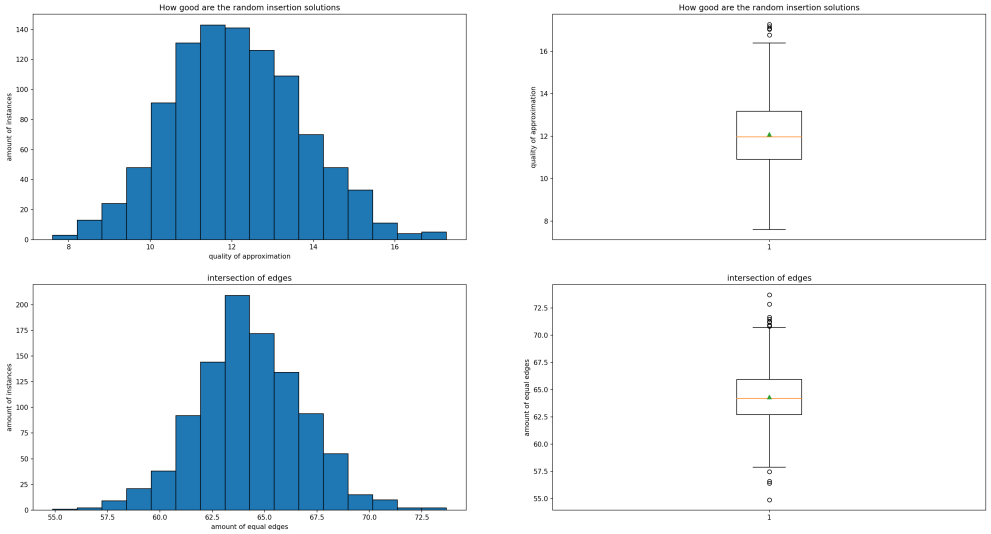

Abbildung 4: Ergebnisse der Analyse der Random Insertion Heuristik

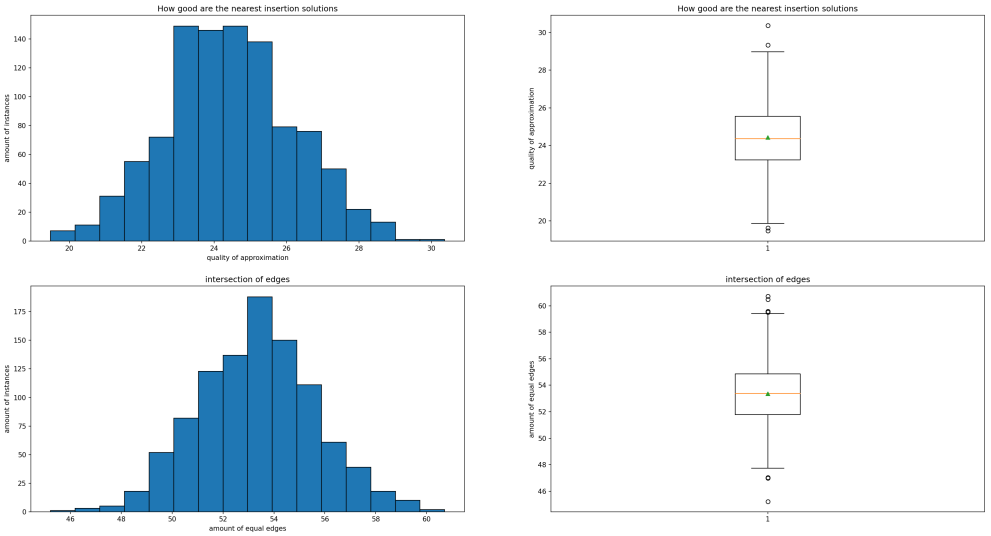

Abbildung 5: Ergebnisse der Analyse der Nearest Insertion Heuristik

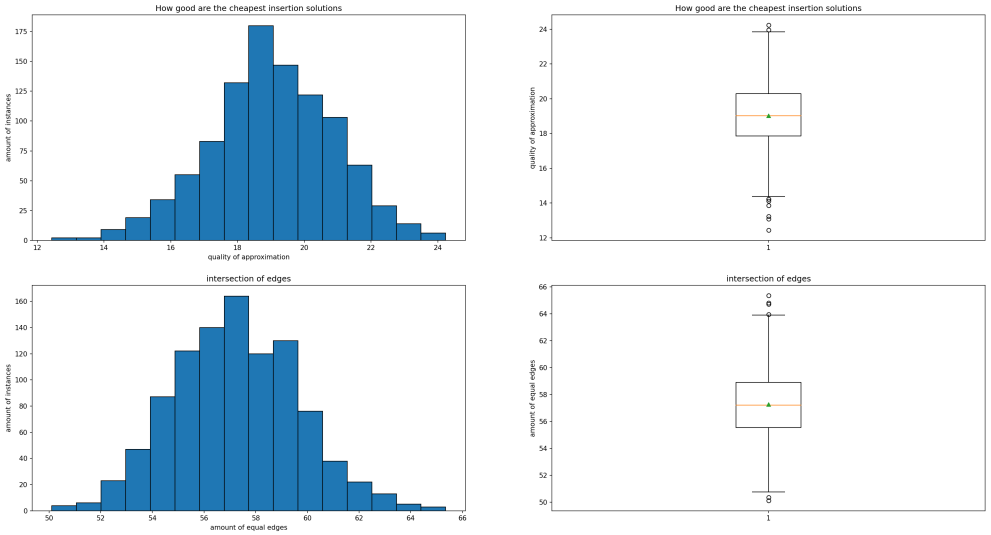

Abbildung 6: Ergebnisse der Analyse der Cheapest Insertion Heuristik

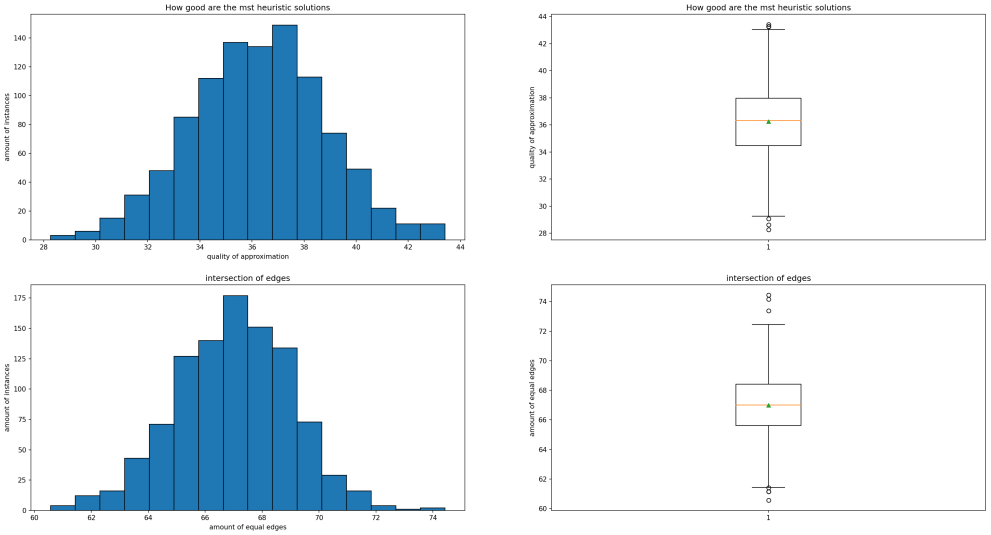

Abbildung 7: Ergebnisse der Analyse der MST Heuristik

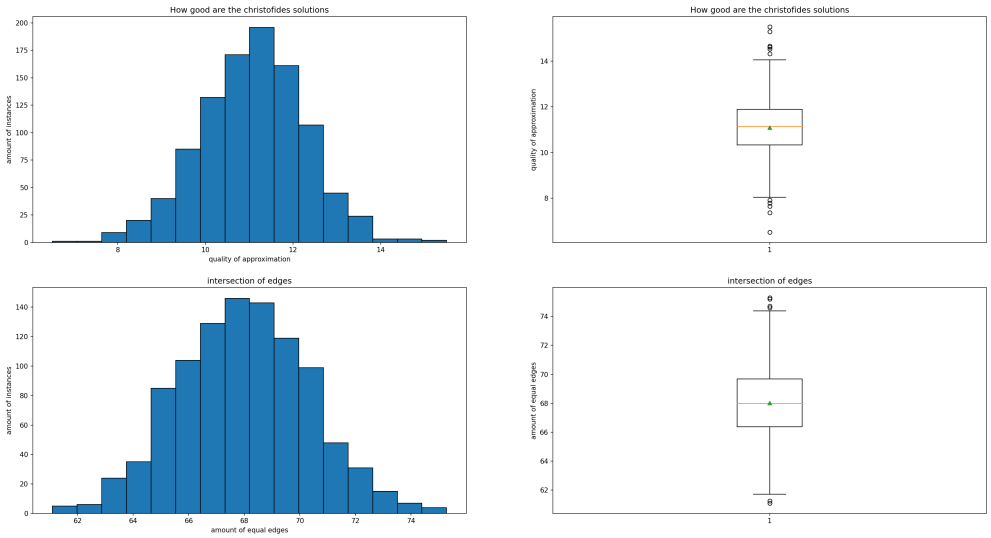

Abbildung 8: Ergebnisse der Analyse des Christofides Algorithmus

An den Ergebnisse kann man sehen, dass die Farthest Insertion Heuristik die beste Approximationsgüte liefert mit einem durchschnittlichen Wert von ca. 10.1%. Im Gegensatz dazu liegen ca. 67% der Kanten der optimalen Lösung auch in den Lösungen der Farthest Insertion Heuristik und damit ist die Heuristik auch nur Mittelmaß mit Blick auf die anderen Heuristiken. Der Greedy Algorithmus liefert zwar mit ca. 17.5% nicht die beste Approximationsgüte, dafür liefert sie mit ca. 74% die meisten Kanten, die auch in der optimalen Lösungen enthalten sind. Ausreißer hingegen sind der Nearest Insertion Heuristik, in der nur ca. 53.8% der Kanten der optimalen Lösung auch in den Lösungen der Heuristik enthalten sind und damit den niedrigsten Wert liefert. Die schlechteste Approximationsgüte hingegen liefert die MST Heuristik mit einem Wert von ca. 36.5%.

4.2 Analyse der MST's

Zusätzlich zu den Heuristiken wurden auch MST's für jedes TSP berechnet. Die MST's gelten als eine untere Schranke für die TSP's. Deshalb wurde die Güte dieser unteren Schranke berechnen und analysiert. Dazu wurde die folgende Funktion verwendet.

$$\frac{L_{MST}}{L_{opt}} * 100 \tag{25}$$

Zudem wurde auch wieder der Prozentsatz der Kanten, die im Schnitt der optimalen Lösung und den MST's liegen, analysiert.

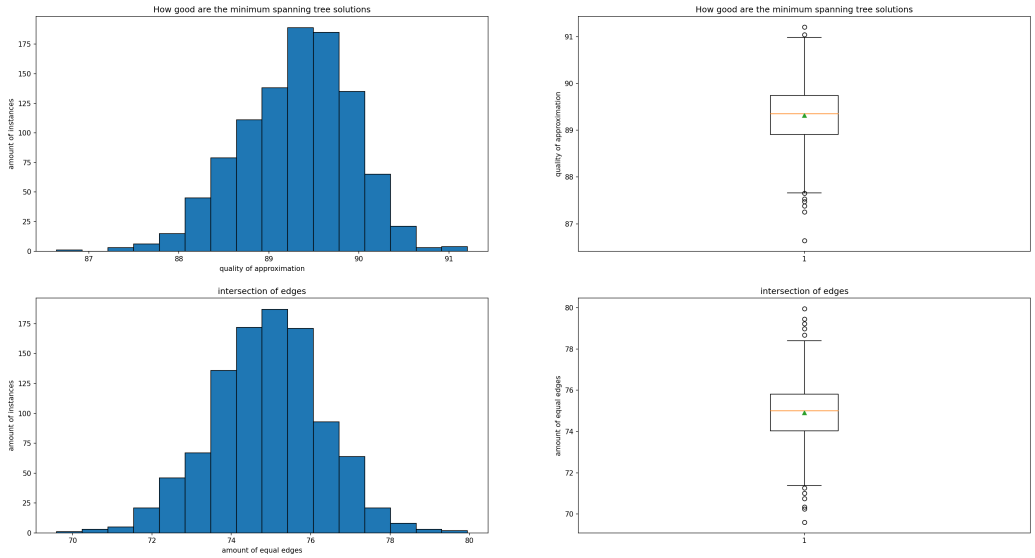

Abbildung 9: Ergebnisse der MST's

Am Ergebnis wird ersichtlich, dass die Güte der unteren Schranke bei ca. 89.5% liegt. Dieser Wert ist nicht vergleichbar zu den anderen Heuristiken, aber es liegt dennoch auf einem hohen Wert und zeigt, dass die MST's als eine untere Schranke nutzbar sind und als ein Merkmal helfen können, Kanten vorherzusagen. Anschließend wurde ein Scatter Plot für die Ergebnisse aller Heuristiken und der MST's erstellt, um alle Ergebnisse leichter vergleichen zu können. Wie schon erwähnt kann die Güte nicht mit den anderen vergleicht werden. Die Ergebnisse der MST's wurden für die Übersicht dennoch in den Scatter Plot mit verwendet.

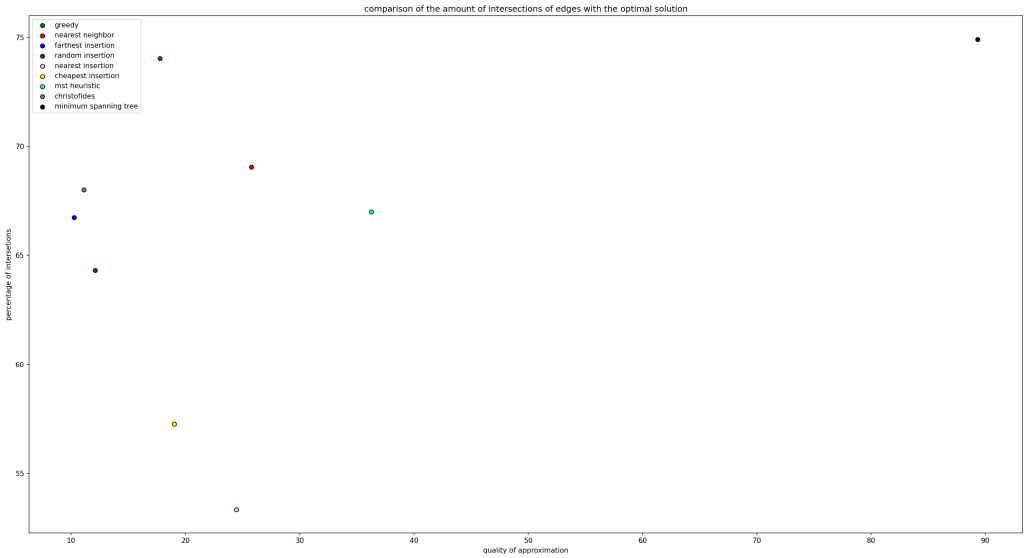

Abbildung 10: Ergebnisse aller Heuristiken und des MST's

4.3 Analyse des k-RNG

Abschließend wurden auch die k-RNG auf 3 Werte analysiert. Dafür wurden die k-RNG mithilfe eines naiven Algorithmus berechnet. Der Berechnung lag dabei bei $O(n^3)$, welches bei der Anzahl an Daten nicht effizient war, aber für diesen Verwendungszweck ausgereicht hat.

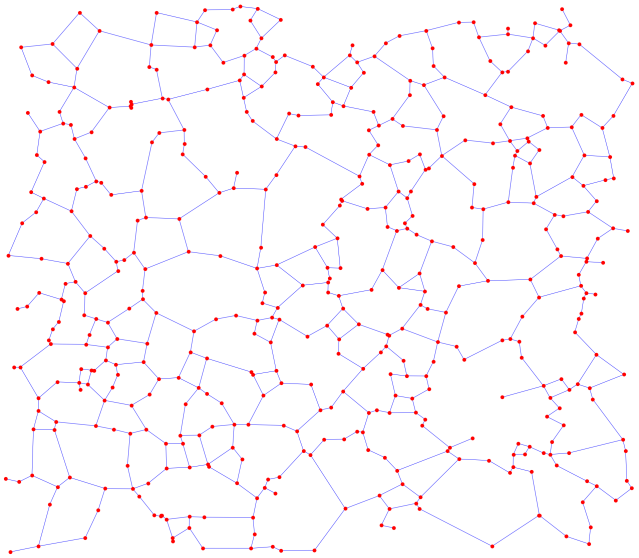

Abbildung 11: 1-RNG für das 1. TSP

Die ersten 2. Werte, die analysiert wurden, waren: Wie viele Kanten einer optimalen Lösung sind auch im 1-RNG enthalten und wie viele Kanten einer Lösung des Greedy Algorithmus sind im 1-RNG enthalten. Zum Schluss wurde noch überprüft, wie viele Kanten einer optimalen Lösung sind im k-RNG enthalten. Für die ersten 2. Werte wurden wieder Histogramme und Boxplots erstellt.

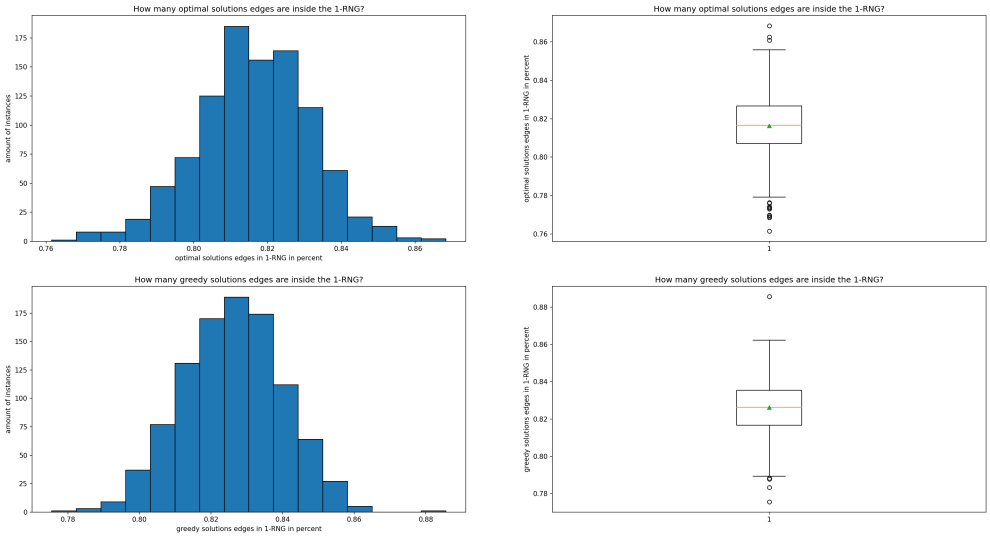

Abbildung 12: Ergebnisse des 1-RNG

An den Ergebnissen ist ersichtlich, dass der durchschnittliche Prozentsatz der Kanten, die im 1-RNG liegen, für die Lösungen des Greedy Algorithmus und den optimalen Lösungen ähnlich ist. Beachtlicherweise liegt der Greedy Algorithmus sogar minimal über den optimalen Lösungen. Die Ergebnisse des 3. Wertes wurden nachfolgend anhand einer Tabelle übersichtlich dargestellt.

k	absoluten Zahlen	Prozentanteile
0	328339	81.87
1	54893	13.69
2	13051	3.25
3	3407	0.85
4	974	0.24
5	288	0.07
6	80	0.02
7	29	0.01
8	8	0
9	1	0
10	0	0
11	1	0

Abbildung 13: Ergebnisse des k-RNG

Hierzu wurden die Werte für die erste Spalte als (k-1) ausgegeben. An den Ergebnissen wird ersichtlich, dass 81.87% der Kanten im 1-RNG liegen. Dieses Ergebnis ist nicht überraschend und bestätigt das Ergebnis des 1. Wertes, welches im Histogramm berechnet wurde. Außerdem liegen 13.69% der optimalen Kanten im 2-RNG und 3.25% im 3-RNG. Je größer das k wird, desto weniger optimale Kanten sind im RNG. Diese Ergebnisse zeigen auf, dass mehr als $\frac{4}{5}$ Kanten der optimalen Lösung im 1-RNG enthalten sind und als ein weiteres Merkmal dienen kann, um Kanten der optimalen Lösung vorhersagen zu können.

5 Schlussfolgerung

Wie schon in 4 erwähnt, konnten die Merkmale und das RFT nicht implementiert werden und dadurch konnte die Forschungsfrage, ob ML bessere Lösungen als Heuristiken liefert, nicht beantwortet werden. Deshalb konnte auch das Ziel, eine solche ML-Methode zu implementieren, nicht

erreicht werden. Die Arbeit gibt dennoch ein Beispiel, wie eine ML-Methode aufgebaut werden kann. Da die Schritte voneinander abgekoppelt sind, kann das Vorgehen so implementiert werden oder es können auch Merkmale ausgetauscht oder eine andere ML-Methode als der RFT genutzt werden. Zudem wurde gezeigt, wie die Heuristiken mit den generierten Daten untereinander und mit Blick auf die optimale Lösung abgeschnitten haben. Es wurden 25 Merkmale aufgezeigt, die der ML-Methode helfen können, die Kanten der optimalen Lösung vorhersagen zu können.

Die Motivation dieser Arbeit war es, eine ML-Methode mithilfe von heuristischen Analysen und Merkmalen zu nutzen, um eine effiziente Lösung für das Traveling Salesman Problem zu berechnen. Die Zeit für die Implementierung einer ML-Methode hängt von den Merkmalen und der genutzten Architektur ab. Die Laufzeiten der vorgestellten Merkmalen ist unterschiedlich, da aber in den meisten nur Vergleiche zwischen 2 Mengen vorgenommen wird, sind auch die meisten Laufzeiten $O(n^2)$ und aufgrund dessen auch nutzbar.

Durch die Popularität der ML-Methoden werden sie auch immer mehr in Zukunft genutzt werden. Diese Arbeit zeigt eine weitere Herangehensweise, wie eine ML-Methode aufgebaut und genutzt werden kann. Zudem kann diese Vorgehensweise auch genutzt werden, um andere kombinatorische Probleme zu lösen.

Literaturverzeichnis

[1] J. Fitzpatrick, D. Ajwani, and P. Carroll, "Learning to sparsify travelling salesman problem instances," 2021.

[2] S. Ait Bouziaren and B. Aghezzaf, "An improved augmented ε -constraint and branch-and-cut method to solve the tsp with profits," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, pp. 195–204, Jan 2019.

[3] D. Nuraiman, F. Ilahi, Y. Dewi, and E. A. Z. Hamidi, "A new hybrid method based on nearest neighbor algorithm and 2-opt algorithm for traveling salesman problem," in *2018 4th International Conference on Wireless and Telematics (ICWT)*, pp. 1–4, 2018.

[4] V. B. Lobo, B. B. Alengadan, S. Siddiqui, A. Minu, and N. Ansari, "Traveling salesman problem for a bidirectional graph using dynamic programming," in *2016 International Conference on Micro-Electronics and Telecommunication Engineering (ICMETE)*, pp. 127–132, 2016.

[5] H. Yang and M. Gu, "A new baseline of policy gradient for traveling salesman problem," in *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–7, 2022.

[6] C. Ding, Y. Cheng, and M. He, "Two-level genetic algorithm for clustered traveling salesman problem with application in large-scale tsps," *Tsinghua Science and Technology*, vol. 12, no. 4, pp. 459–465, 2007.

[7] Z. Xing and S. Tu, "A graph neural network assisted monte carlo tree search approach to traveling salesman problem," *IEEE Access*, vol. 8, pp. 108418–108428, 2020.

[8] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," *INFORMS Journal on Computing*, vol. 3, pp. 376–384, November 1991.

[9] Y. Sun, A. Ernst, X. Li, and J. Weiner, "Generalization of machine learning for problem reduction: a case study on travelling salesman problems," *OR Spectrum*, vol. 43, pp. 607–633, sep 2020.

[10] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," *Technical report*, 2008.