Duale Hochschule Baden-Württemberg
Mannheim

Projektarbeit I

Variation, Analyse und Verbesserung eines Algorithmus zur heuristischen Lösung des Travelling Salesman Problems

Studiengang Wirtschaftsinformatik

Studienrichtung Software Engineering

Verfasser/in:	Benno Grimm
Matrikelnummer:	5331201
Firma:	SAP SE
Abteilung:	Transportation Management
Kurs:	WWI18SEA
Studiengangsleiter:	Prof. Dr. Julian Reichwald
Wissenschaftlicher Betreuer:	Prof. Dr. Julian Reichwald julian.reichwald@dhbw-mannheim.de +49 (0)621 4105 - 1395
Firmenbetreuer:	Peter Wadewitz peter.wadewitz@sap.com +49 6227 7-63730
Bearbeitungszeitraum:	13.05.2019 – 01.09.2019

Kurzfassung

Titel	Variation, Analyse und Verbesserung eines Algorithmus zur heuristischen Lösung des Travelling Salesman Problems
Verfasser/in:	Benno Grimm
Kurs:	WWI18SEA
Ausbildungsstätte:	SAP SE

Hier können Sie die Kurzfassung der Arbeit schreiben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

DHBW	Duale Hochschule Baden-Württemberg
TSP	Travelling Salesman Problem
LE	Längeneinheiten

1 Einleitung

Das Travelling Salesman Problem (TSP) ist... Duale Hochschule Baden-Württemberg (DHBW) DHBW

2 Das Travelling Salesman Problem

2.1 Exakte Löungsverfahren und die Optimale Route

2.2 Heuristische Lösungsverfahren

3 Verwendete Lösungsverfahren

3.1 Insert-First-Verfahren

Das Insert-First-Verfahren ist ein heuristischer Lösungsansatz des TSPs, bei dem das Betrachten der Knoten zum Aufbau eines Graphen in zufälliger Reihenfolge, bzw. in der Reihenfolge ihrer Erzeugung geschieht. Dabei wird zu einem Zeitpunkt genau ein Knoten betrachtet und an der für ihn bestmöglichen Stelle in den bereits bestehenden Graphen eingefügt.

3.1.1 Funktionsweise

Algorithmus 1 Insert-First-Algorithmus

Require: Graph G, Pfad P
Require: $G = K_1, K_2, \ldots, K_n$, $n > 2$
1: $P_1 \leftarrow K_1$ ▷ Setzen der ersten beiden Knoten
2: $P_2 \leftarrow K_2$
3: **for** $a \leftarrow 2$, $a \leq n$, $a \leftarrow a + 1$ **do**
4: $j_S \leftarrow -1$ ▷ Index der geringsten Distanz
5: $d_S \leftarrow -1$ ▷ Geringste Distanz
6: **for** $b \leftarrow 1$, $b \leq n$, $b \leftarrow b + 1$ **do**
7: $d_C \leftarrow$ mergeAt(P, b, K_a) distance ▷ Gesamtdistanz, wenn K_a am Index b in P eingefügt werden würde (siehe Alg. 3 auf Seite 19)
8: **if** $j_S = -1$ **or** $d_C < d_S$ **then**
9: $d_S \leftarrow d_C$ ▷ Kürzeste Distanz wird übernommen
10: $j_S \leftarrow b$ ▷ Und ihr Index
11: **end if**
12: **end for**
13: $P \leftarrow$ mergeAt(P, j_S, K_a) ▷ K_a wird am Index j_S in P eingefügt
14: **end for**
15: **return** new Graph(P) ▷ Graph mit Pfad P wird zurückgegeben

Als Eingabe erhält der Algorithmus einen Graphen mit einer ungeordneten Liste von n Knoten $K_1, K_2, \ldots, K_n$ mit $n > 2$. Jeder Graph ist mit einem Pfad P der Länge n assoziiert, der die Reihenfolge der Knoten bestimmt im Ergebnis des Algorithmus bestimmt. Nun wird K_1, der erste Knoten aus der übergebenen Liste in den Pfad

des Graphs an erster Stelle eingefügt. Dies geschieht so oder ähnlich bei allen Verfahren, um einen statischen Ausgangspunkt zu gewährleisten und somit vergleichbare Ergebnisse zu erzielen. Anschließend wird noch der zweite Knoten, K_2 angehängt.

Das Vorgehen für das Einfügen der restlichen Knoten lässt sich wie folgt beschreiben: Sei G ein Graph mit einer ungeordneten Menge von n Knoten $K_1, K_2, \ldots, K_n$ und bereits teilweise befülltem Pfad $K_1, \ldots, K_m$ mit $m \geq 2$ und $m < n$. Die Knoten, die noch eingefügt werden müssen, werden in der Reihenfolge ihres Auftretens in der übergebenen Liste in den Graphen eingefügt, womit der als nächstes einzufügende Knoten immer K_i mit $i = m + 1$ ist.
Um die beste Stelle zu ermitteln, in die K_i eingefügt werden soll, wird für jede mögliche Stelle die Distanzerhöhung berechnet, zu der das Einfügen von K_i an dieser Stelle führen würde. Um die beste Stelle zu ermitteln, in die K_i eingefügt werden soll, wird für jeden möglichen Index, also jede mögliche Stelle, die Gesamtdistanz des entstehenden Graphen berechnet.

Das niedrigste Ergebnis dieser Möglichkeiten wird zusammen mit dem dazugehörigen Index j vermerkt. Nachdem die niedrigste Distanz für K_i errechnet wurde kann anhand des Index' der Knoten an der bestmöglichen Stelle in den Graphen eingefügt werden. Einfügen bedeutet hier, dass alle Knoten, deren Index gleich oder höher j ist nach hinten verschoben werden. Nachdem alle Knoten auf diese Weise nach hinten verschoben wurden, kann K_i an der Stelle j eingefügt werden, ohne, dass andere Knoten verloren gehen. Beispielhaft sähe das mit den vorher festgelegten Bezeichnungen wie folgt aus:

$P = K_1, K_2, K_4, K_3$ und $K_{i=5}$

Durch das ermitteln der Gesamtdistanzen in Abhängigkeit zu den möglichen Einfügestellen wird bekannt, dass $K_{i=5}$ mit dem Index $j = 4$, also zwischen K_4 und K_3 bestmöglich eingefügt werden kann. Durch das Einfügen nach 3 auf Seite 19 entsteht folgender Pfad:

$P = K_1, K_2, K_4, K_5, K_3$

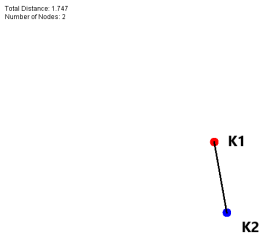
für den Graphen.

3.1.2 Ergebnisse und Schwächen

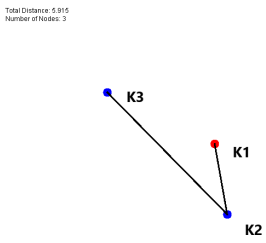
Bevor einige durch den Algorithmus generierte Beispiele betrachtet werden, wird hier das Szenario dieser und aller folgender Beispiele, es sei denn ist anderes angegeben, beschrieben. Alle gezeigten Knoten befinden sich auf einem zweidimensionalem Fläche mit den Maßen zehn mal zehn Längeneinheiten (LE). Folglich kann jedem Knoten eine

X- und Y-Koordinate zwischen jeweils null und zehn zugeordnet werden. Dementsprechend bewegen sich auch die Gesamtdistanzen er gezeigten Graphen in dieser Größenordnung. Weiterhin werden aus Gründen der Übersichtlichkeit für den Großteil der folgenden Beispiele nur Graphen mit fünf Knoten betrachtet.

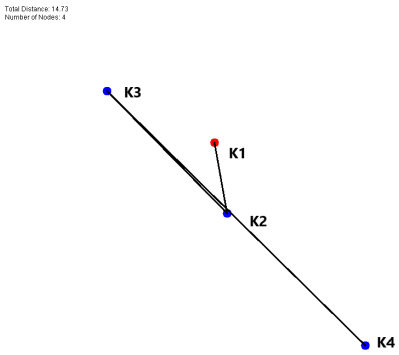
Bei dem Einsatz des oben beschriebenen Algorithmus kommt es zu Ergebnissen, die in ihrer Qualität nah an die optimale Lösung herankommen, teilweise aber auch weit von ihr abweichen können.



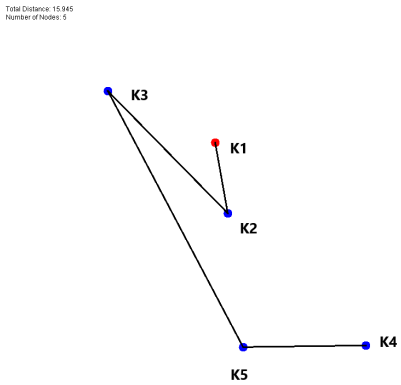
(a) $m = 2$



(b) $m = 3$



(c) $m = 4$



(d) $m = 5$

Abbildung 3.1: Insert-First führt zu schlechtem Ergebnis

Auf 3.1c lässt sich erkennen, dass das Einfügen des vierten Knoten K_4 nicht optimal geschieht. Besser für $m = 4$ wäre hier der Pfad

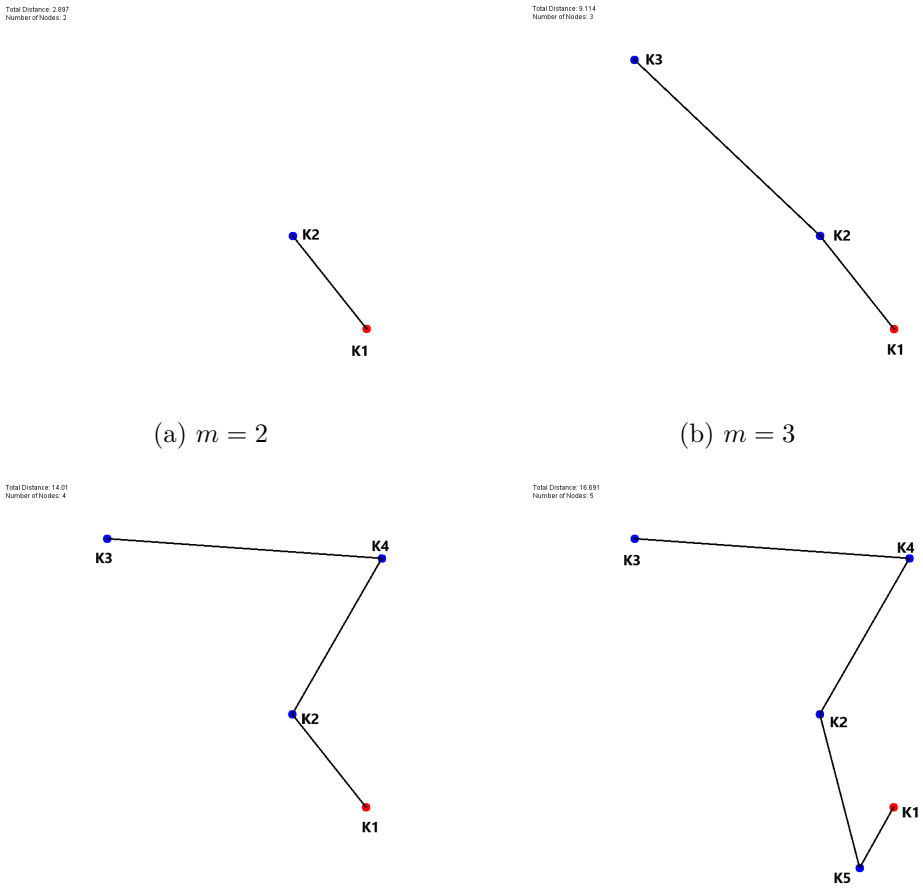
$$P = K_1, K_3, K_2, K_4.$$

Dieser wird allerdings nicht durch das Insert-First-Verfahren gebildet, da dies eine Änderung des bereits erzeugten Graphen in 3.1b auf der vorherigen Seite erfordern würde. Dies ist jedoch nicht möglich, da K_4 nur zwischen bereits im Pfad des Graphen vorhandenen Knoten eingefügt werden kann, sodass der schlussendlich generierte Graph eine Gesamtlänge von 15,945 LE hat. Hier lässt sich auch das grundlegende Problem des Algorithmus erkennen: Das Erstellen einer Route ohne vorherige Betrachtung der Gesamtheit der Knoten. Einzelne Teilschritte des Graphen können gut erzeugt werden, wie beispielsweise im Schritt von 3.1a auf der vorherigen Seite zu 3.1b auf der vorherigen Seite. Andere hingegen, wie vorher erwähnt, nicht. Grund hierfür ist die alleinige Betrachtung des Knotens K_i. Spezifischer bedeutet das, dass das frühe Einfügen von Knoten in den Pfad eines Graphen später zu Komplikationen führen kann, da es objektiv besser gewesen wäre einen anderen Knoten früher einzufügen. Am konkreten Beispiel führt die generierte Reihenfolge von

$$P = K_1, K_2, K_3.$$

in 3.1b auf der vorherigen Seite dazu, dass K_4 nur unter einen vergleichsweise großen Gesamtdistanzzuwachs in den Graphen eingefügt werden kann.

Konträr zu diesem schlechten Beispiel ist der Insert-First-Algorithmus auch in der Lage gute bis optimale Ergebnisse zu generieren.



(a) $m = 2$ (b) $m = 3$

(c) $m = 4$ (d) $m = 5$

Abbildung 3.2: Insert-First führt zu guten Ergebnis

Am Beispiel in 3.2 lässt sich erkennen, wie der Insert-First-Algorithmus einen optimalen Pfad mit den gegebenen Knoten generiert. Gerade im Schritt von 3.2c zu 3.2d ist ein funktionierendes und korrektes Einfügen des Knotens in den Graphen zu sehen, bei dem der Anstieg der Gesamtdistanz der Route sehr gering gehalten wird. Hier wird der aktuelle Knoten mit geringem Zuwachs der schlussendlichen Gesamtdistanz in den Graphen eingefügt, sodass die Gesamtdistanz zum Ende bei 16,691 LE liegt. Die scheint zwar höher als das vorherige schlechte beispiel, das liegt aber an den Positionen der einzelnen Knoten.

Für die Bewertung des Algorithmus müssen also beide Seiten betrachtet werden. Zwar ist Insert-First in der Lage eine gute oder auch optimale Route zu erstellen, allerdings beeinflusst die Reihenfolge der Betrachtung der Knoten stark die Qualität des Endergebnisses.

3.2 Insert-Furthest-Verfahren

Aufbauend auf den Erkenntnissen des Insert-First-Verfahrens können experimentell einige Verbesserungsideen abgeleitet und ihre Auswirkungen auf das Erzeugen eines Graphen betrachtet werden. Beim Insert-First-Verfahren wurde festgestellt, dass eine große Schwäche des Algorithmus die Reihenfolge der Betrachtung der Knoten sein kann. Ein möglicher Ansatz, dieser in 3.1.2 auf Seite 4 beschriebenen Schwäche entgegenzuwirken, ist die Einführung eines Kriteriums zur Betrachtung der Knoten. Eine mögliche Umsetzung eines solchen Kriteriums ist das Insert-Furthest-Verfahren. Hier wird der als nächstes einzufügende Knoten (K_i) durch seine Distanz zum Vorgänger (K_{i-1}) bestimmt.

3.2.1 Funktionsweise

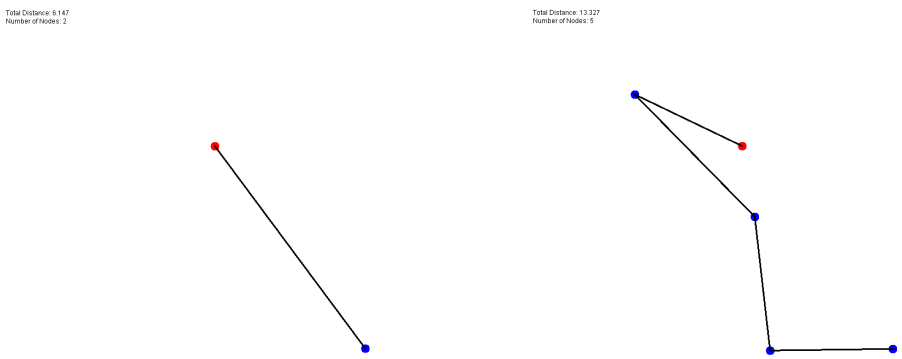
Ähnlich dem Insert-First-Verfahrens wird auch hier ein Graph mit einer Liste von Knoten udn einem zu Beginn leerem Pfad erzeugt. Auch hier wird wieder der erste Knoten der Liste K_1 als initialer Knoten des Pfades gesetzt. Der nächste zu betrachtende Knoten ist nun aber nicht K_2, sondern wird durch die Distanz zu K_1 bestimmt. Ausgewählt wird der Knoten, der am weitesten von K_1, bzw. allgemein am weistesten von K_{i-1}, entfernt ist und nicht bereits Teil des Pfads ist. Dieser Knoten wird nun auf die gleiche Weise wie die Knoten beim Insert-First-Verfahren in den Pfad des Graphen eingefügt; die Stelle mit der geringsten Distanzerhöhung für den Graphen wird gesucht und K_i an dieser Stelle nach dem in 3 auf Seite 19 beschriebenen Verfahren eingefügt. Der Gedanke hinter der dieser Veränderung ist der Versuch Knoten mit größerer Voraussicht als im Insert-First-Verfahren in den Pfad des Graphen einzufügen. Ziel ist es mit den ersten paar Knoten einen Pfad zu generieren, der einen großen Teil der Fläche überspannt, auf der sich Knoten befinden. Das kann insofern zu einem besserem Ergebnis führen, dass die ersten Knoten zwar unter einer, relativ zur schlussendlichen Gesamtlänge des Graphen, hohen Distanzerhöhung eingefügt werden, die nachfolgenden Knoten aber durch geringe Umwege des bestehenden Pfads in den Graphen eingebunden werden können. Auf diese Weise sollen Komplikationen beim Einfügen der letzten Knoten verhindert werden und so suboptimale Graphen wie in 3.1 auf Seite 5 umgangen werden.

3.2.2 Ergebnis und Schwächen

Testet man des Insert-Furthest-Verfahren anhand der Knoten des Beispiels 3.1 auf Seite 5 wird sichtbar, dass der Algorithmus tatsächlich in der Lage ist einen besseren Pfad zu generieren als das Insert-First-Verfahren.

Algorithmus 2 Insert-Furthest-Algorithmus

Require: Graph G, Pfad P
Require: $G = K_1, K_2, \ldots, K_n$, $n > 2$
1: $P_1 \leftarrow K_1$ ▷ Setzen der ersten beiden Knoten
2: $P_2 \leftarrow K_2$
3:
4: $i \leftarrow -1$
5: $d_i \leftarrow -1$
6: **for** $a \leftarrow 2$, $a \leq n$, $a \leftarrow a + 1$ **do**
7: $d_C \leftarrow \text{distance}(K_a, P_m)$ ▷ Distanz zwischen K_a und dem letzten Knoten in P, P_m
8: **if** ($K_a \notin P$ **and** $d_C > d_i$) **or** $i = -1$ **then**
9: $d_i \leftarrow d_C$
10: $i \leftarrow a$
11: **end if**
12: **end for**
13: **for** $a \leftarrow 2$, $a \leq n$, $a \leftarrow a + 1$ **do**
14: $j \leftarrow -1$ ▷ Index der geringsten Distanz
15: $d_S \leftarrow -1$ ▷ Geringste Distanz
16: **for** $b \leftarrow 1$, $b \leq n$, $b \leftarrow b + 1$ **do**
17: $d_C \leftarrow \text{mergeAt}(P, b, K_a)$ distance ▷ Gesamtdistanz, wenn K_a am Index b in P eingefügt werden würde
18: **if** $j = -1$ **or** $d_C < d_S$ **then**
19: $d_S \leftarrow d_C$ ▷ Kürzeste Distanz wird übernommen
20: $j \leftarrow b$ ▷ Und ihr Index
21: **end if**
22: **end for**
23: $P \leftarrow \text{mergeAt}(P, j, K_a)$ ▷ K_a wird am Index j in P eingefügt
24: **end for**
25: **return** new Graph(P) ▷ Graph mit Pfad P wird zurückgegeben

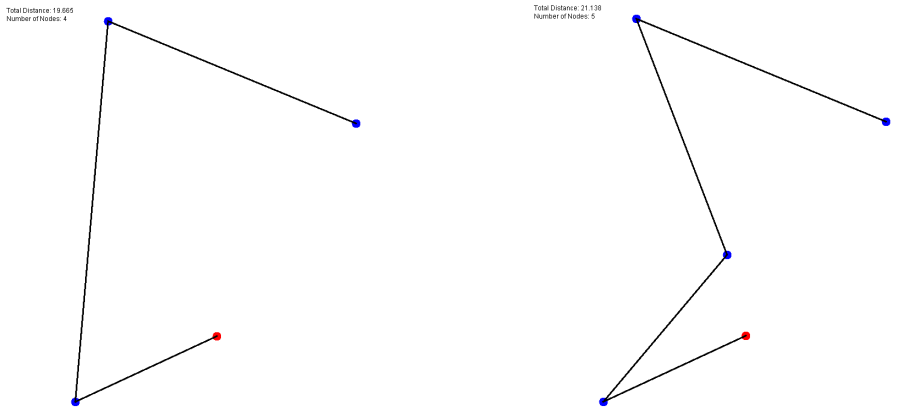


(a) $m = 2$ (b) $m = 5$

Abbildung 3.3: Insert-First führt zu schlechtem Ergebnis

In 3.4a auf der nächsten Seite ist zu erkennen, dass, anstatt wie in 3.1a auf Seite 5 K_2 als erster Knoten, K_4 eingefügt wird. Dieses Verhalten ist nach der in 3.2.1 auf Seite 8 definierten Funktionsweise zu erwarten, da K_4 der Knoten mit der größten Distanz zu K_1 ist und daher als erstes in den Pfad des Graphen eingefügt wird. Nach der Ausführung aller Schritte des Algorithmus erhält man den in 3.4b auf der nächsten Seite zu sehenden Graphen. Dieser hat eine Gesamtdistanz von 13,327 LE und ist somit im Vergleich mit dem in 3.1 auf Seite 5 durch das Insert-First-Verfahren erzeugten Graph 2,618 LE oder 16,41% kürzer.

Ebenso wie das Insert-First-Verfahren kann das Insert-Furthest-Verfahren auch Graphen generieren die in ihrer Gesamtdistanz vom Optimum abweichen. Am folgenden Beispiel wird deutlich, dass auch dieser Algorithmus von ähnlichen Schwächen betroffen ist.



(a) $m = 4$ (b) $m = 5$

Abbildung 3.4: Insert-First führt zu schlechtem Ergebnis

Die der Abbildung 3.4 vorhergehenden Schritte $m = 1$ bis $m = 3$ sind im Anhang zu finden.

Auch hier lässt sich das Problem der Reihenfolge beobachten, welches in 3.1.2 auf Seite 4 beschrieben wurde. Durch das Einfügen des letzten Knotens K_4 in den Pfad entsteht ein suboptimaler Graph. In diesem konkreten Beispiel ist das mit dem menschlichen Auge jedoch nicht sofort ersichtlich. Vergleicht man aber die Distanzen zwischen den Knoten K_4 und K_1 (6,129 LE) mit den zwischen K_1 und K_2 (5,019 LE) wird schnell ersichtlich, dass eine Verminderung der Distanz durch das Umlegen der Knoten erreicht werden kann. Verursacht wird diese Abweichung vom Optimum dadurch, dass K_4 als letztes in den Graphen eingefügt wird, da der Knoten sich, relativ zu den restlichen Knoten, in der Mitte der Fläche befindet und somit aufgrund seiner geringeren Entfernung vom Algorithmus als betrachtet wird. Die optimale Route

`path =` K_1, K_3, K_4, K_2, K_1

würde es jedoch erfordern, dass K_4 früher betrachtet und in den Pfad eingefügt wird. Der durch den Algorithmus erzeugte Graph hat eine Gesamtlänge von 21,138 LE, während durch das Umlegen zum optimalen Graph eine Länge von 20,028 LE, also Reduktion der Distanz um 1,11 LE oder 5,251% erreicht werden kann.

Das Insert-Furthest-Verfahren verhält sich in einigen Fällen, wie in 3.3 auf der vorherigen Seite gezeigt, besser als das Insert-First-Verfahren, weist aber immer noch eindeutige Schwächen, gerade im Bezug auf die Reihenfolge der Betrachtung der Knoten. Das Beispiel in 3.4 zeigt deutlich, wie auch hier die Positionierung der Knoten Einfluss auf das Endergebnis hat.

3.3 Insert-Closest-Verfahren

Aufbauend auf den durch das Insert-First- und Insert-Furthest-Verfahren gewonnen Erkenntnissen ist es mögliche weitere Variationen der Heuristik zu entwickeln und deren Ergebnisse zu betrachten. Da sowohl in 3.1 auf Seite 3 als auch in 3.2 auf Seite 8 festgestellt wurde, dass ein Grund für suboptimal erstellte Routen die Reihenfolge der Betrachtung der Knoten ist, wird für das Insert-Closest-Verfahren eine weitere anderes Kriterium für eben diese Reihenfolge festgelegt. Wie der Name des Verfahrens schon suggeriert, geschieht hier die Auswahl der Knoten wieder nach ihrer Distanz.

3.3.1 Funktionsweise

Das Insert-Closest-Verfahren bezeichnet im Grundprinzip die Umkehrung des Insert-Furthest-Prinzips. Anstatt des am weitesten entfernten Knotens wird hier der dem aktuellen Knoten nächste betrachtet.
Zu Beginn beschreibt sich der Algorithmus identisch zum Insert-Furthest-Verfahren. Auch hier wird ein neuer Graph mit einem leerem Pfad `path` und einer Liste von Knoten $K_1, \ldots, K_n$ der Länge n erzeugt. Auch hier wird der erste Knoten K_1 als erster Knoten des Pfades festgelegt. Der als nächstes einzufügende Knoten wird wie beim Insert-Furthest-Verfahren durch seine Distanz zum vorherigen bestimmt. Für das Insert-Closest-Verfahren wird der dem Vorherigen Knoten nächste Knoten, unter der Bedingung, dass dieser nicht bereits Teil des Graph ist, in den Pfad eingefügt. Um die beste Stelle zum Einfügen des Knotens zu ermitteln wird das gleiche Verfahren wie bei den beiden vorherigen Algorithmen angewandt; die Stelle, die den geringsten Anstieg für die Gesamtdistanz des Pfads wird ausgewählt. Auch das Vorgehen beim Einfügen orientiert sich hier an den vorherigen Algorithmen. Der aktuelle Knoten K_i wird nach dem in **??** auf Seite ?? dargestelltem Prinzip an dem vorher festgelegten Index j in den Graph eingefügt.

3.3.2 Ergebnis und Schwächen

Auch die Ergebnisse dieses Algorithmus gestalten sich sehr divers. Wie bei den beiden anderen Verfahren entstehen hier Graphen, die nahe an die Optimale Route heranreichen oder ihr teilweise auch entsprechen, aber auch solche, die weit von der optimalen Lösung abweichen. Übergibt man dem Algorithmus beispielsweise die gleichen Knoten wie in 3.1 auf Seite 5, einem Szenario, bei dem der Insert-First Algorithmus einen suboptimalen Graph generiert, kommt der Insert-Closest Algorithmus, so wie der Insert-Furthest Algorithmus auch, auf die optimale Route.

(a) $m = 4$ (b) $m = 5$

Abbildung 3.5: Der Insert-Closest Algorithmus kommt zu einem schlechten Ergebnis

So wie die anderen Algorithmen stößt auch der Insert-Closest Algorithmus bei bestimmten Konstellationen von Knoten an seine Grenzen.

Wie in 3.5 zu sehen ist, plant der Algorithmus einen Pfad, der deutlich erkennbar nicht optimal ist. Die einzelnen Schritte, die zur Generierung des Graphen führen finden sich im Anhang in der Abbildung A.1 auf Seite 19. Der in 3.5b gezeigt Pfad hat eine Gesamtdistanz von 15,336 LE. Durch das Ändern der Reihenfolge der Knoten zu

$$\texttt{path} = K_1, K_4, K_5, K_2, K_3$$

könnte eine Verringerung der Distanz im Vergleich zum vorherigen Resultat um 3,208 LE, bzw. 20,918% erreicht werden. Auch hier wird der suboptimal geplante Pfad durch die Reihenfolge der Betrachtung der Knoten verursacht. Im konkreten Beispiel werden die ersten vier Knoten K_1, K_2, K_4 und K_5 zu einem für sich optimalen Pfad zusammengefügt. K_3 wird aufgrund seiner hohen Distanz zu den übrigen Knoten als letztes in den Pfad eingefügt. Im gezeigten Beispiel kommt es also genau zu der Umkehrung des Problems, welches beim Insert-Furthest Algorithmus besteht; dort werden Knoten aufgrund ihrer zu niedrigen Distanz teilweise zu spät eingefügt. Diese Problem hat hier zur Folge, dass K_3 als letzter Knoten des Pfads nach dem von ihm am weitesten entfernten Knoten eingefügt wird, was zu einem hohen Zuwachs der Gesamtdistanz führt.

3.4 Lösungsverfahren im Vergleich

Betrachtet man die Graphen, die durch die drei vorgestellten Algorithmen erzeugt werden, so fallen bei allen schnell Schwächen auf, die ihr Ergebnis beeinträchtigen. Anhand der vorherigen Beispiele wurde deutlich, dass die Algorithmen zwar in der Lage sind gute Ergebnisse zu erzeugen, gleichzeitig aber auch auf sich allein gestellt nicht sehr zuverlässig sind. Dies trifft sowohl auf das Insert-First Verfahren, als auch auf die vorgestellten Algorithmen zu. Weiterhin ist in Abbildung 3.5 auf der vorherigen Seite ist ein Phänomen zu beobachten, welches gerade bei Anwendung der Algorithmen mit mehr Knoten häufig auftritt. Das Entstehen von Überkreuzungen von Teilrouten zwischen zwei Knoten (siehe hierzu Abbildung A.1 auf Seite 19).

4 Verbesserung eines bestehenden Pfads

4.1 Entfernen von Überschneidungen

4.2 Nachbesserung eines Pfads

4.3 Komplexität und Nutzen der Verbesserungen

5 Fallbeispiel Transportation Management SAP

5.1 Implementierung der Algorithmen

5.2 Auswirkungen auf die Routenplanung

6 Zusammenfassung und Ausblick

7 Literaturverzeichnis

A Anhang

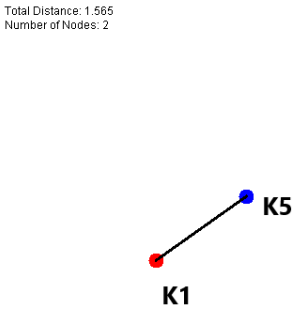
A.1 Bilder

(a) $m = 2$

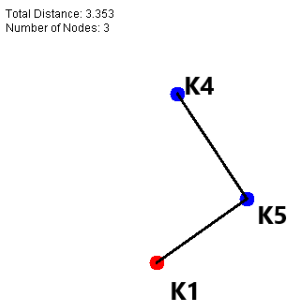
A.2 Algorithmen

Algorithmus 3 Einfügen eines Knoten in einen Pfad

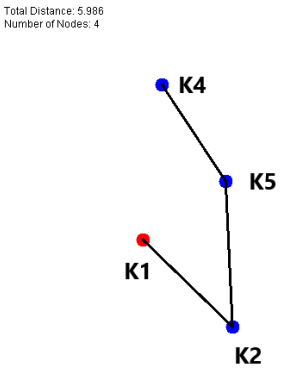
Require: Pfad $P = K_1, \ldots, K_n$
Require: Knoten K^*, Index i, $i \leq n$
1: **for** $a \leftarrow n$, $a \geq i$, $a \leftarrow a + 1$ **do**
2: $\quad P_{a+1} \leftarrow P_a$
3: **end for**
4: $P_i \leftarrow K^*$
5: **return** P


Total Distance: 1.565
Number of Nodes: 2
K5
K1

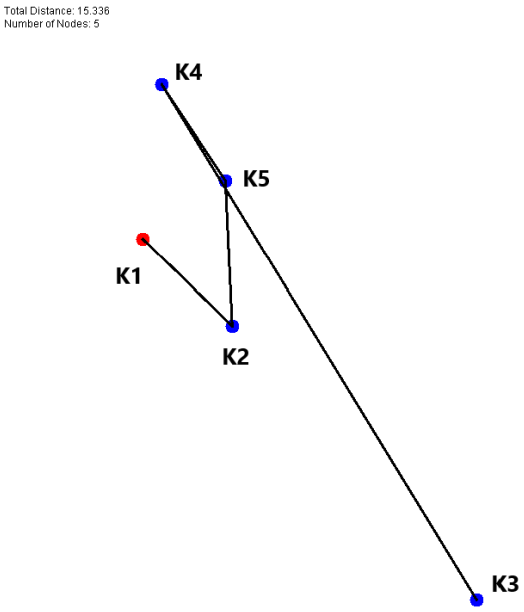
(b) $m = 3$


Total Distance: 3.353
Number of Nodes: 3
K4
K5
K1

(c) $m = 4$

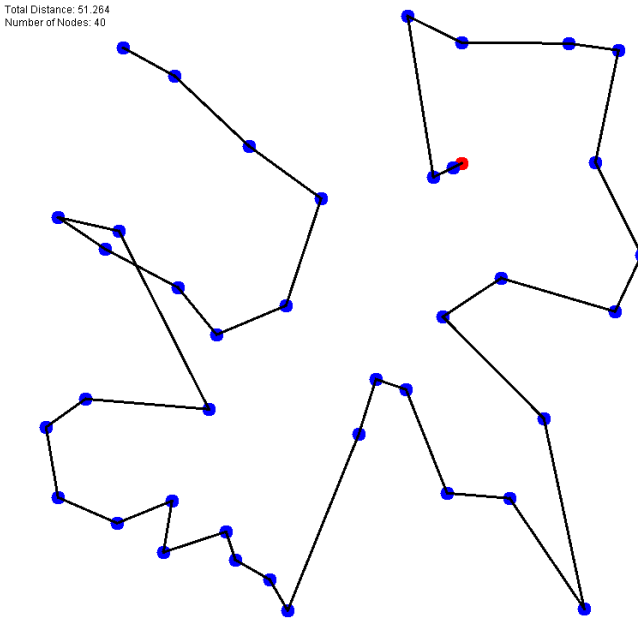


(d) $m = 5$

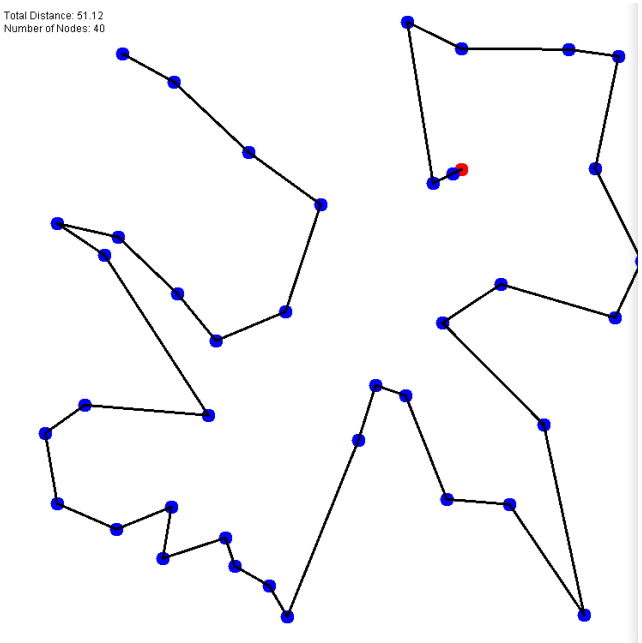


(e) $m = 5$

Abbildung A.-3: Viele Bilder



(a) Pfad mit einem Crossover



(b) Pfad mit aufgelöstem Crossover

Abbildung A.-3: Pfad aus 40 Knoten mit und ohne Crossover

Ehrenwörtliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit mit dem Thema: *Variation, Analyse und Verbesserung eines Algorithmus zur heuristischen Lösung des Travelling Salesman Problems* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort, Datum Benno Grimm