Problem Set 3

CVRP Improvements

Johanna Sacher | 221103072
johanna.sacher@student.uni-halle.de

Christian Paffrath | 221103085
christian.paffrath@student.uni-halle.de

I Problem 1: CVRP – Algorithmic Improvements

I.1 Problemstellung

Analyse your previous algorithms to improve their solution quality.

1. Identify instances on which your algorithms behave suboptimal.
2. Why does your algorithm produce suboptimal solutions? What needs to be improved?
3. Describe central ideas on how you might improve your algorithm.
4. Implement your ideas to create a third heuristic.

I.2 Lösung

Da wir in Abgabe 2 nicht dazu gekommen sind, eine zweite Heuristik zu implementieren, betrachten wir im folgenden den Greedy-Algorithmus.
Da außerdem aus der Aufgabenstellung nicht klar hervorging, ob die *»improved solution quality«* sich auf die Verbesserung der Laufzeit oder die Verbesserung (also Verringerung) der insgesamt gefahrenen Distanz bezieht, haben wir uns dazu entschieden zu versuchen, die Gesamtdistanz zu verringern.

I.2.1 Suboptimales Verhalten

(1) Außenseiter (Distanzen) Falls einzelne Knoten »Außenseiter« sind, also sehr weit von den meisten anderen Knoten entfernt liegen, werden diese vom Greedy- Algo-

rithmus erst anvisiert, wenn für eine Route keine näher gelegenen Knoten mit passendem `Demand` mehr vorhanden sind. In diesem Fall werden gegebenenfalls große Distanzen zurückgelegt, um diese Außenseiter zu erreichen, obwohl sie vorher von einem anderen Knoten aus sogar besser erreichbar gewesen wären – auch wenn sie dort nicht der tatsächlich nächste Nachbar waren. Dieses Verhalten resultiert aus dem Fokus des Greedy-Approaches auf das lokale Optimum, wobei das globale Optimum außer Acht gelassen wird (immer den nächsten Nachbarn finden, ohne an zukünftige Touren zu denken).

(2) Außenseiter (Demands) Bei Instanzen mit einer großen Spanne unterschiedlicher `Demands` der Kunden kann es passieren, dass große Distanzen zwischen Kunden zurückgelegt werden müssen, um den nächstliegenden Kunden mit passendem `Demand` zu finden. Sind alle `Demands` jedoch grob im selben Größenbereich, kommen Routen eher ohne große Distanzen aus.

I.2.2 Was müsste verbessert werden?

Beim Greedy-Algorithmus wird für jede Route immer der nächstgelegene Knoten des aktuellen Standpunktes bestimmt, dessen `Demand` höchstens so groß ist wie die verbleibende Ladung des Fahrzeuges. Dieser Knoten wird besucht, dann wiederum von dort aus der nächste Nachbar mit passendem `Demand` bestimmt, und so weiter. Für jede Route wird also immer nur der nächst »beste« Knoten betrachtet, nie der zweitbeste oder andere Kandidaten. Dadurch wird zwar immer die aktuell anstehende Tour optimiert, aber nicht die gesamte Route, da zukünftige Touren nicht mit in die Kalkulation einbezogen werden. Auf diesen müssen dann möglicherweise größere Distanzen zurücklegt werden um Knoten zu erreichen, die in der aktuellen Route mit geringerem Aufwand hätten besucht werden können, wenn man beispielsweise die drei bis fünf nächstgelegenen Knoten betrachten würde, statt immer nur den absolut nächsten. Aus diesem Grund werden beim Greedy Algorithmus zwar anfänglich kurze Touren für eine Route berechnet, aber dies kann, wie in Teilaufgabe I.2.1 beschrieben, auf Kosten der späteren Routen geschehen.

I.2.3 Zentrale Ideen zur Verbesserung

Das in Teilaufgabe I.2.2 beschriebene Problem der weit außerhalb liegenden Knoten wollen wir versuchen zu verbessern, in dem wir durchschnittliche Distanzen zwischen noch unbesuchten Knoten mit in die Kalkulation des nächsten Knotens einbeziehen. Wir betrachten dafür sowohl die knotenspezifische Durchschnittsdistanz von einem Kunden zu allen anderen unbesuchten Kunden, als auch die globale Durchschnittsdistanz zwischen allen noch unbesuchten Kunden. Uns ist bewusst, dass durch diese zusätzlichen Vergleiche und vor allem durch das ständige Neuberechnen der Durchschnittsdistanzen zu allen noch nicht besuchten Knoten, die Laufzeit unseres Algorithmus stark zunehmen wird. Der Fokus unerer Verbesserungen liegt aber wie gesagt auf dem Ergebnis (also verringerte Kosten), nicht auf der Laufzeit.
Wie in Teilaufgabe I.2.2 bereits erwähnt, wollen wir nun nicht mehr nur den nächsten Nachbarn finden, sondern eine Auswahl aus den fünf am nächsten gelegenen Knoten treffen, deren `Demand` mit unserer verbleibenden Ladung abgedeckt werden kann. Zuerst wollen wir dann herausfinden, ob einer dieser Knoten ein »Außenseiter« ist. Dazu vergleichen wir die individuelle durchschnittliche Distanz iad jedes dieser Kandidaten zu allen anderen Knoten mit der allgemeinen Durchschnittsdistanz gad. Ist $iad > gad$, handelt es sich in der Tat um einen »Außenseiter«. Wenn nun auch noch die Distanz vom aktuellen Knoten zu diesem Außenseiter geringer ist als die gad, dann nehmen wir an, dass es auf lange Sicht wahrscheinlich günstig ist, diesen Knoten jetzt zu besuchen, da er von den verbleibenden Knoten im Durchschnitt weiter entfernt liegt als die gad, von unserem aktuellen Standpunkt aus aber näher dran ist. Finden wir unter den Kandidaten keinen »Außenseiter«, wählen wir als nächstes Ziel, wie zuvor, den am nächsten gelegenen Nachbarn.

I.2.4 Implementierung einer 3. Heuristik

Zur Realisierung unserer Ideen haben wir einen AverageDistanceSolver implementiert, der im Python-Script average_distance_heuristic.py aufgerufen wird. Die grundsätzliche Struktur ist wie beim GreedySolver, aber an der Stelle, wo der nächste Nachbar ausgewählt wird, erhalten wir nun eine Liste mit fünf Kandidaten, aus der wir dann, wie in Teilaufgabe I.2.3 beschrieben, auswählen.
Zentral dabei sind folgende Methoden:

- find_nearest_five(current_node, max_demand) – gibt die fünf am nächsten gelegenen Knoten für `node_id` zurück
- total_average_distance() – berechnet die durchschnittliche Distanz zwischen allen noch verbleibenden Knoten
- average_distance(node_id) – berechnet die durchschnittliche Entfernung eines bestimmten Knotens zu allen noch verbleibenden Knoten

II Problem 2: CVRP – Hypothesis Tests

II.1 Problemstellung

Apply a hypothesis test to assess whether your new heuristic is superior to a previous one.

1. Formulate appropriate null and alternative hypotheses.
2. Design, run, and evaluate the Sign Test (see Slide 178).
3. Design, run, and evaluate the Wilcoxon Signed-Rank Test (see Slide 179).

Summarize your experimental setup and results in a short report.

II.2 Lösung

Wir wollen herausfinden, ob unsere Heuristik mit den Durchschnittsdistanzen (D) für das CVRP bessere Ergebnisse erzielt als der Greedy-Approach (G). Die Unterschiede zwischen D und G bezeichnen wir mit $Z = D - G$. $Z > 0$ g.d.w. D besser ist als G. Für den Vergleich nutzen wir die Kosten (also die Gesamtdistanzen), die die beiden Algorithmen für dieselben `11` Instanzen berechnen.

II.2.1 Nullhypothese und Alternativhypothese

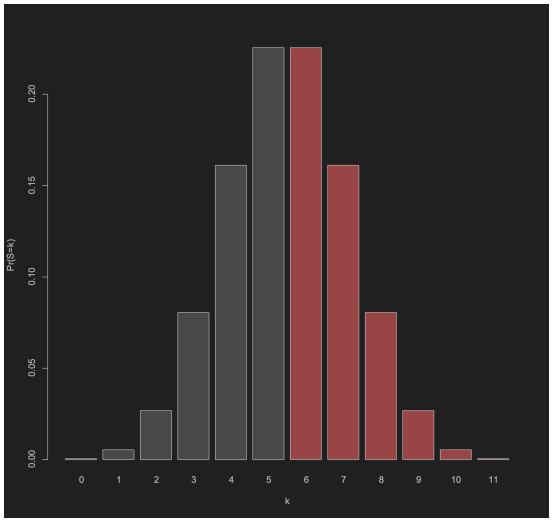
$H_0 =$ Kein Algorithmus ist besser als der andere: $median(Z) = 0$

$H_1 = D$ ist G überlegen: $median(Z) > 0$

$H_2 = D$ und G unterscheiden sich

II.2.2 Vorzeichentest

Den Vorzeichentest haben wir basierend auf dem zur Verfügung gestellten R-Script durchgeführt. Schon an Abbildung 1a ist deutlich zu erkennen, dass die Ergebnisse von D und G ausgewogen sind. Auch die in Abbildung 1b dargestellten Ergebnisse des eigentliche Test zeigen eindeutig, dass H_0 angenommen und H_1 und H_2 abgelehnt werden müssen: der p-Wert von $0,5$ ist deutlich größer als das angenommene Signifikanzniveau von $\alpha = 0,05$.



(a) Barplot der Teststatistik

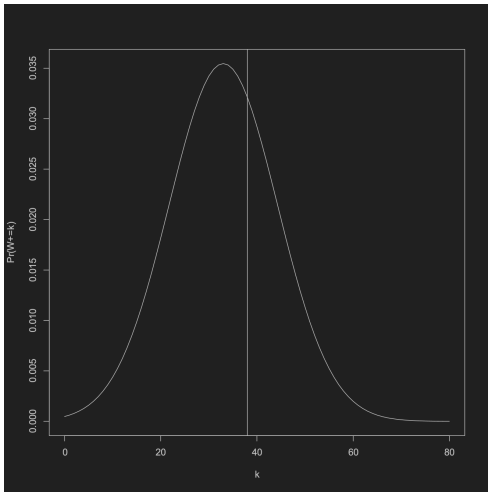
```
    One-sample Sign-Test

data:  Z
s = 6, p-value = 0.5
alternative hypothesis: true median is greater than 0
95 percent confidence interval:
 -557.0364       Inf
sample estimates:
median of x
       127

Achieved and Interpolated Confidence Intervals:

                  Conf.Level    L.E.pt U.E.pt
Lower Achieved CI     0.8867 -451.0000    Inf
Interpolated CI       0.9500 -557.0364    Inf
Upper Achieved CI     0.9673 -586.0000    Inf
```

(b) Ergebnisse

Abbildung 1: Vorzeichentest

II.2.3 Wilcoxontest

Auch den Wilcoxon-Rank Test haben wir basierend auf dem bereitgestellten R-Script durchgeführt. Und auch bei diesem Test lässt sich bereits am Plot der Teststatistik in Abbildung 2a gut erkennen, dass bei Algorithmen zu etwa gleich guten Ergebnissen kommen. Beim eigentlichen Test erhalten wir einen p-value von $0,35$, was ebenfalls deutlich über dem Signifikanzniveau von $\alpha = 0,05$ liegt, wenn auch nicht ganz so deutlich wie bei Teilaufgabe II.2.2. Dennoch suggeriert auch der Wilcoxon-Test klar, dass wir H_0 annehmen und H_1 und H_2 ablehnen müssen.



(a) Plot der Teststatistik

```
data:  Z
V = 38, p-value = 0.3501
alternative hypothesis: true location is greater than 0
```

(b) Ergebnisse

Abbildung 2: Wilcoxon Signed Rank Test

II.2.4 Ergebnis

Insgesamt müssen wir feststellen, dass unser Versuch, den Greedy-Algorithmus zu verbessern, gescheitert ist. Nicht nur haben wir durch die zusätzlichen Rechenschritte eine deutlich längere Laufzeit bekommen – der Aufwand lohnt sich noch nicht einmal mit Hinblick auf das Ergebnis. Die Hypothesentests zeigen deutlich, dass beide Algorithmen zu etwa gleich guten Ergebnissen kommen.

III Problem 3: CVRP – Virtual Running Time

III.1 Problemstellung

For the following task, choose any of your algorithms for the CVRP.

1. Select a small subset S of lines such that the set α_S of operations is representative (see Slide 150). Elaborate why α_S is in fact representative.
2. Choose a set $\mathcal{I}$ of at least $|S|$ instances of increasing size. For each instance $I \in \mathcal{I}$, measure the running time $\text{CPU}(I)$ of your algorithm and determine the numbers $\alpha k(I)$ for each representative line $k \in S$. Present your results in a table.
3. Identify a line $k \in S$ such that α_k is an *asymptotic bottleneck operation* (see Slide 153) or show that no such operation exists. For this purpose, draw the curves α_k / α_S over increasing instance size.

III.2 Lösung

Wir haben für diese Aufgabe unsere in Aufgabe I beschriebene Implementierung, den AverageDistanceSolver, betrachtet.

III.2.1 Repräsentative Operationen

Unser Set S besteht aus folgenden Codezeilen a_k:

a_1 Zeile 25: Für jeden aktuellen Standpunkt werden die fünf nächsten Nachbarn herausgesucht. Dazu müssen alle noch nicht besuchten Knoten betrachtet werden. Diese Zeile ist repräsentativ, da die $(n-1)$-mal ausgeführte find_nearest_five()-Funktion durch den `for`-Loop im Worstcase zu einer Laufzeit von $\mathcal{O}(n^2)$ führen kann.

a_2 Zeile 37: Die Berechnung der allgemeinen Durchschnittsdistanz aller noch nicht besuchten Knoten muss für jede Tour zwischen zwei Knoten aufs neue geschehen, da ja immer ein Knoten weniger dabei ist. Diese Zeile ist repräsentativ, da die total_average_distance()-Funktion mit dem doppelten `for`-Loop im Worstcase eine Laufzeit von $\mathcal{O}(\binom{n}{2})$ hat, was insgesamt zu einer Laufzeit von $\mathcal{O}(n^2)$ führen kann.

a_3 Zeile 44: Für jede Tour zwischen zwei Knoten muss für jeden der bis zu fünf Kandidaten die aktuelle individuelle Durchschnittsdistanz neu berechnet werden. Diese Zeile ist repräsentativ, da die $(n-1)$-mal ausgeführte average_distance()-Funktion durch den for-Loop im Worstcase eine Laufzeit von $\mathcal{O}(5n)$ hat, was insgesamt zu einer Laufzeit von $\mathcal{O}(n^2)$ führen kann.

III.2.2 Tabelle Laufzeiten & Operation Counts

Instanz-Name	**Größe**	CPU(I) **(sec.)**	$a_1(I) = (n-1)\cdot n$	$a_2(I) = \sum_{i=1}^{n}\binom{i}{2}$	$a_3(I) = (n-1)\cdot 5n$
X-n101-k25	101	0,031918	10.100	171.700	50.500
ORTEC-n242-k12	242	0,335361	58.322	2.362.041	291.610
Loggi-n401-k23	401	1,386524	160.400	10.746.800	802.000
X-n561-k42	561	3,743836	314.160	29.426.320	1.570.800
X-n599-k92	599	4,5182280	358.202	35.820.200	1.791.010
ORTEC-n701-k64	701	7,146705	490.700	57.411.900	2.453.500
Loggi-n901-k42	901	15,197763	810.900	121.905.300	4.054.500
X-n916-k207	916	15,873696	838.140	128.095.730	4.190.700
X-n1001-k43	1.001	20,345868	1.001.000	167.167.000	5.005.000
Leuven1	3.001	543,5946	9.003.000	4.504.501.000	45.015.000
Leuven2	4.001	1286,0946	16.004.000	10.674.668.000	80.020.000

III.2.3 Asymptotic Bottleneck Operation

a_2 ist eine *asymptotic bottleneck operation*. Das lässt sich schon aus Tabelle III.2.2 deutlich ablesen: Mit zunehmender Instanz-Größe nimmt die Laufzeit immer stärker zu.