Problem Set 4

CVRP Solution Quality

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

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

I Problem 1: Random Instance Generator

Design and implement an instance generator to produce random CVRP instances. Your generator should have at least one parameter n, the number of locations, and should produce a variety of different instances.
Briefly describe

- the generator's parameters
- its key ideas
- and how to use it

I.a Grundidee & Parameter

Wir haben den Instanz-Generator als Klasse implementiert, deren Konstruktor die Anzahl N der zu erstellenden Instanzen und die Anzahl n der Locations als Parameter entgegennimmt. Die Implementierung ist, wie immer, in unserem GitHub-Repository zu finden und wird im Folgenden näher erläutert.

Die Grundidee unseres Instanz-Generators ist, dass alle Parameter bis auf die Anzahl der Locations (begrenzt) zufällig ausgewählt werden. Diese Auswahl geschieht mit Hilfe des `random` Python-Moduls, welches pseudo-zufällige Ganzzahlen generieren kann. Natürlich ist das nicht *echt* zufällig, aber für unsere Testzwecke dennoch ausreichend.

Der Instanz Generator ruft die in Abbildung 1 gezeigte Funktion `create_locations()` auf. Wie in den Zeilen 2-3 zu sehen, setzen wir für die `x-` und `y-Koordinaten`, den `Demand` der einzelnen Kunden und die Vehicle-`Capacity` per Default obere Grenzen, die aber aller optional verändert werden können.

In der eigentlichen Funktion wird zuerst ein LocationsContainer der richtigen Größe instanziiert (s. Abbildung 1, Z. 6-7), der anschließend mit zufällig erzeugten Locations gefüllt wird (Z. 8-18). In Zeile 23 wird die Capacity zufällig bestimmt und in Zeile 28 mit den so erstellten Daten ein Problem-Objekt instanziiert.

```
1  def create_locations(number_of_locations,
2                       max_x=1000, max_y=1000,
3                       max_demand=101, capacity_range=(180, 400),
4                       instance_id=0):
5
6      locations = lc.LocationsContainer()
7      locations.init_location_list(int(number_of_locations))
8      for n in range(0, int(number_of_locations)):
9          node_id = n
10         x = random.randint(0, max_x)
11         y = random.randint(0, max_y)
12         if node_id == 0:
13             demand = 0
14         else:
15             demand = random.randint(1, max_demand)
16
17         node = ln.LocationNode(node_id=n, x_coord=x, y_coord=y, demand=demand)
18         locations.add_location(node)
19
20     locations.set_depot_id(0)
21     locations.create_distance_matrix()
22
23     capacity = random.randint(capacity_range[0], capacity_range[1])
24     filename = "Custom-n" + str(number_of_locations)
25               + "-" + "q" + str(capacity)
26               + "_" + str(instance_id) + ".vrp"
27
28     problem = prbl.Problem(locations=locations,
29                            capacity=capacity,
30                            file_name=filename)
31
32     return problem
```

Abbildung 1: create_locations() Funktion des Random Instance Generators

Die Instanzen werden als einzelne .vrp-Files abgespeichert, wobei der Dateiname das Format Custom-n<locations>-q<capacity>__<instance-ID>.vrp hat.
Die instance_ID (s. auch Abbildung 1, Z. 4) soll bei der Erstellung mehrerer Instanzen für dasselbe n verhindern, dass Instanzen, die zufällig auch dieselbe Capacity haben, sich gegenseitig überschreiben.

I.b How to use

```
1  from classes import InstanceGenerator as IG
2
3
4  def main():
5      vrp_folder = "../data/custom/"
6      numbers_of_locations = [100, 150, 200, 250, 300, 350, 400, 500, 600, 700]
7      number_of_instances = 1000
8
9      IG.InstanceGenerator(numbers_of_locations=numbers_of_locations,
10                          number_of_instances=number_of_instances,
11                          vrp_folder=vrp_folder)
```

Abbildung 2: Wie der InstanceGenerator genutzt werden kann

Das Script instance_generator.py zeigt beispielhaft, wie der Instanz Generator genutzt werden kann. Benötigt werden der Dateipfad, wo die .vrp-Dateien später abgelegt werden sollen (Abbildung 2, Z.5), die Anzahl der Locations n als Liste (Z.6) und die Anzahl N der Instanzen, die für jedes n generiert werden sollen (Z.7). Mit diesen Parametern wird der InstanceGenerator instanziiert (Z.9). Die Funktion create_locations() wird in dessen Konstruktor aufgerufen (s. Abbildung 3):

```
1  class InstanceGenerator:
2
3      def __init__(self, numbers_of_locations, number_of_instances, vrp_folder,
4                   max_x=1000, max_y=1000,
5                   max_demand=101, capacity_range=(180, 400),
6                   instance_id=0
7                   ):
8          for n in numbers_of_locations:
9              for i in range(number_of_instances + 1):
10                 problem = create_locations(n,
11                                            max_x, max_y,
12                                            max_demand,
13                                            capacity_range,
14                                            instance_id=i)
15
16                 problem.write_vrp_file(n, vrp_folder)
```

Abbildung 3: Konstruktor des Instance Generators

II Problem 2: Average Solution Quality

Experimentally analyse the average solution quality of any of your previous algorithms.

Wir haben unseren Greedy-Algorithmus je 1000 mit dem Random Instance Generator aus Aufgabe I erzeugte Instanzen für zehn verschiedene n lösen lassen. Dazu haben wir ein Script geschrieben, das rekursiv durch einen gegebenen Ordner läuft, alle `.vrp`-Dateien einliest, mit dem Greedy Solver löst und alle für Aufgabe II und Aufgabe III benötigten Parameter berechnet. Die einzelnen Parameter haben wir mit den in Abbildung 4 gezeigten Hilfsfunktionen berechnet und in einer eigenen Datenstruktur zwischengespeichert.

```
1  def mean(x_values):
2      res = sum(x_values) / len(x_values)
3      return round(res, 3)
4
5
6  def variance(x_values, x_mean):
7      var = sum([(x - x_mean) ** 2 for x in x_values]) / (len(x_values) - 1)
8      return round(var, 3)
9
10
11 def covariance(x_values, x_mean, y_values, y_mean):
12     cov = sum([(x - x_mean) * (y - y_mean) for x, y in zip(x_values, y_values)])
13     return round((cov / (len(x_values) - 1)), 3)
14
15
16 def random_variate(x_values, y_values, y_variance, covariance):
17     rd_variates = [x - (covariance / y_variance) * (y - y_variance)
18                    for x, y in zip(x_values, y_values)]
19     return rd_variates
```

Abbildung 4: Hilfsfunktionen für die Berechnung der statistischen Parameter

1 `mean()` berechnet das arithmetische Mittel $\bar{X} = \frac{1}{N}\sum_{j=1}^{N} X_j$

6 `variance()` berechnet die Varianz $V(X) = \frac{1}{N-1}\sum_{j=1}^{N}(X_j - \bar{X})^2$

11 `covariance()` berechnet die Covarianz $Cov(X,Y) = \frac{1}{N-1}\sum_{j=1}^{N}(X_j - \bar{X})(Y_j - \bar{Y})$

16 `random_variate()` berechnet $Z = X - a(Y - \bar{Y})$ für $a = \frac{Cov(X,Y)}{V(Y)}$

Die Ergebnisse haben wir uns als JSON-Datei ausgeben lassen, um sie für die Plots einfacher weiterverarbeiten zu können.

II.a Mean und Variance der Tourlängen

Run your algorithm on a set of $N = 1000$ random instances (for some fixed value of n) and output the total tour lengths $X_1, X_2, ..., X_N$.
Calculate

- the sample mean $\bar{X} = \frac{1}{N} \sum_{j=1}^{N} X_j$
- and sample variance $V(X) = \frac{1}{N-1} \sum_{j=1}^{N} (X_j - \bar{X})^2$

Repeat this experiment for 10 different values of n. Present $\bar{X}$ and $V(X)$ in a table.

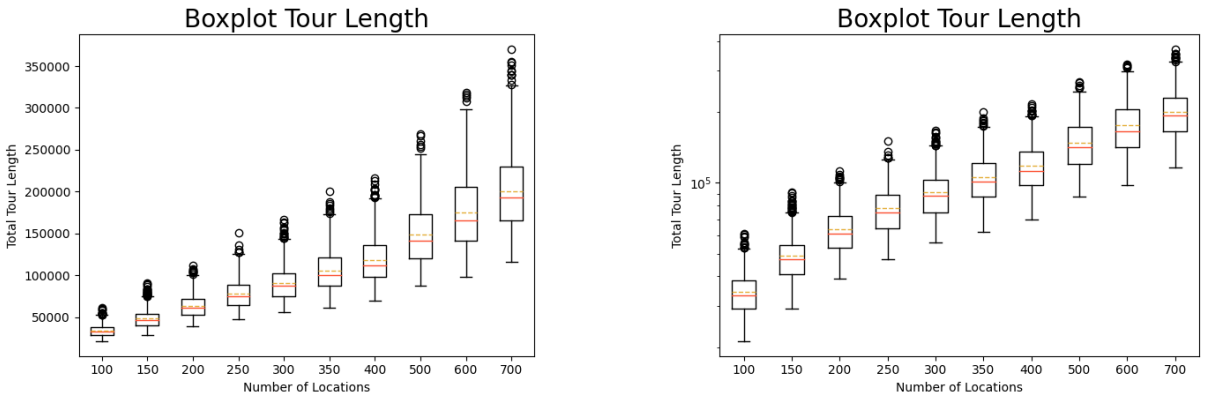
Tabelle 1 zeigt die durchschnittliche Tourlänge $\bar{X}$ und die entsprechende Varianz $Var(X)$ für die 1000 gelösten Instanzen für jedes n.

No. Locations	Tour Lengths	
n	$\bar{X}$	$Var(X)$
100	34 431.36	50 032 158.108
150	48 854.83	114 454 431.743
200	63 530.045	193 129 869.434
250	78 183.071	326 588 157.695
300	90 847.992	425 357 931.395
350	105 743.34	615 483 525.571
400	118 545.003	748 776 697.085
500	148 407.506	1 276 202 989.058
600	175 437.801	1 993 049 679.232
700	200 774.71	2 231 183 047.503

Tabelle 1: Durchschnittliche Tourlängen und Varianz der Tourlängen für verschiedene n

II.b Plot der Ergebnisse

Visualize your data by drawing a box-and-whisker diagram (see Slide 164) in which the x-axis displays the number n of locations and the y-axis the total tour lengths X. Think about the scaling of the axes and whether normalization might be a good idea.



(a) y-Achse nicht skaliert

(b) y-Achse logarithmisch skaliert

Abbildung 5: Boxplot der Tourlängen

Abbildung 5 zeigt den Boxplot der Tourlängen für verschiedene n. Da unsere gewählten n nicht allzu weit auseinander liegen und die Grids der Kunden in allen Instanzen die selben Dimensionen haben, kann man bei uns auch mit linearer Skalierung, wie in Abbildung 5a, die Tour Length Distribution recht gut erkennen und vergleichen. Gut zu sehen ist, dass die durchschnittliche Gesamtlänge der Touren ansteigt, je mehr Kunden zu besuchen sind – ein vorhersehbares Ergebnis. Zum Vergleich zeigt Abbildung 5b dieselben Daten mit logarithmisch skalierter y-Achse. Hier lässt sich die Verteilung noch etwas genauer erkennen, aber einen großen Vorteil bietet die Skalierung an dieser Stelle nicht.
Andererseits verzerrt die logarithmische Skalierung aber auch das Ergebnis: In Abbildung 5a lässt sich gut erkennen, dass die Varianz der durchschnittlichen Tourlängen ebenfalls mit größer werdendem n ansteigt. In Abbildung 5b sieht es hingegen so als, als würde die Varianz für alle n etwa gleich groß bleiben, während nur die durchschnittliche Tourlänge ansteigt.

III Variance Reduction

Refine the previous experiment by using control variates (see Slide 209).

III.a Mean & Variance der Subtouren

Next to the tour length X, now also output the number Y of subtours in your solution. Calculate the sample mean $\bar{Y}$ and sample variance $V(Y)$. Add $\bar{Y}$ and $V(Y)$ to your table.

Auch die Anzahl der Subtouren Y haben wir uns in der JSON-Datei wie in Aufgabe II beschrieben mit ausgeben lassen. Die Ergebnisse für den Durchschnitt und die Varianz von Y haben wir in Tabelle 2 in den Spalten 4 und 5 abgebildet.

III.b Covarianz von Tourlänge und Anzahl der Subtouren

Calculate the covariance

$$Cov(X,Y) = \frac{1}{N-1}\sum_{j=1}^{N}(X_j - \bar{X})(Y_j - \bar{Y})$$

Add the values of $Cov(X,Y)$ to the table. Is X positively correlated with Y? If so, why?

Die Kovarianz wurde wie in Aufgabe II beschrieben berechnet und in Tabelle 2 in Spalte 6 eingefügt.
Die berechneten Werte zeigen deutlich, dass X und Y tatsächlich positiv korrelieren. Alle Kovarianzen sind positiv, was bedeutet, dass wenn X steigt, Y ebenfalls steigt, und umgekehrt.

III.c Random Variate Z

Now consider the random variate $Z = X - a(Y - \bar{Y})$ for $a = \frac{Cov(X,Y)}{V(Y)}$.
Calculate the sample mean $\bar{Z}$ and sample variance $V(Z)$ and add these to the table, too.

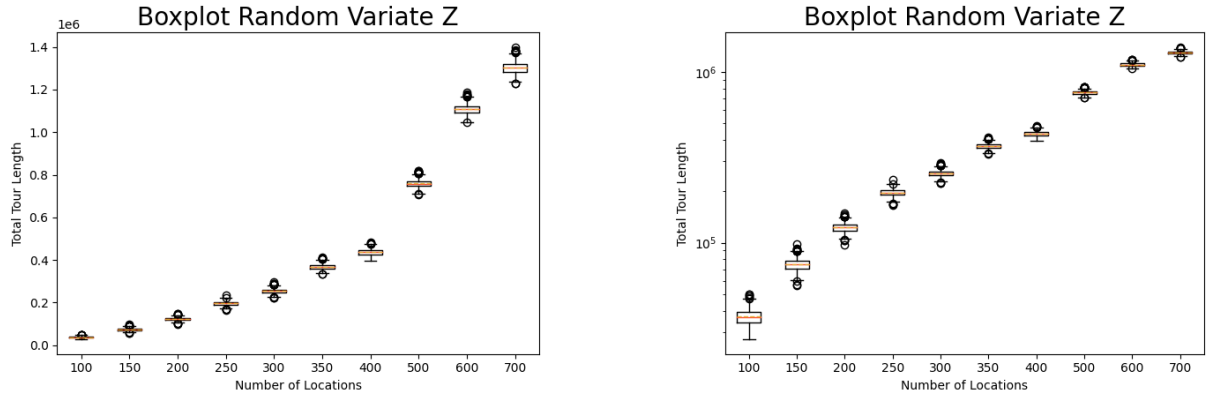
Die Zufallsvariable Z wurde wie in Aufgabe II beschrieben berechnet. Die Durchschnittswerte $\bar{Z}$ und Varianzen $Var(Z)$ wurden in den Spalten 7 und 8 in Tabelle 2 eingefügt.

No. Locations	Tour Lengths		No. Subtours		Covariance	Random Variate	
n	$\bar{X}$	$Var(X)$	$\bar{Y}$	$Var(Y)$	$Cov(X,Y)$	$\bar{Z}$	$Var(Z)$
100	34 431.36	50 032 158.108	19.367	21.251	27 444.486	36 863.967	14 588 591.892
150	48 854.83	114 454 431.743	28.455	48.026	63 270.75	74 637.594	31 099 522.95
200	63 530.045	193 129 869.434	38.094	83.632	109 072.763	122 920.532	50 878 004.755
250	78 183.071	326 588 157.695	47.129	134.056	183 012.846	196 855.372	76 740 423.3
300	90 847.992	425 357 931.395	56.052	178.316	239 409.218	255 001.047	103 924 085.56
350	105 743.34	615 483 525.571	66.08	268.028	350 013.035	369 463.565	158 407 081.905
400	118 545.003	748 776 697.085	74.358	313.078	416 468.234	436 098.928	194 774 608.416
500	148 407.506	1 276 202 989.058	95.476	558.809	736 604.147	759 157.616	305 234 646.22
600	175 437.801	1 993 049 679.232	113.42	785.08	1 089 203.314	1 107 284.033	481 911 335.838
700	200 774.71	2 231 183 047.503	131.961	1 002.671	1 269 484.802	1 303 183.338	623 884 188.791

Tabelle 2: Tabelle aller Ergebnisse von Teilaufgabe III.a, Teilaufgabe III.b und Teilaufgabe III.c

III.d Varianzreduktion

Visualize the random variates Z by drawing a box-and-whisker diagram similar to Teilaufgabe II.b. Do you observe a reduced variance? Critically discuss your results.



(a) y-Achse nicht skaliert **(b)** y-Achse logarithmisch skaliert

Abbildung 6: Boxplot der Zufallsvariable Z

Wie in Abbildung 6 zu sehen ist, kann eine deutliche Reduzierung der Varianz der Tour Lengths im Vergleich zu Abbildung 5 festgestellt werden. Dies war auch durch die positive Korrelation von X und Y zu erwarten.

IV Randomized Horse Race Experiment

Choose two of your algorithms for the CVRP. Setup an experiment to test which algorithm gives better results on average, when tested on the random instances produced by your instance generator. For this purpose, evaluate the difference $D = A - B$ between the total tour lengths A and B of your respective algorithms.

Zur Bearbeitung dieser Aufgabe haben wir ein Script geschrieben, das die jeweiligen Tourlängen zwischenspeichert und dann die benötigten Parameter direkt berechnet und als JSON-Datei ausgibt.

IV.a Durchschnittliche Tour-Länge Average Distance

Fix the number n of locations to some meaningful value and generate a large number of N random instances. Run your first algorithm to produce the total tour lengths $A_1, A_2, ..., A_N$ and calculate the sample mean $\bar{A}$.

Wir haben $n = 300$ gesetzt und unseren Average Distance Solver die 1000 n-300 Instanzen aus Aufgabe II lösen lassen.
Die durchschnittliche Tourlänge dabei war $\bar{A} = 92537.445$

IV.b Durchschnittliche Tour-Länge Greedy

Create N new random instances (using the same value of n) and run your second algorithm to produce the total tour lengths $B_1, B_2, ..., B_N$. Calculate the sample mean $\bar{B}$ and the difference $D = \bar{A} - \bar{B}$ of the sample means.

Mit unserem Random Instance Generator aus Aufgabe I haben wir 1000 neue Instanzen für $n = 300$ generiert und diese von unserem Greedy Solver lösen lassen.
Die durchschnittliche Tourlänge dabei war $\bar{B} = 91485.823$.
Das ergibt eine positive Differenz von $D = \bar{A} - \bar{B} = 1051.622$, was bedeutet, dass der Greedy Algorithmus (B) im Schnitt kürzere Touren gefunden hat als der Average Distance Algorithmus (A).

IV.c Durchschnittliche Differenz

Now run both algorithms on the same N random instances to produce the differences $D_i = A_i - B_i$ in tour length for $i = 1, 2, ..., N$. Calculate the sample mean $\bar{D}$. Compare with the value obtained in Teilaufgabe IV.b. Discuss your findings.

Wir haben beide Algorithmen auf beiden Instanz-Mengen laufen lassen. Für die erste Menge von Instanzen ergibt sich eine durchschnittliche Differenz $\bar{D}_1$ von 1713.293. Für die zweite Menge von Instanzen ist die durchschnittliche Differenz $\bar{D}_2 = 1743.042$.

Auch hier sind beide Differenzen positiv, was bedeutet, dass der Greedy Algorithmus durchschnittlich bessere Ergebnisse erzielt als unsere Average-Distance Heuristic. Gleichzeitig ist die Differenz aber auch nicht übermäßig groß, was sich mit den Ergebnissen unseres Hypothesentests aus Aufgabenblatt 3 deckt. Hier hatten die statistischen Tests nahegelegt, die Nullhypothese, dass die beiden Algorithmen sich nicht unterscheiden, anzunehmen.