3.3 Heuristiken: schnell eine gute Lösung finden

Leitidee

Brute-Force findet immer die beste Rundreise – aber schon bei wenigen Orten wird das Ausprobieren sehr mühsam. Eine Paketbotin mit 100 Adressen kann unmöglich alle Reihenfolgen durchrechnen. Was macht sie stattdessen? Vermutlich etwas ganz Naheliegendes: Sie fährt immer zur nächstgelegenen Adresse, die sie noch nicht beliefert hat. So eine Faustregel heißt in der Informatik **Heuristik**. In diesem Abschnitt lernst du, wie solche Faustregeln funktionieren, wendest eine davon von Hand an und prüfst, wie gut ihr Ergebnis ist.

Lernziele

Nach diesem Abschnitt kannst du

- den Begriff Heuristik definieren und an einem Beispiel erklären,
- beschreiben, was eine Greedy-Heuristik auszeichnet,
- die Nearest-Neighbor-Heuristik von Hand auf eine Probleminstanz anwenden,
- das Ergebnis einer Heuristik mit der optimalen Lösung vergleichen und die Abweichung in Prozent berechnen,
- Vor- und Nachteile von Heuristik und Brute-Force gegenüberstellen.

3.3.1 Faustregeln statt Ausprobieren

Im Alltag benutzt du ständig Faustregeln. An der Supermarktkasse stellst du dich an die kürzeste Schlange. Das geht schnell und klappt meistens gut. Eine Garantie gibt es aber nicht: Vielleicht hat die Person vor dir einen riesigen Einkauf, und an der Nachbarkasse wärst du schneller gewesen. Um sicherzugehen, müsstest du alle Einkaufswagen genau ansehen – das dauert länger als das Warten selbst.

Genau solche Faustregeln nennt man in der Informatik **Heuristiken**. Das Wort kommt vom griechischen *heuriskein*, „finden“. Eine Heuristik verzichtet darauf, alle Möglichkeiten zu prüfen. Dafür ist sie schnell.

Definition: Heuristik

Eine **Heuristik** ist eine Lösungsstrategie, die **schnell** eine zulässige Lösung liefert, ohne alle Möglichkeiten zu prüfen. Die Lösung ist meist gut, aber **nicht garantiert optimal**.

Merke

Eine **Heuristik** ist eine Lösungsstrategie, die **schnell** eine (meist gute, aber nicht garantiert optimale) Lösung liefert, ohne alle Möglichkeiten zu prüfen.

3.3.2 Greedy: immer das, was gerade am besten aussieht

Eine besonders einfache Art von Heuristik geht so vor: Sie baut die Lösung Schritt für Schritt auf. In jedem Schritt nimmt sie das, was **im Moment** am besten aussieht. Was später passiert, bedenkt sie nicht. Eine einmal getroffene Entscheidung wird nie mehr geändert.

Solche Heuristiken heißen **Greedy-Heuristiken** (englisch *greedy*, gesprochen „griedi“, heißt „gierig“). Denk an ein Kind am Buffet, das sich zuerst das größte Stück Kuchen nimmt – ohne zu überlegen, ob danach noch Platz für den Nachtisch bleibt.

Definition: Greedy-Heuristik

Eine **Greedy-Heuristik** baut eine Lösung schrittweise auf. In jedem Schritt trifft sie die Entscheidung, die **im Moment** am besten erscheint, ohne die Auswirkungen auf spätere Schritte zu berücksichtigen.

Eine Greedy-Strategie hast du übrigens schon in Klasse 9 kennengelernt: Beim Lernen eines Entscheidungsbaums wählt man in jedem Schritt das Merkmal, das die Daten im Moment am besten aufteilt. Ob eine andere Wahl später zu einem besseren Baum geführt hätte, prüft man nicht.

Merke

Eine **Greedy-Heuristik** („gierig“) trifft in jedem Schritt die Entscheidung, die im **Moment** am besten erscheint, ohne spätere Auswirkungen zu berücksichtigen.

3.3.3 Die Nearest-Neighbor-Heuristik

Für das Problem des Handlungsreisenden gibt es eine naheliegende Greedy-Heuristik. Sie heißt **Nearest Neighbor** (englisch für „nächster Nachbar“, gesprochen „nierest neiber“). Ihre Regel lautet: Fahre immer zum nächstgelegenen Ort, an dem du noch nicht warst.

So gehst du Schritt für Schritt vor:

1. Beginne im Startort. Er ist der **aktuelle Ort**.
2. Suche unter allen **noch nicht besuchten** Orten den, der vom aktuellen Ort am wenigsten weit entfernt ist. Sind zwei Orte gleich weit entfernt, nimm den, der im Alphabet zuerst kommt.
3. Fahre dorthin. Dieser Ort ist jetzt der aktuelle Ort und gilt als besucht.
4. Wiederhole die Schritte 2 und 3, bis alle Orte besucht sind.
5. Fahre zurück zum Startort.

Merke

Beim Problem des Handlungsreisenden wählt die Greedy-Heuristik **Nearest Neighbor** von der aktuellen Stadt aus immer die **nächstgelegene, noch nicht besuchte** Stadt.

Beispiel: Nearest Neighbor von Hand, Start in A

Wir benutzen wieder die Entfernungstabelle der vier Orte:

	A	B	C	D
A	–	10	15	20
B	10	–	35	25
C	15	35	–	30
D	20	25	30	–

Schritt	aktueller Ort	noch nicht besucht (Entfernung)	Wahl
1	A	B 10, C 15, D 20	B (10 km)
2	B	C 35, D 25	D (25 km)
3	D	C 30	C (30 km)
4	C	alle besucht	zurück nach A (15 km)

Rundreise: A → B → D → C → A. Länge: $10 + 25 + 30 + 15 = 80$ km.

Lies die Tabelle Zeile für Zeile:

- **Schritt 1:** Von A aus ist B am nächsten (10 km). C und D sind weiter weg.
- **Schritt 2:** In B schaust du nur noch auf C und D, denn A ist schon besucht. D ist mit 25 km näher als C mit 35 km.
- **Schritt 3:** Von D aus bleibt nur noch C übrig. Hier gibt es nichts zu entscheiden.
- **Schritt 4:** Alle Orte sind besucht. Zum Schluss geht es zurück zum Start.

Vergleiche mit dem Ergebnis von Brute-Force aus dem letzten Abschnitt: Die optimale Rundreise ist A → B → D → C → A mit 80 km. Die Heuristik hat hier also mit nur vier kurzen Blicken in die Tabelle die optimale Lösung gefunden. Ist das immer so? Das untersuchst du jetzt.

Stolperstein

Die drei häufigsten Fehler bei Nearest Neighbor:

- Du wählst einen Ort, der **schon besucht** ist (meistens den Startort). Streiche besuchte Orte am besten in der Tabelle durch.
- Du vergisst am Ende die **Rückfahrt** zum Start.
- Du schaust „voraus“ und wählst einen Ort, weil es danach günstiger weitergeht. Das ist zwar klug, aber nicht mehr die Nearest-Neighbor-Heuristik. Sie schaut nur auf den aktuellen Schritt.

Arbeitsauftrag 26: Anderer Startpunkt

Wende die Nearest-Neighbor-Heuristik **von Hand** an, aber starte diesmal in Ort **C**. Lege dazu eine Tabelle an wie im Beispiel. Vergleiche Rundreise und Länge mit dem Ergebnis für den Start in A. Was fällt dir auf?

Deine Tabelle und dein Vergleich:

Arbeitsauftrag 27: Start in D

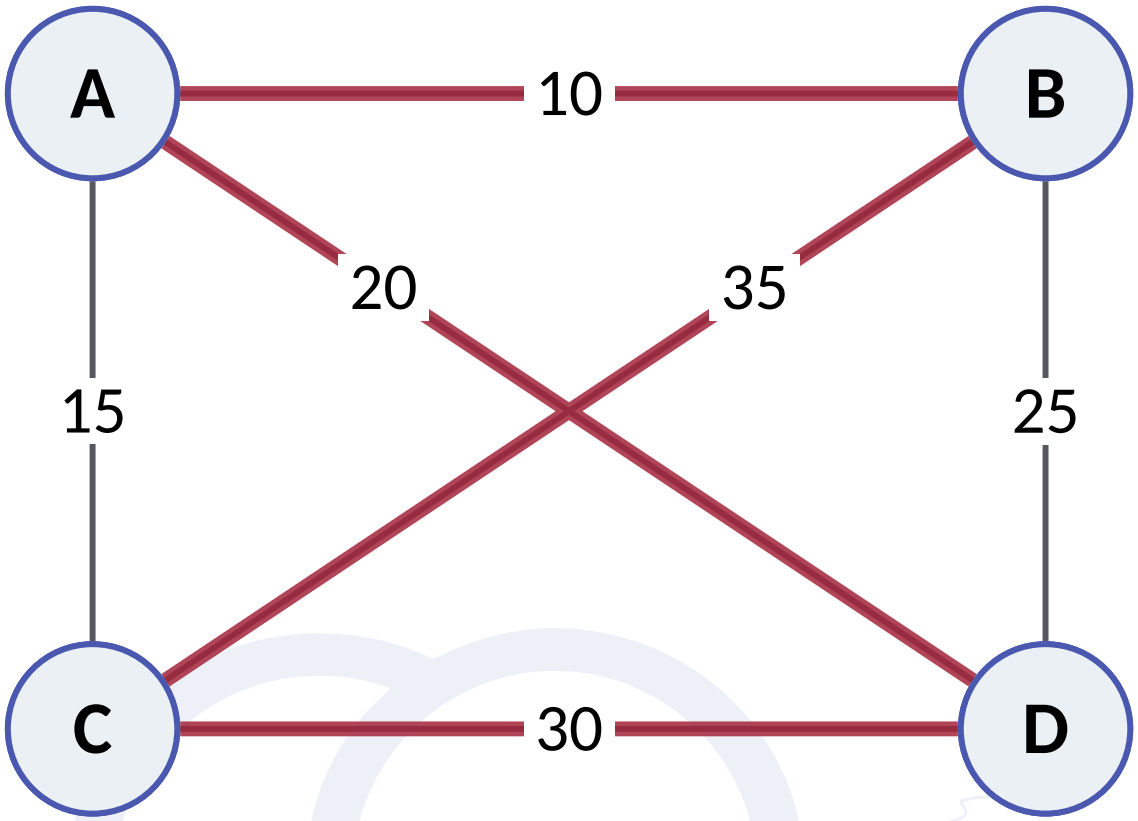
Wende die Nearest-Neighbor-Heuristik noch einmal an, jetzt mit Start in **D**. Ist das Ergebnis optimal? Erkläre, an welcher Stelle die Heuristik „in eine Falle läuft“.

Gymnasium Ottweiler

Deine Tabelle und deine Erklärung:

Die Heuristik liefert also je nach Startort unterschiedlich gute Ergebnisse. Bei Start in A, B oder C kommt sie auf 80 km, bei Start in D auf 95 km. Im Graphen siehst du, woran es liegt: Die Rundreise vom Start D aus benutzt die teure Kante B–C.

Gymnasium Ottweiler



Nearest Neighbor mit Start in D liefert D → A → B → C → D (rot) mit $20 + 10 + 35 + 30 = 95$ km. Optimal wären 80 km

Merke

Heuristiken sind **schneller** als Brute-Force, liefern aber **nicht immer** die optimale Lösung – manchmal führt eine „im Moment beste“ Entscheidung in eine Sackgasse.

3.3.4 Wie gut ist die Heuristik? Die Abweichung in Prozent

„95 km statt 80 km“ – ist das viel oder wenig? Um das zu beurteilen, rechnet man aus, um **wie viel Prozent** die Lösung der Heuristik schlechter ist als die optimale Lösung. Das kennst du aus Mathe als prozentuale Änderung:

$$\text{Abweichung} = \frac{\text{Länge der Heuristik} - \text{optimale Länge}}{\text{optimale Länge}} \cdot 100\,\%$$

Für den Start in D:

$$\text{Abweichung} = \frac{95 - 80}{80} \cdot 100\,\% = \frac{15}{80} \cdot 100\,\% = 18{,}75\,\% \approx 18{,}8\,\%$$

Die Rundreise der Heuristik ist also knapp 19 % länger als nötig. Bei einer Abweichung von 0 % hat die Heuristik die optimale Lösung getroffen.

Stolperstein

Teile immer durch die **optimale** Länge, nicht durch die Länge der Heuristik. $\frac{15}{95} \approx 15{,}8\,\%$ wäre falsch. Die Frage lautet ja: Um wie viel ist die Heuristik schlechter **als das Optimum**?

Arbeitsauftrag 28: Heuristik und Optimum

Nimm deine eigene Probleminstanz mit fünf Orten aus dem Auftrag *Neue Probleminstanz mit fünf Orten* (oder die dort vorgeschlagene Tabelle).

1. Wende die Nearest-Neighbor-Heuristik mit Start in A an. Gib die Rundreise und ihre Länge an.
2. Vergleiche mit der optimalen Rundreise, die du mit Brute-Force gefunden hast. Berechne die **prozentuale Abweichung**.

Deine Tabelle und deine Rechnung:

Gymnasium Ottweiler

3.3.5 Heuristik oder Brute-Force?

Warum benutzt man überhaupt Heuristiken, wenn sie nicht immer optimal sind? Weil sie **viel weniger Arbeit** machen. Zähle einmal, wie oft man bei Nearest Neighbor eine Entfernung ansehen muss: Bei vier Orten sind es im ersten Schritt 3 Entfernungen, im zweiten 2 und im dritten 1, zusammen also 6. Bei 10 Orten sind es $9+8+7+\ldots+1 = 45$. Brute-Force müsste bei 10 Orten dagegen $9! = 362\,880$ Rundreisen durchrechnen.

	Brute-Force	**Nearest Neighbor**
Ergebnis	garantiert optimal	zulässig, meist gut, nicht immer optimal
Arbeit bei 4 Orten	6 Rundreisen berechnen	6 Entfernungen ansehen
Arbeit bei 10 Orten	362 880 Rundreisen berechnen	45 Entfernungen ansehen
geeignet für	kleine Probleminstanzen	auch große Probleminstanzen

So kannst du es sagen

So vergleichst du die beiden Ansätze in Fachsprache:

- *„Die Heuristik ist schneller, aber nicht so gut."* → *„Die Nearest-Neighbor-Heuristik benötigt deutlich weniger Rechenschritte als der Brute-Force-Ansatz, liefert aber nicht immer eine optimale Lösung."*
- *„Hier war sie 15 km schlechter."* → *„Bei Start in D weicht die Lösung der Heuristik um 18,75 % von der optimalen Lösung ab."*

3.3.6 Für Interessierte: Nearest Neighbor mit Python

Dieser Unterabschnitt ist freiwillig. Die Entfernungstabelle und die Funktionen `laenge` und `als_text` kennst du aus dem Abschnitt über Brute-Force. Neu ist die Funktion `nearest_neighbor`.

```
1  namen = ["A", "B", "C", "D"]
2  entfernung = [[0, 10, 15, 20],
3                [10, 0, 35, 25],
4                [15, 35, 0, 30],
5                [20, 25, 30, 0]]
6
7  def laenge(reise):
8      summe = 0
9      for i in range(len(reise) - 1):
10         summe += entfernung[reise[i]][reise[i + 1]]
11     return summe
12
13 def als_text(reise):
14     text = namen[reise[0]]
15     for i in range(1, len(reise)):
16         text = text + " -> " + namen[reise[i]]
17     return text
18
19 def nearest_neighbor(start):
20     reise = [start]
21     aktuell = start
22     while len(reise) < len(namen):
23         naechster = -1
```

```
24          kleinste = 1000000
25          for ort in range(len(namen)):
26              if ort not in reise:
27                  if entfernung[aktuell][ort] < kleinste:
28                      kleinste = entfernung[aktuell][ort]
29                      naechster = ort
30          reise.append(naechster)
31          aktuell = naechster
32      reise.append(start)
33      return reise
34
35  for start in range(len(namen)):
36      reise = nearest_neighbor(start)
37      print(als_text(reise), "Länge:", laenge(reise))
```

AUSGABE

```
A -> B -> D -> C -> A Länge: 80
B -> A -> C -> D -> B Länge: 80
C -> A -> B -> D -> C Länge: 80
D -> A -> B -> C -> D Länge: 95
```

- **Zeilen 19–21:** Die Funktion bekommt die Nummer des Startorts. Die Rundreise beginnt als Liste mit nur dem Startort. Er ist auch der aktuelle Ort.
- **Zeile 22:** Solange noch nicht alle Orte in der Liste stehen, wird ein weiterer Ort gesucht.
- **Zeilen 23–29:** Das ist die Suche nach dem nächsten Nachbarn. `kleinste` startet mit einer riesigen Zahl. Die Schleife geht alle Orte durch. `ort not in reise` ist wahr, wenn der Ort noch nicht besucht wurde. Ist er außerdem näher als der bisher nächste, merkt sich das Programm ihn. Bei gleicher Entfernung bleibt der zuerst gefundene Ort gemerkt, denn `<` ist nur bei einer echt kleineren Entfernung wahr. Das ist der Ort, der im Alphabet zuerst kommt – wie in der Regel oben.
- **Zeilen 30–31:** Der nächste Nachbar wird an die Reise angehängt und zum aktuellen Ort (**Greedy-Entscheidung**, die nie mehr geändert wird).
- **Zeilen 32–33:** Zum Schluss wird der Startort noch einmal angehängt (Rückfahrt), und die fertige Rundreise wird zurückgegeben.
- **Zeilen 35–37:** Die Heuristik wird für jeden Startort ausgeführt. Die Ausgabe bestätigt deine Handrechnungen.

Beachte: `in` funktioniert bei Listen genauso wie bei Zeichenketten. `3 in [0, 1, 3]` ist `True`.

Versuch: Ändere die Tabelle

Trage im Programm die Tabelle mit fünf Orten ein (eine fünfte Zeile, in jeder Zeile eine fünfte Zahl, und `"E"` in `namen`). Die Funktion `nearest_neighbor` musst du nicht ändern. Vergleiche die Ausgabe mit deiner Lösung zum Auftrag *Heuristik und Optimum*.

3.3.7 Übungen

Arbeitsauftrag 29: Nearest Neighbor von allen Startorten

Wende die Nearest-Neighbor-Heuristik auf die vier Orte A, B, C und D mit Start in **B** an. Fasse danach die Ergebnisse für alle vier Startorte in einer Tabelle zusammen (Startort, Rundreise, Länge, Abweichung vom Optimum 80 km).

Deine Rechnung und deine Tabelle:

Arbeitsauftrag 30 (Zusatz): Was würdest du empfehlen?

Ein Lieferdienst muss jeden Tag 40 Kundinnen und Kunden beliefern. Die Disponentin überlegt, ob sie mit einem Computer die optimale Route per Brute-Force berechnen lassen soll oder ob die Nearest-Neighbor-Heuristik reicht. Beurteile die beiden Möglichkeiten und gib eine begründete Empfehlung.

Gymnasium Ottweiler

Deine Beurteilung:

3.3.8 Weitere Übungen

Hier kannst du das Gelernte weiter üben. Die Punkte im Titel zeigen, wie schwer eine Aufgabe ist: ●○○ leicht, ●●○ mittel, ●●● knifflig. Fang mit den leichten an.

Arbeitsauftrag 31: Ausprobieren oder Faustregel? ●○○

Entscheide bei jeder Vorgehensweise, ob es sich um einen **Brute-Force-Ansatz** oder um eine **Heuristik** handelt. Gib bei den Heuristiken an, ob sie **greedy** sind, also in jedem Schritt das nehmen, was im Moment am besten aussieht.

Gymnasium Ottweiler

Vorgehensweise	Brute-Force oder Heuristik?	greedy?
Ben hat die Zahl seines Zahlenschlosses vergessen und probiert alle Zahlen von 000 bis 999 der Reihe nach aus.		
Beim Kartenspiel spielt Ida in jeder Runde ihre höchste Karte.		
Elif schreibt alle Reihenfolgen auf, in denen sie ihre vier Hausaufgaben machen kann, und wählt die, die am wenigsten Zeit kostet.		
Beim Wandern nimmt Paul an jeder Kreuzung den Weg, der am steilsten bergauf führt, um möglichst schnell zum Gipfel zu kommen.		
Eine Kassiererin gibt Wechselgeld, indem sie immer die größte Münze nimmt, die noch passt.		

Arbeitsauftrag 32: Nearest Neighbor beim Partyeinkauf ●○○

Mia startet zu Hause (Z) und fährt zur Bäckerei (B), zum Getränkemarkt (G) und zum Supermarkt (S). Die Entfernungen in km:

	Z	B	G	S
Z	–	3	7	6
B	3	–	9	5
G	7	9	–	4
S	6	5	4	–

Wende die Nearest-Neighbor-Heuristik mit Start in **Z** an. Fülle die Tabelle aus und gib die Rundreise und ihre Länge an.

Schritt	aktueller Ort	noch nicht besucht (Entfernung)	Wahl
1	Z		
2			
3			
4		alle besucht	

Rundreise und Länge:

Arbeitsauftrag 33: Start in der Bäckerei ●○○

Mias Freundin Ela wohnt direkt neben der Bäckerei. Sie startet deshalb in **B** und fährt am Ende wieder nach B zurück. Wende die Nearest-Neighbor-Heuristik mit Start in B an (Tabelle aus dem Auftrag *Nearest Neighbor beim Partyeinkauf*). Vergleiche die Länge mit der optimalen Rundreise (19 km). Um wie viele km ist Elas Rundreise länger?

Deine Tabelle und dein Vergleich:

Arbeitsauftrag 34: Stimmt das? ●○○

Entscheide, ob die Aussage richtig oder falsch ist. Begründe kurz.

Aussage	richtig oder falsch?	Begründung
Eine Heuristik prüft alle Möglichkeiten.		
Nearest Neighbor kann die optimale Rundreise finden.		
Nearest Neighbor liefert immer eine zulässige Rundreise.		
Eine Greedy-Heuristik ändert eine Entscheidung später noch, wenn sie sich als schlecht herausstellt.		
Heuristiken sind meistens schneller als Brute-Force.		

Arbeitsauftrag 35: Abweichung in Prozent ●●○

1. Berechne für jede Zeile die prozentuale Abweichung der Heuristik vom Optimum. Runde auf eine Stelle nach dem Komma.
2. Paul hat für die erste Zeile $\frac{22-19}{22} \cdot 100\,\% \approx 13{,}6\,\%$ gerechnet. Erkläre seinen Fehler.

Länge der Heuristik	optimale Länge	Rechnung	Abweichung
22 km	19 km		
60 km	50 km		
47 km	43 km		
30 km	30 km		

Pauls Fehler:

Arbeitsauftrag 36: Fehler im Protokoll ●●○

Lina hat Nearest Neighbor für die Radtour mit fünf Zielen (Auftrag *Radtour mit fünf Zielen* im Abschnitt über Brute-Force) mit Start in J angewendet. Hier ist die Tabelle noch einmal:

	J	B	S	T	W
J	–	9	11	10	13
B	9	–	6	14	4
S	11	6	–	15	9
T	10	14	15	–	12
W	13	4	9	12	–

Linas Protokoll:

Schritt	aktueller Ort	noch nicht besucht (Entfernung)	Wahl
1	J	B 9, S 11, T 10, W 13	B (9 km)
2	B	J 9, S 6, T 14, W 4	W (4 km)
3	W	S 9, T 12	T (12 km), weil es danach zum See geht
4	T	S 15	S (15 km)

Linas Ergebnis: J → B → W → T → S, Länge $9 + 4 + 12 + 15 = 40$ km.

1. Finde **drei** Fehler in Linas Protokoll und erkläre sie.
2. Führe die Heuristik richtig durch. Vergleiche mit der optimalen Rundreise (43 km) und berechne die Abweichung.

Linas Fehler und deine richtige Lösung:

Arbeitsauftrag 37: Die Falle selbst bauen ●●●

In dieser Entfernungstabelle fehlt eine Zahl: Die Entfernung zwischen R und S heißt x.

	P	Q	R	S
P	–	4	6	8
Q	4	–	7	5
R	6	7	–	x
S	8	5	x	–

1. Setze $x = 5$ ein. Wende Nearest Neighbor mit Start in P an. Bestimme mit Brute-Force die optimale Rundreise (3 verschiedene Rundreisen genügen). Ist die Heuristik optimal?
2. Setze $x = 20$ ein und wiederhole beides. Berechne die Abweichung.
3. Untersuche allgemein: Welche Rundreise wählt Nearest Neighbor, egal wie groß x ist? Ab welchem Wert von x ist sie nicht mehr optimal? Begründe.

Deine Rechnungen und deine Begründung:

Gymnasium Ottweiler

3.3.9 Das Wichtigste in Kürze

- Eine **Heuristik** liefert schnell eine zulässige, meist gute Lösung, ohne alle Möglichkeiten zu prüfen. Eine optimale Lösung ist nicht garantiert.
- Eine **Greedy-Heuristik** trifft in jedem Schritt die Entscheidung, die im Moment am besten aussieht, und ändert sie nie wieder.
- Die **Nearest-Neighbor-Heuristik** fährt immer zum nächstgelegenen, noch nicht besuchten Ort und am Ende zurück zum Start.
- Das Ergebnis kann vom **Startort** abhängen: Bei den vier Orten liefert der Start in D 95 km statt der optimalen 80 km.
- Die **Abweichung** berechnet man als (Länge der Heuristik − optimale Länge) : optimale Länge · 100 %.
- Heuristiken machen viel weniger Arbeit als Brute-Force. Deshalb verwendet man sie bei großen Probleminstanzen.

Selbstcheck

- ☐ Ich kann den Begriff Heuristik definieren und ein Beispiel aus dem Alltag nennen.
- ☐ Ich kann erklären, was „greedy“ bei einer Heuristik bedeutet.
- ☐ Ich kann die Nearest-Neighbor-Heuristik von Hand mit einer Tabelle durchführen.
- ☐ Ich kann die Abweichung einer Heuristik vom Optimum in Prozent berechnen.
- ☐ Ich kann an einem Beispiel erklären, warum eine Greedy-Entscheidung in eine Sackgasse führen kann.
- ☐ Ich kann begründen, wann man eine Heuristik statt Brute-Force einsetzt.

Gymnasium Ottweiler