DS 25 – Heuristiken: Der Greedy-Ansatz

Informatik 10 – MINT-Zweig | Informatisches Problemlösen

Lernziel

Ich definiere den Begriff der Heuristik, beschreibe eine Heuristik zur Problemlösung und wende sie auf Probleminstanzen an.

🕘 Stundenverlaufsplan (Doppelstunde, 90 Minuten)

Orientierungsrahmen für 11 SuS: Die ersten 80 Minuten bilden den Kern. Die letzten 10 Minuten sind bewusst flexibel und werden nicht mit neuem Pflichtstoff belegt.

Phase	Zeit	Inhalt/Aktion	Sozialform
Einstieg und Aktivierung	10'	Das Problem des schnell wachsenden Brute-Force-Aufwands erneut aufgreifen.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Erarbeitung	25'	Nearest Neighbor als schrittweise lokale Entscheidung aus Schülerideen entwickeln.	Unterrichtsgespräch; Beiträge der SuS werden sichtbar festgehalten
Gemeinsame Umsetzung und Sicherung	20'	Die Route ab A gemeinsam bestimmen und mit dem Optimum vergleichen.	Plenum; Live-Coding, Tafelbild oder gemeinsame Simulation
Kurzer Transfer	15'	Die Heuristik von einem anderen Startpunkt aus von Hand anwenden.	Einzelarbeit mit anschließendem Tandemabgleich (4 Tandems + 1 Trio)
Auswertung	10'	Vorteil, fehlende Optimalitätsgarantie und Einfluss der Instanz beurteilen lassen.	Plenum; Lösung erklären und Fachsprache sichern
Reserve / Vertiefung	10'	Gegenbeispiel suchen; Python nur als optionale Kontrolle.	Flexibel; bei Bedarf für Fragen, Fehlersuche oder Abschluss nutzen

🗣️ Sprachhilfe

- „Es handelt sich um ein Problem/eine Probleminstanz, weil …“
- „Der Ansatz prüft/wählt … und benötigt dabei …“
- „Die Heuristik ist hier geeignet/nicht geeignet, weil …“

📌 Merksätze

- Eine **Heuristik** ist eine Lösungsstrategie, die **schnell** eine (meist gute, aber nicht garantiert optimale) Lösung liefert, ohne alle Möglichkeiten zu prüfen.
- Eine **Greedy-Heuristik** ("gierig") trifft in jedem Schritt die Entscheidung, die im **Moment** am besten erscheint, ohne spätere Auswirkungen zu berücksichtigen.
- Beim TSP: Die Greedy-Heuristik **Nearest Neighbor** wählt von der aktuellen Stadt aus immer die **nächstgelegene, noch nicht besuchte** Stadt.
- Heuristiken sind **schneller** als Brute-Force, liefern aber **nicht immer** die optimale Lösung – manchmal führt eine "im Moment beste" Entscheidung in eine Sackgasse.

💡 Beispiel: Nearest Neighbor von Hand

Mit der Entfernungstabelle aus DS 23/24, Start in `A`:

1. Von `A` aus: nächstgelegene unbesuchte Stadt ist `B` (Entfernung 10).
2. Von `B` aus: nächstgelegene unbesuchte Stadt ist `D` (Entfernung 25, da `C` mit 35 weiter ist).
3. Von `D` aus: einzige verbleibende Stadt ist `C` (Entfernung 30).
4. Zurück nach `A` (Entfernung 15).

Route: `A → B → D → C → A`, Länge: `10 + 25 + 30 + 15 = 80`

Die Route `A → B → D → C → A` hat die Länge **80** und ist für diese Instanz zugleich optimal. Das Beispiel zeigt also: Eine Heuristik kann das Optimum treffen, garantiert dies aber nicht.

🐍 Kontrolle: Nearest-Neighbor-Heuristik in Python

```python
entfernungen = {
    ("A", "B"): 10, ("A", "C"): 15, ("A", "D"): 20,
    ("B", "C"): 35, ("B", "D"): 25, ("C", "D"): 30,
}

def entfernung(ort1, ort2):
    if (ort1, ort2) in entfernungen:
        return entfernungen[(ort1, ort2)]
    return entfernungen[(ort2, ort1)]

def nearest_neighbor(staedte, start):
    unbesucht = set(staedte)
    unbesucht.remove(start)
    route = [start]
    aktuell = start

    while unbesucht:
        naechste = min(unbesucht, key=lambda stadt: entfernung(aktuell, stadt))
        route.append(naechste)
        unbesucht.remove(naechste)
        aktuell = naechste

    return route

def rundreise_laenge(reise):
    laenge = 0
    for i in range(len(reise) - 1):
        laenge += entfernung(reise[i], reise[i + 1])
    laenge += entfernung(reise[-1], reise[0])
    return laenge

staedte = ["A", "B", "C", "D"]
route = nearest_neighbor(staedte, "A")
print("Nearest-Neighbor-Route:", route)
print("Länge:", rundreise_laenge(route))
```

```
Nearest-Neighbor-Route: ['A', 'B', 'D', 'C']
Länge: 80
```

📝 Übung 1 – Anderer Startpunkt

Wendet die Nearest-Neighbor-Heuristik **von Hand** an, aber startet diesmal in Stadt `C`. Vergleicht die Route und Länge mit dem Ergebnis von Start `A`. Was fällt euch auf?

🚀 Optionale Vertiefung – Heuristik vs. Optimum

Vergleicht für eure **eigene 5-Städte-Instanz aus DS 24 Übung 1**: Wie lang ist die Route der Nearest-Neighbor-Heuristik im Vergleich zur optimalen (Brute-Force-)Route? Berechnet die **prozentuale Abweichung**.

Lösung

✅ Lösung Übung 1

Lösung

```
route_c = nearest_neighbor(staedte, "C")
print("Route ab C:", route_c)
print("Länge:", rundreise_laenge(route_c))
```

```
Route ab C: ['C', 'A', 'B', 'D']
Länge: 80
```