DS 29 – Bewertung von Heuristiken & Abschluss

Informatik 10 – MINT-Zweig | Informatisches Problemlösen

Lernziel

Ich beurteile, wie gut eine Heuristik bei verschiedenen Probleminstanzen funktioniert, und festige die Inhalte des Themenfelds.

🕘 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'	Kriterien sammeln, nach denen eine schnelle, nicht garantiert optimale Lösung beurteilt werden kann.	Plenum; kurze Einzeldenkzeit
Fragend-entwickelnde Erarbeitung	25'	Prozentuale Abweichung und die Notwendigkeit mehrerer Instanzen aus Beispielen entwickeln.	Unterrichtsgespräch; Beiträge der SuS werden sichtbar festgehalten
Gemeinsame Umsetzung und Sicherung	20'	Das vorbereitete Experiment gemeinsam auswerten; Python bleibt Demonstrationswerkzeug.	Plenum; Live-Coding, Tafelbild oder gemeinsame Simulation
Kurzer Transfer	15'	Eine begründete Bewertung der Nearest-Neighbor-Heuristik formulieren.	Einzelarbeit mit anschließendem Tandemabgleich (4 Tandems + 1 Trio)
Auswertung	10'	Urteile anhand von Optimum, Abweichung, Aufwand und Instanzvielfalt vergleichen.	Plenum; Lösung erklären und Fachsprache sichern
Reserve / Vertiefung	10'	Abschlussfragen als Lernstandscheck oder Vorbereitung auf einen Kurztest.	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

- Um eine Heuristik zu **bewerten**, vergleicht man ihre Lösung mit der **optimalen** Lösung (sofern diese bekannt bzw. berechenbar ist) – üblich ist die **prozentuale Abweichung**.
- Eine Heuristik kann bei **manchen** Probleminstanzen die optimale Lösung treffen und bei **anderen** deutlich schlechter abschneiden – deshalb reicht ein einzelner Test nicht aus.
- Zur verlässlichen Bewertung testet man eine Heuristik über **mehrere, möglichst unterschiedliche** Probleminstanzen und betrachtet z. B. die **durchschnittliche** Abweichung.

🐍 Experiment: Nearest-Neighbor über mehrere zufällige Instanzen

Die folgende Zelle erzeugt mehrere zufällige TSP-Instanzen, löst sie sowohl mit Brute-Force (optimal) als auch mit Nearest-Neighbor (Heuristik) und berechnet die durchschnittliche Abweichung.

```python
import random
from itertools import permutations

def zufaellige_instanz(anzahl_staedte, seed):
    random.seed(seed)
    staedte = list(range(anzahl_staedte))
    entfernungen = {}
    for i in range(anzahl_staedte):
        for j in range(i + 1, anzahl_staedte):
            entfernungen[(i, j)] = random.randint(5, 50)
    return staedte, entfernungen

def entfernung(entfernungen, a, b):
    return entfernungen[(a, b)] if (a, b) in entfernungen else entfernungen[(b, a)]

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

def optimale_route(staedte, entfernungen):
    beste, beste_laenge = None, None
    for perm in permutations(staedte[1:]):
        reise = [staedte[0]] + list(perm)
        laenge = rundreise_laenge(entfernungen, reise)
        if beste_laenge is None or laenge < beste_laenge:
            beste, beste_laenge = reise, laenge
    return beste, beste_laenge

def nearest_neighbor(staedte, entfernungen, start):
    unbesucht = set(staedte)
    unbesucht.remove(start)
    route = [start]
    aktuell = start
    while unbesucht:
        naechste = min(unbesucht, key=lambda s: entfernung(entfernungen, aktuell, s))
        route.append(naechste)
        unbesucht.remove(naechste)
        aktuell = naechste
    return route

abweichungen = []
for seed in range(10):
    staedte, entfernungen = zufaellige_instanz(6, seed)
    _, opt_laenge = optimale_route(staedte, entfernungen)
    nn_route = nearest_neighbor(staedte, entfernungen, staedte[0])
    nn_laenge = rundreise_laenge(entfernungen, nn_route)
    abweichung = (nn_laenge - opt_laenge) / opt_laenge * 100
    abweichungen.append(abweichung)
    print(f"Instanz {seed}: Optimal={opt_laenge}, Nearest-Neighbor={nn_laenge},
Abweichung={abweichung:.1f}%")

print(f"\nDurchschnittliche Abweichung: {sum(abweichungen)/len(abweichungen):.1f}%")
```

```
Instanz 0: Optimal=119, Nearest-Neighbor=119, Abweichung=0.0%
Instanz 1: Optimal=92, Nearest-Neighbor=92, Abweichung=0.0%
Instanz 2: Optimal=79, Nearest-Neighbor=79, Abweichung=0.0%
Instanz 3: Optimal=122, Nearest-Neighbor=124, Abweichung=1.6%
Instanz 4: Optimal=81, Nearest-Neighbor=81, Abweichung=0.0%
Instanz 5: Optimal=113, Nearest-Neighbor=113, Abweichung=0.0%
Instanz 6: Optimal=78, Nearest-Neighbor=109, Abweichung=39.7%
```

```
Instanz 7: Optimal=67, Nearest-Neighbor=75, Abweichung=11.9%
Instanz 8: Optimal=83, Nearest-Neighbor=87, Abweichung=4.8%
Instanz 9: Optimal=114, Nearest-Neighbor=135, Abweichung=18.4%

Durchschnittliche Abweichung: 7.7%
```

📝 Übung 1 – Auswertung

Beantwortet anhand der Ausgabe oben:

1. Gab es eine Instanz, bei der die Heuristik die optimale Lösung getroffen hat (Abweichung 0 %)?
2. Wie groß war die größte beobachtete Abweichung?
3. Ist die Nearest-Neighbor-Heuristik eurer Einschätzung nach für den praktischen Einsatz geeignet? Begründet.

(Eure Antworten hier)

📝 Übung 2 (Abschlussaufgabe) – Zusammenfassung

Beantwortet in eigenen Worten (als Wiederholung des gesamten Themenfelds):

1. Was ist der Unterschied zwischen einem Problem und einer Probleminstanz?
2. Wann liefert ein Brute-Force-Ansatz garantiert die optimale Lösung, und wo liegt sein Nachteil?
3. Was zeichnet eine Greedy-Heuristik aus, und welches Risiko birgt sie?
4. Nennt je ein Beispiel für ein Problem mit faktoriellem und eines mit exponentiellem Brute-Force-Aufwand.

(Eure Antworten hier – als Vorbereitung für einen möglichen Kurztest)