Grundlagen der Intelligenten Systeme (WS2024/25)

Programmierübung 1

Deadline für die Einreichung: 03.12.2024
Anzahl der Aufgaben: 1
Maximal erreichbare Punktezahl: 100 Punkte

Allgemeines

- Diese ist eine von zwei Programmierübungen in *Grundlagen der intelligenten Systeme*. Sie können bis zu **100 Punkte** auf diese Übung erhalten.
- Die Deadline für die Einreichung dieser Programmierübung ist der **03.12.2024** und ist nicht verlängerbar. Für verspätete Abgaben wird Ihre erreichte Punktezahl um 25 % pro angefangenen Tag reduziert. Konkret bedeutet dies Folgendes: Angenommen, Sie machen eine um t Tage verspätete Abgabe, die wenn Sie zeitgerecht erfolgt wäre, mit p Punkten beurteilt wäre. In diesem Fall würden Sie $\max\{0, p \cdot (1 - 0.25t)\}$ Punkte für Ihre Abgabe erhalten.
- Bei Fragen und Problemen kontaktieren Sie uns rechtzeitig—primär über Moodle [1].
- Lösen Sie die Programmierübungen in **Python 3**. Verwenden Sie **keine** Bibliotheken die nicht explizit erlaubt ist.
- Stellen Sie sicher, dass Ihr Code ausführbar ist und das geforderte Interface implementiert. Jupyter Notebooks werden nicht akzeptiert.
  ⚠ Wenn Ihr Code nicht ausführbar ist oder die angeführten Spezifikationen nicht einhält, erhalten Sie für Ihre Abgabe Null Punkte.
- Lösen Sie die Programmieraufgaben eigenständig. Wir verwenden Tools wie Turnitin zur Detektion von Plagiaten.
  ⚠ Wenn wir ein Plagiat entdecken, werden Sie in der Lehrveranstaltung mit einem "X" beurteilt.
- Bennen Sie Ihre ausführbaren Python-Skripte mit `aufgabeZ.py`, wobei `Z` die Nummer der gelösten Aufgabe ist. Zippen Sie alle Ihre Skripte und benennen Sie die Datei `matrikelnummer_nachname_vorname.zip`, wobei Sie `matrikelnummer` durch Ihre Matrikelnummer, `nachname` durch Ihren Nachnamen und `vorname` durch Ihren Vornamen ersetzen. **Halten Sie sich nicht an diese Benennungsregeln, ist ihr Code nicht ausführbar.**

1 Lernziele

Ziel dieser Programmieraufgabe ist es, ausgewählte, Suchalgorithmen in Python zu implementieren und für diese ein tieferes Verständnis zu entwickeln.

2 Aufgaben

Lösen Sie alle Aufgaben durch das Erstellen von Python-Skripten. Sie können folgende Bibliotheken verwernden: `numpy`, `os`, `pyyaml`, `argparse`.

Aufgabe 1: A* Search

(Maximal erreichbare Punktezahl: 100 Punkte)

Implementieren Sie den A* Suchalgorithmus in der Graph-search Variante. Entwickeln Sie ein Python-Skript welches als Kommandozeilenargument eine Dateinamen akzeptiert. Diese Datei enthält eine Beschreibung des mittels Suche zu lösende Problems sowie Informationen die Sie zur Entwicklung einer Heuristik, d.h. einer Schätzfunktion für die Kosten zum Zielzustand, nutzen können um die Suche effizienter zu gestalten (Details siehe unten). Ihre Implementierung soll das eingelesene Problem mittels Suche lösen und Informationen zu Ihrer gefundenden inklusive deren Kosten ausgeben (Details siehe unten). In einem zweiten Schritt soll die Suche durch Nutzung einer geeigneten Heuristik verbessert werden und wieder sollen entsprechende Informationen ausgegeben werden. Die Laufzeit Ihres Programms ist pro Aufgabe auf jeweils 1 Minute begrenzt und muss in unseren Tests Probleme mit bis zu 10.000 Orten lösen. Stellen Sie sicher, dass Ihr Programm ausreichend effizient ist um diese Beschränkung einzuhalten (orientieren Sie sich an der bereitgestellten Beispielaufgabe mit 1.000 bzw. 10.000 Orten). Ihr Skript soll mit einem `ValueError` beendet werden, wenn ein Problem eintritt (z.B., die Eingabedatei is nicht lesbar oder das Problem ist nicht lösbar).

Problembeschreibung. Bei den zu lösenden Problemen handelt es sich um Probleme der Routenfindung. Die Problembeschreibung erfolgt mittels eines yaml-Files, welche folgendes Format aufweist (die Namen der Städte, hier `0`, `1`, `2`, `3`, können beliebige alphanumerische Zeichenfolgen ohne Whitespaces sein):

```
problem:
  cities:
  - '0'
  - '1'
  - '2'
  - '3'
```

```
  city_0:
    connects_to:
      '2': 218.0
  city_1:
    connects_to:
      '2': 105.0
      '3': 91.0
  city_2:
    connects_to:
      '0': 219.0
      '1': 121.0
      '3': 85.0
  city_3:
    connects_to:
      '1': 88.0
      '2': 68.0
  city_end: '3'
  city_start: '0'
```

Der "Block" `problem` enthält die Problembeschreibung:

- `city_start` beschreibt den Startzustand (Ort in dem die zu findende Routen beginnt).
- `city_end` beschreibt den Zielzustand (den Ort, der erreicht werden soll).
- Für jede Stadt `X` gibt es eine Beschreibung im Eintrag `city_X`, der beschreibt, welche Städte von `X` aus erreicht werden können. Konkret besteht `connects_to` aus einer Liste von Einträgen, die die jeweilige erreichbare Stadt `Y` und die hierfür anfallenden Kosten `c` darstellen: `Y: c`.

Zusatzinformation für das Erstellen von Heuristiken. Darüber hinaus werden Informationen zur Erstellung von Heuristiken bereitgestellt (`additional_information`).

```
additional_information:
  city_0:
    altitude_difference: 161.18780820302896
    line_of_sight_distance: 78.23894417994319
  city_1:
    altitude_difference: 33.78587006556745
```

```
    line_of_sight_distance: 43.221505253674174
  city_2:
    altitude_difference: 11.116275824444926
    line_of_sight_distance: 54.962335331489754
  city_3:
    altitude_difference: 0.0
    line_of_sight_distance: 0.0
```

In den Einträgen `city_X` sind potentiell relevante Informationen bezüglich der Erreichung des Zielortes enthalten:

- Luftliniendistanz zum Zielort `line_of_sight_distance`.
- Absolute Höhendifferenz `altitude_difference`.

Ausgabe des Programms. Ihr Programm muss eine yaml-Datei mit folgendem Format erzeugen (beispielhaft dargestellt):

```
solution:
  cost: 210
  path:
  - 0
  - 2
  - 3
  expanded_nodes: 10

  heuristic:
    city_0: 10
    city_1: 3
    city_2: 5
    city_3: 0
```

Der Pfad `path` muss sowohl den Startknoten als auch den Endknoten enthalten. Die Gesamtkosten des Pfades sind in `cost` zu speichern. Der Eintrag `expanded_nodes` zählt die Gesamtzahl der expandierten Knoten (inklusive Startknoten, exklusive Endknoten). Die Heuristik, die für die Routenfindung genutzt wurde ist in `heuristic` zu speichern.

Teilaufgabe 1: Lösen des Suchproblems ohne Heuristik

[30 Punkte]. Verwenden Sie $h(n) = 0$ als Heuristik in ihrem A* Suchalgorithmus um das Suchproblem zu lösen. Ihr Algorithmus entspricht dann der UNIFORM-COST SEARCH. Speichern Sie die Ergebnisse in der Datei `aufgabe1-1.yaml` in dem oben beschriebenen Ausgabeformat.

- 10 Punkte: Formal korrekte, mit dem Problem konsistente, Ausgabe wird erzeugt.
- 10 Punkte: Ein Pfad vom Start bis zum Ziel inklusiver korrekter Kosten wird gefunden. Die Information zur Heuristik sind im Einklang mit der Aufgabe.
- 10 Punkte: Der Pfad vom Start zum Ziel ist optimal. Die Anzahl der Expansionen entspricht der, der Referenzimplementation gegen die wir Ihre Abgabe testen.

Teilaufgabe 2: Lösen des Suchproblems mit simpler Heuristik.

[30 Punkte]. Nutzen Sie die `line_of_sight_distance`-Information um die Performance der Suche zu verbessern (weniger Expansionen). Speichern Sie die Ergebnisse in der Datei `aufgabe1-2.yaml` in dem oben beschriebenen Ausgabeformat.

- 20 Punkte: Die Heuristik unterscheidet sich von der in der vorherigen Aufgabe. Das Ergebnis der Suche ist mit der verwendeten Heuristik konsistent. Die optimale Lösung wird gefunden.
- 10 Punkte: Die gewählte Heuristik reduziert auf den Testproblemen (`test?.yaml`) , die auf Moodle bereitgestellt sind die Anzahl der Expansionen. Die optimale Lösung wird gefunden.

Teilaufgabe 3: Verbessern der Heuristik und erneute Lösung des Suchproblems

[20 Punkte] Nutzen Sie weitere Informationen um die Performance Ihres Suchalgorithmus zu verbessern (weniger Expansionen). Identifizieren Sie eine Heuristik, die auf den zur Verfügung gestellten Problemen die Anzahl der Expansionen reduziert. Speichern Sie die Ergebnisse in der Datei `aufgabe1-3.yaml`.

- 20 Punkte: Für die zur Verfügung gestellten Testprobleme (`test?.yaml`) reduziert die Heuristik die Anzahl an Expansionen im Vergleich zur vorhergehenden Aufgabe. Die optimale Lösung wird gefunden.

Teilaufgabe 4: Generalisierung zu weiteren Suchproblemen.

[20 Punkte] Stellen Sie sicher, dass Ihr Suchalgorithmus auch auf anderen als der zur Verfügung gestellten Problembeschreibung funktioniert. Wir testen Ihre Einreichung auf weiteren Problemen (mehr oder weniger Städte, andere Kosten, usw.). Sie erhalten Punkte entsprechend dem Anteil der Probleme, die Ihre Einreichung korrekt löst. Die Probleme umfassen bis zu 10.000 Orte. Stellen Sie also eine ausreichend effiziente Implementierung sicher (Zeitlimit 1 Minute). Ein Überschreiten des Zeitlimits wird gewertet, wie wenn das gestellte Problem nicht gelöst werden konnte.

3 References

[1] *Grundlagen der intelligenten Systeme, Moodle Page.* URL: `https://moodle.univie.ac.at/course/view.php?id=437205`.