

Übungsblatt: Suchverfahren im Vergleich

1 Einführung

Auf diesem Übungsblatt beschäftigen wir uns mit Suchverfahren am Beispiel des 8-Puzzle Problems aus dem Skript. Wir werden gemeinsam die unterschiedlichen Suchverfahren (Tiefensuche, Breitensuche, iterative Tiefensuche, und die zwei A*-Heuristiken) am 8-Puzzle-Problem praktisch testen. Sie müssen dazu nicht programmieren, aber wir werden gemessene Programm-Laufzeiten (Code siehe tiles.py) diskutieren und die Auswirkungen der verschiedenen Parameter betrachten.

Machen Sie sich zunächst nochmals mit dem 8-Puzzle vertraut (siehe Skript und https://en.wikipedia.org/wiki/15_puzzle).

Das folgende Puzzle stellt den Zielzustand des Problems dar:

1	2	3
8		4
7	6	5

Beim Zustandsübergang überführen wir einen Zustand in einen benachbarten Zustand, das heißt wir verschieben ein Puzzleteil auf das freie Feld.

Um die Qualität unserer heuristischen Suchverfahren beurteilen zu können, benötigen wir einen Ausgangszustand, dessen Schwierigkeit wir beurteilen bzw. beeinflussen können. Es gibt dazu eine Funktion, die ein Puzzle mit einer definierten maximalen Schwierigkeit erzeugt. Hierzu nimmt sie den oben beschriebenen Zielzustand als Ausgangszustand, und führt dann eine bestimmte Anzahl n zufälliger Zustandsübergänge durch. Mit anderen Worten: Wir nehmen das fertige Puzzle, und verschieben n -mal nacheinander ein beliebiges Teil¹.

Wie in der Vorlesung gesehen, können wir die Zustände unseres Problems als Baum darstellen. Knoten stellen hierbei Zustände dar, Verbindungen zwischen Knoten sind Zustandsübergänge. Die Zustände j , die durch einen einzelnen Zustandsübergang von Zustand i erreicht werden können, sind im Baum dementsprechend die Kinder von i .

Alle Suchverfahren, die wir in der Vorlesung kennen gelernt haben, operieren auf diesem Baum. Der derzeitige Knoten/Zustand wird expandiert, d.h., seine Kindknoten werden erzeugt, und es wird überprüft ob eines der Kinder dem Zielzustand entspricht. Falls keines der Kinder dem Zielzustand entspricht,

¹Auch ein im-Kreis-schieben kann vorkommen, daher 'maximale' Schwierigkeit.

müssen die Kinder (in einem der nächsten Schritte) expandiert werden. Dazu werden sie zu einer Warteschlange hinzugefügt, die alle Knoten enthält, die noch expandiert werden müssen (siehe <https://de.wikipedia.org/wiki/Datenstruktur#Warteschlange>). Der erste Knoten der Warteschlange wird im nächsten Schritt expandiert. Die Suchverfahren unterscheiden sich nun dadurch, wie die Warteschlange aufgebaut bzw. sortiert wird. Beispielsweise können neue Knoten *am Ende* der Warteschlange eingefügt werden (first in, first out, FIFO) oder *am Anfang* (last in, first out, LIFO). Falls nötig, kann jeder Knoten zusätzliche Informationen speichern (z.B. Informationen über Kosten). Ein neuer Knoten kann dann so in die Warteschlange eingefügt werden, dass die Sortierung erhalten bleibt.

2 Datenstruktur Warteschlange

Beschreiben Sie, wie die folgenden Verfahren Knoten zur Warteschlange hinzufügen bzw. die Warteschlange sortieren. Ziel ist immer, dass der erste Knoten der Warteschlange derjenige ist, den die Suchstrategie als nächstes auswählt.

1. Tiefensuche
2. Breitensuche
3. Iterative Tiefensuche
4. A*-Suche

3 Zeit- und Platzkomplexität

Wir untersuchen nun empirisch, d.h. mit Hilfe eines Experiments (Code siehe `tiles.py`), die Zeit- und Platzkomplexität der Suchstrategien. Wir testen alle Suchverfahren, d.h. Tiefensuche, Breitensuche, iterative Tiefensuche, A*-Heuristiken (“Wrong Tiles” und “Manhattan Distance”). Die Tests wurden mit einer Schwierigkeit von 5, 10, bzw. 15 Mischvorgängen durchgeführt. Für repräsentative Ergebnisse führen wir jeweils 500 Versuche durch und berechnen das arithmetische Mittel der Laufzeiten über alle 500 Versuche.

Ergebnisse für AMD Ryzen 9 7950X (@4.50 GHz)

Tabelle 1: Average results for 5 shuffled moves over 500 runs.

Strategy	Iterations	Max Queue Size	Time (ms)
BFS	26.5	20.5	0.03
DFS (random walk)	160054.1	1.0	206.35
DFS (without cycles)	29013.7	14782.6	40.39
IDS	51.5	6.0	0.06
Heuristic (wrong tiles)	7.0	7.8	0.02
Heuristic (Manhattan)	5.8	6.5	0.03

Tabelle 2: Average results for 10 shuffled moves over 500 runs.

Strategy	Iterations	Max Queue Size	Time (ms)
BFS	400.1	302.2	0.39
DFS (random walk)	187908.2	1.0	227.65
DFS (without cycles)	36386.5	21639.9	51.18
IDS	891.8	9.0	0.91
Heuristic (wrong tiles)	41.6	37.9	0.13
Heuristic (Manhattan)	17.2	16.4	0.08

Tabelle 3: Average results for 15 shuffled moves over 500 runs.

Strategy	Iterations	Max Queue Size	Time (ms)
BFS	5063.0	3779.1	5.97
DFS (random walk)	193422.1	1.0	236.41
DFS (without cycles)	39331.9	23361.3	55.21
IDS	12401.5	13.4	12.34
Heuristic (wrong tiles)	376.6	318.4	3.12
Heuristic (Manhattan)	77.2	67.9	0.45

- **Wie würden Sie die oben genannten Ergebnisse bewerten?** Welches der zwei *uninformierten* Suchverfahren würden Sie empfehlen, wenn keine Heuristiken zur Verfügung stehen würden?
- **Schätzen Sie was passiert,** wenn wir die maximale Schwierigkeit des Puzzles noch höher setzen, z.B. auf 25 oder 30?