Praktikum 01 – Grundlagen der KI

Name: Umar Farooq

Datum: 17.10.2025

Search 01- Problemformalisierung und Zustandsform

1- Problemformalisierung (Zustände, Aktionen, Start und Endzustand)

Zustände: (E,O,P)

E = Anzahl der Elben am linken Ufer

O = Anzahl der Orks am linken Ufer

P = {L, R} = Position des Pferdes (L = links, R = rechts)

Es gibt insgesamt 3x Elben, 3x Orks und 1x Pferd

Starzustand: (3, 3, L)

Alle Elben und Orks befinden sich am linken Ufer, das Pferd ebenfalls.

Zielzustand: (0, 0, R)

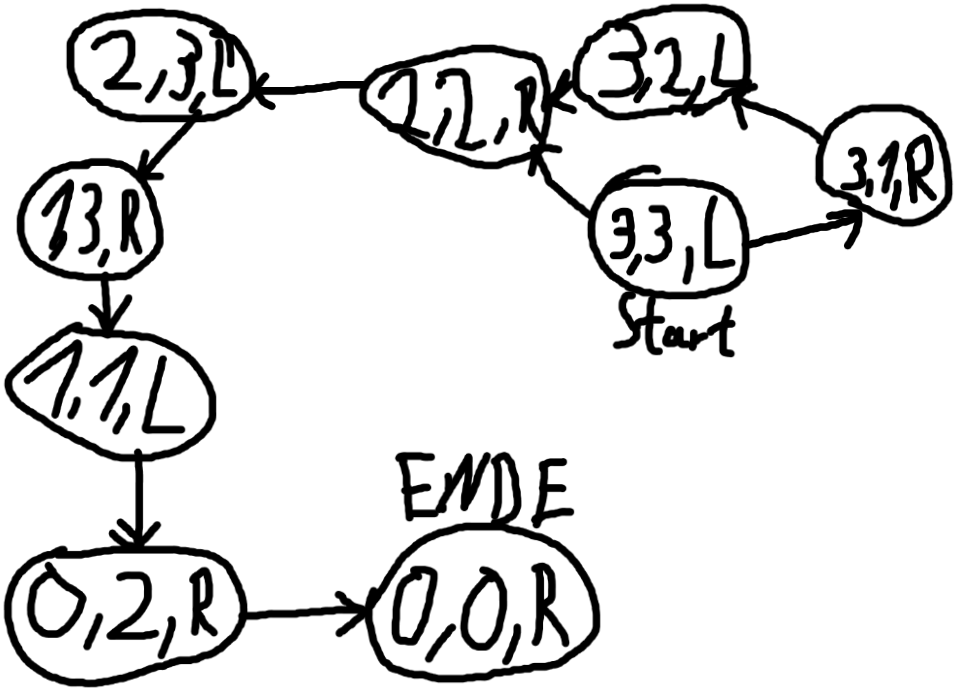
Alle Elben und Orks sind am rechten Ufer, das Pferd ebenfalls.

Aktionen: max 2 Wesen für Transport

Wenn P = L, Bewegung von links nach rechts

Wenn P = R, Bewegung rechts nach limks

2- Problemgraph- Skizze



Search 02- Suchverfahren

Tiefensuche

Pfad: Würzburg -> Frankfurt -> Mannheim -> Karlsruhe -> Augsburg -> München

217 + 80 + 250 + 84 = **631 km**

Breitensuche

Pfad: Würzburg -> Nürnberg -> München

103 + 167 = 270 km

A*

Pfad: Würzburg -> Nürnberg -> München

f(n) minimal = 270

Vergleich der drei Algorithmen durch Handsimulation

Die **Tiefensuche** durchsucht den Graphen immer so weit wie möglich in die Tiefe, bevor sie zurückspringt. Dadurch ist die maximale Anzahl an Einträgen in der Datenstruktur gering, weil nur der aktuelle Pfad gespeichert wird. Die Hauptschleife wird für jeden besuchten Knoten einmal durchlaufen. Das Verfahren ist speichereffizient, findet aber nicht unbedingt den kürzesten Weg.

Die **Breitensuche** durchsucht den Graphen Ebene für Ebene und hält daher in der Datenstruktur alle Knoten einer Ebene gleichzeitig. Dadurch wird die Warteschlange deutlich größer als bei der Tiefensuche. Sie benötigt mehr Schleifendurchläufe, findet aber immer den kürzesten Weg (in diesem Fall von Würzburg über Nürnberg nach München).

Der **A*-Algorithmus** kombiniert die Vorteile beider Verfahren, indem er nur Knoten mit den kleinsten geschätzten Gesamtkosten $f(n)=g(n)+h(n)$f(n) = g(n) + h(n)f(n)=g(n)+h(n) erweitert. Die Datenstruktur ist größer als bei der Tiefensuche, aber kleiner als bei der Breitensuche. A* benötigt weniger Durchläufe als die Breitensuche und liefert dennoch die optimale Lösung, sofern die Heuristik zulässig ist.

Restkostenabschätzung in A*

Nein weil, sie **nicht alle zulässig** sind. Um die Heuristik zu korrigieren, müssen diese Werte so angepasst werden, dass sie kleiner oder gleich der tatsächlichen Minimaldistanz sind, zum Beispiel: h(Nürnberg) = 160 km und h(Stuttgart) = 250 km

Search 03- Dominanz

Bedeutung

Eine Heuristik h1 (n) **dominiert** eine andere Heuristik h2 (n), wenn für alle Knoten n gilt: h1 (n) >= h2 (n)

und beide Heuristiken **zulässig** sind, sie überschätzen die tatsächlichen Kosten nie.

Auswirkung in A*

Wenn A* eine dominierende Heuristik h1 verwendet, werden **weniger Zustände expandiert**, weil h1 näher an den tatsächlichen Restkosten liegt und die Suche stärker in Richtung Ziel lenkt.
A* bleibt dabei **optimal**, weil die Zulässigkeit weiterhin gilt, ist aber **effizienter** als mit h2 .

Selbstgewähltes Beispiel

Ein **Pikachu** möchte das **Poké-Center** in einer Stadt erreichen.
Die Karte besteht aus Straßen (man kann nur in vier Richtungen laufen).

Wir betrachten zwei mögliche Heuristiken:

- h1 (n): **Luftlinie** zwischen Pikachu und dem Poké-Center (direkter Abstand in Metern)
- h2 (n): **Manhattan-Distanz**, also nur horizontale und vertikale Schritte

Da die Luftlinie nie kleiner ist als die Manhattan-Distanz, gilt: h1 (n) >= h2 (n)

Wendet man A* mit h1 an, sucht Pikachu gezielter in Richtung Poké-Center und prüft weniger Zwischenwege – die Suche verläuft schneller, aber das Ergebnis bleibt optimal.

Search 04- Beweis der Optimalität von A*

Der A*-Algorithmus ist in der Tree-Search-Variante optimal, wenn die verwendete Heuristik zulässig ist. Eine Heuristik gilt als zulässig, wenn sie die tatsächlichen minimalen Kosten zum Ziel nie überschätzt.

Angenommen, der Algorithmus findet eine Lösung mit den Kosten $g(n*)$, es gäbe aber eine bessere Lösung mit geringeren Kosten $g(n')<g(n*)$. In diesem Fall müsste auf dem Weg zu dieser besseren Lösung ein Knoten existieren, dessen f-Wert kleiner oder gleich $g(n*)$ ist. Da A* alle Knoten mit kleinerem f(n) zuerst expandiert, wäre dieser Knoten bereits vor der gefundenen Lösung untersucht worden. Damit kann eine suboptimale Lösung nicht zuerst gefunden werden.

Folglich liefert A* mit einer zulässigen Heuristik immer eine Lösung mit minimalen Gesamtkosten und ist somit **optimal**.