Käfer im Labyrinth

Nils Wiemer

817971
wiemer@uni-potsdam.de
2.Semester
2.Fachsemester
Informatik/Computational Sience

beaufsichtigt von
Dr. Andrey Cherstvy
[PHY_131d] Simulation und Modellierung

30. September 2023

Inhaltsverzeichnis

1 Einführung

Das Lösen eines Labyrinths mit Hilfe komplexer Algorithmen, welche von höheren Lebensformen angewandt werden können, ist eine Aufgabe in der es inzwischen darum geht Algorithmen zu finden, welche immer schneller und weniger speicheraufwendig sind. Um es zu ermöglichen alltägliche Aufgaben wie die Suche nach der besten Route in den Urlaub oder die effizienteste Route für den Postboten schneller und schneller zu ermöglichen oder Straßennetze effektiver zu planen und bauen und Häuser mit Strom, Wasser und Internet zu versorgen und dabei die Materialien so effizient wie möglich einzusetzten.

Mit Blick auf niedrigere Lebewesen wie Käfer, Milben und ähnliches kommt die Frage auf ob es möglich ist ein Labyrinth auch mit simplen Algorithmen oder ganz ohne Algorithmen zu lösen. In dem wissenschaftlichen Artikel "Development of an automatic truntable-type multiple T-maze device and observation of pill bug behavior"[2] wird das Bewegungsverhalten von Rollasseln (Armadillidiidae) in einem endlichen Labyrinth untersucht, um festzustellen, ob einer Wiederholung von Richtungswechseln in die gleiche Richtung eine Fehlfunktion ist. Es wird von einer Fehlfunktion gesprochen, da bei Organismen mit simplen Nervensystemen davon ausgegangen wird, dass diese Entscheidungen mechanisch/instinktiv getroffen werden. Die Studie zeigt aber, dass eine wiederholte Drehung in eine Richtung in Maßen angewandt wird um einen Bewegungsablauf aus zu führen.

Um Labyrinthe zu lösen werden Algorithmen bewusst oder unbewusst angewandt. So werden für Computeralgorithmen Labyrinthe in Graphen umgewandelt, wobei jede Kreuzung im Labyrinth zu einem Knoten im Graphen wird und die Strecke zwischen diesen Kreuzungen zu Kanten im Labyrinth. Um nun den das Labyrinth zu lösen muss lediglich die kürzeste Kombination von Knoten und Kanten gefunden werden welche zwischen dem Start und dem Ziel des Labyrinths liegen. Auf dieser Grundlage kann man die verschiedensten Suchalgorithmen aus der Graphentheorie anwenden um die Verbindung zwischen den zwei Knoten zu finden.[5] Um in einem Labyrinth mit mehreren Lösungswegen den besseren zu finden, ist es möglich die Kanten zu gewichten, ihnen Werte basierend auf bestimmten Eigenschaften zuzuweisen, wie zum Beispiel die Länge des Weges oder die Schwierigkeit diesen zu passieren.

Ergebnisse dieses Experiments können genutzt werden um Vergleiche für das Denkverhalten von Lebewesen zu erhalten. So kann man feststellen ob simplere Lebewesen sich nach einem Muster bewegen oder zufällig ein Labyrinth lösen, indem man den Durchschnitt der benötigten Zeit von mehren Durchläufen mit den Daten dieses Experiments vergleicht, was helfen kann Entscheidungen dieser Lebewesen in der Umwelt nachzuvollziehen.

2 Problem

Die folgenden Seiten befassen sich mit dem Problem ein Labyrinth zu lösen. Dabei soll ein Käfer dieses Labyrinth durchlaufen, während dieses Durchlaufs werden die Anzahl der Schritte sowie die Zeit die der Käfer benötigt gemessen. Ein Schritt findet statt, wenn der Käfer eine Position im Labyrinth weitergeht. Eine Zeiteinheit verstreicht, wenn der Käfer eine Aktion durchführt, wobei eine Aktion ein Schritt sein kann oder das Prüfen ob ein Schritt möglich ist. Dieser Versuch wird in zwei Variationen durchgeführt. In der ersten Variation ist es dem Käfer möglich in alle vier Himmelsrichtung zu laufen. In der zweiten Variation kann der Käfer aber nicht rückwärts gehen und muss sich somit drehen, dies erhöht die Zeit ebenfalls. Diese Variationen werden im weiteren Verlauf als erster und zweiter Versuch bezeichnet.

In diesem Experiment gilt es, herauszufinden wie sich die Größe des Labyrinths auf die Zeit und die Schritte, welche der Käfer benötigt auswirken. Genauer wird geguckt in welchem Verhältnis Schritte und Zeit in einem einzelnen Durchlauf stehen und in welchem Verhältnis sie in Durchläufen mit unterschiedlich großen Labyrinthen stehen. Darauf hin gilt es die Ergebnisse der ersten Variante mit denen der zweiten Variante zu vergleichen. Um das Experiment erfolgreich durchführen zu können, ist vorauszusetzen, dass das Labyrinth auch lösbar ist, also ein Weg zwischen Start und Ziel immer besteht. Aber nicht nur das Labyrinth muss an sich lösbar sein, auch der Käfer muss mit einer simplen Bewegungsabfolge durch das Labyrinth kommen ohne sich an einer Stelle im Kreis zu drehen oder sich gar nicht mehr zu bewegen. Dem ist hinzuzufügen, dass im nicht Kreisdrehen heißt, dass er irgendwann aus diesem Kreis wieder heraus kommt und nicht, dass er nicht öfters die gleiche Strecke ablaufen darf.

Der Versuch ein Labyrinth zu lösen, sei dies nun die Vernetzung im Internet, die Fahrt durch die Stadt oder das Labyrinth auf dem Jahrmarkt, ist eine Aufgabe die der Mensch oder ein anderes Lebewesen nicht so effizient wie der Computer bewäligen kann. Somit ist es von Vorteil ein Algorithmus zu finden der angewandt werden kann um ein Labyrinth jeglicher Form auf einfachste Weise zu lösen. Der einfachste Weg ist es einfach solang durch die Gegend zu laufen bis man an seinem Ziel angekommen ist. Das Problem an dieser Sache ist, dass es bei großen Labyrinthen lange dauern kann und um herauszufinden wie lange es dauern kann, gilt es zu finden wie sich die Zeit zum lösen eines Labyrinths zu dessen Größe verhält. Betrachtet man nun das Problem der Labyrinthlösung im Tierreich stösst man auf Limitationen wie die Unfähigkeit einiger Tiere sich rückwärts zu bewegen. Sollte ein Lebewesen mit dieser Einschränkung ein Labyrinth lösen müssen, muss es sich erst drehen, bevor es rückwärts gehen kann. Was bedeutet, dass es wahrscheinlicher nur vorwärts geht. Zu wissen welche Weise effektiver ist und wo die Vor- und Nachteile liegen kann nicht nur in der Entwicklung von Algorithmen zum Lösen von Labyrinthen helfen sondern auch dabei Verhaltensweisen von Tieren zu erklären.

3 Aufbau

3.1 Erzeugen des Labyrinths

Um dem Käfer die Möglichkeit zu geben das Labyrinth in jedem Fall lösen zu können, muss dieses ein "perfektes"Labyrinth sein. Das heißt, dass es möglich ist von jedem Punkt im Labyrinth zu jedem anderen zu kommen. Zudem hat dieses Labyrinth weder Schleifen noch Pfade die sich kreuzen.[4] Weiterhin sollen die Labyrinthe zufällig erzeugt werden, um den Ablauf des Programms zu vereinfachen und so wenig wie mögliche Durchläufe auf dem gleichen Labyrinth zu haben.

Ein Labyrinth ist wie in Kapitel 2 erklärt zu verstehen. Um ein perfektes Labyrinth aus einem Graphen zu erzeugen, muss man einen Spannbaum aus diesem Graphen finden. Um dies zu tun gibt es verschiedene Algorithmen. Desweiteren wird sich mit dem Kruskal [1] Algorithmus befasst.

Der Kurskal Algorithmus findet in einem gewichteten Graphen den minimalen Spannbaum, in dem er die minimale Summe aller Gewichte findet und die Knoten so verbindet. Dabei fängt er mit dem niedrigsten Gewicht an und verbindet die zwei Knoten zu einem Baum. Dieser Schritt wird nun wiederholt, bis alle Knoten miteinander verbunden sind. Sollte währenddessen ein Knoten mit einem Baum oder ein Baum mit einem Baum verbunden werden, so werden diese zu einem einzigen Baum zusammen gefasst. Dabei ist aber zu beachten, dass nur Knoten mit einander verbunden werden, wenn diese nicht im gleichen Baum sind, da sonst eine Schleife entstehen würde. Dadurch entsteht am Ende ein minimaler Spannbaum auf dem Graphen.

Um ein Labyrinth zu erstellen sind die Gewichte der Kanten nicht weiter relevant, wodurch man sie weg lassen kann. Das führt dazu, dass die Knoten nicht anhand der Gewichte verbunden werden, sonder zufällig zwei Knotenpaare ausgewählt werden und anschließend verbunden. Es entsteht ein zufälliger Kruskal Algorithmus. Die Umsetzung erfolgt wie folgt und beruht auf der Pythonbibliothek Mazelib [3]

Es wird die Höhe und Breite des Labyrinths verdoppelt um die Wände einzufügen und eins addiert um das Labyrinth auf allen Seiten mit Wänden zu umfassen. Folgend werden nochmal zwei addiert um die Außenwände „zwei dick“ zu machen. Dies ist nötig, da beim Lösen des Labyrinths Schritte über zwei Felder getan werden, um die Wände des Labyrinths aus zu schließen, da Wände die Dicke eines Feldes haben und diese nicht mit gezählt werden sollen und an den Ränder sonnst das Labyrinth verlassen wird. Ein Beispiel dafür ist in Kapitel 3.2 zu finden.

```
1 # Bug.py
2 def __init__(self, h, w):
3         self.H = 2*h+3
4         self.W = 2*w+3
5         self.grid = None
```

Anschließend wird ein Gitter aus Einsen erstellt und in der dritten Zeile und Spalte die Eins auf Null gesetzt um einen Knoten zu symbolisieren.

Dies wird mit einer Lücke von Eins nach rechts und unten fortgesetzt und einer Liste an Knoten hinzugefügt. Anschließend werden alle verbleibenden Einsen,

```
[1 1 1 1 1 1 1 1 1]
[1 1 1 1 1 1 1 1 1]
[1 1 0 1 0 1 0 1 1]
[1 1 1 1 1 1 1 1 1]
[1 1 0 1 0 1 0 1 1]
[1 1 1 1 1 1 1 1 1]
[1 1 0 1 0 1 0 1 1]
[1 1 1 1 1 1 1 1 1]
[1 1 1 1 1 1 1 1 1]
```

Abbildung 1: Knoten des Labyrinths

welche horizontal nicht durch nullen getrennt werden und alle welche vertical nicht vertical getrennt werden einer Liste als Kanten einer Liste hinzugefügt. Zuletzt werden die Kanten gemischt.

```
# Bug.py
def generate(self):
        self.grid = np.ones((self.H,self.W), dtype=np.int8)

    wald = []
    Kanten_h=[]
    Kanten_v=[]
    for row in range(2, H - 2, 2):
        for col in range(2, W - 2, 2):
            wald.append([(row,col)])
            grid[row][col] = 0

    # Hinzufuegen aller verticaler Knoten.
    for row in range(3, H - 2, 2):
        for col in range(2, W - 2, 2):
            Kanten_v.append((row,col))
    # Hinzufuegen aller horizontaler Knoten.
    for row in range(2, H - 2, 2):
        for col in range(3, W - 2, 2):
            Kanten_h.append((row,col))

    shuffle(Kanten_h)
    shuffle(Kanten_v)
```

Aus den Listen an Kanten wird solange das erste Element heraus genommen, bis die Länge der Knoten Eins beträgt. Für Kanten, welche vertical zwischen zwei Knoten liegen wird die Position der Knoten in der Liste der Knoten gesucht und gespeichert. Wenn diese Positionen nicht gleich sind, heißt wenn sie sich nicht im gleichen Baum befinden, werden sie verbunden und zu einem Baum zusammen gefügt. Das gleiche Verfahren wird für die Liste von horizontalen Knoten verwendet.

```
# Bug.py
while len(wald) > 1:
  row_v, col_v = Kanten_v[0]
    Kanten_v = Kanten_v[1:]
    row_h, col_h = Kanten_h[0]
```

```
    Kanten_h = Kanten_h[1:]

    top_v = 0
    bottom_v = 0
    left_h = 0
    right_h = 0

    for i,edge in enumerate(wald):
        if(row_v-1, col_v) in edge:
            top_v = i

        if(row_v+1, col_v) in edge:
            bottom_v = i


    if top_v != bottom_v:
        Baum = wald[top_v] + wald[bottom_v]
        sec = wald[bottom_v]
        wald.pop(top_v)
        if top_v < bottom_v:
            wald.pop(bottom_v-1)
        else:
            wald.pop(bottom_v)
        wald.append(Baum)
        grid[row_v, col_v]=0

    for i,edge in enumerate(wald):
        if(row_h, col_h-1) in edge:
            left_h = i
        if(row_h, col_h+1) in edge:
            right_h = i


    if left_h != right_h:
        Baum = wald[left_h] + wald[right_h]
        sec = wald[right_h]
        wald.pop(left_h)
        if left_h < right_h:
            wald.pop(right_h-1)
        else:
            wald.pop(right_h)
        wald.append(Baum)
        grid[row_h, col_h] = 0
```

Dieser Vorgang kann wie folgt dargestellt werden. Die rot markierte Null stellt jeweils die gerade ausgewählte Kante da. Die blau markierte Null die schon früher ausgewählten Kanten. Sollten die beiden Nullen daneben oder da drüber und da drunter nicht im gleichen Baum liegen so werden diese Verbunden.

Wenn nur noch ein Baum in der Liste ist bricht die While-Scheife ab und der minimale Spannbaum wurde gefunden.

3.2 Durchlaufen des Labyrinths

Nach dem das Labyrinth erzeugt wurde liegt eine Liste von Listen mit Nullen und Einsen gefüllt vor, wobei eine Eins eine Wand symbolisiert und eine Null einen Weg.

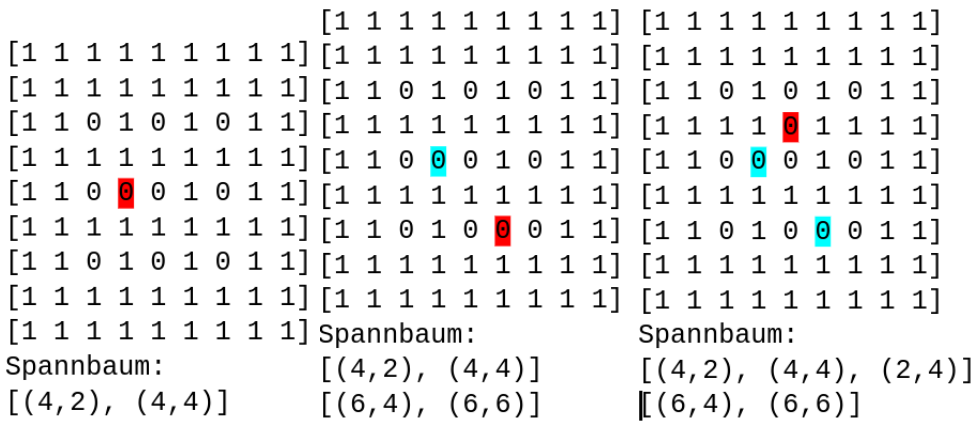

Abbildung 2: Knoten des Labyrinths

Um nun das Labyrinth durchlaufen zu können, wird als Startpunkt die obere linke Ecke gewählt und als Ziel die untere rechte Ecke. Desweiteren gilt es zu beachten, dass eine Bewegung immer um zwei Felder passiert, da die Wände ebenfalls als Feld gewertet werden. Wir also bei einem 3x3 Feld bis zu fünf Schritten in eine Richtung laufen könnten, statt 3. Nehmen wir als Beispiel Abbildung 3, in der ein 3x3 Labyrinth abgebildet ist. Wird der Käfer (pink) einen Schritt machen landet dieser auf dem grünen Feld oder in der unteren linken Ecke des Labyrinths, da Wände nicht als Schritt gewertet werden sollen. Das soll heißen, dass Wände eben nicht so dick wie dargestellt sein müssen, sondern auch ein simpler strich sein können.

Abbildung 3: Schritte im 3x3 Labyrinth

Im ersten Versuch einen simplen Bewegungsablauf zu finden bestand der Versuch daraus, im Uhrzeigersinn von oben beginnend zu prüfen ob sich in die Richtung eine Wand befindet. Ist dies der Fall wird die nächste Richtung geprüft, ansonsten bewegt sich der Käfer ein Feld in diese Richtung.

Bei diesem Versuch trat jedoch das Problem auf, dass der Käfer das Labyrinth nicht lösen kann, da er sobald er nach unten geht im nächsten schritt zwingend wieder nach oben geht. Somit bewegt sich der Käfer effektiv auf zwei Feldern. Und konnte mehrere Felder nur nach oben oder links laufen. Aufgrund seiner Startposition war die Bewegung nach oben jedoch nicht möglich.

Im zweiten Versuch wurde nur die Reihenfolge der Richtungen in welcher geprüft wird geändert. Es ging immer noch im Uhrzeigersinn, jedoch wurde nun zuerst die rechte Seite geprüft.

Das Problem, im zweiten Versuch bewegte sich der Käfer nun nach links und rechts zwischen den selben zwei Feldern.

Diese beiden Versuche ergaben das egal von welcher Richtung im Uhrzeigersinn oder dagegen gegangen würde, nach wenigen Schritten würde der Käfer also nur noch zwischen zwei Punkten springen. Somit konnten zwar mit Glück kleinere Labyrinthe gelöst werden aber dies geschah nicht zuverlässig und war somit nicht nutzbar oder auf größere Labyrinthe anwendbar.

Der dritte Versuch basiert auch auf dem Prüfen der Richtung im Uhrzeigersinn, nur dass in diesem die Ausgangsrichtung zufällig gewählt wird. Dazu wird eine Zahl zufällig mit der in Python verbauten random Bibliothek erzeugt. Dies zufällige Zahl zwischen 0 und 3 entspricht jeweils einer Richtung, wobei 0 für rechts steht und von dort aus im Uhrzeigersinn gegangen wird. In dem Versuch wo der Käfer nicht rückwärts laufen kann, wird, wenn er rückwärts laufen muss, der Käfer einmal rotiert, anschließend prüft er nach vorne und bewegt sich dann zur Seite. Auf diese Weise liessen sich alle Labyrinthgrößen lösen mit denen getestet wurde. Diese Tests liefen bis zur Größe 20. Folgend ein Beispiel wie der Käfer in einem Gang nach links läuft.

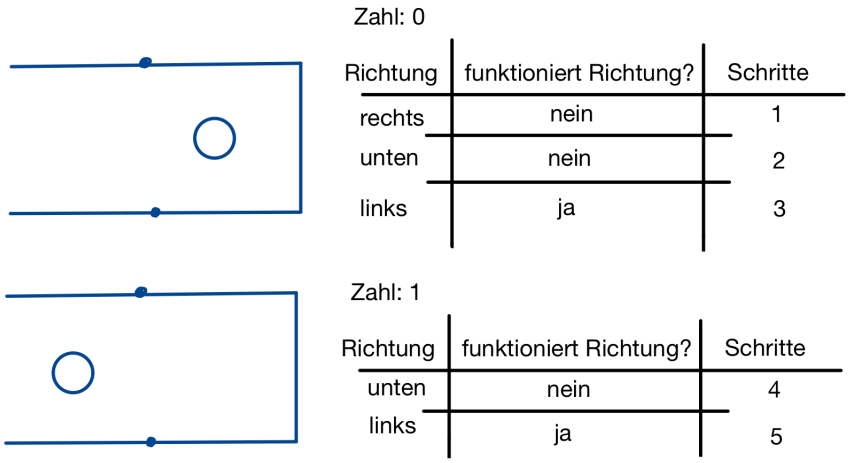

Abbildung 4: Bewegung des Käfers

3.3 Daten sammeln und speichern

Gespeichert wird welcher Durchlauf es insgesamt war, wie groß das Labyrinth war, welcher Durchlauf es für die bestimmte Größe war, in welcher Zeit und mit wie vielen Schritten der Durchlauf abgeschlossen wurde und wie das Labyrinth aussah. Die Zeit wird berechnet je nachdem ob eine Prüfung nach einer Wand stattgefunden hat oder ob ein Schritt gemacht wurde, heißt wenn ein Schritt gemacht wird, wird die Prüfung nach einer Wand nicht auf die Zeit gerechnet. Dies ist so vorzustellen als würde der Käfer gegen die Wand rennen, wenn eine dort ist hat er eine Zeiteinheit oben drauf wenn keine dort ist hat er sie ebenfalls oben drauf ist aber einen Schritt weiter. Die Schritte werden nur berechnet, wenn

sich der Käfer ein Feld weiter bewegt hat. Diese Daten werden anschließend für jeden Labyrinthgröße in einer CSV-Datei gespeichert.

4 Auswertung

Es wurden Labyrinthe von der Größe drei bis zur Größe 38 für den Versuch verwendet. Die Größe von 38 wird im Fazit erkärt. Jedes Labyrinth wurde dabei 100 mal gelöst und die Daten gespeichert und für diese Auswertung herangezogen. Nach dem ein Labyrinth einmal gelöst wurde wird ein neues generiert, somit werden 7600 Labyrinthe generiert. Somit sind das 100 pro Durchlauf 38 Durchläufe bei zwei Versuchen. Als Versuch eins bzw. zwei werden im Folgenden das Experiment betitelt, in welchem sich der Käfer in alle vier Richtungen bewegen konnte und in welchem er sich nur in drei Richtungen bewegen konnte. Ein Durchlauf sind alle Lösungen eines Labyrinths einer bestimmten Größe, Druchlauf 29 sind alle Lösungen der Labyrinthgöße 29.

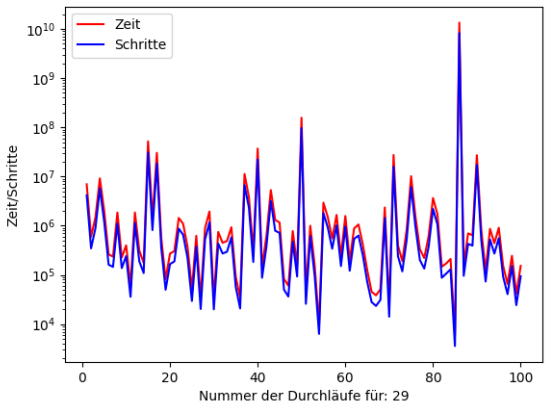

Abbildung 5: Logplot Durchlauf 29, erster Versuch

Bei der Betrachtung der Lösung eines einzelnen Labyrinths fällt auf, dass die Zeit und Schritte, welche benötigt wurden um dieses Labyrinth zu lösen, bis auf ein paar Ausreißer, in einem gewissen Bereich gleich nah aneinander liegen. Ein hervorragendes Beispiel dafür ist der Durchlauf 29, zu sehen in Abbildung 5, aus dem ersten Versuch. Dort ist nur ein Ausreißer deutlich zu erkenne, ansonsten liegen die Werte nah bei einander.

Zu beachten ist jedoch, dass die Werte nicht durchgehend die selben sind, abgesehen von den zuvor erwähnten Ausreißern, wie man es aus Abbildung 5 annehmen kann. Dass die Werte unterschiedlich sind und doch in der Nähe der anderen Werte liegen sieht man in Abbildung 6, wo der Durchlauf 3 des ersten Versuches zu sehen ist.

Nimmt man nun den Durchschnitt der Zeit und Schritte jeder Labyrinthsgröße und ermittelt dafür jeweils den Durchschnitt, so kann man anhand eines Plots ablesen, wie sich die benötigten Schritte und Zeit im Verhältnis zur Labyrinthsgröße verhalten.

In Abbildung 7 ist der erste Versuch abgebildet. In diesem ist zu erkennen, dass sich bis Labyrinthgröße 22 keine starke Veränderung der benötigten Zeit

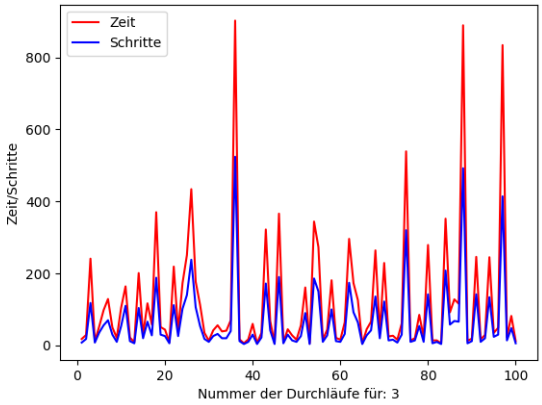

Abbildung 6: Plot Durchlauf 3, erster Versuch

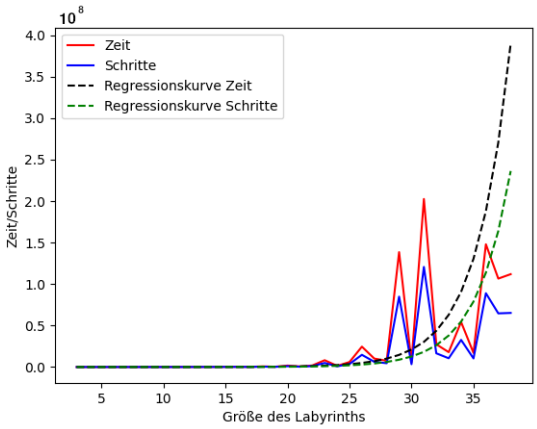

Abbildung 7: erster Versuch

oder Schritte einstellt. Bei Labyrinthgröße 23 ist der erste kleine Ausschlag zu erkennen und bei Labyrinthgröße 29 und 31 befinden sich die ersten beiden starken Ausschläge, wobei sich Größe 30 wieder in der Nähe der Größen bis 22 befindet. Anschließend findet ein leichter Anstieg statt und ein starker Ausschlag bei Größe 36. Die Ausschläge sind auf einzelne Extremwerte in den Daten des jeweiligen Druchlaufs zurück zu führen. Als Beispiel ist Durchlauf 29 im ersten Versuch anzuführen, rechnet man den Ausschlag des 85 Labyrinths von 13414446518 nicht mit ein so beträgt der Mittelwert nur 4373697 stat 138474425. Und würde den Ausschlag deutlich reduzieren. Nimmt man nun für kleinere Labyrinthe mehr Werte also bis zu einer Zahl von 300 Labyrinthen für eine Größe, so erhählt man den Graphen aus Abbildung 8 in dem zu erkennen ist, dass dies die Fluktuation im Bereich um Labyrinthgröße 30 nicht beieinflusst. Und ehr zu noch Fluktuation in dem Bereich der kleineren Labyrinthen führt.

Wird nun eine Regressionskurve eingefügt, so ist zu erkennen, dass der Anstieg einer Exponentialfunktion ähnelt. Im logarithmischen Plot Abbildung 9 ist dies eindeutig zu sehen, da die Regressionskurve zu einer perfekten Gerade geworden ist. Zu beachten ist aber das der Anstieg nicht perfekt exponentiel ist, da im Bereich um Größe 30 die werte fluktuieren. Somit ist die Regressionskurve eine

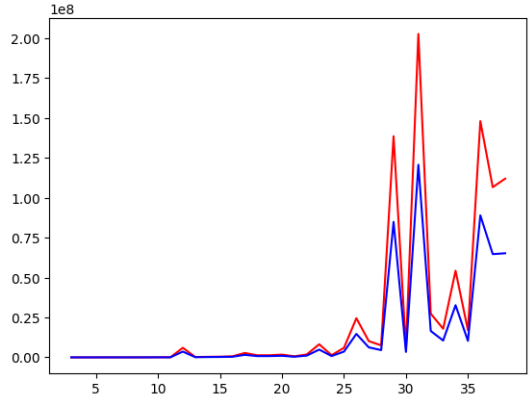

Abbildung 8: erster Versuch mit mehr kleinen Labyrinthen

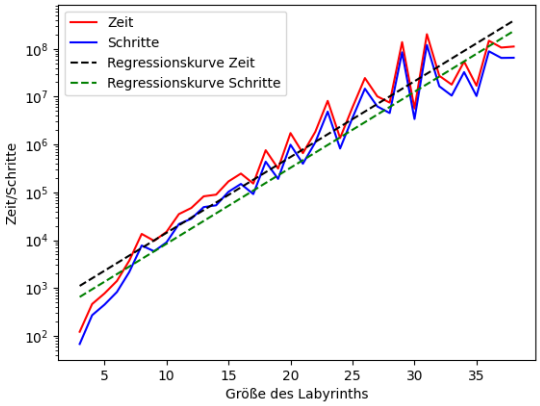

Abbildung 9: erster Versuch, Log-Plot

Exponentialfunktion und die Schritte bzw. Zeit steigen exponentiel.

Die gleiche Betrachtung ist ebenfalls auf den zweiten Versuch, Abbildung 10 anwendbar, hier ist auffällig, dass es nur einen Ausschlag gibt, welcher auffällt. Dieser liegt bei Labyrinthgröße 37. Durch Einfügen einer Regressionkurve sieht man in diesem Plot ebenfalls den Ansatz exponentielem Wachstums, welche sich in Abbildung 11 anhand der Geraden im logarithmischen Plot bestätigen lässt.

Betrachtet man nun den Anstieg der Exponentialfunktionen in Abbildung 7 und Abbildung 10 so fällt auf, dass der zweite Versuch, wo der Käfer nur in drei Richtung laufen kann, einen weniger starken Anstieg hat, als der Käfer im ersten Versuch, der in alle vier Richtungen laufen kann. Dies ist der Fall obwohl der Käfer im zweiten Versuch mehr Schritte machen muss um Rückwärts zu laufen. Begründen lässt es sich damit, dass es dem Käfer damit auch schwerer fällt seinen Weg wieder rückwärts zu gehen, da dies nur passiert, wenn entweder rückwärts der einzige Weg ist, er als Ausgang in die Richtung guckt, welche rückwärts ist, er in der Routine vorwärts, rechts oder vorwärts, rechts, links den weg Rückwärts beschreitet. Also ist es dem Käfer nur in vier Fällen möglich rückwärts zu laufen, während der Käfer aus dem ersten Versuch in folgen Fällen rückwärts laufen kann. In vier Fällen, wenn er in einer Sackgasse ist, in drei Fällen wenn ein Weg hinter ihm ist und einer wo anders, in zwei Fällen wenn ein Weg hinter ihm und

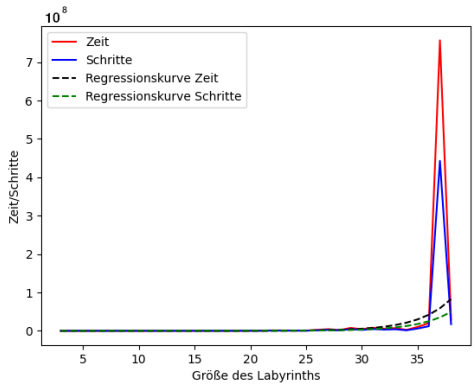

Abbildung 10: zweiter Versuch

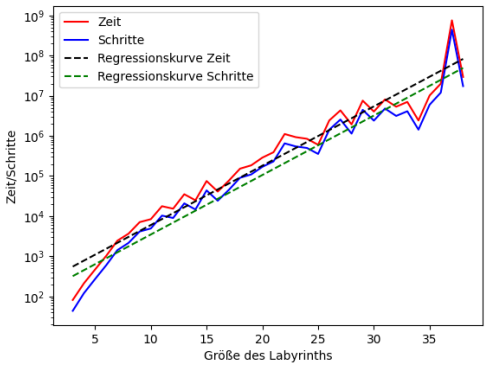

Abbildung 11: zweiter Versuch, Log-Plot

zwei Wege vor ihm sind und in einem Fall wen alle Wege offen sind. Somit ist es dem Käfer in 4 Fällen möglich rückwärts zu gehen. Folglich kann der Käfer aus dem ersten Versuch potenziel vier mal so oft seinen Weg wieder zurück gehen als der Käfer aus Versuch zwei. Es ist aber nur nötig rückwärts zu gehen, wenn der Käfer in einer Sackgasse gelandet ist. Somit ist der Käfer aus dem zweiten Versuch schneller als der Käfer aus dem ersten Versuch.

5 Fehlerbetrachtung

Auf Grund eines kleinen Datensets sind die Daten fehlerbehaftet, da ein einziger Durchlauf, in dem die Zahl der Schritte deutlich höher ist alles alle anderen den Durchschnitt hochzieht. Als Beispiel ist Abbildung 5 anzuführen, dort ist erkennbar, dass bis auf einen Fall die Zeiten an nährend gleich sind. Mit diesem Ausfall liegt der Durchschnitt jedoch über diesen Zeiten. Dem könnte mit einer höheren Anzahl an Werten entgegen gewirkt werden, unter der Bedingung, dass ähnlich große Ausschläge nicht gleich Häufig auftreten, wie in dem Fall von 100 Labyrinthen. Ebenfalls ist in dem zweiten Versuch, siehe Abbildung 10, zu erkennen, dass der exponentiele Anstieg ebenfalls durch einen Ausfall der Messwerte entsteht. Um sicherzustellen, dass der Anstieg exoponentiel ist, müssten weitere Werte für größere Labyrinthe erhoben werden. Im Bereich der kleineren Labyrinthe ist ebenfalls anzumerken, dass nich jedes unterschiedlich ist, da es nicht genügend Kombinationen gibt um diese zu erreichen.

6 Fazit

In dem Experiment, wurde ein Labyrinth erzeugt, welches immer lösbar ist und nur einen richtigen Weg besitzt. Durch dieses Labyrinth wurde ein Käfer geschickt, welcher im Uhrzeigersinn geprüft hat, ob sich an dieser Stelle eine Wand befand und wenn nicht sich auf diesen Punkt bewegt hat, dabei wurde die erste Wand welche geprüft wird zufällig ausgewählt. Im zweiten Versuch war es dem Käfer nicht möglich direkt rückwärts zu laufen, um dies zu tun musste er sich um 90° drehen. Dabei wurden die Anzahl der Schritte und die Anzahl der erfolgten Aktionen, als Zeit definiert, gespeichert.

Nach Betrachtung der Daten sind folgende Erkenntnisse aufgetreten. Es viel auf, das obwohl die Richtung, in welche sich der Käfer bewegt, zufällig gewählt wurden, der Käfer unterschiedliche Labyrinthe von der selben Größe in etwa der selben Zeit und Schrittzahl löst. Des weiteren wurde festgestellt, dass bei Betrachtung der benötigten Zeit und Schritte, über die verschiedenen Labyrinthgrößen, ein exponentieler Anstieg zu erkennen ist. Dabei ist aufgefallen, das der Anstieg während des ersten Versuchs ungefähr vier mal so groß ist, wie während des zweiten Versuchs, was darauf zurück zu führen ist, dass der Käfer während des zweiten Versuch aufgrund seiner Einschränkung nicht rückwärts laufen zu können ohne eine Drehung diese Bewegung weniger oft ausführte. Dadurch bewegte er sich weniger häufig auf Felder, auf welchen er sich vorher befand. Daraus entsteht die Erkenntnis, dass es schneller ist ein Labyrinth zu lösen, in dem nur zurückgegangen wird, wenn eine Sackgasse erreicht wird.

Probleme während der Durchführung des Experiments, waren die Sicherstellung der Lösbarkeit des Labyrinths sowie das Speichern der Daten. Beim Speichern der Daten war es geplant ebenfalls den genauen Pfad des Käfers zu speichern, um zu prüfen wie extreme Ausschläge genau zu stande kamen. Dies erwies sich jedoch als nicht möglich, da mir drei Daten der Labyrinthgröße 15, 17 und 18 eine Speichermenge von ca. 154Gb erreicht wurde, was ca 17% des gesamt verfügbaren Speichers entsprach. Ebenfalls geplant war es bis zu einer Labyrinthgröße von 50 zu testen. Dies war ebenfalls nicht möglich, da nach Labyrinthgröße 38 im ersten Versuch und Lagyrinthgröße 41 im zweiten Versuch die für das Programm verfügbaren CPU Kerne zu 100% genutzt waren und das Programm nicht weiter lief. Das letzte Problem war, das durch einen Fehler im Code die Daten 100 mal in der selben Datei gespeichert wurden anstatt einmal. Dies wurde behoben und hat auch die Daten nicht weiter beeinflusst, da die exakt gleichen Werte per Hand entfernt wurden.

Literatur

[1] Buck Jamis: „Mazes for Programmers. Code Your Own Twisty Little Passages“ Raleigh: Pragmatic Bookshelf, 2015, S. 158–163.

[2] Takaharu Shokaku u. a.: „Development of an automatic turntable-type multiple T-maze device and observation of pill bug behavior“ In: *Rev Sci Instrum* 91 (2020) H. 10. S. 104104.

[3] John Stilley: „Mazelib“ URL: https://github.com/john-science/mazelib (besucht am 29. 09. 2023).

[4] Bellot Victor u. a.: „How to Generate Perfect Mazes?“ In: *Information Sciences* 10.1016 (2021).

[5] Hollis Williams: „The Mathematics of Mazes“ In: (Aug. 2020).