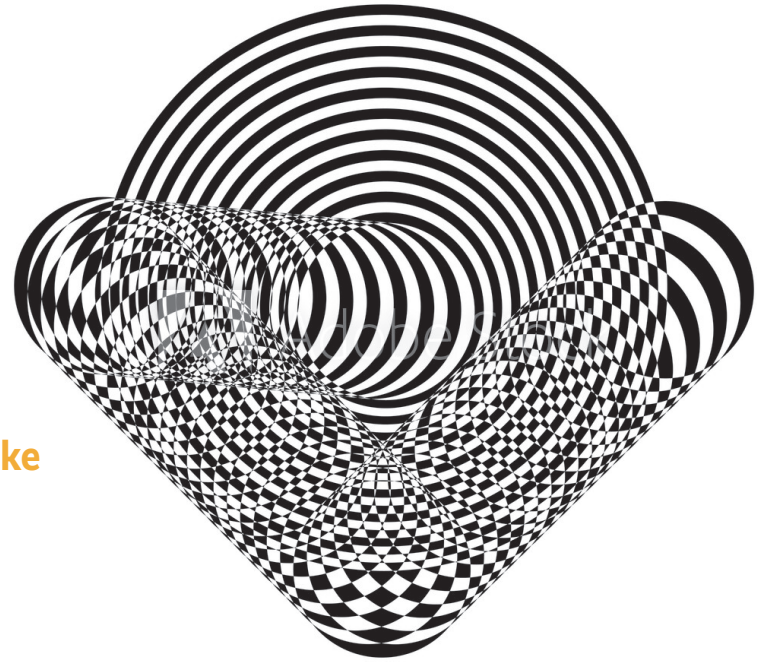
Klassische versus Quantenprogrammierung: der Random Walk

Überlagerte Wege



Sebastian Bock, Prof. Dr. Adrian Paschke

Den Gesetzen der Quantenmechanik folgend wohnt der Quantenprogrammierung eine vollständig andere Logik inne als der traditionellen.

Entwicklerinnen sind mit der Softwareentwicklung auf klassischen Rechnern vertraut. Intuitive Programmiersprachen, die auf bekannten Denk- und Sprachmustern aufbauen, ermöglichen auch Neulingen einen schnellen Einstieg und erste Erfolge bei kleinen Anwendungen. Bei der Programmierung eines Quantencomputers ist die Lage durch die zugrunde liegenden Gesetze der Quantenmechanik komplizierter und deutlich abstrakter. Die Unterschiede bei der Programmierung auf einem klassischen und einem Quantencomputer soll ein Beispiel verdeutlichen.

Steffen fliegt in den Urlaub. Sofort zieht es ihn auf die Strandpromenade. Um fünf Uhr morgens stolpert er reichlich angetrunken aus einer Bar und kann sich nicht mehr erinnern, in welcher Richtung sein Hotel liegt. Dort muss er aber schnellstmöglich hin, will er sich pünktlich um sechs Uhr eine Liege in der ersten Reihe am Hotel-Pool reservieren. Steffen denkt nach: Das Hotel muss irgendwo an dieser Straße liegen. In einer Mathevorlesung vor mehreren Jahren hatte der Professor etwas über Random Walks erzählt und dass der Walker nach beliebig vielen Schritten jeden Punkt auf einer Linie erreichen kann.

Steffen fasst also den Entschluss, eine Münze zu werfen. Bei Kopf geht er nach links, bei Zahl nach rechts jeweils bis zum nächsten Hotel und schaut, ob es das richtige ist. Falls nicht, wirft er wieder eine Münze und geht entsprechend nach links oder rechts. Wenn der Professor recht hatte, müsste er irgendwann an seinem Hotel ankommen (siehe Abbildung 1).

Steffens Vorgehen entspricht einem einfachen, diskreten Random Walk in einer Dimension, also auf einer Linie. Tatsächlich erreicht man damit nach beliebig vielen Schritten jeden Punkt auf dieser Linie. Der Haken: Steffen hat nicht beliebig viel Zeit, er will in einer Stunde im Hotel sein. Zwei kleine Programme – ein klassisches und ein Quantenprogramm – sollen das Umherirren von Steffen modellieren.

Die klassische Irrfahrt

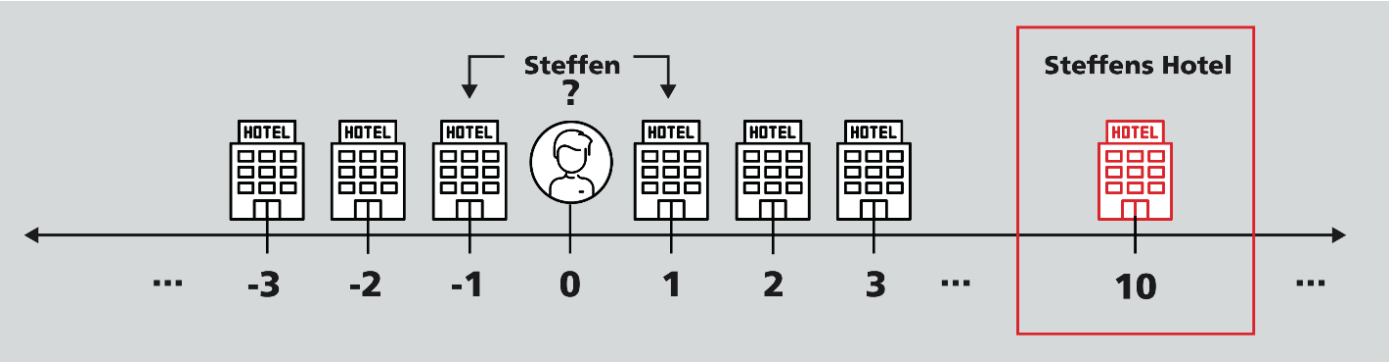
Für diesen einfachen Fall kann der Ablauf des klassischen Python-Programms der Beschreibung des Random Walk folgen. Eine Funktion `Randwalk(n)` für den Random Walk bekommt die Anzahl der zu gehenden Schritte `n` übergeben. Die Funktion erhält außerdem eine Variable für die Position und eine für die verstrichene Zeit. In der `for`-Schleife, die n-mal durchlaufen wird, übernimmt die Funktion `random.choice` den Münzwurf. Sie gibt mit einer Wahrscheinlichkeit von jeweils 0,5 False – also Zahl – oder True – für Kopf – zurück. Bei False wird die Positionsvariable `x` um 1 erhöht, Steffen geht nach rechts, bei True wird `x` um eins verringert, Steffen geht nach links. Nach Durchlaufen der Schleife gibt die Funktion die Position `x` nach `n` Zeitschritten zurück (siehe Listing 1).

Mit diesem kleinen Programm lässt sich untersuchen, wie die Wahrscheinlichkeitsverteilung nach beispielsweise 16 Zeitschritten aussieht. In diesem Fall hat Steffen etwa vier Minuten pro Münzwurf und Gang zum nächsten Hotel, um pünktlich anzukommen. Dazu wird die oben definierte Funktion sehr oft, zum Beispiel 100 000 Mal, aufgerufen und die Ergebnisse werden in einem Histogramm dargestellt (siehe Abbildung 2).

Die Wahrscheinlichkeit, dass Steffen nach genau 16 Zeitschritten wieder am Ausgangspunkt ankommt, ist mit knapp 20 Prozent am größten und die Wahrscheinlichkeit, dass er sein Hotel erreicht hat, mit weniger als einem Prozent viel kleiner. Diese Darstellung vereinfacht die Situation aber drastisch. Bei einer genaueren Betrachtung, die alle Mög-

X-TRACT

- Traditionelle und Quantencomputing-Programmierung unterscheiden sich vor allem in der Programmlogik.
- Dass klassische und Quantenprogramme zu völlig unterschiedlichen Ergebnissen kommen können, zeigt das mathematische Problem des Random Walk.
- Auch für die Quantenprogrammierung ist es unerlässlich, die Ausgangsaufgabe auf klassischer Ebene zu verstehen. Danach ist sie in die Sprache der Quantenmechanik zu übersetzen.



Steffen hat die Orientierung verloren und will zu seinem Hotel zurück. Er wirft eine Münze und geht mit einer Wahrscheinlichkeit von 50 Prozent nach links und mit einer Wahrscheinlichkeit von 50 Prozent nach rechts (Abb. 1).

lichkeiten berücksichtigt, in denen Steffen in höchstens 16 Zeitschritten, also etwa auch schon nach 14 Schritten, sein Hotel erreicht, kommt man auf eine Wahrscheinlichkeit von 1,27 Prozent. Immer noch ist es sehr unwahrscheinlich, dass Steffen mit dem klassischen Random Walk sein Hotel rechtzeitig erreicht.

Die Quantenirrfahrt

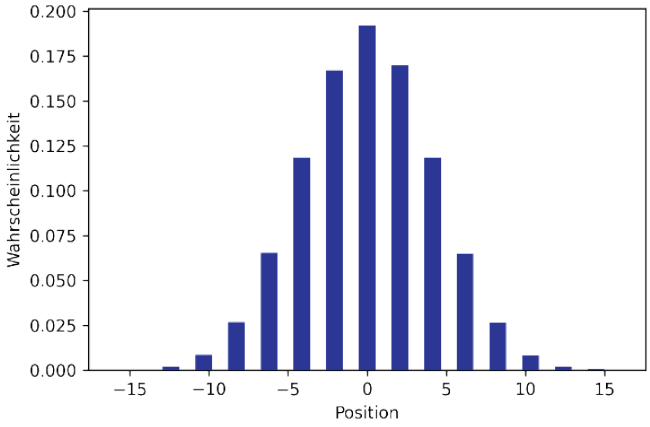
Ganz anders ginge die Situation aus, wenn Steffen ein Quantenteilchen wäre, zum Beispiel ein Photon. Um zu schauen, wie sich die Aufenthaltswahrscheinlichkeit im Vergleich zum klassischen Random Walk ändert, soll für dieses Quantenteilchen ebenfalls der Random Walk beschrieben und in einem Programm modelliert werden.

Dazu ist das Problem zunächst in der Sprache der Quantenmechanik auszudrücken, das heißt, die Position des Teilchens und der Münzwurf sind als quantenmechanische Zustände zu beschreiben und in Qubits zu speichern. Die Beschreibung soll möglichst naiv und analog zum klassischen Fall geschehen. Dass sich Steffen an Position 0 befindet, drückt der quantenmechanische Zustand $|0\rangle$ aus, Position -1 entspricht $|-1\rangle$ und so weiter. Den Münzwurf beschreibt ein zusätzliches Qubit. Der Zustand $|0\rangle$ soll der Zahl und der Zustand $|1\rangle$ dem Kopf entsprechen. An dieser Stelle besteht ein wesentlicher Unterschied zum klassischen Random Walk: Qubits können sich in einem Überlagerungszustand von $|0\rangle$ und $|1\rangle$ befinden:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

Das ähnelt dem Münzwurf schon sehr. Vorstellen kann man sich das so: Man legt die Münze auf den Daumen, dabei liegt Kopf $|1\rangle$ oder Zahl $|0\rangle$ oben. Die Münze wird in die Luft geschnipst, fängt an sich zu drehen und befindet sich gewissermaßen in einem Überlagerungszustand von Zahl und Kopf. Solange sie fliegt und sich dreht, ist das Ergebnis unbekannt. Die Wahrscheinlichkeit für beide Ereignisse liegt bei 50 Prozent.

An dieser Stelle gibt es einen entscheidenden Unterschied: Beim klassischen Random Walk wartet Steffen, bis die Münze in seiner Hand landet, schaut sich das Ergebnis an und geht dann nach links oder rechts. Beim Quantum Random Walk hingegen geht das Programm sozusagen schon während des Münzwurfes bei Kopf nach links und bei Zahl nach rechts.



Die Wahrscheinlichkeiten eines einfachen, diskreten Random Walk nach 16 Schritten und 100 000 Durchläufen folgen der Binomialverteilung (Abb. 2).

Dies führt dazu, dass sich das Quantenteilchen, genau wie die Münze, nach dem ersten Schritt in einem Superpositionszustand aus |–1⟩ und |1⟩ befindet. Anschaulicher darstellen lässt sich das mit einem Galtonbrett. Statt aus Stäben und Kugeln besteht dieses jedoch aus Strahlteilern und Photonen (siehe Abbildung 3).

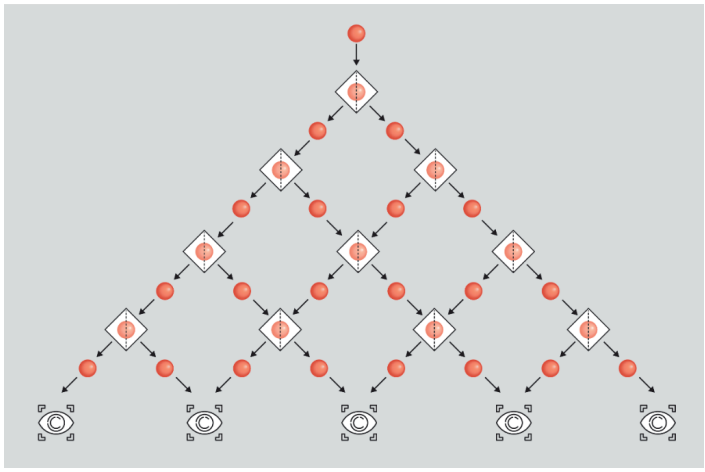
Nicht nach jedem Münzwurf hinschauen

Der erste Strahlteiler zerlegt das Photonenbündel in zwei Hälften. Eine geht nach rechts, die andere nach links ab. Werden die Photonen weiter geteilt, überlagern sich die beiden mittleren Bündel bereits auf der dritten Strahlteilerebene. Dieser Überlagerungseffekt ist charakteristisch für den Quantum Walk. Erst in der untersten Ebene werden dann die Photonen gemessen, das heißt, erst hier zeigt sich, welches Auge ein Lichtsignal registriert.

Mit der Beschreibung des Quantum Walk und dem abgewandelten Galtonbrett lässt sich nun ein Python-Programm für Quantenrechner modellieren. Zuerst ist eine Quantum-Walk-Funktion zu definieren und ihr der Quantenschaltkreis, das Quantenregister, auf dem das Teilchen läuft, das Quantenregister für die Münze und die Anzahl der Schritte zu übergeben. In der Schleife, die das Programm n-mal durchläuft, führt es zuerst auf dem Münz-Qubit ein Hadamard-Gatter aus – und wirft damit die Münze. Die `increment`-Funktion erhöht dann die Position um 1 für den Fall, dass das Münz-Qubit in Zustand |1⟩ ist, die `decrement`-Funktion verringert die Position

Listing 1: Der klassische Random Walk

```
# n: Anzahl der Zeitschritte
def Randwalk(n):
  x = 0

  for i in range(n):
    # 'Zahl' False, 'Kopf' True
    coin = random.choice([True, False])
    # Schritt nach rechts wenn 'Zahl'
    if coin == False:
      x += 1

    # Schritt nach links wenn 'Kopf'
    if coin == True:
      x += -1
  # gibt Position x nach n Schritten zurück
  return x
```



Stellt man den Quantum Random Walk mit Photonen und Strahlteilern dar, sieht man, dass die Photonen erst am Ende gemessen werden, nachdem sie sich schon mehrfach überlagert haben (Abb. 3).

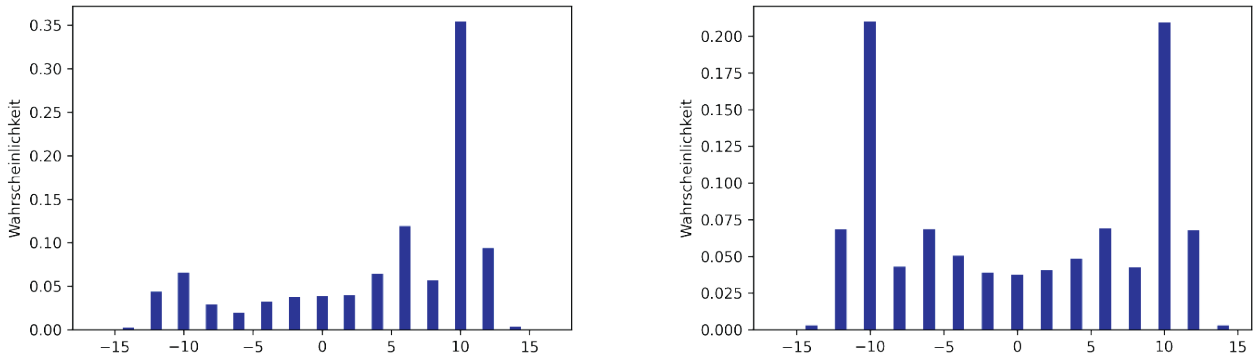
um 1, falls das Münz-Qubit in |0⟩ ist. Nach Durchlaufen der Schleife wird der so erzeugte Quantenschaltkreis zurückgegeben (siehe Listing 2).

Der weitere Teil des Programms dient dazu, die Quantenregister und dazugehörigen klassischen Register, die man für die Messung benötigt, sowie den Quantenschaltkreis zu erzeugen. Danach ist noch die Anzahl der Zeitschritte festzulegen, der richtige Anfangszustand zu definieren und der Quantum Walker zu starten. Am Ende misst das Programm den Zustand der Qubits.

Die Ergebnisse für jeweils 100 000 Durchläufe mit einem symmetrischen und einem rechtslastigen Quantum Walker berechnete in diesem Fall ein Quantensimulator (siehe Abbildung 4). Die Wahrscheinlichkeitsverteilung für den symmetrischen und den rechtslastigen Quantum Walker hängt dabei vom gewählten Anfangszustand der Quantenmünze ab. Ein echter Quantencomputer hat eine zu hohe Fehlerrate, als dass er hierfür sinnvolle Ergebnisse liefern würde. Beim Quantum Random Walk erreicht Steffen bei der richtigen Wahl des Anfangszustandes mit einer Wahrscheinlichkeit von über 35 Prozent sein Hotel nach 16 Zeitschritten und damit rechtzeitig.

Listing 2: Der Quantum Random Walk

```
def QuantumWalk(qc, q, s, n):  # implementiert Quantum Walk

  for i in range (steps):
    qc.h(s)
    increment(qc, q, s)        # erhöht Zustand q um 1, wenn s in |1⟩
    decrement(qc, q, s)        # verringert Zustand q um 1, wenn s in |0⟩
  return qc

number_qubits = 7              # Anzahl der Qubits

# Quantumregister und Classicalregister (für Messungen)
# für die Schaltung, auf der der RandomWalk läuft
qnodes = QuantumRegister(number_qubits,'qc')
cnodes = ClassicalRegister(number_qubits,'cr')

# Quantumregister und Classicalregister (für Messungen) für die Münze
qcoin = QuantumRegister(1,'qanc')
ccoin = ClassicalRegister(1,'canc')

# Quantum Walk Circuit (qwc)
qwc = QuantumCircuit(qnodes, qcoin, cnodes, ccoin)

n = 16                         # Anzahl der Zeitschritte

qwc.x(qsubnodes[0])            # Sorgt für "Rechtslastigkeit" des Walks

QuantumWalk(qwc, qnodes, qcoin, n)
qwc.measure(qnodes, cnodes)
```



Die Wahrscheinlichkeitsverteilung für den symmetrischen (links) und den rechtslastigen Quantum Walker (rechts) zeigt: Die Verteilung hängt vom gewählten Anfangszustand der Quantenmünze ab (Abb. 4).

Die mathematische Frage allerdings, wie genau der richtige Anfangszustand aussieht und warum die Wahrscheinlichkeitsverteilung überhaupt vom ihm abhängt, würde den Rahmen dieses Artikels sprengen (siehe ix.de/zzhq).

Fazit

Das Beispiel des eindimensionalen Random Walk zeigt, wie komplex die Quantenmechanik ist. Ein kleines Quantenprogramm lässt sich verhältnismäßig einfach schreiben, auch wenn dazu gute Vorkenntnisse und mathematische Vorbetrachtungen notwendig sind, die hier stark verkürzt dargestellt wurden. Drei wesentliche Punkte sind bei der Implementierung des Quantum Walk, aber auch bei anderen Algorithmen entscheidend: Zunächst ist es wichtig, das zu lösende Problem auf klassischer Ebene zu verstehen, was schon sehr komplex sein kann. Danach lässt sich die Aufgabe in die Sprache der Quantenmechanik übersetzen und einer tiefergehenden mathematischen Betrachtung unterziehen. Erst nach dieser Vorarbeit kann man im dritten und letzten Schritt den Algorithmus in ein Quantenprogramm umsetzen. (sun@ix.de)

Quellen

[1] Weiterführende Literatur zum Random Walk siehe ix.de/zzhq
[2] Norbert Henze; Irrfahrten – Faszination der Random Walks; Springer Spektrum 2018

Sebastian Bock

arbeitet als wissenschaftlicher Mitarbeiter am Fraunhofer FOKUS im Bereich Quantum Computing. Die Leidenschaft für den Fachbereich entwickelte er während seines Physikstudiums in Chemnitz und Siegen.

Prof. Dr. Adrian Paschke

ist Professor im Bereich Corporate Semantic Web am Institut für Informatik der Freien Universität Berlin und Leiter des Data Analytics Center am Fraunhofer FOKUS. Er beschäftigt sich mit der Anwendungsforschung in der künstlichen Intelligenz, u. a. mit quantenunterstützter KI im vom BMBF-geförderten Projekt PlanQK.