Kolmogoroff-Komplexität

Vortragsausarbeitung (1. Teil)

Sandro Köppen
Mirko Hochmuth

Berlin, 2001

Einleitung
In folgender Vortragsausarbeitung wollen wir uns mit der so genannten Kolmogoroff – Komplexität beschäftigen. Nach einigen kurzen Daten zum Namensgeber Andrei Nikolaevich Kolmogoroff beginnen wir mit einem einführendem Beispiel und definieren daraus die Kolmogoroff-Komplexität. Mithilfe der Kolmogoroff-Komplexität wird dann die universelle Wahrscheinlichkeitsverteilung beschrieben. Mit diesen lässt sich dann zeigen, dass eben unter dieser Verteilung die Laufzeiten von jedem (endlichen) Algorithmus im Durchschnittsfall und im schlechtesten Fall bis auf einen konstanten Faktor gleich sind.

Andrei Nikolaevich Kolmogoroff
Der russische Mathematiker wurde am 25.04.1903 in Tambov (Russland) geboren. Er studierte an der Universität von Moskau. Ab 1930 lehrte er als Professor an dieser Universität. Kolmogoroff beschränkte sich nicht nur auf ein Teilgebiet der Mathematik sondern betrieb seine Forschungen in fast jede Richtung: Mengenlehre, Konstruktionslogik, Informationstheorie, Ballistik, Wahrscheinlichkeitstheorie um nur einige zu nennen. Anwendung fand seine Forschung unter anderem in der Biologie und Geologie (Kristallwachstum). Er erhielt in seinem Leben viele Auszeichnungen darunter den Stalin- und den russischen Staatspreis und den Bolzano-Preis. Auch im Ausland war Kolmogoroff ein anerkanntes Mitglied der mathematischen Gesellschaft. Er hat unter anderem Ehrentitel der Universitäten von Paris und Berlin und war Mitglied diverser Akademien wie der Königlichen Akademie der Niederlande, der Königlichen Akademie Londons, der USA National Society und Akademien in Frankreich, Polen und Rumänien.
Die von uns nun betrachtete Kolmogoroff-Komplexität oder auch „Algorithmic Information Theory“ genannte Theorie entwickelte Kolmogoroff in den 70er Jahren.
Er starb im Alter von 84 Jahren am 20.10.1987 in Moskau.

Ein einleitendes Beispiel
Am Anfang soll mit einem einfachen Beispiel begonnen werden. Daran wollen wir die unterschiedliche Komplexität von Wörtern oder Strings zeigen. Unser Alphabet besteht aus den Zeichen 0 und 1 . Es können also ua. folgende Wörter oder Zahlenfolgen gebildet werden:

10101010101010101010101010101010101010 (a)

und

11110110101010010101001110001011010101111 (b)

Die erste Folge sieht sehr einfach aus, es sind einfach 20 mal die Zeichen 0 und 1 im Wechsel geschrieben. Bei dem zweiten Beispiel kann man keine Regelmäßigkeit mit dem ersten Blick erkennen. Zu bedenken ist aber, dass beide Folgen mit der Länge 40 im Sinne der Wahrscheinlichkeitsrechnung gleich wahrscheinlich sind, beide haben die gleiche Anzahl von Nullen und Einsen. Es gibt also 2^{40} Folgen oder Wörter der Länge 40 über unserem Alphabet. Jedes tritt mit einer Wahrscheinlichkeit von $1/(2^{40})$ auf. Wir können mit der Wahrscheinlichkeitsrechnung offensichtlich nicht erkennen ob ein Wort zufällig entstanden ist oder einem Bildungsgesetz folgt.

Folgender Algorithmus oder Programm erzeugt die erste Zeichenfolge:

```
FOR i=1 TO 20
        PRINT '10'
NEXT i
```

Die zweite Folge wird mit diesem Programm erzeugt:

```
PRINT ' 11110110101010010101001110001011010101 11'
```

Mit n=40 hat nun das erste Programm die Länge O(1) + log (n/2) und das zweite O(1) + n. O(1) steht hier für den eigentlichen Programmtext (z.B. PRINT im zweiten Beispiel).
Wie leicht zu erkennen ist, haben wir im ersten Fall ein einfaches Bildungsgesetz gefunden, das uns die Zeichenfolge erzeugt. Im zweiten Fall blieb uns nichts anderes übrig, als die gesamte Folge hinzuschreiben.

Kolmogoroff-Komplexität
Es lässt sich nun wie folgt die (absolute) Kolmogoroff-Komplexität definieren:
Sei x ϵ {0,1}*. Dann bezeichnet K(x) die Länge des kleinsten binären Programms, das in der Lage ist, ohne Eingabe in endlicher Zeit das Wort x auszugeben.

Hier ist K(a) = O(1) + log n.

Aus der Definition folgt sofort:
$0 \leq K(x) \leq O(1) + n$

Zufälligkeit
Ein Wort x heiße in unserem Sinne zufällig, wenn $K(x) \geq n$ für die Kolmogoroff-Komplexität dieses Wortes gilt. Es gibt also keinen Algorithmus der x ausgeben kann, ohne im Wesentlichen n Bits zu benötigen.

Interessant ist zu zeigen, dass Wörter, die zu einem Viertel aus Einsen und dem Rest (also drei Vierteln) aus Nullen bestehen, in unserem Sinne nicht mehr zufällig sein können.
Beweis:

Es gibt nur eine begrenzte Anzahl von Wörter der Länge n mit dieser Eigenschaft, nämlich :

$$\binom{n}{\frac{3n}{4}} = \binom{n}{\frac{n}{4}} = \frac{n!}{\left(\frac{3}{4}n\right)! \cdot \left(\frac{1}{4}n\right)!}$$

Nach der Sterlingschen Näherungsformel für die Fakultät sind dies ungefähr:

$$\sqrt{\frac{8}{3\pi n}} \cdot \left(\frac{4}{3}\right)^{\frac{3}{4}n} \cdot 4^{\frac{1}{4}n} = \sqrt{\frac{8}{3\pi n}} \cdot 1.7547^{\,n}$$

Diese Wörter lassen sich nun über eine Indexierung eindeutig zuordnen, sodass für einen Algorithmus zur Bestimmung des i-ten Wortes der Länge n mit dieser Eigenschaft ein Platzbedarf von O(1) + *log i* folgt.
Damit ist auch die Kolmogoroff-Komplexität dieses i-ten Wortes beschränkt auf

$K(x) \leq O(1) + \log i$

Wobei für *log i* folgende obere Schranke gilt:

$$\log(i) = \log\left(\sqrt{\frac{8}{3\pi n}} \cdot 1.7547^{\,n}\right) \leq 0.8112\ n$$

Dies ist nun aber ein Widerspruch zu unserer Zufälligkeitsdefinition, da wir weniger als O(1) + n Platz benötigen, um x zu berechnen!

qed.

Man kann das erste Programm noch dahingehend erweitern das wir die Länge des zu erzeugenden Wortes nicht konstant Vorgeben sondern als Eingabe erwarten:

```
INPUT n
FOR i=1 TO n/2
    PRINT '10'
NEXT i
```

Die Länge des Programmtextes schrumpft somit auf O(1) da wir keine Konstante mehr im Programm angeben müssen. Das zweite Beispiel lässt sich nicht so erweitern.

Definition „relative Kolmogoroff-Komplexität“:
Seien $x,y \in \{0,1\}^*$. Dann bezeichnet $K(x|y)$ die Länge des kleinsten (binären) Programms, das in der Lage ist unter der Eingabe von y in endlicher Zeit das Wort x auszugeben.

Zwei kleine Beispiele
Um das Prinzip der Kolmogoroff-Komplexität zu verdeutlichen, sollen nun zwei Beispiele angebracht werden. Wir wollen für bestimmte Wörter oder Algorithmen die K-Komplexität angeben.

1. Beispiel:

Frage ist:

Wie groß ist die Kolmogoroff-Komplexität $K(\pi_n \mid n)$ wenn π_n die Folge der ersten n Binärstellen der Zahl π ist?

Antwort:

$K(\pi_n \mid n) \leq c$

Begründung:

Es existiert mindestens ein Programm bzw. Algorithmus der zum Beispiel die Zahl π im Näherungsverfahren berechnen kann. Wenn dieser Algorithmus π bis auf die n-te Stelle berechnet hat stoppt er und gibt sein Ergebnis aus. Dieser Algorithmus hat konstante Länge.

2. Beispiel:

Behauptung ist:

Es gibt eine Konstante c, sodass für alle x gilt $K(x \mid x) \leq c$.

Beweis:

Wenn wir folgendes Programm betrachten:

```
INPUT w
OUTPUT w
```

erkennen wir, dass das Programm bei Eingabe eines Wortes x genau das Wort x auch ausgibt. Weiterhin hat das Programm eine konstante Länge und erfüllt somit obige Behauptung.

Es kann nicht „viele“ Wörter mit geringer Kolmogoroff-Komplexität geben, da es höchstens 2^k Wörter x mit K(x)=k gibt. Die Verteilung dieser 2^k möglichen Wörter lässt sich wie folgt beschreiben:

Anzahl Wörter mit K(x)=0 ist ≤ 1
Anzahl Wörter mit K(x)=1 ist ≤ 2
Anzahl Wörter mit K(x)=2 ist ≤ 4
...
Anzahl Wörter mit K(x)=n-1 ist $\leq 2^{n-1}$

Oder allgemein: Die Anzahl der Wörter mit $K(x) \leq k$ ist kleiner als $1 + 2 + ... + 2^k = 2^{k+1}-1$.

Interessant ist auch zu zeigen, dass die Funktion **x → K(x) nicht berechenbar** ist.
Beweis:

Angenommen, sie wäre doch berechenbar.
Das (immer stoppende) Programm dazu heiße M.
Wir benutzen M als Unterprogramm in folgendem Programm P_m:

```
x:=e;
REPEAT
  x:= Nachfolger von x
UNTIL M(x) ≥ m;
OUTPUT x;
```

Dieses Programm berechnet offenbar das lexikografisch erste Wort x mit K(x)=m.

Dilemma: Das Programm hat lediglich die Länge O(1) + log m, d. h. wir haben nun das Wort x (welches ja eine K-Komplexität von m hat) mit O(1)+log m beschrieben. Dies ist ein Widerspruch, also kann es kein M geben, welches K(x) berechnet.

Universelle Wahrscheinlichkeit

Im Folgenden definieren wir die universelle Wahrscheinlichkeit auf den Wörtern x einer bestimmten Länge.
Wir **wählen** m(x) proportional zu $1/2^{2K(x|n)}$.

Das sorgt dafür, dass Wörter mit geringer Kolmogoroff-Komplexität eine höhere Wahrscheinlichkeit besitzen als Wörter mit einer großen Komplexität. Dies hat vor allem philosophische Überlegungen zum Hintergrund, da regelmäßige und pseudozufällige Wörter in der Praxis wohl häufiger auftreten als wirkliche Zufallswörter.

Es ist zu beachten, dass $\sum_{x:|x|=n} 2^{-2K(x|n)}$ gegen einen von n unabhängigen Wert d konvergiert, damit man für die universelle Wahrscheinlichkeitsverteilung definieren kann:

$$m(x)=c\cdot 2^{-2K(x|n)} \qquad (c = 1/d)$$

denn nur dann gilt:

$$\sum_{x:|x|=n} M(x) = 1.$$

Laufzeit von Algorithmen (worst-case vs. average-case)

Nun wollen wir zeigen, dass das Komplexitätsverhalten von Algorithmen im schlechtesten Fall und im mittleren Fall unter Verwendung der oben definierten Wahrscheinlichkeits-verteilung bis auf einen konstanten Faktor zusammenfallen.

Zu Anfang wollen wir dies am Beispiel von Quicksort betrachten. Die mittlere Komplexität von Quicksort ist O(n * log n) und im schlechtesten Fall $O(n^2)$.
Dieses schlechte Verhalten hat der Algorithmus gerade bei der Eingabe eines bereits sortierten Wortes bzw. Feldes. Die Kolmogoroff-Komplexität dieses Wortes ist $K((1,2,3,...,n)|n)=O(1)$. Unter der von uns definierten Wahrscheinlichkeitsverteilung m ist das Auftreten dieses Wortes besonders groß und konstant für alle n:

$$m((1,2,3,...,n)) = 2^{-2K((1,2,3,...,n)|n)} = 2^{-2O(1)} := a$$

Weiterhin definieren wir die Funktion $T_A(x)$, die die Rechenzeit eines Algorithmus A bei Eingabe x berechnet.

Somit erhalten wir unter der Wahrscheinlichkeitsverteilung m folgenden Erwartungswert für die Rechenzeit von Quicksort

$$\sum_{x:|x|=n} m(x) \cdot T_{Quicksort}(x) \geq m((1,2,...,n)) \cdot T_{Quicksort}((1,2,...,n))$$
$$= a\cdot O(n^2) = O(n^2)$$

Quicksort hat nun also auch im mittleren Fall eine Laufzeit von $O(n^2)$.

Am Ende des ersten Teils wollen wir mit dem Beweis für folgenden Satz schließen:

> Unter der Wahrscheinlichkeitsverteilung **m** gilt für jeden immer stoppenden Algorithmus **A, dass seine Durchschnittskomplexität bis auf einen konstanten Faktor der Schlechtesten-Fall-Komplexität entspricht.**

Oder anders:

$$\mathbf{T_{average\ case} = c \cdot T_{worst\ case}}$$

Um den Beweis führen zu können, betrachten wir vorher ein Programm, das bei Eingabe von n ein bestimmtes Wort der Länge n ausgibt:

```
INPUT n
w:=0
FOR (alle y mit |y|=n, in lexikografischer Ordnung)
      v = T_A(y)
      IF v>w THEN
            w:=v
            x:=y
      END IF
NEXT y
OUTPUT x
```

Dieses Programm gibt genau ein x_n für die Länge n aus bei dem der Algorithmus A seine Schlechtester-Fall-Komplexität erreicht. Das Programm hat eine feste Länge c. Es gilt bei Eingabe n gibt das Programm x_n aus, also ist $K(x_n \mid n) \leq c$.
Da wir m(x) proportional zu $1/2^{2K(xn \mid n)}$ und wir $K(x_n \mid n) \leq c$ in diese Formel einsetzen können gilt:

$m(x_n) \geq 1 / 2^{2c}$

$1 / 2^{2c}$ ist konstant also können wir dafür schreiben:

$m(x_n) \geq a$

Nun folgt der eigentliche **Beweis:**
Wir definieren:

$T_{Awc}(n) = \max\{T_A(x) \mid |x|=n\}$ als die Schlechtester-Fall-Komplexität oder worst-case-Komplexität von **A**

und

$$T_{Aav\mathbf{m}}(n) = \sum_{x\,:|x|=n} m(x)\, T_A(x)$$

die erwartete Komplexität bzw. Durchschnittskomplexität von **A** bei Eingabe **x** unter der Wahrscheinlichkeitsverteilung **m.**

Mit obigem Programm haben wir beschrieben das $T_A(x_n) = T_{Awc}(n)$ ist, also worst-case-Komplexität hat und $m(x_n) \geq a$ gilt.

Damit ergibt sich:

$$T_{Aavm}(n) = \sum_{x\,:|x|=n} m(x)\, T_A(x)$$

$$\geq m(x_n) \cdot T_A(x_n) \qquad \text{// weil } \mathbf{wc} \text{ in Summe enthalten}$$

$$= m(x_n) \cdot T_{Awc}(n) \qquad \text{// mit } T_A(x_n) = T_{Awc}(n)$$

$$\geq a \cdot T_{Awc}(n) \qquad \text{// mit } a = m(x_n)$$

also:

$$T_{Aavm}(n) = b \cdot T_{Awc}(n) \qquad \text{// mit } b = a \cdot k$$

qed.

Ausblick auf den zweiten Teil

Im zweiten Teil wollen wir auf weitere Anwendungen der Kolmogoroff-Komplexität eingehen. Dazu zählen Themen wie *Primzahlen*, in diesem Zusammenhang wollen wir den *schwachen Primzahlsatz* beweisen, dann wollen wir uns mit *Schaltkreiskomplexität* befassen, mit *Turingmaschinen* und *Sprachen*.

Bei allen genannten Themen lassen sich untere Schranken mithilfe der Kolmogoroff-Komplexität zeigen.