Karlsruher Institut für Technologie

Stefan Walzer
ITI Prof. Sanders
Wintersemester 2023/2024

Übungsblatt 3 – Important Random Variables and How to Sample Them

Randomisierte Algorithmik

Aufgabe 1 – Ber(1/3) aus Ber(1/2)

Entwerfe einen Algorithmus der, gegeben eine Folge $B_1, B_2, \ldots \sim \text{Ber}(1/2)$ von Zufallsbits in erwarteter Zeit $O(1)$ ein Sample $B \sim \text{Ber}(1/3)$ berechnet.

Lösung 1

Wir interpretieren $B_1, B_2, B_3, \ldots$ als Binärdarstellung einer Zahl $U = (0.B_1B_2B_3\ldots)_2$. Für diese gilt dann $U \sim \mathcal{U}([0,1])$. Wir definieren $B := \mathbb{1}_{U<1/3}$. Daraus folgt unmittelbar $B \sim \text{Ber}(1/3)$ wie gewünscht. Die Binärdarstellung von $1/3$ ist $1/3 = (0.01010101\ldots)_2$. So ergibt sich folgender Algorithmus, der immer die nächsten zwei Ziffern der Binärdarstellung von U hernimmt und schaut ob diese eine Entscheidung zulassen:

for $i = 1$ **to** ∞ **do**
  $(x, y) \leftarrow (B_{2i-1}, B_{2i})$
  if $(x, y) = (0, 0)$ **then**
    return 1
  else if $(x, y) = (1, 0)$ *or* $(x, y) = (1, 1)$ **then**
    return 0

Jede Runde führt mit Wahrscheinlichkeit 3/4 zu einer Entscheidung. Wenn R die Anzahl von Runden ist, dann gilt

$$\mathbb{E}[R] \overset{\text{TSF}}{=} \sum_{i \in \mathbb{N}_0} \Pr[R > i] = \sum_{i \in \mathbb{N}_0} \frac{1}{4^i} = \frac{1}{1 - \frac{1}{4}} = \frac{4}{3}.$$

Bemerkung: In der Praxis würde man das nicht so machen, sondern wie in der nächsten Aufgabe davon ausgehen, dass man $U \sim \mathcal{U}([0,1])$ direkt sampeln kann (so genau wie es Floating-Point Zahlen eben hergeben).
Bemerkung: Die Laufzeit ist unbeschränkt und das ist auch unvermeidbar. Das können wir durch einen Widerspruchsbeweis zeigen. Angenommen ein Algorithmus kommt für ein festes $C \in \mathbb{N}_0$ stets mit dem Präfix $B_1, \ldots, B_C$ der Zufallsfolge aus. Dann ist seine Ausgabe B eine Zufallsvariable $B : \Omega \to \{0,1\}$, die auf dem Wahrscheinlichkeitsraum $\Omega = \{0,1\}^C$ mit Gleichverteilung definiert ist (dessen Ergebnisse sind alle Möglichkeiten für $B_1, \ldots, B_C$). Jedes Ergebnis hat eine Wahrscheinlichkeit von 2^{-C}. Jedes Ereignis – und damit auch das

Ereignis $\{B = 1\}$ ist eine Menge von Ergebnissen und hat damit eine Wahrscheinlichkeit, die ein ganzzahliges Vielfaches von 2^{-C} ist. Das steht im Widerspruch dazu, dass diese Wahrscheinlichkeit $1/3$ sein soll.

Aufgabe 2 – $\mathrm{Ber}(p)$ und $\mathcal{U}(\{1,\dots,n\})$ aus $\mathcal{U}([0,1])$

Wir nehmen nun ein Maschinenmodell an in dem reelle Zahlen verarbeitet werden können und in dem wir $U \sim \mathcal{U}([0,1])$ sampeln können. Zeige, dass wir auch $B \sim \mathrm{Ber}(p)$ für $p \in [0,1]$ und $X \sim \mathcal{U}(\{1,\dots,n\})$ für $n \in \mathbb{N}$ sampeln können.

Hinweis: Für den Rest des Blattes und des Semesters nehmen wir Voraussetzung und Ergebnis dieser Aufgabe als gegeben an.

Lösung 2

Gegeben $U \sim \mathcal{U}([0,1])$ definieren wir $B := \mathbb{1}_{U<p}$ und $X := \lceil U \cdot n \rceil$. Dann gilt wie gewünscht:

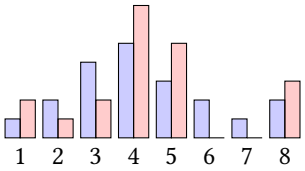
$$\Pr[B = 1] = \Pr[U < p] = p \text{ sowie}$$
$$\text{für } 1 \le i \le n\text{: } \Pr[X = i] = \Pr\left[U \cdot n \in (i-1, i]\right] = \Pr\left[U \in (\tfrac{i-1}{n}, \tfrac{i}{n}]\right] = \tfrac{1}{n}.$$

Bemerkung: Streng genommen ist für $U \sim \mathcal{U}([0,1])$ das Ergebnis $U = 0$ möglich was zu $X = 0$ führt, obwohl eigentlich $X \in \{1,\dots,n\}$ gelten soll. Das passiert allerdings mit Wahrscheinlichkeit 0. Man kann das reparieren indem man zum Beispiel sagt, dass 0 einfach auch zu 1 aufgerundet wird. Oder man kehrt solche nervigen Spitzfindigkeiten einfach unter den Teppich.

Aufgabe 3 – Rejection Sampling Allgemein

Seien $\mathcal{D}_1$ und $\mathcal{D}_2$ Verteilungen auf einer endlichen Menge D. Wir nehmen an:

1. Wir können in Zeit $O(1)$ ein Sample $X \sim \mathcal{D}_1$ erzeugen.
2. Wir können wir ein gegebenes $x \in D$ in Zeit $O(1)$ die Wahrscheinlichkeiten $p_1(x) := \Pr_{X\sim\mathcal{D}_2}[X = x]$ und $p_2(x) := \Pr_{X\sim\mathcal{D}_1}[X = x]$ berechnen.
3. Es existiert $C > 0$ sodass für alle $x \in D$ gilt:
$$p_2(x) \le C \cdot p_1(x).$$



Mögliches Histogramm für $\mathcal{D}_1$ (blau, links) und $\mathcal{D}_2$ (rot, rechts). Es gilt stets "rot $\le 2 \cdot$ blau", also ist Bedingung (3) mit $C = 2$ erfüllt.

Entwerfe einen Algorithmus, der in erwarteter Zeit $O(C)$ ein Sample $Y \sim \mathcal{D}_2$ erzeugt.

Lösung 3

Der Algorithmus geht folgendermaßen:

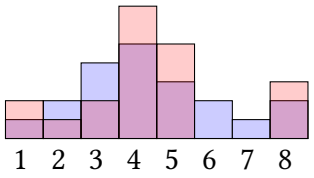
while True **do**
  sample $X \sim \mathcal{D}_1$ // $O(1)$
  sample $U \sim \mathcal{U}([0,1])$ // $O(1)$
  if $U < \frac{p_2(X)}{C \cdot p_1(X)}$ **then** // $O(1)$
    return X

Wir überprüfen zunächst seine Korrektheit. Wichtig ist hierbei, dass $\frac{p_2(X)}{p_1(X) \cdot C} \in [0,1]$ gilt, was aus Vorraussetzung (3) folgt. Sei Y das Ergebnis eines Schleifendurchgangs, das heißt $Y = X$ falls X zurückgegeben wird und $Y = \perp$ falls die Schleife ohne Ergebnis endet. Dann gilt für $x \in D$:

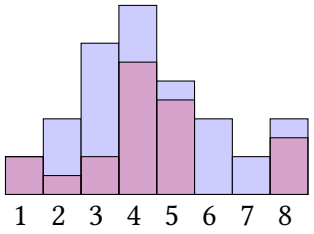
$$\Pr[Y = x] = \Pr[X = x] \cdot \Pr\left[U < \frac{p_2(x)}{C \cdot p_1(x)}\right] = p_1(x) \cdot \frac{p_2(x)}{C \cdot p_1(x)} = \frac{p_2(x)}{C}.$$

Insbesondere ist die Wahrscheinlichkeit $\Pr[Y = x]$ *proportional* zu $p_2(x)$. Damit gilt $\Pr[Y = x \mid Y \neq \perp] = p_2(x)$. In anderen Worten: Wenn etwas zurückgegeben wird, dann liegen die Wahrscheinlichkeiten von $\mathcal{D}_2$ zugrunde, wie gewünscht. Die Erfolgswahrscheinlichkeit einer Runde ist $\Pr[Y \neq \perp] = \sum_{x \in D} \Pr[Y = x] = \frac{1}{C}$. Damit sind im Erwartungswert C Runden nötig bis ein Erfolg eintritt.

Intuition: Man kann sich den Algorithmus auch verbildlichen. Wenn wir die Balken der Histogramme übereinander malen ergibt sich folgendes:



Wenn wir nun die blauen Balken hochskalieren, dann erhalten wir ein Bild in dem die roten Balken stets kleiner sind als die blauen.



Um aus der roten Verteilung zu ziehen genügt es, einen zufälligen roten Punkt zu ziehen und den Index des Balkens zurückzugeben in dem er liegt. Dazu ziehen wir einen zufälligen blauen Punkt (in der Darstellung: blau oder lila ist) und behalten ihn, falls er rot ist.

Im Algorithmus wird ein zufälliger blauer Punkt gezogen indem wir einen Balken X wählen und dann eine zufällige Höhe $U \cdot C \cdot p_1(X)$ auf diesem Balken. Das wird dann mit der Höhe des roten Balkens verglichen.

Aufgabe 4 – $G \sim \mathrm{Geom}_1(p)$ mit Inverse Transform Sampling

Entwerfe einen Algorithmus der für gegebenes $p \in (0, 1]$ eine Zufallsvariable $G \sim \mathrm{Geom}_1(p)$ in Zeit $O(1)$ sampelt.

Lösung 4

Die kumulative Verteilungsfunktion von G ist:

$$F_G(i) = \Pr[G \leq i] = 1 - (1-p)^i.$$

Für die (verallgemeinerte) Inverse gilt dann für $u \in (0, 1]$:

$$\begin{aligned} F_G^{-1}(u) &:= \min\{i \in \mathbb{N}_0 \mid F_G(i) \geq u\} = \min\{i \in \mathbb{N}_0 \mid 1 - (1-p)^i \geq u\} \\ &= \min\left\{i \in \mathbb{N}_0 \mid i \geq \frac{\log(1-u)}{\log(1-p)}\right\} = \left\lceil \frac{\log(1-u)}{\log(1-p)} \right\rceil. \end{aligned}$$

Nach der Methode sollte also folgendes funktionieren:

sample $U \sim \mathcal{U}([0, 1])$
return $G = \left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil$

Dass alles geklappt hat können wir auch nochmal überprüfen in dem wir nachrechnen, dass das G aus dem Algorithmus die gewünschte Verteilungsfunktion hat:

$$\begin{aligned} \Pr[G \leq i] &= \Pr\left[\left\lceil \frac{\log(1-U)}{\log(1-p)} \right\rceil \leq i\right] = \Pr\left[\frac{\log(1-U)}{\log(1-p)} \leq i\right] = \Pr[\log(1-U) \geq i \log(1-p)] \\ &= \Pr[1 - U \geq (1-p)^i] = \Pr[U \leq 1 - (1-p)^i] = 1 - (1-p)^i. \end{aligned}$$

Aufgabe 5 – Ziehen ohne Zurücklegen

Wir betrachten Algorithmen, die für $k, n \in \mathbb{N}$ mit $0 \leq k \leq n/2$ eine Menge $S \subseteq [n]$ der Größe k berechnen, die aus allen Teilmengen von $[n]$ der Größe k uniform zufällig gewählt ist.

(a) Warum dürfen wir $k \leq n/2$ ohne Beschränkung der Allgemeinheit annehmen?

(b) Beschreibe einen Algorithmus, der eine erwartete Laufzeit von $O(k \log k)$ hat.
Hinweis: Rejection Sampling und Suchbaum.

(c) **Bonus:** Entwerfe einen Algorithmus, der eine Worst-Case Laufzeit von $O(k \log k)$ hat.

(d) **Bonus:** Recherchiere, wie man eine Worst-Case Laufzeit von $O(k)$ erreichen kann:

`https://stackoverflow.com/a/67850443`

Lösung 5

(a) $S \subseteq [n]$ ist eine zufällige Menge der Größe k genau dann, wenn $[n] \setminus S$ eine zufällige Menge der Größe $n - k$ ist.

(b) Von der Idee her zieht der Algorithmus *mit* Zurücklegen, speichert sich die Ergebnisse in einem Suchbaum und ignoriert Ergebnisse, die schonmal aufgetreten sind. Das macht er bis er k verschiedene Ergebnisse gesehen hat. Das ist eine Form von Rejection Sampling und es ist ziemlich klar, dass das korrekt ist.

```
Algorithm ZieheOhneZurücklegen(n, k):
    S ← ∅ // als Suchbaum
    while |S| < k do
        sample X ~ U({1, ..., n})
        if X ∉ S then
            S ← S ∪ {X}
    return S
```

Nach Annahme aus (a) und der Schleifenbedingung gilt am Anfang jedes Schleifendurchlaufs $|S| < k \leq n/2$. Damit ist die Wahrscheinlichkeit, etwas zu ziehen, was wir schon haben, stets höchstens $1/2$. Daraus folgt, dass die Anzahl F erfolgloser Schleifendurchläufe erwartet höchstens der Anzahl erfolgreicher Schleifendurchläufe entspricht, also gilt $\mathbb{E}[F] \leq k$.

Die Gesamtlaufzeit beträgt $T = (k + F) \cdot O(\log k)$ weil es $k + F$ Schleifendurchläufe gibt und jede mit Suchbaumoperationen in $O(\log k)$ durchführbar ist. Somit ergibt sich $\mathbb{E}[T] = O(k \log k)$.

(c) Die Idee ist, dass wir die Menge der noch ziehbaren Elemente explizit verwalten. Im Folgenden wird Array$[1..n]$ verwendet, was stets eine Permutation der Menge $\{1, \ldots, n\}$ enthält. Am Anfang von Schleifendurchlauf i enthält Array$[1..i-1]$ die bereits gezogenen Elemente und Array$[i..n]$ die noch ziehbaren Elemente.

```
Algorithm ZieheOhneZurücklegen(n, k):
    Array = [1, 2, ..., n] // alles noch ziehbar
    for i = 1 to k do
        sample j ~ U({i, ..., n})
        swap Array[j] and Array[i] // tut nichts falls j = i
    return Array[1..k]
```

Dummerweise ergibt sich so eine Laufzeit von $O(n + k)$ wegen der Initialisierung des Arrays. Das lässt sich aber reparieren. Offenbar kann stets für höchstens $2k$ Indizes i gelten, dass Array$[i] \neq i$ ist. Es genügt also sich diese Ausnahmepositionen in einem Suchbaum zu speichern. Dann erhält man Laufzeit $O(k \log k)$.

(d) Siehe `https://github.com/ciphergoth/sansreplace/blob/master/cardchoose.md`