5 Monte Carlo Algorithmen

Bisher haben wir randomisierte Algorithmen kennengelernt, deren Laufzeit vom Zufall abhängt. Das Ergebnis des Algorithmus war jedoch immer korrekt. Solche Algorithmen nennt man auch *Las Vegas Algorithmen*. In diesem Kapitel werden wir einen Algorithmus kennenlernen, der nur mit hoher Wahrscheinlichkeit das korrekte Ergebnis berechnet. Für diesen Algorithmus können wir eine Laufzeitschranke angeben, die nicht vom Zufall abhängt. Wir nennen solche Algorithmen auch *Monte Carlo Algorithmen*.

5.1 Minimale Schnitte in Graphen

Wir werden nun einen Monte Carlo Algorithmus kennenlernen, der einen *minimalen Schnitt* in einem Graph mit ganzzahligen Kantengewichten berechnet. Zur Vereinfachung der Präsentation des Algorithmus werden wir den Graph jedoch als Multigraph darstellen, d.h. eine Kante mit Gewicht k wird durch k einzelne Kanten dargestellt. Wir nehmen weiterhin an, dass der Graph keine Selbstloops hat. Ein Schnitt in einem Graph ist definiert wie folgt:

Definition 4 *Ein Schnitt in einem zusammenhängenden Multigraph ist eine Menge von Kanten C, so dass der Multigraph nach dem Entfernen von C nicht mehr zusammenhängend ist.*

5.1.1 Ein einfacher Algorithmus

Als erstes werden wir nun einen sehr einfachen Algorithmus kennenlernen, der einen Schnitt in einem Multigraphen $G = (V, E)$ berechnet. Wir werden dann zeigen, dass der berechnete Schnitt mit Wahrscheinlichkeit $1/n^2$ ein minimaler Schnitt ist.

Der Algorithmus wählt in jedem Schritt eine zufällige Kante (u, v) aus dem aktuellen Multigraph aus. Danach wird diese Kante *kontrahiert*. Dabei werden die Knoten u und v durch einen neuen Knoten w ersetzt. Jede Kante der Form (x, u) für $x \neq v$ wird durch eine Kante (x, w) ersetzt. Jede Kante der Form (x, v) für $x \neq u$ ebenso. Kanten zwischen u und v fallen weg. Wird eine Kante kontrahiert, so liegen die beiden zugehörigen Knoten nach dem Entfernen des berechneten Schnitts in derselben Zusammenhangskomponente. Der neue Knoten repräsentiert die beiden alten Knoten.

Sind nur noch zwei Knoten übrig, so gibt der Algorithmus den Schnitt zwischen den Knotenmengen im Ursprungsgraph aus, die durch die übriggebliebenen Knoten repräsentiert werden.

CONTRACT(G)
H = G;
while H hat mehr als zwei Knoten **do**
    Wähle Kante (x, y) zufällig und gleichverteilt aus den Kanten von H
    Kontrahiere die Kante $(x; y)$ in H
Sei K die Menge der Kanten in H (zu diesem Zeitpunkt sind nur 2 Knoten übrig)
Gib die Kanten aus G aus, die K entsprechen

Das folgende Beispiel soll den Algorithmus veranschaulichen.

Die Laufzeit des Algorithmus CONTRACT beträgt (bei geschickter Implementierung) $O(n^2)$. Dabei benutzt man die Darstellung des Graphen durch eine Adjazenzmatrix. Der Wert der Matrix für zwei Knoten u, v gibt die Anzahl der Kanten an, die zwischen den beiden Knoten verlaufen. Um eine Kante schnell auszuwählen, halten wir außerdem noch die Summe der Einträge in jeder Zeile der Matrix aufrecht. Nun können wir in $O(n)$ Zeit eine Kante zufällig und gleichverteilt auswählen, indem wir zunächst die Summe der Einträge in der Matrix aufsummieren, dann eine zufällige ganze Zahl zwischen 1 und der Summe der Einträge wählen und die entsprechende Kante mit Hilfe der Summen der Zeilen in $O(n)$ Zeit bestimmen (Details: Übung).

5.1.2 Analyse des einfachen Algorithmus

Wir wollen nun zeigen, dass der von Algorithmus CONTRACT berechnete Schnitt mit Wahrscheinlichkeit mindestens $1/n^2$ ein minimaler Schnitt ist. Wir bezeichnen mit e_i die Kante, die in Iteration i des Algorithmus kontrahiert wird ($1 \le i \le n-2$) und mit H_i den Graph, der nach der Kontraktion von e_1 bis e_i entstanden ist. H_0 bezeichnet den Ausgangsgraph G und H_{n-2} den Graph, der am Ende des Algorithmus übrigbleibt.

Entfernt man eine Menge von Kanten C aus H_i und es entstehen dabei zwei oder mehr Zusammenhangskomponenten, so gilt dies auch, wenn man C aus $H_0 = G$ entfernt. Daher entspricht jeder Schnitt in einem der Graphen H_i einem Schnitt in G. Daraus folgt natürlich sofort, dass die Menge C, die der Algorithmus berechnet, ebenfalls einen Schnitt in G bildet. Außerdem gilt damit sofort, dass die Kardinalität eines minimalen Schnitts in H_i nicht kleiner wird als die eines minimalen Schnitts in G.

Satz 9 *Der von Algorithmus* CONTRACT *berechnete Schnitt ist mit Wahrscheinlichkeit* $1/n^2$ *ein minimaler Schnitt.*

Beweis: Sei K ein minimaler Schnitt in G. Wird im Verlauf des Algorithmus keine Kante von K kontrahiert, so ist der zurückgegebene Schnitt K. Wir werden zeigen, dass die Wahrscheinlichkeit dafür mindestens $1/n^2$ ist. Es folgt

$$\begin{aligned}
\mathbf{Pr}\big[C \text{ ist ein minimaler Schnitt}\big] &\geq \mathbf{Pr}\big[C = K\big] \\
&= \mathbf{Pr}\big[\bigwedge_{1\leq i\leq n-2} (e_i \notin K)\big] \\
&= \prod_{1\leq i\leq n-2} \mathbf{Pr}\big[e_i \notin K | \bigwedge_{1\leq j<i} (e_j \notin K)\big] \qquad (5.1)
\end{aligned}$$

Behauptung 1 *Für* $1 \leq i \leq n-2$ *gilt*

$$\mathbf{Pr}\big[e_i \in K | \bigwedge_{1\leq j<i} (e_j \notin K)\big] \leq \frac{2}{n-i+1} \ .$$

Beweis: Wir betrachten die Kontraktion von Kante e_i, die den Graph H_{i-1} in den Graph H_i überführt. Sei n_{i-1} die Anzahl der Knoten und m_{i-1} die Anzahl der Kanten in H_{i-1}. Da jede Kante mit derselben Wahrscheinlichkeit gewählt wird, gilt

$$\mathbf{Pr}\big[e_i \in K | \bigwedge_{1\leq j<i} (e_j \notin K)\big] = \frac{|K|}{m_{i-1}} \ .$$

Weiterhin gilt $m_{i-1} \geq |K| \cdot n_{i-1}/2$, weil es ansonsten einen Knoten mit Grad kleiner als $|K|$ gibt und seine inzidenten Kanten einen Schnitt mit Kardinalität $\leq |K|-1$ bilden würden (Widerspruch zur Minimalität von K). Also folgt $\frac{|K|}{m_{i-1}} \leq \frac{2}{n_{i-1}}$. Nun folgt das Lemma aus $n_{i-1} = n-i+1$. □

Wir benutzen nun Behauptung 1, um den Beweis von Satz 9 zu vervollständigen. Dazu setzen wir das Ergebnis der Behauptung in Gleichung 5.1 ein und erhalten

$$\begin{aligned}
\mathbf{Pr}\big[C \text{ ist ein minimaler Schnitt}\big] &\geq \prod_{1\leq i\leq n-2} \left(1 - \frac{2}{n-i+1}\right) \\
&= \prod_{1\leq i\leq n-2} \frac{n-i-1}{n-i+1} \\
&= \frac{2}{n\cdot(n-1)} \geq 1/n^2
\end{aligned}$$

□

5.1.3 Amplifikation

Wir haben nun einen Algorithmus entwickelt, der mit Wahrscheinlichkeit $1/n^2$ einen minimalen Schnitt berechnet. Um nun diese Wahrscheinlichkeit zu erhöhen, lassen wir den Algorithmus mehrmals laufen und geben den kleinsten der berechneten Schnitte zurück.

Wird in einem Durchlauf ein minimaler Schnitt berechnet, so haben wir einen minimalen Schnitt gefunden. Wenn wir den Algorithmus k-mal ausführen, so gilt

$$\mathbf{Pr}\left[\text{Kein Durchlauf berechnet einen minimalen Schnitt}\right] \leq (1 - 1/n^2)^k \ .$$

Für $k = n^2 \ln x$ folgt damit

$$\mathbf{Pr}\left[\text{Kein Durchlauf berechnet einen minimalen Schnitt}\right] \leq (1-1/n^2)^{n^2 \ln x} \leq e^{-lnx} = 1/x \ .$$

Wir können also die Fehlerwahrscheinlichkeit beliebig klein machen. Für konstante Fehlerwahrscheinlichkeit erhalten wir auf diese Weise einen Algorithmus mit Laufzeit $O(n^4)$, da jeder Durchlauf $O(n^2)$ Zeit benötigt.

5.1.4 Ein schnellerer Algorithmus

Wir wollen nun den Algorithmus so modifizieren, dass er mit größerer Wahrscheinlichkeit einen minimalen Schnitt berechnet. Dabei wollen wir die Laufzeit möglichst wenig erhöhen. Unsere Verbesserung basiert auf folgender Beobachtung: Wenn wir den Algorithmus Contract laufen lassen, bis ein Graph mit t Knoten entstanden ist, dann ist die Wahrscheinlichkeit, dass wir noch keine Kante eines festen minimalen Schnitts K genommen haben, mindestens (folgt aus dem Beweis von Satz 9)

$$\prod_{1 \leq i \leq n-t} (1 - \frac{2}{n-i+1}) = \prod_{1 \leq i \leq n-t} \frac{n-i-1}{n-i+1} = \frac{t \cdot (t-1)}{n \cdot (n-1)} \ .$$

Also ist die Wahrscheinlichkeit, dass der Algorithmus bei den ersten $n/2$ Kontraktionen keine Kante aus K nimmt, mindestens $\frac{n/2 \cdot (n/2-1)}{n(n-1)} \approx 1/4$. Obwohl unser Algorithmus nur mit Wahrscheinlichkeit $1/n^2$ einen minimalen Schnitt berechnet, nimmt er während der ersten $n/2$ Kontraktionen nur mit Wahrscheinlichkeit $1/4$ eine Kante des minimalen Schnitts. Also entsteht die hohe Fehlerwahrscheinlichkeit in erster Linie durch die letzten Kontraktionen des Algorithmus. Andererseits ist aber der Algorithmus auf kleinen Graphen viel schneller als auf großen.

Die Grundidee. Je kleiner der Graph im Verlauf des Algorithmus wird, umso mehr Durchläufe führen wir für den aktuellen Graph durch. Immer dann, wenn wir mehrere Kopien laufen lassen, wählen wir die Kopie, die den kleinsten Schnitt berechnet hat. Durch viele Kopien bei kleinen Graphen können wir die erhöhte Fehlerwahrscheinlichkeit ausgleichen.

FastCut(G)
  if $n \leq 6$ **then** berechne minimalen Schnitt 'brute force'
  else
    $t = \lceil 1 + n/\sqrt{2} \rceil$
    benutze Algorithmus Contract um unabhängig voneinander zwei Graphen H_A, H_B mit t Knoten zu berechnen
    FastCut(H_A);
    FastCut(H_B);
    gibt den kleineren der beiden berechneten Schnitte zurück

Die Laufzeit des Algorithmus $T(n)$ wird durch die Rekursion

$$T(n) = 2T\big(\lceil 1 + \frac{n}{\sqrt{2}} \rceil\big) + O(n^2)$$

beschrieben. Mit Hilfe des Master Theorems erhält man $T(n) = O(n^2 \log n)$.

Satz 10 *Algorithmus* FastCut *gibt mit Wahrscheinlichkeit* $\Omega(1/\log n)$ *einen minimalen Schnitt zurück.*

Beweis: Wir halten einen minimalen Schnitt K fest. Wir haben bereits gesehen, dass die Wahrscheinlichkeit, dass bei einer Kontraktion bis auf t Knoten keine Kante aus K gewählt wurde, mindestens

$$\frac{t \cdot (t-1)}{n \cdot (n-1)} = \frac{\lceil 1 + n/\sqrt{2} \rceil}{n} \cdot \frac{\lceil n/\sqrt{2} \rceil}{n-1} \geq 1/2$$

ist. Wir bezeichnen nun mit $P(G)$ die Wahrscheinlichkeit, dass FastCut(H) einen minimalen Schnitt berechnet. Wir wollen nun eine Rekursion für $P(G)$ aufstellen. Die Wahrscheinlichkeit, dass FastCut auf dem Rechenweg über H_A einen minimalen Schnitt berechnet, ist mindestens $1/2P(H_A)$. Gleiches gilt für den Rechenweg über H_B. Damit ist die Wahrscheinlichkeit, dass einer der beiden Wege zu einem minimalen Schnitt führt,

$$P(G) \geq 1 - \big(1 - \frac{P(H_A)}{2}\big) \cdot \big(1 - \frac{P(H_B)}{2}\big) \ . \tag{5.2}$$

Wir definieren nun die Höhe des Rekursionsbaums als die Anzahl Kanten auf einem kürzesten Weg zu einem Blatt, d.h. Blätter haben Tiefe 0. Sei $p(k)$ die Wahrscheinlichkeit, dass der Algorithmus FastCut einen minimalen Schnitt berechnet, wenn der Rekursionsbaum Tiefe k hat (die Tiefe des Rekursionsbaums hängt natürlich nur von der Anzahl der Knoten im Baum ab und nicht vom Verlauf des Algorithmus). Aus der Ungleichung 5.2 folgt

$$p(k) \geq 1 - \big(1 - \frac{p(k-1)}{2}\big)^2$$

für $k \geq 1$ und mit $p(0) = 1$. Wir wollen nun zeigen, dass $p(k) \geq \frac{1}{k+1}$ gilt. Der Induktionanfang gilt wegen $p(0) = 1$. Wir nehmen an, dass die Behauptung für $k-1$ gilt.

$$\begin{aligned}
p(k) &\geq 1 - \left(1 - \frac{p(k-1)}{2}\right)^2 \\
&\geq 1 - \left(1 - \frac{1}{2k}\right)^2 \\
&= \frac{1}{k} - \frac{1}{4k^2} \\
&= \left(\frac{(4k-1)(k-1)}{4k^2}\right) \cdot \left(\frac{1}{k+1}\right) \\
&= \left(\frac{4k^2 + 4k - k - 1}{4k^2}\right) \cdot \left(\frac{1}{k+1}\right) \\
&\geq \frac{1}{k+1}
\end{aligned}$$

Da die Rekursiontiefe für einen Graphen mit n Knoten $O(\log n)$ ist, folgt Satz 10. □

5.2 Der Miller Rabin Primzahltest

Als nächstes werden wir einen Monte-Carlo Algorithmus entwickeln, mit dessen Hilfe man testen kann, ob eine gegebene Zahl eine Primzahl ist. Die Laufzeit des Algorithmus ist polylogarithmisch in der Größe der Zahl, d.h. polynomiell in der Eingabegröße (bei binärer Kodierung). Man kann also mit Hilfe dieses Algorithmus testen, ob eine Zahl eine Primzahl ist. Ist eine Zahl keine Primzahl, so findet der Algorithmus jedoch keine Zerlegung der Zahl in kleinere Faktoren, er bemerkt lediglich, dass die Zahl zusammengesetzt sein muss.

Wir werden zunächst ohne Beweis einige Sätze aus der Zahlentheorie wiedergeben, die wir zur Analyse des Miller-Rabin Primzahltests benötigen.

Satz 11 (Chinesischer Restsatz.) *Seien $m_1, \ldots, m_n$ paarweise teilerfremde positive ganze Zahlen, dann existiert für jedes Tupel ganzer Zahlen $a_1, \ldots, a_n$ eine ganze Zahl r, die die folgende simultane Kongruenz erfüllt:*

$$r = a_i \mod m_i \text{ für } i = 1, \ldots, n$$

Alle Lösungen dieser Kongruenz sind kongruent modulo $M = m_1 \cdot m_2 \cdot m_3 \cdots m_n$.

Wir benutzen im Folgenden die Bezeichnungen $\mathbb{Z}_p = \{0, \ldots, p-1\}$, $\mathbb{Z}_p^* = \{x \in \mathbb{Z}_p | ggT(x, p) = 1\}$. Falls p prim ist, so gilt $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$.

Satz 12 (Kleiner Satz von Fermat.) *Sei p prim und $a \in \mathbb{Z}_P*$. Dann gilt*

$$a^{p-1} = 1 \mod p \ .$$

Erste Idee für Primzahltest (Fermat-Test). Wir wollen einen Test entwickeln, ob eine Zahl p eine Primzahl ist. Eine sehr einfache Idee ist, einfach eine beliebige andere Zahl a zufällig zu wählen und mit Hilfe des Euklidischen Algorithmus zu testen, ob $ggT(a,p) \neq 1$ ist. Ist dies der Fall, so haben wir einen Teiler gefunden. Wenn wir diesen Test häufig genug wiederholen und nie ein Teiler gefunden wird, dann können wir irgendwann mit hoher Wahrscheinlichkeit sagen, dass kein solcher Teiler existiert. Leider gibt es natürlich auch Zahlen, die das Produkt zweier Primzahlen sind. In diesem Fall würden wir $\Omega(p)$ Wiederholungen unseres Tests benötigen, um einen solchen Teiler mit konstanter Wahrscheinlichkeit zu finden. Wir könnten also genausogut alle Teiler von 2 bis p ausprobieren. Dieser Test alleine ist also zu schwach.

Aber was passiert, wenn wir ihn mit dem kleinen Satz von Fermat kombinieren?

FermatTest(p)
  Wähle $a \in \mathbb{Z}_p \setminus \{0,1\}$ zufällig und gleichverteilt.
  if $ggT(a,p) \neq 1$ **then output** zusammengesetzt
  if $a^{p-1} \neq 1 \mod p$ **then output** zusammengesetzt
  output prim

Zunächst einmal wollen wir die Laufzeit dieses Algorithmus analysieren. Den ggT zweier Zahlen a, p mit $a \leq p$ kann man in $O(\log^2 p)$ Zeit mit dem Euklidischen Algorithmus berechnen. Die Zahl $a^{p-1} \mod p$ kann man durch wiederholtes Quadrieren schnell berechnen. Dazu stellen wir $p-1$ in Binärdarstellung dar, d.h. $p-1 = \sum_{i=0}^{\lfloor \log p \rfloor} p_i \cdot 2^i$. Dann gilt:

$$a^{p-1} = \prod_{i:p_i \neq 0} a^{2^i} \mod p \ .$$

Wenn wir also $a^{2^i} \mod p$ schnell berechnen können, dann können wir auch schnell $a^{p-1} \mod p$ berechnen. $a^{2^i} \mod p$ können wir aber einfach aus $b = a^{2^{i-1}} \mod p$ berechnen, indem wir $b^2 \mod p$ ausrechnen. Insgesamt benötigen wir dann $O(\log^3)$ Zeit ($O(\log^2 p)$ für jede der $O(\log p)$ Multiplikationen).

Leider gibt es Zahlen, die die dritte Zeile des Fermat-Tests für alle $a \in \mathbb{Z}_p^*$ bestehen und die aus zwei Primzahlen zusammengesetzt sind. Es gibt also Fälle, in denen wir mit dem Fermat-Test nichts gegenüber unserem einfachen Test gewinnen.

Also brauchen wir noch ein weiteres Kriterium, um einen guten Primzahltest zu entwickeln. Dazu werden wir benutzen, dass die Zahl 1 genau zwei Quadratwurzeln modulo jeder Primzahl p hat, nämlich 1 und -1. Für viele zusammengesetze Zahlen gilt aber, dass die Zahl 1 vier oder mehr Quadratwurzeln hat. Insbesondere haben die Zahlen, für die der Fermat-Test nicht funktioniert, diese Eigenschaft. Wenn nun eine Zahl den Fermat-Test für a besteht, dann suchen wir eine Quadratwurzel von a und bestimmen, ob diese Quadratwurzel 1 oder -1 ist. Wenn dies nicht der Fall ist, dann wissen wir, dass die Zahl nicht prim ist.

Wenn p den Fermat-Test mit a besteht, dann gilt $a^{p-1} = 1 \mod p$. Also ist $a^{\frac{p-1}{2}} \mod p$ eine Quadratwurzel von 1. Ist dieser Wert wieder 1, dann halbieren wir den Exponenten solange bis entweder der Exponent keine ganze Zahl mehr ist oder bis wir eine von 1 verschiedene Zahl erhalten. Wir formalisieren nun unser Kriterium und geben danach den Algorithmus und seine formale Analyse an.

Lemma 12 *Falls* $a \neq +-1 \mod p$ *und* $a^2 = 1 \mod p$*, dann ist* p *nicht prim.*

Beweis: Es gilt

$$a^2 - 1 = (a+1) \cdot (a-1) = 0 \mod p$$

und damit

$$(a+1) \cdot (a-1) = c \cdot p$$

für ein $c \in \mathbb{N}$. Wegen $a \neq +-1 \mod p$ gilt $0 < a+1 < p$ und $0 < a-1 < p$. Also muß p zusammengesetzt sein. □

Dieses Kriterium wird im Primzahltest von Miller und Rabin verwendet.

MillerRabinTest(p)
  if p ist gerade **then**
    if $p = 2$ **then output** prim
    else output zusammengesetzt
  Wähle $a_1, \ldots, a_k \in \mathbb{Z}_p^*$ zufällig und gleichverteilt
  for $i = 1$ **to** k **do**
    Sei $p - 1 = 2^\ell \cdot s$ mit s ungerade
    $j = 0$
    while $a_i^{s \cdot 2^j} \neq 1 \mod p$ **do**
      $j = j + 1$
    if $j > 0$ und $a_i^{s \cdot 2^{j-1}} \neq -1 \mod p$ **then output** zusammengesetzt
  ouput prim

Wir zeigen zunächst, dass MillerRabinTest jede Primzahl akzeptiert.

Lemma 13 *Wenn* p *eine Primzahl ist, dann akzeptiert* MillerRabinTest *immer.*

Beweis: Sei p eine Primzahl. Ist $p = 2$, so akzeptiert der Test. Ansonsten ist p ungerade. Wegen des kleinen Satzes von Fermat gilt stets $a_i^{p-1} = 1 \mod p$. Nach Lemma 12 gilt, dass p nicht prim ist, wenn es eine Zahl a gibt mit $a \neq +-1 \mod p$ und $a^2 = 1 \mod p$. Falls MillerRabinTest in Zeile 10 akzeptiert, so liefert $a = a_i^{s \cdot 2^{j-1}}$ eine Zahl mit genau diesen Eigenschaften und p wäre nicht prim. Widerspruch. □

Nun betrachten wir den Fall, dass p zusammengesetzt ist.

Lemma 14 *Sei* $p = q \cdot r$ *zusammengesetzt, dann gilt*

$$\mathbf{Pr}\big[\text{MillerRabinTest } \textit{liefert Ausgabe prim}\big] \leq 2^{-k}$$

Beweis: Sei $a \in \mathbb{Z}_p^*$. Wir nennen a einen Zeugen, wenn a die Ausgabe 'zusammengesetzt' liefert. Ansonsten nennen wir a einen Nicht-Zeugen. Wir zeigen, dass mindestens die Hälfte aller $a \in \mathbb{Z}_p^*$ Zeugen sind. Daraus folgt dann sofort

$$\mathbf{Pr}\left[a \text{ ist Zeuge}\right] \geq 1/2 \ .$$

Wir konstruieren eine injektive Abbildung, die jedem Nicht-Zeugen einen Zeugen zuordnet. Sei a mit $a^{s \cdot 2^{j-1}}$ ein Nicht-Zeuge. Sei h ein Nicht-Zeuge mit maximalem j. Betrachte $t \in \mathbb{Z}_p$ mit

$$t = h \mod p \text{ und } t = 1 \mod r \ .$$

Nach dem Chinesischen Restsatz ist t eindeutig. Wegen $h^{s \cdot 2^{j-1}} = -1 \mod p$ und $p = qr$ folgt sofort

$$t^{s \cdot 2^{j-1}} = -1 \mod q \text{ und } t^{s \cdot 2^{j-1}} = 1 \mod r$$

Daraus folgt, dass $t^{s \cdot 2^{j-1}} \neq +-1 \mod p$ ist, denn sonst wäre entweder $t^{s \cdot 2^{j-1}} = 1 \mod q$ und $t^{s \cdot 2^{j-1}} = 1 \mod r$ bzw. $t^{s \cdot 2^{j-1}} = -1 \mod q$ und $t^{s \cdot 2^{j-1}} = -1 \mod r$. Ausserdem ist $t^{s \cdot 2^j} = 1 \mod p$. Daher ist t ein Zeuge.

Als nächstes zeigen wir, dass $dt \mod p$ zu jedem Nicht-Zeugen d einen Zeugen liefert. Da h ein Nicht-Zeuge mit maximalem j ist, gilt $d^{s \cdot 2^{j-1}} = +-1 \mod p$ und $d^{s \cdot 2^j} = 1 \mod p$. also ist $(dt)s \cdot 2^{j-1} \neq +-1 \mod p$ und $(dt)s \cdot 2^{j-1} = 1 \mod p$. damit ist $dt \mod p$ ein Zeuge.

Nun müssen wir noch die Injektivität unserer Zuordnung zeigen. Seien dazu d_1 und d_2 unterschiedliche Nicht-Zeugen. Annahme: Es gilt $td_1 \mod p = td_2 \mod p$. Dann folgt

$$d_1 = t \cdot t^{s \cdot 2^j - 1} d_1 \mod p = t \cdot t^{s \cdot 2^j - 1} d_2 \mod p = d_2 \ ,$$

da $t^{s \cdot 2^j} = 1 mod p$ und somit $t \cdot t^{s \cdot 2^j - 1} = 1 mod p$ gilt. Widerspruch zur Unterschiedlichkeit von d_1 und d_2!

Also ist die konstruierte Abbildung injektiv und es gibt daher mindestens genauso viele Zeugen wie Nicht-Zeugen. □