BACHELORARBEIT

Erzeugen von zufälligen Graphen und Spannbäumen

ausgeführt am

Institut für
Diskrete Mathematik und Geometrie
TU Wien

unter der Anleitung von

Benedikt Stufler

durch

Cederic Demoulin
Matrikelnummer: 11901874

Wien, am 10. Mai 2024

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Wien, am 10. Mai 2024

Cederic Demoulin

Inhaltsverzeichnis

1 Einleitung

Graphen und ihre Teilstruktur, die sogenannten Spannbäume, spielen eine fundamentale Rolle in der Informatik, Optimierung und verschiedenen Anwendungsgebieten davon. Die Fähigkeit, diese Strukturen zu analysieren, zu verstehen und effizient zu generieren, hat weitreichende Auswirkungen auf zahlreiche algorithmische Probleme und praxisrelevante Anwendungen. Diese Bachelorarbeit widmet sich Algorithmen zur zufälligen Generierung von Graphen und Spannbäumen.

Graphen bieten eine abstrakte Darstellung von Beziehungen und Verbindungen zwischen Objekten. In zahlreichen Disziplinen, einschließlich der Analyse von sozialen Netzwerken, Verkehrsplanung, Netzwerkdesign und Optimierung von Prozessen, sind Graphenmodelle unerlässlich.

Die Generierung von zufälligen Graphen und Spannbäumen ist ein interessantes Forschungsgebiet, da sie tiefe Einblicke in die Struktur und das Verhalten von Graphen ermöglicht. Zufallsgraphen dienen als Modelle für reale Netzwerke, in denen die genauen Verbindungen zwischen den Elementen nicht im Voraus bekannt sind. Die Entwicklung effizienter Algorithmen zur zufälligen Generierung von Graphen und Spannbäumen hat direkte Auswirkungen auf die Bewertung von Algorithmen in der durchschnittlichen Fallkomplexität und ermöglicht eine realistischere Simulation von Netzwerken.

Da Graphen eine große Rolle in Bereich Data Science spielen, und Python eine sehr beliebte Programmiersprache in diesem Feld ist, wird in dieser Arbeit die Implementierung von Algorithmen in Python gezeigt. Die Bibliothek `networkx` hat sich als die de facto Standardbibliothek für Graphen in Python etabliert. Mit ihr lassen sich sehr leicht zufällige Graphen generieren und analysieren. In dieser Arbeit werden die Algorithmen die `networkx` verwendet, um zufällige Erdős-Rényi Graphen zu generieren, analysiert und verglichen. In ihrer Arbeit *Fast Random Graph Generation* haben Nobari et al. [NLKB11] verschiedene Algorithmen zur zufälligen Generierung von ebendiesen Graphen in `C++` implementiert und verglichen. In dieser Arbeit werden 2 dieser Algorithmen in Python implementiert und mit denen von `networkx` verglichen.

Von zufälligen Graphen gehen wir zu zufälligen Spannbäumen von Graphen über. Dazu beschreiben wir den Groundskeeper Algorithmus und zeigen, dass der daraus resultierende Spannbaum gleichverteilt ist. Eine Implementierung in Python wird auch hier wieder gezeigt. Für reguläre Graphen wird ein Algorithmus vorgestellt, der wesentlich effizienter ist als der Groundskeeper Algorithmus. Die wichtigste Quelle für dieses Kapitel ist die Arbeit *The random walk construction of uniform spanning trees and uniform labelled trees* von D. J. Aldous [Ald90].

Im letzten Kapitel wird die Wahrscheinlichkeit betrachtet, dass ein Knoten in einem zufälligen Spannbaum eines Graphen ein Blatt ist. Dafür werden wir diese Wahrscheinlichkeit nach oben sowie nach unten abschätzen. Die Schranken stammen ebenfalls aus der Arbeit von Aldous. Zum Schluss wird die Schranke empirisch getestet, es wird der Groundskeeper

Algorithmus verwendet, um zufällige Spannbäume zu generieren und der Anteil der Bäume berechnet, in denen ein bestimmter Knoten ein Blatt ist. Die Schranken erweisen sich als äußert großzügig.

2 Grundlagen

Einige grundlegende Definitionen und Konzepte aus der Graphentheorie und der Statistik werden in diesem Kapitel vorgestellt.

2.1 Graphen

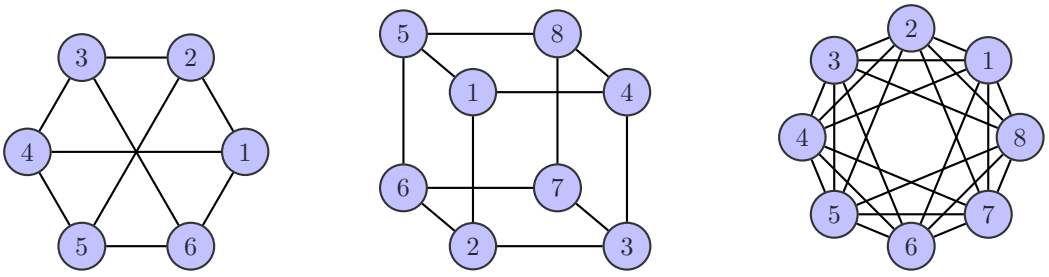
Definition 2.1.1 (einfacher Graph)**.** [Aig15] Ein einfacher Graph G ist ein Tupel (V, E) mit einer einer endlichen Menge V von Knoten und einer endlichen Menge E von Kanten. $E \subseteq \{\{v, w\} | v, w \in V, v \neq w\}$. Zwei Knoten v, w werden *benachbart* genannt, wenn sie durch eine Kante verbunden sind: $\exists e \in E : e = \{v, w\}$. Wir schreiben in diesem Fall auch $v \sim w$. Die Menge der benachbarten Knoten von v wird als $N(v)$ bezeichnet. Die Anzahl der Knoten in $N(v)$ wird als Grad von v bezeichnet. Einen Graphen $G' = (V', E')$ nennen wir *Teilgraph* von $G = (V, E)$, wenn $V' \subseteq V$ und $E' \subseteq E$ gilt.

Wir werden nur mit einfachen Graphen arbeiten weswegen wir statt von einem einfachen Graphen nur von einem Graphen sprechen werden.

Definition 2.1.2 (regulärer Graph)**.** Ein Graph G heißt regulär falls gilt:

$$\forall v \in V : |N(v)| = k$$

also alle Knoten den gleichen Grad haben. Man nennt einen Graphen k-regulär, wenn jeder Knoten Grad k hat.



(a) 3-regulärer Graph mit 6 Knoten (b) 3-regulärer Graph mit 8 Knoten (c) 6-regulärer Graph mit 8 Knoten

Abbildung 2.1: Beispiele für reguläre Graphen

Definition 2.1.3 (zusammenhängender Graph)**.** Ein Graph $G = (V, E)$ heißt zusammenhängend, wenn gilt:

$$\forall v, w \in V : \exists v_1, \ldots, v_n \in V : v_1 = v, v_n = w \wedge \forall i \in \{1, \ldots, n-1\} : \{v_i, v_{i+1}\} \in E$$

Es gibt zwischen jedem Knotenpaar einen Pfad.

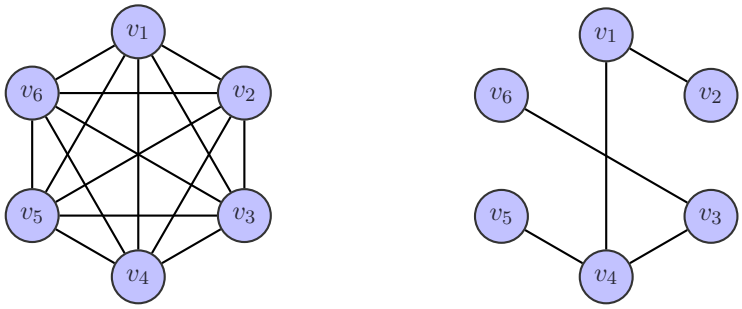
Definition 2.1.4 (vollständiger Graph)**.** Ein Graph G heißt vollständig, wenn gilt:

$$\forall v, w \in V : \{v, w\} \in E$$

also jeder Knoten mit jedem anderen Knoten verbunden ist.

Definition 2.1.5 (Baum)**.** Ein Baum ist ein Graph T, in dem es keine Folge $v_1, \ldots, v_n$ von Knoten gibt, sodass v_i und v_{i+1} für $i = 1, \ldots, n-1$ benachbart sind und $v_n = v_1$.

Definition 2.1.6 (Spannbaum)**.** Ein Teilgraph $T = (V_T, E_T)$ eines Graphen $G = (V, E)$ heißt Spannbaum, wenn T ein Baum ist und $V_T = V$ gilt.



(a) vollständiger Graph mit 6 Knoten (b) Spannbaum von 2.2a

Abbildung 2.2: vollständiger Graph K_6 und Spannbaum davon

2.2 Statistik

Definition 2.2.1 (stochastischer Prozess)**.** [MS05] Sei $(\Omega, \mathcal{F}, \mathbb{P})$ ein Wahrscheinlichkeitsraum und $(Z, \mathcal{Z})$ Messraum und T eine Indexmenge. Dann heißt eine Familie $X = (X_t)_{t \in T}$ messbarer Abbildungen

$$X_t : \Omega \to Z, t \in T$$

stochastischer Prozess (mit Zustandsraum Z).

Definition 2.2.2 (stationärer stochastischer Prozess)**.** [Kle13] Ein stochastischer Prozess $(X_t)_{t \in T}$ mit der Indexmenge T heißt stationär, wenn die Verteilung von $(X_{s+t})_{t \in T}$ nicht von der Verschiebung $s \in T$ abhängt, also wenn gilt

$$\mathbb{P}_X((X_{s+t})_{t \in T}) = \mathbb{P}_X((X_t)_{t \in T})$$

für alle $s \in T$

Definition 2.2.3 (Markov-Kette)**.** [Pri18] Ein stochastischer Prozess $(X_t)_{t \in \mathbb{N}_0}$ der nur Werte aus einem höchstens abzählbaren Zustandsraum Z annimmt, wird Markov-Kette genannt, wenn gilt:

$$\begin{aligned}\mathbb{P}(x_{t+1} = z_{t+1} | x_t = z_t, x_{t-1} = z_{t-1}, \ldots, x_0 = j_0) \\ = \mathbb{P}(x_{t+1} = z_{t+1} | x_t = z_t)\end{aligned}$$

für alle $t \in \mathbb{N}_0$ und alle $(z_0, \ldots, z_{t+1}) \in Z^{t+2}$. Diese Eigenschaft nennt man auch Gedächtnislosigkeit. Die Größen

$$p_{z,v}(t) = \mathbb{P}(x_{t+1} = v | x_t = z)$$

werden Übergangswahrscheinlichkeiten genannt. Sind diese nicht von t abhängig, so spricht man von stationären Übergangswahrscheinlichkeiten und einer homogenen Markov-Kette. Die Matrix $P(t)$ mit Einträgen $p_{z,v}(t)$ mit $z, v \in V$ ist dann die Übergangsmatrix der Markov-Kette. Da wir nur mit homogenen Markov-Ketten zu tun haben, werden wir P für die Übergangsmatrix schreiben.

Definition 2.2.4 (stationäre Verteilung)**.** [MS05] Sei $(X_t)_{t \in T}$ eine Markov-Kette mit Indexmenge T, Zustandsraum Z und Übergangsmatrix P. Eine Verteilung $\boldsymbol{\pi}$ heißt stationär, falls für alle $v \in Z$ gilt:

$$\sum_{z \in Z} \boldsymbol{\pi}(z) p_{z,v} = \boldsymbol{\pi}(v) \tag{2.1}$$

Fasst man $\boldsymbol{\pi}$ als Zeilenvektor auf, so kann man 2.1 auch in der Form

$$\boldsymbol{\pi} P = \boldsymbol{\pi}$$

beschreiben.

Definition 2.2.5 (erreichbar, kommunizierend)**.** [MS05] Sei $(X_t)_{t \in \mathbb{N}_0}$ eine Markov-Kette, mit Zustandsraum Z, Übergangsmatrix P und zwei Zuständen $i, j \in Z$. Der Zustand j heißt von i aus erreichbar, falls es einen Pfad von i nach j gibt. Das heißt,

$$\exists\, n \geq 1 : \quad \mathbb{P}_X(X_{t+n} = j | X_t = i) > 0 \qquad t \in \mathbb{N}_0.$$

Ist i auch von j aus erreichbar, so heißen i und j kommunizierend.

Definition 2.2.6 (irreduzibel)**.** [MS05] Ist $C \subset Z$ eine Teilmenge des Zustandsraums Z einer Markov-Kette und kommunizieren alle $i, j \in C$ miteinander, so heißt C irreduzibel. Ist Z irreduzibel, so heißt die Markov-Kette irreduzibel.

3 Erdős-Rényi Graphen in Python

In diesem Kapitel steht die Erzeugung von Erdős-Rényi Graphen im Mittelpunkt, wobei vier unterschiedliche Algorithmen untersucht werden. Die Python-Bibliothek `networkx`, welche die de-facto Standardbibliothek für das Erstellen, Manipulieren und Analysieren von Graphen in Python ist, beinhaltet die Algorithmen `gnp_random_graph` und `fast_gnp_random_graph`, welche näher beschreiben werden. In den Paper von Batagelj und Brandes [BB05], welches auch in `networkx` referenziert wird, werden diese und weitere Algorithmen zu Generierung von Graphen vorgestellt und verglichen. Zwei weitere Algorithmen, die in diesem Paper vorgestellt werden, sind `PreLogZER` und `PreZER`, welche in diesem Kapitel ebenfalls beschrieben werden. Zum Schluss werden die Algorithmen verglichen und die Ergebnisse diskutiert. [HSSC08]

gnp_random_graph

Die Funktion `gnp_random_graph` aus dem Modul `networkx.generators.random_graphs` mit den Parametern n, p und *seed* generiert einen Graphen mit n Knoten und einer Wahrscheinlichkeit p, dass eine Kante zwischen zwei Knoten existiert. Leicht vereinfacht, arbeitet die Funktion `gnp_random_graph` wie in Abbildung 3.1 dargestellt. Die Funktion geht alle Kanten des Graphen durch und entscheidet für jede Kante einzeln, ob sie im generierten Graphen existiert oder nicht. Dabei wird für jede Kante eine Zufallszahl zwischen 0 und 1 generiert. Ist diese Zufallszahl kleiner als p, so wird die Kante hinzugefügt, wenn nicht, existiert die Kante im generierten Graphen nicht. Die Anzahl der Schleifendurchläufe ist also abhängig von der Anzahl der Kanten in einem vollständigem Graphen mit n Knoten. Dies ist $\binom{n}{2} = \frac{n(n-1)}{2}$. Der Aufwand dieser Funktion ist somit $O(n^2)$. [HSSC08][Gil59]

fast_gnp_random_graph

Betrachtet man die Verteilung der Anzahl an Kanten in dem durch `gnp_random_graph` generierten Graphen, bemerkt man, das es sich hierbei um eine Binomialverteilung handelt. Wir bezeichnen mit E die Menge der Kanten in dem generierten Graphen, dann gilt:

$$\forall 0 \leq k \leq \binom{n}{2} : \mathbb{P}(|E| = k) = \binom{\binom{n}{2}}{k} p^k (1-p)^{\binom{n}{2}-k}$$

Nach dem Erwartungswert der Binomialverteilung gilt:

$$\mathbb{E}(|E|) = \sum_{k=0}^{\binom{n}{2}} p = p\binom{n}{2}$$

```python
import networkx as nx
import itertools
import random

def gnp_random_graph(n: int, p, seed=None) -> nx.Graph:
    """
    creates a random graph with n nodes and a probability p
    """
    random.seed(seed)
    edges = itertools.combinations(range(n), 2)

    G = nx.Graph()
    G.add_nodes_from(range(n))
    for e in edges:
        if random.random() < p:
            G.add_edge(*e)
    return G
```

Abbildung 3.1: `gnp_random_graph` in `networkx`

Dieser Wert ist also die erwartete Anzahl an Schleifendurchläufen, in denen wir eine Kante hinzufügen. Um die Anzahl der Schleifendurchläufe zu verringern, wird in `fast_gnp_random_graph` nicht für jede Kante eine Zufallszahl generiert, die über deren Existenz entscheidet, sondern die Zufallszahl entscheidet in wie vielen Schleifendurchläufen die nächste Kante hinzugefügt wird. Dazu betrachten wir für $k \geq 1$ die Wahrscheinlichkeit, mit der die nächste Kante erst wieder nach $k-1$ Schleifendurchläufen hinzugefügt wird. Dies ist eine einfache, geometrische Verteilung mit Parameter p.

$$\forall 0 \leq k : \mathbb{P}(\text{nächste Kante in } k \text{ Schleifendurchläufen}) = (1-p)^{k-1} \cdot p$$

Da wir weiterhin nur aus dem Interval $[0,1)$ samplen wollen, müssen wir das Interval $[0,1)$ auf die Wartezeiten k abbilden. Dazu verwenden wir die Funktion:

$$I(k) := \sum_{i=1}^{k} (1-p)^{k-1} \cdot p = 1 - (1-p)^k$$

Wir bilden $r \leftarrow [0,1)$ wie folgt auf $k \geq 1$ ab:

$$\begin{aligned} \kappa : [0,1) &\to \mathbb{N} \\ r &\mapsto \kappa(r) := \min\{k \in \mathbb{N} \mid r < I(k)\} \end{aligned} \tag{3.1}$$

Es ist zu zeigen, dass κ wohldefiniert und surjektiv ist.
Surjektivität: Sei $k \in \mathbb{N}$ beliebig, dann gilt:

$$\begin{aligned} \mathbb{P}(\kappa(r) = k) &= \mathbb{P}(I(k-1) \leq r < I(k)) \\ &= (1-p)^{k-1} \cdot p \end{aligned}$$

Somit ist κ surjektiv.
Wohldefiniertheit: Für die Wohldefiniertheit ist zu zeigen: $\forall r \in [0,1) : \exists! k \in \mathbb{N} : \kappa(r) = k$. Für die Existenz reicht es zu zeigen, dass $\lim_{k\to\infty} I(k) = 1$.

$$\begin{aligned}
\lim_{k\to\infty} I(k) &= \sum_{k=1}^{\infty}(1-p)^{(k-1)} \cdot p \\
&= p \cdot \sum_{k=0}^{\infty}(1-p)^{k} \\
&\overset{\text{geometische Reihe}}{=} p \cdot \frac{1}{1-(1-p)} \\
&= 1
\end{aligned}$$

Die Eindeutigkeit folgt aus der Monotonie von I. Wir müssen außerdem nachrechnen, dass κ tatsächlich r auf $\mathbb{N}$ nach der geometrischen Verteilung abbildet. Sei $k \in \mathbb{N}$ beliebig, dann gilt:

$$\begin{aligned}
\mathop{\mathbb{P}}_{r\leftarrow[0,1)}(\kappa(r) = k) = \mathop{\mathbb{P}}_{r\leftarrow[0,1)}(I(k-1) \le r < I(k)) &= 1-(1-p)^k - (1-(1-p)^{k-1}) \\
&= (1-p)^{k-1} \cdot p
\end{aligned}$$

Wir können k auch explizit durch r ausdrücken. Sei dazu $r \in [0,1)$ beliebig, dann gilt wegen 3.1:

$$\begin{array}{ccccc}
\kappa(r)-1 & \le & r & < & \kappa(r) \\
\sum_{i=1}^{\kappa(r)-1}(1-p)^{i-1}\cdot p & \le & r & < & \sum_{i=1}^{\kappa(r)}(1-p)^{i-1}\cdot p \\
1-(1-p)^{\kappa(r)-1} & \le & r & < & 1-(1-p)^{\kappa(r)} \\
(1-p)^{\kappa(r)-1} & \ge & 1-r & > & (1-p)^{\kappa(r)} \\
\kappa(r)-1 & \le & \log(1-r)/\log(1-p) & < & \kappa(r)
\end{array}$$

Somit gilt:

$$\kappa(r) = \left\lceil \frac{\log(1-r)}{\log(1-p)} \right\rceil$$

was mit `1+ int(math.log(1-r)/math.log(1-p))` in Python implementiert werden kann und in Abbildung 3.2 dargestellt ist. Durch die zwei Schleifen, die insgesamt einmal die Knoten und die Kanten durchgehen, ist der Algorithmus linear in der Anzahl der Knoten und Kanten, also $\mathcal{O}(n+m)$. [BB05]

Der Algorithmus geht in lexikografischer Ordnung die Kanten des vollständigen Graphen durch. Anders betrachtet, geht der Algorithmus die untere Dreiecksmatrix Reihe für Reihe durch. Es werden `int(lr/lp)` Kanten übersprungen und die nächste Kante hinzugefügt. Als Beispiel generieren wir einen Graphen mit 4 Knoten. Sei die Kantenwahrscheinlichkeit $p = 0.5$ und die Knotenmenge $V = \{v_1, v_2, v_3, v_4\}$. Dadurch ergibt sich für Zeile `12` in Algorithmus 3.2: `lp = log(0.5)`. Dann sind die Kanten in Lexikografischer Ordnung:

$$E = \{(v_1,v_0),(v_2,v_0),(v_2,v_1),(v_3,v_0),(v_3,v_1),(v_3,v_2)\}$$

```python
def fast_gnp_random_graph(n: int,p, seed=42) -> nx.Graph:
    """
    :param n: number of nodes
    """
    random.seed(seed)
    G = nx.empty_graph(n)
    if p <= 0 or p >= 1:
        return nx.gnp_random_graph(n, p, seed=seed)
    lp = log(1.0 - p)
    v = 1
    w = -1
    while v < n:
        lr = log(1.0 - random.random())
        k = 1 + int(lr / lp)
        w = w + k
        while w >= v and v < n:
            w = w - v
            v = v + 1
        if v < n:
            G.add_edge(v, w)
    return G
```

Abbildung 3.2: schnellerer Algorithmus für kleine p um einen random Graph in `networkx` zu generieren

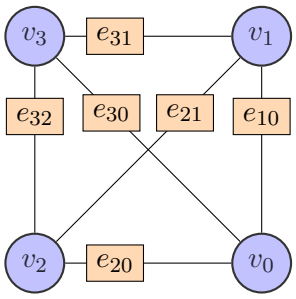


Abbildung 3.3: K_4 mit beschrifteten Kanten wie in `fast_gnp_random_graph`

Die Werte der Zufallsvariablen r im folgenden Beispiel wurden mit `random.random()` generiert.

- Initialisiere $v = 1$ und $w = -1$
- $r \leftarrow 0.084 \Rightarrow k = 1 + \left\lfloor \frac{\log(0.916)}{\log(0.5)} \right\rfloor = 1 \Rightarrow w = 0 \Rightarrow E = \{e_{10}\}$
- $r \leftarrow 0.51 \Rightarrow k = 1 + \left\lfloor \frac{\log(0.49)}{\log(0.5)} \right\rfloor = 2 \Rightarrow w = 2 \Rightarrow w = 1 \wedge v = 2 \Rightarrow E = \{e_{10}, e_{21}\}$

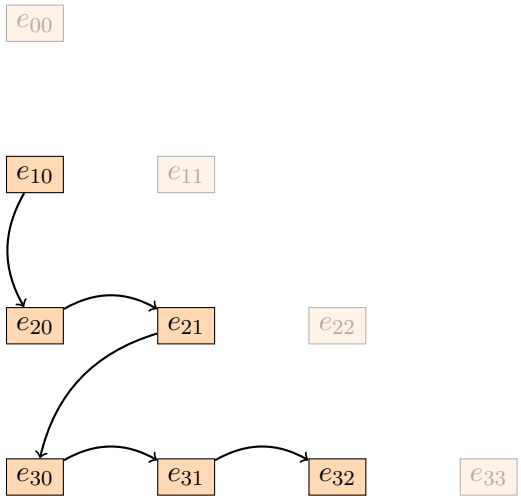


Abbildung 3.4: Reihenfolge der Kanten in Algorithmus 3.2 anhand der unteren Dreiecksadjazenzmatrix

- $r \leftarrow 0.864 \Rightarrow k = 1 + \left\lfloor \frac{\log(0.136)}{\log(0.5)} \right\rfloor = 3 \Rightarrow w = 4 \Rightarrow w = 2 \wedge v = 3 \Rightarrow E = \{e_{10}, e_{21}, e_{32}\}$
- Das Programm terminiert, da nach nächster Iteration $w > 2$ wäre und so $w \geq v(= 2)$ gelten würde wodurch v erhöht wird, weswegen dann $v \geq n = 3$ gilt, was die äußere `while`-Schleife terminieren lässt.

PreLogZER

Eine weitere Möglichkeit, die Laufzeit von `fast_gnp_random_graph` für gewisse Umstände zu verringern, kann durch das Berechnen der Logarithmen der Zufallszahlen im Voraus geschehen. Das ist von Vorteil, wenn die Anzahl an Schleifendurchläufen größer ist, als die Anzahl der verschiedenen Zufallszahlen, die generiert werden. Pythons `random` Modul generiert Zufallszahlen mit gleichem Abstand und einer Präzision von 53 bit. [pyt] Das bedeutet auch, dass die Funktion `random.random()` $2^{53} - 1$ verschiedene Zufallszahlen im Halboffenen Intervall $[0, 1)$ generieren kann. Für die Anwendung der `random.random()` Funktion in den Implementierungen `gnp_random_graph` 3.1 und `fast_gnp_random_graph` 3.2 heißt das, dass diese nur für eine Wahrscheinlichkeit p bis zu einer Präzision von 2^{53} sinnvoll sind. Um das einzusehen, betrachten wir die Zahl $p_1 = 0.05954861408025609$, welche nicht durch die Funktion `random.random()` generiert werden kann, da sie kein Vielfaches von 2^{-53} ist. Somit gilt für p_1 und $p_2 = 536366232364542 \cdot 2^{-53}$, das nächst größere Vielfache von 2^{-53}:

$$\nexists r \in \text{random.random}() : p_1 \leq r < p_2$$

Dadurch werden die Implementierungen für p_1, p_2 und gleichen Seed auch immer die gleichen Graphen liefern.

Wenn wir die Auswahl der Möglichen Werte des Parameter $p \in (0, 1)$ einschränken, können wir auch die Genauigkeit der Zufallszahl r einschränken und so ab einer gewissen Anzahl von Knoten n die Anzahl der Aufrufe der Logarithmusfunktion nach oben beschränken. In

der Implementierung `fast_gnp_random_graph` 3.2 ist die erwartete Anzahl an Aufrufen der Logarithmusfunktion $p \cdot \frac{n(n-1)}{2}$. In Abhängigkeit, der Anzahl der Knoten n, der Wahrscheinlichkeit p und der Anzahl der möglichen Zufallszahlen `maxrand` werden somit unter folgender Bedingung in der Implementierung von `PreLogZER` 3.5 weniger Logarithmusfunktionen aufgerufen als in `fast_gnp_random_graph`:

$$p \cdot \frac{n(n-1)}{2} > \texttt{maxrand}$$
$$\Leftrightarrow n > \frac{1}{2} + \sqrt{\frac{1}{4} + 2 \cdot \frac{\texttt{maxrand}}{p}} \in \mathcal{O}\left(\left(\frac{\texttt{maxrand}}{p}\right)^{\frac{1}{2}}\right)$$

Unter der Annahme, dass das Lesen eines Wertes aus einer Liste mit den zuvor berechneten Logarithmen der Zufallszahlen r, schneller ist, als das berechnen des Logarithmus einer Zufallszahl, können wir so die Laufzeit verringern.

PreZER

Im vorherigen Algorithmus haben wir sichergestellt, dass jeder Logarithmus nur maximal ein Mal berechnet wird. Da die Berechnung des Logarithmus lediglich dazu dient, Abstufungen der Verteilungsfunktion zu errechnen, können wir den Mehraufwand durch die Berechnung des Logarithmus ganz umgehen, in dem wir die Abstufungen im Voraus berechnen. Die Abstufungen sind für verschiedene p in 3.6 dargestellt. Für große p ist die Verteilungsfunktion schon für kleine k nah an 1, was bedeutet, dass die Wahrscheinlichkeit in `fast_gnp_random_graph` ein großes k zu samplen, klein ist. Natürlich können wir nicht alle Abstufungen im Voraus berechnen, da es unendlich viele gibt. Wir können aber, bis zu einem gewissen, von uns festgelegtem `max_k`, alle Funktionswerte $F_p(k)$ berechnen und so $F_p(\texttt{max\_k}) \cdot 100\%$ aller möglichen Fälle und Zufallszahlen $r \in [0, 1)$ abdecken, für die wir dann keinen Logarithmus `log(1-r)` mehr berechnen müssen. Die Wahl von `max_k` ist dabei die entscheidende Frage. Das hängt von der Verbesserung der Laufzeit, die die Berechnung der Abstufungen mit sich bringt ab.

Vergleich der Algorithmen

Der Vergleich der Beschriebenen Algorithmen verlief nicht wie erwartet, da sich die Laufzeit der Algorithmen in Python nicht wie in [NLKB11] verhält. In [NLKB11, Nobari et. al.] wurden die Laufzeiten der Algorithmen anhand von Graphen mit einer Knotenanzahl von 10000 gemessen. Die Algorithmen wurden in C++ implementiert und die durchschnittliche Laufzeit anhand von 10 erstellten Graphen gemessen.

Zunächst viel auf, dass Implementierung von `PreLogZER` sich nicht mit den anderen Algorithmen vergleichen lässt, da durch die Einschränkung der Domain der Zufallsvariablen r nicht die hoch optimierte `random.random()` Funktion verwendet werden kann. Stattdessen wurde `random.randint()` verwendet. Dadurch ist die Laufzeit von `PreLogZER` um ein vielfaches langsamer als die der anderen Algorithmen.

Für die anderen 3 Algorithmen wurde die Laufzeit für Graphen mit 10000 Knoten gemessen, wobei für jede Kantenwahrscheinlichkeit $p = 0.1, ...0.6$ 6 Graphen erzeugt wurden und deren durchschnittliche Laufzeit genommen wurde.

```python
import networkx as nx
from math import log
import random

def PreLogZER(n: int,
              p: float,
              randmax: int = 65536,
              seed=None) -> nx.Graph:
    """
    :param n: number of nodes

    """
    random.seed(seed)
    lr = []
    for i in range(randmax):
        lr.append(log((i+1)/randmax))
    G = nx.empty_graph(n)
    if p <= 0 or p >= 1:
        return nx.gnp_random_graph(n, p, seed=seed)
    lp = log(1.0 - p)
    v = 1
    w = -1
    while v < n:
        r = random.randint(0, randmax-1)
        w += 1 + int(lr[r] / lp)
        while w >= v and v < n:
            w = w - v
            v = v + 1
        if v < n:
            G.add_edge(v, w)
    return G
```

Abbildung 3.5: `PreLogZER`

Es stellt sich heraus, dass die Vorausberechnung der Abstufungen keinen Geschwindigkeitsvorteil im Vergleich zu `fast_gnp_random_graph` und `gnp_random_graph` bringt. Im Gegenteil, der Graph 3.8 zeigt, dass die Laufzeit besonders für große p steigt. Es scheint so, als würde das ersetzen des Logarithmus durch einen Lookup-Table in Python nicht schneller sein. Recht gibt uns die `networkx` Library, die zum generieren von Erdős–Rényi Graphen nur die Algorithmen 3.1 und 3.2 verwendet, ersten für große p und zweiten für dünne Graphen.

In 3.8 kann man erkennen, dass bis zu einer Kantenwahrscheinlichkeit von ungefähr $p = 0,4$ der Algorithmus `fast_gnp_random_graph` 3.2 schneller ist und danach gegen-

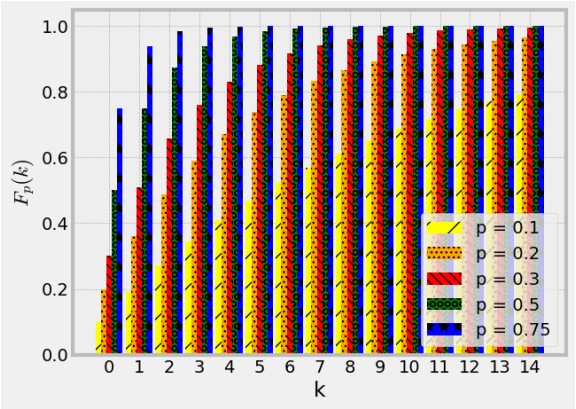


Abbildung 3.6: Verteilungsfunktionen der geometrischen Verteilung $F_p(k) = (1-p)^{k-1} \cdot p$ für $p \in \{0.1, 0.2, 0.3, 0.5, 0.75\}$

über `gnp_random_graph` 3.1 an Laufzeit verliert. Das liegt an der geringeren Anzahl an Schleifendurchläufen aber dem größeren Aufwand durch die Berechnung des Logarithmus, welche sich erst bei großen p bemerkbar macht. Für beide scheint die Laufzeit linear in der Kantenwahrscheinlichkeit zu sein.

```python
import networkx as nx
from math import log,pow,floor
import random
from time import process_time
import numpy as np
MAX_M = 9
def PreZER(n: int, p, max_m: int = MAX_M, seed = 42) -> nx.Graph:
    """
    :param n: number of nodes
    :param p: probability of edge creation
    :param max_m: number of precomputed breakpoints of the comulative \
        distribution
    """
    random.seed(seed)
    G = nx.empty_graph(n)
    if p <= 0 or p >= 1:
        return nx.gnp_random_graph(n, p, seed=seed)
    F = np.array([1-pow(1-p,k) for k in range(1,max_m+1)])
    lp = log(1.0 - p)
    v = 1
    w = -1
    while v < n:
        r = random.random()
        j = 0
        while j < max_m:
            if r < F[j]:
                k = j
                break
            j += 1
        else:
            k = 1 + floor(log(1.0 - r) / lp)
        w += k
        while w >= v and v < n:
            w -= v
            v += 1
        if v < n:
            G.add_edge(v, w)
    return G
```

Abbildung 3.7: `PreZER`

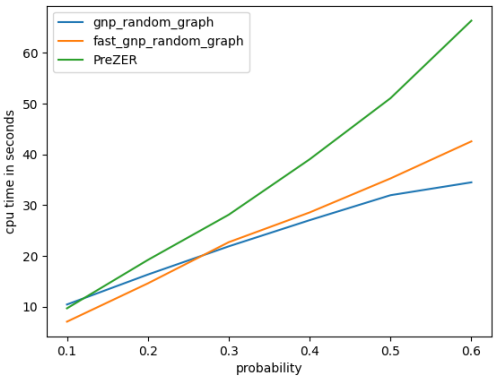


Abbildung 3.8: Vergleich der durchschnittlichen Laufzeit von `gnp_random_graph`, `fast_gnp_random_graph` und `PreZER` für Graphen mit 10000 Knoten und 6 generierten Graphen pro p

4 Spannbäume generieren

Im vorigen Kapitel haben wir gesehen, wie man zufällige Graphen generiert. In diesem Kapitel werden wir 2 Algorithmen kennenlernen, die uns erlauben, einen Spannbaum eines Graphen zu generieren. Der erste Algorithmus ist der Groundskeeper Algorithmus, welcher durch einen Random Walk auf einem Graphen einen Spannbaum liefert. Wir werden Beweisen, dass dieser Spannbaum gleichverteilt auf allen Spannbäumen des Graphen ist. Da der Groundskeeper Algorithmus nicht sehr effizient ist, werden wir im Anschluss einen schnelleren Algorithmus kennenlernen, der einen Spannbaum von einem vollständigen Graphen liefert. Wir werden mit Hilfe des Beweises über den Baum aus dem Groundskeeper Algorithmus zeigen, dass auch der Spannbaum aus dem schnelleren Algorithmus gleichverteilt auf allen Spannbäumen des vollständigen Graphen ist.

4.1 Groundskeeper Algorithmus

Konstruktion des Spannbaums

Sei im folgenden $G = (V, E)$ ein zusammenhängender einfacher Graph wie in 2.1.1 und 2.1.3 definiert. Mit r_v bezeichnen wir den Grad, die Anzahl der Nachbarn, eines Knotens v aus V. Mit $(X_j; j \geq 0)$ bezeichnen wir einen Random Walk auf dem Graphen G mit einem zufällig ausgewählten Startknoten X_0. Für einen zufälligen Startknoten v gilt dann

$$X_j = \begin{cases} v & \text{if } j = 0 \\ w \text{ für } w \in \{w \in V : w \sim X_{j-1}\} & \text{if} j > 0 \end{cases}$$

wobei jedes $w \in \{w \in V : w \sim X_{j-1}\}$ gleich wahrscheinlich ist. Das heißt, dass für alle Knoten in V gilt, dass die Wahrscheinlichkeit, dass ein Knoten v aus V der erste Knoten $X_0 = v$ ist, $1/|V|$ ist. Für einen beliebigen Schritt $X_j = v$ mit $j \geq 0$ des Random Walks und eine Kante $\{v, w\}$ aus E gilt, dass die Wahrscheinlichkeit, dass der nächste Schritt $X_{j+1} = w$ ist, gleich $1/r_v$ ist. Der Random Walk terminiert, wenn alle Knoten von V erschlossen wurden. Da G endlich ist, terminiert ein Random Walk mit Wahrscheinlichkeit 1. Eine Implementierung des Random Walks in Python ist in Abbildung 4.1 zu sehen.

Auf Grundlage dieses Random Walks konstruieren wir einen Spannbaum des Graphen G und gehen dabei folgendermaßen vor. Wir betrachten die verwendeten Kanten eines Random Walks, also die Menge

$$\big\{\{v, w\} \in E \mid \exists\, i \geq 0 : X_i = v \ \wedge \ X_{i+1} = w\big\}$$

und entfernen die Kanten, durch die kein neuer Knoten durch den Random Walk erschlossen wurde. Für eine genauere Beschreibung definieren wir den Zeitpunkt, zu dem ein Knoten

```python
import random
import networkx as nx

def random_walk(graph, seed=42):
    random.seed(seed)
    assert nx.is_connected(graph), "Graph nicht zusammenhängend"
    #Startknoten X_0
    random_node = random.choice(list(graph))
    #Liste zum Speichern der Nodes
    randomwalk = [random_node]
    while set(randomwalk) != set(graph):
        nachbarn = list(graph.neighbors(random_node))
        nächster_knoten = random.choice(nachbarn)
        randomwalk.append(nächster_knoten)
    return(randomwalk)
```

Abbildung 4.1: Implementierung des Random Walks in Python

das erste Mal entdeckt wurde. Wir bezeichnen diesen Zeitpunkt für jeden Knoten v als T_v, der folgendermaßen definiert ist:

$$T_v := \min\{j \geq 0 : X_j = v\}$$

Da der Random Walk mit Wahrscheinlichkeit 1 terminiert, sind die T_v wohldefiniert. Wir können nun einen Teilgraph von G definieren, mit der Kantenmenge

$$E' := \{\{X_{T_v-1}, X_{T_v}\} | v \in V \setminus X_0\} \tag{4.1}$$

Wir definieren den Teilgraph

$$\mathcal{T} := (V, E')$$

Proposition 4.1.0.1. Sei $N(G)$ die Anzahle der Spannbäume t von G. Dann ist

$$\mathbb{P}(\mathcal{T} = t) = \frac{1}{N(G)}$$

für alle Spannbäume t von G.

Den Beweis dieser Proposition werden wir in zwei Schritten führen. Zunächst zeigen wir, dass $\mathcal{T}$ ein Spannbaum ist in dem wir beweisen, das $\mathcal{T}$ zusammenhängend und kreisfrei ist. Im zweiten Schritt und Hauptteil zeigen wir, dass $\mathcal{T}$ auf der Menge aller Spannbäume gleichverteilt ist.

```python
def baum_aus_Randomwalk(randomwalk:list) -> nx.Graph:
    """
    creates Tree from randomwalk like in the Aldous paper
    """
    T = nx.Graph()
    #liste in reihenfolge der Entdeckung
    T_geordnete_knoten = list(dict.fromkeys(randomwalk))
    for node in T_geordnete_knoten[1:]:
        i = randomwalk.index(node)
        T.add_edge(randomwalk[i-1],node)
    assert nx.is_tree(T), "T ist kein Baum"
    return(T)
```

Abbildung 4.2: Funktion in Python um einen Spannbaum aus einem Random Walk zu konstruieren

Konstruktion ist zusammenhängend

Um zu zeigen, dass $\mathcal{T}$ ein Spannbaum ist, zeigen wir zunächst, dass $\mathcal{T}$ zusammenhängend ist. Dazu nummerieren wir die Knoten aus V in der Reihenfolge ihrer Entdeckung im Random Walk $(X_j; j \geq 0)$ vermöge $(v_1, \ldots, v_{|V|}) := (X_{T_{v_1}}, \ldots, X_{T_{v_{|V|}}})$ wobei $T_{v_i} < T_{v_j}$ für $i < j$. Wir zeigen, dass der Graph, der durch die Knoten $V_N = \{v_1, \ldots, v_N\}$ und die Kantenmenge $E_N = \{(X_{T_v - 1}, X_{T_v}) | v \in V_N \setminus X_0\}$, $N \leq |V|$, definiert ist, zusammenhängend ist. Dazu führen wir einen Induktionsbeweis über N.

Induktionsanfang $N = 1$:
Der Graph $V_1 = (\{v_1\}, \emptyset)$ ist als trivialer Graph zusammenhängend.
Induktionsvoraussetzung:
Der Graph $G_N = (\{v_1, \ldots, v_N\}, \{(X_{T_v - 1}, X_{T_v}) | v \in V_N \setminus X_0\})$ mit $N < |V|$ ist zusammenhängend.
Induktionsschritt $N \to N + 1$:
Der Knoten von dem aus v_{N+1} entdeckt wurde, ist der Knoten $X_{T_{v_{N+1}} - 1}$. Da dieser Knoten zuvor entdeckt worden sein muss, ist $X_{T_{v_{N+1}} - 1}$ in V_N. Da nach der Induktionsvoraussetzung V_N zusammenhängend ist, existiert ein Pfad zwischen v_1 und v_N. Somit können wir diesen Pfad durch die Kante $(X_{T_{v_{N+1}} - 1}, X_{T_{v_{N+1}}}) \in E_{N+1}$ erweitern und haben einen Pfad zwischen v_1 und v_{N+1} gefunden. Somit ist also v_1 mit jedem anderen Knoten in V_{N+1} verbunden, wodurch G_{N+1} zusammenhängend ist.

Wir haben also bewiesen, dass für $N \leq |V|$ der Graph G_N zusammenhängend ist. Dadurch ist insbesondere der Graph $G = G_{|V|} =$ zusammenhängend, was wir zeigen wollten.

Konstruktion ist kreisfrei

Um zu zeigen, dass G ein Spannbaum ist, bleibt noch zu zeigen, dass es in G keine Kreise gibt. Dazu zeigen wir zunächst folgende Lemmata:

Lemma 4.1.1. Ein Graph $G = (V, E)$ mit $|E| < |V|$ hat mindestens ein Blatt, also einen Knoten mit nur einem Nachbarn.

Beweis. Wir nehmen an, ein Graph $G = (V, E)$ mit $|E| < |V|$ habe kein Blatt. Dann sind alle Knoten von G mindestens vom Grad 2. Summiert man die Grade der Knoten von V auf, zählt man alle Kanten doppelt, somit ergibt sich:

$$2|E| = \sum_{v \in V} r_v \geq 2|V|.$$

Und dadurch

$$|E| \geq |V|$$

was ein Widerspruch zur Annahme $|E| < |V|$ ist. Also hat jeder Graph $G = (V, E)$ mit $|E| < |V|$ mindestens ein Blatt. □

Lemma 4.1.2. Ein zusammenhängender Graph G mit n Knoten hat mindestens $n - 1$ Kanten.

Beweis. Für $n \leq 3$ lassen sich alle möglichen Graphen, wie in 4.3 zu sehen, leicht aufzeichnen, um die Aussage zu verifizieren.

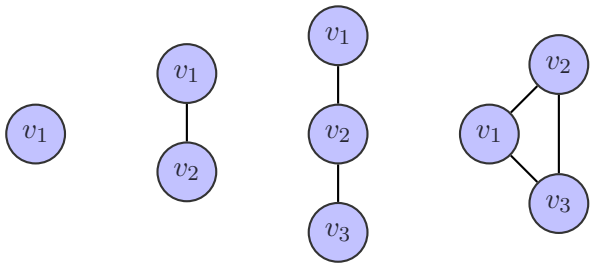


Abbildung 4.3: Alle zusammenhängenden Graphen mit 3 oder weniger Knoten

Seil also nun $n \geq 4$. Um einen Widerspruch zu erzeugen, betrachten wir den Graphen G mit minimaler Knotenanzahl n, dessen Anzahl der Kanten nicht größer als $n - 2$ ist. Wir entfernen einen Knoten vom Grad 1, welcher aufgrund von Lemma 4.1.1 existiert, und dessen zugehörige Kante. Dadurch erhalten wir einen neuen zusammenhängenden Graphen G' mit Knotenanzahl $n - 1$ und weniger als $n - 2$ Kanten. Dieser Graph ist zusammenhängend und hat mindestens 2 Kanten weniger als Knoten, wodurch der ursprüngliche Graph G nicht der Graph mit dieser Eigenschaft und minimaler Knotenanzahl gewesen sein kann. Somit kann dieser Graph G nicht existieren und ein zusammenhängender Graph G mit n Knoten hat mindestens $n - 1$ Kanten. □

Satz 4.1.3. Der durch die Kanten in 4.1 definierte Graph ist kreisfrei.

Beweis. Wir nehmen an es gäbe in dem durch 4.1 definierten Graphen $G = (V, E')$ einen Kreis. Dann können wir eine Kante e aus diesem Kreis entfernen, sodass der $G' = (V, E' \setminus e)$ immer noch zusammenhängend ist. Allerdings gilt $|E'| = |V| - 1$ und somit $|E' \setminus e| = |V| - 2$. Wir haben aber in Lemma 4.1.2 gezeigt, dass ein zusammenhängender Graph mit Knotenzahl $|V|$ mindestens $|V| - 1$ Kanten haben muss. Das ist ein Widerspruch und somit ist G kreisfrei.

□

Damit haben wir gezeigt, dass der durch 4.1 definierte Graph $\mathcal{T}$ ein Spannbaum des originalen Graph G ist. Jeder Spannbaum von G ist durch diese Konstruktion möglich. Einem bestimmten Spannbaum t könnten mehrere verschiedene Random Walks zu Grunde liegen. Einer ist aber gerade jener, der durch die Tiefensuche auf dem Baum t bestimmt wird.

Konstruktion ist gleichverteilt

Wir zeigen im Folgenden, dass die durch Random Walks definierten Bäume gleichverteilt sind. Dazu brauchen wir den Begriff eines stationären stochastischen Prozesses aus 2.2.2. Für uns ist ein Random Walk $(X_j; j \geq 0)$ ein stochastischer Prozess wie in 2.2.1 mit dem Raum Ω aller möglichen Random Walks auf G. Der Zustandsraum Z ist die Menge V der Knoten von G und T die Indexmenge $\mathbb{N}_0$.

Wir bezeichnen im folgenden die Anzahl der Spannbäume von G mit $N(G)$ und die Menge aller gewurzelten Spannbäume von G mit $\mathcal{S}$. Ein gewurzelter Spannbaum ist ein Spannbaum mit einem festen Wurzelknoten. Da es $|V|$ Möglichkeiten gibt, einen Knoten als Wurzelknoten zu wählen, gilt $|\mathcal{S}| = |V| \cdot N(G)$. Um zu zeigen, dass die Verteilung der Spannbäume (ohne Wurzel) uniform ist, also

$$\mathbb{P}(\mathcal{T} = t) = \frac{1}{N(G)} = \frac{|V|}{|\mathcal{S}|},$$

mit einem Spannbaum t, betrachten wir zunächst die gewurzelten Spannbäume, die durch einen Random Walk $(X_j; j \geq 0)$ definiert sind aber die Konstruktion wie in 4.1 zu einem späteren Zeitpunkt m im Random Walk startet. Bezeichne mit T_v^m den Index des ersten Besuches des Knotens v ab dem Index m, also

$$T_v^m = \min\{j \geq m : X_j = v\}.$$

Dann ist

$$S_m = (V, \{(X_{T_v^m - 1}, (X_{T_v^m}) | v \in V \setminus X_m\}) \in \mathcal{S}$$

der Spannbaum mit Wurzel X_m, der durch den Random Walk $(X_m, X_{m+1}, X_{m+2}, \dots)$ mit $m \geq 0$ definiert wird. Dadurch erhalten wir eine Folge von gewurzelten Spannbäumen $(S_m)_{m \geq 0}$. Im nächsten Schritt betrachten wir einen Random Walk $(X_j; -\infty < j < \infty)$ auf G, der mit den ganzen Zahlen indexiert ist. Der Random Walk $(X_j; -\infty < j < \infty)$ induziert dann eine ebenfalls über die ganzen Zahlen indexierte Folge von gewurzelten Spannbäumen $(S_m; -\infty < m < \infty)$. Wir werden uns eine solche Folge von gewurzelten Spannbäumen in Rückwärtszeit

$$(S_m, S_{m-1}, S_{m-2}, \dots), \tag{4.2}$$

welche bei einem Index $m \in \mathbb{Z}$ beginnt, genauer ansehen.

Lemma 4.1.4. Sei $P \in [0,1]^{n \times n}$ die Übergangsmatrix einer irreduziblen Markov-Kette (2.2.6) und sei $A = [P - I, \mathbf{1}]$ die Matrix $P - I$ mit einer zusätzlichen letzten Spalte mit nur 1 als Einträgen. Dann hat A vollen Rang.

Beweis. Da die Zeilen jeder Übergangsmatrix P aufsummiert 1 ergeben, gilt $P\mathbf{1} = \mathbf{1}$ und somit hat die Gleichung $Ax = 0$ die Lösung $(\mathbf{1}, 0)^T$. Sollte $\operatorname{rang}(A) = n$ nicht gelten, so müsste es eine weitere nicht triviale Lösung $(\mathbf{y}, \alpha)^T$ geben, die orthogonal zu $(\mathbf{1}, 0)^T$ (Gram-Schmidt) ist. Also muss gelten

$$\langle (\mathbf{1}, 0)^T, (\mathbf{y}, \alpha)^T \rangle = \sum_i \mathbf{y}_i = 0. \tag{4.3}$$

Dadurch können die Einträge von $\mathbf{y}$ nicht alle gleich sein, das sonst $\sum_i \mathbf{y}_i = n \cdot \mathbf{y}_1 = 0$ gelten würde, was nur für $\mathbf{y} = 0$ gilt. Daraus würde aber folgen, dass $\alpha = 0$ ist, was im Widerspruch zu $(\mathbf{y}, \alpha)^T \neq \mathbf{0}$ steht.

Wegen $A(\mathbf{y}, \alpha)^T = 0$ gilt $P\mathbf{y} + \alpha\mathbf{1} = \mathbf{y}$. Jeder Eintrag von $\mathbf{y}$ ist also eine Konvexkombination der Einträge von $\mathbf{y}$ plus α. Da die Markov-Kette irreduzibel ist, gibt es einen Zustand k, dessen zugehöriger Eintrag im Vektor $\mathbf{y}$ maximal ist und welcher auf dem Übergangsgraphen der Markov-Kette Nachbar von einem Zustand l ist, dessen zugehöriger Eintrag im Vektor $\mathbf{y}$ geringer ist. Würde dieses Tupel (k, l) nicht existieren, wären die Zustände mit maximalen Einträgen in $\mathbf{y}$ eine Mengen von Zuständen, die nicht mit anderen Zuständen kommunizieren, welche geringere Einträge in $\mathbf{y}$ haben. So wäre aber $\mathbf{y}$ nicht irreduzibel. Somit ist $p_{kl} \neq 0$ und dadurch $\mathbf{y}_k > \sum_i p_{ki}\mathbf{y}_i$. Da laut Annahme $\mathbf{y}_k = \sum_i p_{ki}\mathbf{y}_i + \alpha$ gelten muss, ist also $\alpha > 0$. Analog lässt sich ein Zustand k' finden, dessen zugehöriger Eintrag im Vektor $\mathbf{y}$ minimal ist und welcher auf dem Übergangsgraphen der Markov-Kette Nachbar von einem Zustand l' ist, dessen zugehöriger Eintrag im Vektor $\mathbf{y}$ größer ist. So erhalten wir $\alpha < 0$ und somit einen Widerspruch. Somit kann es keine zweite nicht triviale Lösung von $Ax = 0$ geben, wodurch $\operatorname{rang}(A) = n$ gilt. [BHK20] □

Korollar 4.1.4.1. $dim(\{\boldsymbol{\pi} : \boldsymbol{\pi}P = \boldsymbol{\pi}\}) \leq 1$.

Beweis. Für ein $\boldsymbol{\pi} \in \mathbb{R}^n$ mit $\sum_i \boldsymbol{\pi}_i = 1$ und $\boldsymbol{\pi}$ ist Linkseigenvektor von P muss gelten, $\boldsymbol{\pi}A = (\mathbf{0}, 1)$. Aus $\boldsymbol{\pi}A = (A^T\boldsymbol{\pi}^T)^T$ und $\operatorname{rang}(A) = \operatorname{rang}(A^T)$ folgt dann durch 4.3 $\dim(\{xA | x \in \mathbb{R}^n\}) = n$, womit die Abbildung $x \mapsto xA$ injektiv ist. Somit hat $\boldsymbol{\pi}A = (\mathbf{0}, 1)$ höchstens eine Lösung und durch skalieren dieser Lösung erhalten wir den Raum $\{\lambda\boldsymbol{\pi} : \lambda \in \mathbb{R}, \boldsymbol{\pi}A = (\mathbf{0}, 1)\} = \{\boldsymbol{\pi} : \boldsymbol{\pi}P = \boldsymbol{\pi}\}$, womit die Aussage gezeigt ist. [Eis14] □

Definition 4.1.1. [BHK20] Sei $\mathbf{p}(t)$ die Verteilung der Zustände einer Markov-Kette nach t Schritten, $\mathbf{p}(t)_i$ bezeichnet die relative Häufigkeit des Auftretens der Zustands i. Somit gilt klarerweise $\sum_i \mathbf{p}(t)_i = 1$ für jedes t. Bezeichne mit $\mathbf{a}(t)$ die längerfristige Verteilung der Zustände.

$$\mathbf{a}(t) \coloneqq \frac{1}{t}(\mathbf{p}(0) + \cdots + \mathbf{p}(t-1))$$

Satz 4.1.5 (Fundamentalsatz für Markov-Ketten)**.** Für eine irreduzible Markov-Kette existiert eine eindeutige Verteilung $\boldsymbol{\pi}$, welche $\boldsymbol{\pi}P = \boldsymbol{\pi}$ erfüllt und für längerfristige Verteilung $\mathbf{a}(t)$ gilt stets $\lim_{t\to\infty} \mathbf{a}(t) = \boldsymbol{\pi}$

Beweis.

$$\begin{aligned}\mathbf{b}(t) &= \mathbf{a}(t)P - \mathbf{a}(t)\\ &= \frac{1}{t}(\mathbf{p}(1) + \cdots + \mathbf{p}(t)) - \frac{1}{t}(\mathbf{p}(0) + \cdots + \mathbf{p}(t-1))\\ &= \frac{1}{t}(\mathbf{p}(t) - \mathbf{p}(0))\end{aligned}$$

Also gilt $|\mathbf{b}(t)| \leq \frac{2}{t}$ und somit konvergiert $\mathbf{b}(t) = \mathbf{a}(t)P - \mathbf{a}(t)$ gegen $\mathbf{0}$. Dadurch konvergiert $\mathbf{a}(t)$ gegen eine Verteilung $\boldsymbol{\pi}$ für die $\boldsymbol{\pi}P = \boldsymbol{\pi}$ gilt. Diese Verteilung ist durch 4.1.4.1 eindeutig. □

Wir werden zeigen, dass die Folge in 4.2 eine Markov-Kette ist und dass $(S_m; -\infty < m < \infty)$ ein stationärer stochastischer Prozess ist , wodurch wir über die stationäre Verteilung dieser Markov-Kette, die Verteilung aller gewurzelten Spannbäume erhalten.

Lemma 4.1.6. Ein gewurzelter Spannbaum S_i aus der Folge 4.2 mit $i > m$ ist vollständig durch (S_{i+1}, X_i) bestimmt.

Beweis. Dem gewurzelten Spannbaum S_{i+1} liegt der Random Walk $(X_j; j \geq i+1)$ zugrunde. Beginnen wir nun den Random Walk bei X_i, anstelle von X_{i+1}, müssen wir eventuell eine neue Kante zu unserem Baum hinzufügen und zwar die Kante (X_i, X_{i+1}). Falls diese Kante bereits in S_{i+1} vorhanden war, gilt $S_i = S_{i+1}$, ansonsten ist die Kante, die hinzugefügt wurde als X_i in S_{i+1} entdeckt wurde in S_i nicht mehr vorhanden, da ja X_i der erste Knoten war. Der Rest des Baumes bleibt hingegen unverändert.[Dan19] □

Der ausschlaggebende Punkt ist, dass wir die im zweiten Fall überflüssige Kante eindeutig durch S_{l+1} bestimmen können. Diese Kante ist nämlich die Letzte, vom eindeutigen Weg von X_{i+1} nach X_i, im Baum S_{i+1}, was folgende Grafik illustrieren soll.

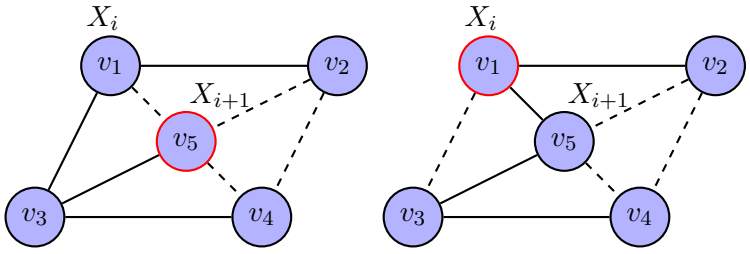


Abbildung 4.4: Beispiel zweier Spannbäume mit Wurzel in rot. Links der Spannbaum S_{i+1} und Rechts S_i. Die Kante (v_3, v_1) wurde als letzte Kante vom Weg von X_{i+1} nach X_i entfernt.

Wir haben somit gezeigt, dass die Folge aus 4.2 gedächtnislos ist. Fassen wir Ω bzw. Z aus 2.2.1 als Menge aller Folgen von gewurzelten Spannbäumen bzw. $\mathcal{S}$ (die Menge aller gewurzelter Spannbäume) auf, dann ist eine Familie $S^m = (S_i)_{i \leq m}$ mit $m \in \mathbb{Z}$ ein stochastischer Prozess. Somit ist jede Folge wie in 4.2 eine Markov-Kette.

Ein Random Walk ist ebenfalls eine Markov-Kette, da die Übergangswahrscheinlichkeiten nur vom aktuellen Knoten abhängen. Da der Graph G zusammenhängend ist, ist die Markov-Kette die einen Random Walk beschreibt irreduzibel und somit existiert durch 4.1.5 eine eindeutige stationäre Verteilung.

Lemma 4.1.7. Die stationäre Verteilung $\boldsymbol{\pi}$ eines Random Walks auf einem zusammenhängenden, endlichen, ungerichteten Graphen $G = (V, E)$ ist proportional zu dem Grad der Knoten. Genauer:

$$\boldsymbol{\pi}(v) = \frac{r_v}{2|E|} \tag{4.4}$$

Beweis. Dazu müssen wir zeigen, dass der Vektor $\boldsymbol{\pi}$ ein Linkseigenvektor zum Eigenwert 1 der Übergangsmatrix einer Markov-Kette $(X_j; m \leq j < \infty)$ mit $m \in \mathbb{Z}$ ist. Sei P die Übergangsmatrix, mit Einträgen $p_{v,w} = \mathbb{P}(X_{j+1} = w | X_j = v)$ für $v, w \in V$, dann soll also gelten

$$\boldsymbol{\pi}^T P = \boldsymbol{\pi}^T$$

und somit

$$\sum_v \boldsymbol{\pi}(v) p_{v,w} = \boldsymbol{\pi}(w)$$

für alle $w \in V$. Durch Einsetzen erhalten wir für festes w

$$\sum_v \boldsymbol{\pi}(v) p_{v,w} = \sum_v \frac{r_v}{2|E|} p_{v,w} = \sum_{v \sim w} \frac{r_v}{2|E|} \frac{1}{r_v} = \frac{r_w}{2|E|} = \boldsymbol{\pi}(w)$$

und somit die Behauptung. □

Somit ist in einem Random Walk $(X_j; -\infty < j < \infty)$ das Auftreten eines bestimmten Knotens nicht von der Zeit abhängig und dadurch zu jedem Zeitpunkt gleich wahrscheinlich. Ein Random Walk indexiert mit den ganzen Zahlen, ist also ein stationärer stochastischer Prozess. Da ein gewurzelter Spannbaum $S_m,\ m \in \mathbb{Z}$ mit Wahrscheinlichkeit 1 von einer endlichen Folge von Knoten $(X_m, X_{m+1}, \ldots X_{m+n})$ abhängt, ist ein gewurzelter Spannbaum zu jedem Zeitpunkt gleich wahrscheinlich und $(S_m; -\infty < m < \infty)$ ein stationärer stochastischer Prozess.

Um die stationäre Verteilung der Markov-Kette von gewurzelten Spannbäumen in Rückwärtszeit wie in 4.2 zu ermitteln, wollen wir die Übergangsmatrix und deswegen die Übergangswahrscheinlichkeiten

$$\mathbb{P}(S_m = u | S_{m+1} = t)$$

für Bäume u und t betrachten, also die Wahrscheinlichkeiten, dass ein gewurzelter Spannbaum u auftritt bedingt durch den Nachfolger t. Dazu sind die Übergangswahrscheinlichkeiten eines Random Walks $(X_j; -\infty < j < \infty)$ in Rückwärtszeit von Bedeutung.

Lemma 4.1.8. Für einen Random Walk $(X_j; -\infty < j < \infty)$, beliebiges $m \in \mathbb{Z}$, $v, w \in V$ mit $v \sim w$ gilt

$$\mathbb{P}(X_{m-1} = w | X_m = v) = \frac{1}{r_v}$$

Beweis. Für den Beweis nutzen wir, dass für beliebiges $v \in V$ und einen beliebigen Schritt $m \in \mathbb{Z}$ im Random Walk $(X_j; -\infty < j < \infty)$ gilt,

$$\mathbb{P}(X_m = v) = \boldsymbol{\pi}(v),$$

mit $\boldsymbol{\pi}$, der stationären Verteilung des Random Walks aus 4.1.7. Dann erhalten wir nach Definition der bedingten Wahrscheinlichkeit für $w \sim v$

$$\begin{aligned}
\mathbb{P}(X_{m-1} = w | X_m = v) &= \frac{\mathbb{P}(X_{m-1} = w, X_m = v)}{\mathbb{P}(X_m = v)} \\
&= \frac{\boldsymbol{\pi}(w)\mathbb{P}(X_m = v | X_{m-1} = w)}{\boldsymbol{\pi}(v)} \\
&= \frac{\frac{r_w}{2|E|}\frac{1}{r_w}}{\frac{r_v}{2|E|}} \\
&= \frac{1}{r_v}
\end{aligned}$$

und somit die Aussage. □

Bezeichne für einen Baum t, den Grad seiner Wurzel mit $r(t)$. Bedingt durch einen gewurzelten Spannbaum $S_i = u$ mit $i \leq m$ der Folge 4.2 von gewurzelten Spannbäumen, betrachten wir jetzt die Wahrscheinlichkeit für einen gewurzelten Spannbaum S_{i-1}. Dem Baum u liegt der Random Walk $(X_j; j \geq i)$ zugrunde. Für den Vorgängerknoten X_{i-1} kommen dadurch $r(u)$ Knoten in Frage, wobei durch 4.1.8 jeder Knoten davon gleich wahrscheinlich ist. Somit gibt es auch $r(u)$ Bäume aus $\mathcal{S}$ die für S_{i-1} in Frage kommen und die alle gleich wahrscheinlich sind. Bezeichne die Menge dieser Bäume mit $\mathcal{D}(u)$. Dann gilt für festes $u \in \mathcal{S}$

$$\mathbb{P}(S_{i-1} = t | S_i = u) = \begin{cases} \frac{1}{r(u)} & \text{falls } t \in \mathcal{D}(u) \\ 0 & \text{sonst} \end{cases} \tag{4.5}$$

Gehen wir in 4.5 allerdings von festem $S_{i-1} = t \in \mathcal{S}$ aus, so gibt es in der Markov-Kette 4.2 der gewurzelten Spannbäume, $r(t)$ Nachfolger S_i von t, für die die Gleichung gilt. Die Menge dieser Bäume bezeichnen wir mit $\mathcal{C}(t)$.

Mit 4.5 können wir die Übergangsmatrix der Markov-Kette $(S_m, S_{m-1}, S_{m-2}, \ldots)$ für $i \leq m$ aufstellen:

$$P = \begin{pmatrix} \mathbb{P}(S_{i-1} = t_1 | S_i = t_1) & \mathbb{P}(S_{i-1} = t_2 | S_i = t_1) & \mathbb{P}(S_{i-1} = t_3 | S_i = t_1) & \ldots \\ \mathbb{P}(S_{i-1} = t_1 | S_i = t_2) & \mathbb{P}(S_{i-1} = t_2 | S_i = t_2) & \ddots & \\ \mathbb{P}(S_{i-1} = t_1 | S_i = t_3) & \ddots & & \\ \vdots & & & \end{pmatrix}$$

Diese Matrix hat aufgrund von

$$\begin{aligned}\sum_{t\in\mathcal{S}} r(t)\mathbb{P}(S_{i-1}=t'|S_i=t) &= \sum_{t\in\mathcal{C}(t')} r(t)\mathbb{P}(S_{i-1}=t'|S_i=t)\\ &= \sum_{t\in\mathcal{C}(t')} r(t)\frac{1}{r(t)}\\ &= r(t')\end{aligned}$$

den Linkseigenvektor $(r(t))_{t\in S}$ zum Eingenwert 1.[Ald90] Um die stationäre Wahrscheinlichkeit der Markov-Kette der gewurzelten Spannbäume zu erhalten, müssen wir diesen Vektor noch normieren.

$$\sum_{t\in\mathcal{S}} r(t) = N(G)\sum_{v\in V} r_v = 2N(G)|E|$$

Somit ist der Vektor

$$\frac{1}{2N(G)|E|}(r(t)_{t\in\mathcal{S}})$$

die gesuchte und wegen Satz 4.1.5 eindeutige stationäre Verteilung $\boldsymbol{\pi}$ der Markov-Kette.

Da $(S_m; -\infty < m < \infty)$ ein stationärer Prozess ist, ist die Verteilung $\boldsymbol{\pi}$ genau der Vektor der Wahrscheinlichkeiten

$$(\mathbb{P}(S_m = t))_{t\in\mathcal{S}}$$

wodurch dann

$$\mathbb{P}(S_m = t) = \frac{r(t)}{2|E||N(G)|}$$

für einen gewurzelten Spannbaum t gilt. Somit hängt die Wahrscheinlichkeit des Auftretens eines Spannbaumes nur vom Grad seiner Wurzel hab. Kehren wir nun wieder zu dem ursprünglichen Spannbaum $\mathcal{T}$, der durch den Random Walk $(X_j; j \geq 0)$ definiert wird, zurück. Ist die Verteilung des Startknotens X_0 die der stationären Verteilung 4.4 des Random Walks über die ganzen Zahlen, so können wir auch die stationäre Wahrscheinlichkeit des Spannbaums anwenden. So ist dann

$$\mathbb{P}(\mathcal{T} = t) = \frac{r(t)}{2|E||N(G)|}$$

für einen gewurzelten Baum t. Wenn wir diese Wahrscheinlichkeit mit der Bedingung eines bestimmten Startknotens $X_0 = w$ versehen, wobei $w \in V$ beliebig ist, dann ist jeder Spannbaum mit w als Wurzel gleich wahrscheinlich, da diese Bäume alle den selben Grad der Wurzel haben. Da wir jeden Spannbaum von G mit jeder Wurzel w auffassen können, sind die entstehenden Bäume ohne Wurzel gleichverteilt. Ist nun X_0 bzw. w uniform, so ist immer noch jeder Spannbaum gleich wahrscheinlich.

■

4.2 schnellerer Algorithmus

Es ist eine schöne Vorstellung, dass der Spannbaum der durch einen Random Walk auf einem Graphen entsteht wirklich gleichverteilt auf der Menge seiner Spannbäume ist. In der Praxis ist dieser Algorithmus allerdings vor allem nicht praktikabel um schnell Spannbäume zu generieren, denn dafür muss der Random Walk erst alle Knoten besuchen, was bei großen Graphen sehr lange dauern kann.

In diesem Abschnitt werden wir einen Algorithmus vorstellen, der sehr schnell Spannbäume von vollständigen Graphen generieren kann. Wir werden im Beweis die Gleichverteiltheit auf der Menge aller Spannbäume des, durch den neuen Algorithmus generierten, Spannbaums auf die Gleichverteilung des Spannbaumes aus dem Groundskeeper Algorithmus zurückführen.

Sei für ein $n \in \mathbb{N} \setminus \{0,1\}$ $K_n = (V = \{v_1, \dots, v_n\}, E = \{e_1, \dots, e_{\binom{n}{2}}\})$ der vollständige Graph mit n Knoten wie in 2.1.4 definiert.

Algorithmus 4.2.1.

(i) Für $2 \leq i \leq n$ sei U_i gleichverteilt und unabhängig auf der Knotenmenge V. Füge die Kanten von v_i nach $v_{\min(U_i, i-1)}$ hinzu.

(ii) Bezeichne die Knoten $v_1, \dots, v_n$ als $v_{\pi(1)}, \dots v_{\pi(n)}$ wobei $\pi \in S_n$ eine zufällige (gleichverteilte) Permutation ist.

Der Algorithmus beginnt mit einer leeren Kantenmenge. Da in der ersten Schleife mit $i = 2$ begonnen wird und somit $\min(U_2, 1) = 1$ schon fest steht, werden zunächst Knoten v_1 und v_2 durch eine Kante verbunden. In jedem weiteren Schritt $i = 3, \dots, n$ wird dann mit Wahrscheinlichkeit $1 - \frac{i-2}{n}$ der Knoten v_i mit dem Knoten v_{i-1} verbunden. Für alle anderen Knoten $v_1, \dots, v_{i-2}$ ist die Wahrscheinlichkeit $\frac{1}{n}$, da diese nur ausgewählt werden können, wenn $U_i < i - 1$ ist.

Proposition 4.2.0.1. Der durch 4.2.1 konstruierte Baum T_n ist gleichverteilt auf der Menge aller Spannbäume von K_n: $P(\mathcal{T}_n = t) = 1/n^{n-2}$ für alle Spannbäume t von K_n.

Beweis. Die Familie $\mathbf{Z} = (Z_i)_{i \geq 0}$ sei iid auf der Knotenmenge V. Die Familie $(\xi_j)_{1 \leq j \leq n}$ bezeichnet die Indizes der Familie $\mathbf{Z}$, an denen ein Knoten aus V das erste mal vorkommt. Das lässt sich definieren als:

$$\begin{aligned} \xi_1 &= 0 \\ \forall 1 \leq j \leq n : \xi_j &= \min\{i > 0 | Z_i \notin \{Z_0, \dots, Z_{\xi_{j-1}}\}\} \end{aligned} \tag{4.6}$$

Die Familie $(\pi_j)_{1 \leq j \leq n}$ seien die Konten von V, in der Reihenfolge, in der ein Knoten als Zustand von $\mathbf{Z}$ das erste mal vorkommt.

$$\begin{aligned} \pi_1 &= Z_0 \\ \forall 1 \leq j \leq n : \pi_j &= Z_{\xi_j} \end{aligned} \tag{4.7}$$

Die Familie $\mathbf{L} := (L_k)_{2 \leq k \leq n}$ sei die Familie der Knoten der Vorgänger der Knoten $(Z_{\xi_j})_{2 \leq j \leq n}$. (Der erste Knoten, $Z_{\xi_1} = Z_0$, hat keinen Vorgänger) Genauer:

$$\forall 2 \leq k \leq n : L_k = Z_{\xi_k - 1}$$

```python
def algorithm2(graph: nx.Graph, seed: int = 42) -> nx.Graph:
    number_of_nodes = len(graph)
    assert len(graph.edges) == number_of_nodes*(number_of_nodes-1)/2,\
        "graph is not complete"
    random.seed(seed)
    u = [random.choice(list(graph)) for _ in range(2, len(graph))]
    T = nx.Graph()
    nodes = [str(i) for i in range(len(graph))]
    T.add_edge(nodes[0],nodes[1])
    for i in range(2,len(graph)):
       T.add_edge(nodes[i],str(min(int(u[i-2]),i-1)))
    random.shuffle(nodes)
    permutation = dict(zip(T.nodes, nodes))
    T = nx.relabel_nodes(T, permutation)
```

Abbildung 4.5: Implementierung des Algorithmus 4.2.1 in Python

Wir betrachten den Baum, in dem für alle $j = 1, \dots, n-1$ der Knoten π_{j+1} mit dem Knoten L_{j+1} durch eine Kante verbunden ist. Der Unterschied ist, dass die Familie $\mathbf{Z}$ kein Random Walk ist, da ein Knoten in der Familie $\mathbf{Z}$ mehrmals hintereinander vorkommen kann. Verwendet man für die Konstruktion des Baumes die Familie $\mathbf{Z}'$, in der alle Zustände Z_i entfernt wurden, die mit Z_{i-1} übereinstimmen, entsteht der selbe Graph. Da die Reihenfolge der entdeckten Knoten gleich bleibt, ändert das entfernen dieser Z_i nichts an der Familie $\boldsymbol{\pi}$. Es ändert sich nur die Familie $\boldsymbol{\xi}$. Diese neue Familie $\mathbf{Z}'$ ist dadurch, dass es keine aufeinanderfolgenden Zustände gibt, ein Random Walk auf dem vollständigen Graphen K_n. Da wir jedes π_i für $i \geq 2$ mit seinem vorgänger verbinden, entspricht also der Algorithmus dem Groundskeeper Algorithmus 4.1. Dadurch ist der Baum gleichverteilt auf der Menge aller Bäume von G.

Algorithmus 4.2.2.

(i) für alle $j = 1, \dots, n-1$ verbinde v_{j+1} mit $\pi^{-1}(L_{j+1})$.

(ii) benenne die Knoten $v_1, \dots, v_n$ in $\pi_i, \dots \pi_n$ um.

Hier bezeichnet die Funktion π die durch die π_i in 4.7 definierte Permutation.

$$\begin{pmatrix} v_1 & v_2 & \dots & v_n \\ \pi_1 & \pi_2 & \dots & \pi_n \end{pmatrix}$$

Man verifiziert leicht, das 4.2.2 den selben Algorithmus wie der Groundskeeper Algorithmus beschreibt: Wendet man (ii) auf die in (i) zu verbindenden Knoten an, so verbindet man für alle $j = 1, \dots, n-1$ den Knoten $v_{\pi_{j+1}}$ mit dem Knoten $v_{L_{j+1}}$. Um zu beweisen, dass 4.2.2 den gleichen Algorithm wie 4.2.1 beschreibt, vergleichen wir die Wahrscheinlichkeiten. Wir halten ein beliebiges $j \geq 1$ und einen beliebigen Prozess $(Z_i : i \leq \xi_j)$ bis zum Index ξ_j fest.

Die Wahrscheinlichkeit, dass Z_{ξ_j+1} ein Knoten ist, der nicht in $\{\pi_1, \dots, \pi_j\}$ vorkommt, ist $1 - \frac{j}{n}$. Das ist gleichbedeutend mit $\xi_{j+1} = \xi_j + 1$ und $L_{j+1} = \pi_j$ und $\pi^{-1}(L_{j+1}) = v_j$. Wird π_{j+1} nicht direkt nach π_j entdeckt, so gibt es ein $M \geq 1$ mit $\xi_{j+1} = \xi_j + M + 1$. Die Knoten auf dem Abschnitt $(Z_{\xi_j}, \dots, Z_{\xi_j+M} = Z_{\xi_{j+1}-1})$ sind unabhängig voneinander und gleichverteilt auf der Menge der Knoten zuvor besuchten Knoten $\{\pi_1, \dots, \pi_j\}$. Somit ist für den Fall $\xi_{j+1} = \xi_j + M + 1$ mit $M \geq 1$, L_{j+1} gleichverteilt auf der Menge der Knoten $\{\pi_1, \dots, \pi_j\}$ und somit $\pi^{-1}(L_{j+1})$ gleichverteilt auf der Menge $\{1, \dots, j\}$.

Zusammen ergibt sich für die Wahrscheinlichkeiten:

$$
\begin{aligned}
&\forall i \in \{1, \dots, j-1\} : \\
&\mathbb{P}(\pi^{-1}(L_{j+1}) = v_i | Z_{\xi_1}, \dots, Z_{\xi_j}) = \mathbb{P}(L_{j+1} = \pi_i | Z_{\xi_1}, \dots, Z_{\xi_j}) = \frac{1}{n} \\
&\mathbb{P}(\pi^{-1}(L_{j+1}) = v_j | Z_{\xi_1}, \dots, Z_{\xi_j}) = \mathbb{P}(L_{j+1} = \pi_j | Z_{\xi_1}, \dots, Z_{\xi_j}) \\
&\qquad = \mathbb{P}(L_{j+1} = \pi_j | Z_{\xi_1}, \dots, Z_{\xi_j} \wedge Z_{\xi_j+1} = Z_{\xi_{j+1}}) \\
&\qquad\quad + \mathbb{P}(L_{j+1} = \pi_j | Z_{\xi_1}, \dots, Z_{\xi_j} \wedge Z_{\xi_j+M+1} = Z_{\xi_{j+1}}) \\
&\qquad = 1 - \frac{j}{n} + \frac{1}{n} = 1 - \frac{j-1}{n}
\end{aligned}
$$

Die Wahrscheinlichkeiten stimmen somit mit 4.2.1 überein. Die generierten Bäume sind also gleichverteilt auf der Menge aller Spannbäume von K_n.

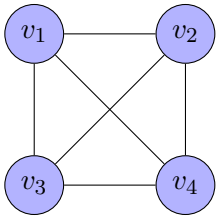
□

Der Punkt ist, dass wir für den Groundskeeper Algorithmus nur einen endlichen Teil des Random Walks brauchen um den Spannbaum zu konstruieren. Wir benötigen nur die Reihenfolge der Entdeckung der Knoten und von welchem Knoten aus ein Knoten entdeckt wurde. Mit 4.5 können wir diese Eckpunkte des Random Walks direkt generieren, ohne einen ganzen Random Walk durchlaufen zu müssen.

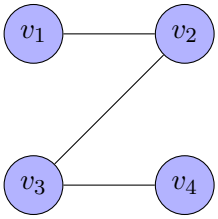
Beispiel 4.2.0.1. Als Beispiel betrachten wir den vollständigen Graphen K_4.

i	0	1	2	3	4	5	6	7	8	9	10	11	12
Z_i	v_2	v_1	v_2	v_2	v_1	v_1	v_2	v_3	v_3	v_4	v_1	v_1	v_3
π_i		v_2	v_1	v_3	v_4								
ξ_i		0	1	7	9								
L_i			v_2	v_2	v_3								

Abbildung 4.6: Beispiel für die Familie $\mathbf{Z}$ und die daraus konstruierten Familien $\boldsymbol{\xi}$, $\boldsymbol{\pi}$ und $\boldsymbol{L}$



(a) vollständiger Graph mit 4 Knoten



(b) Spannbaum von K_4 durch 4.6

5 Blätter von Spannbäumen

In den vorangegangenen Kapiteln, Kapitel 3 und 4, haben wir detailliert dargestellt, wie man zufällige Graphen und Spannbäume generieren kann. Die Untersuchung von Eigenschaften zufälliger Graphen ist von besonderem Interesse, da sie als eine Art Nullhypothese für die Struktur realer Systeme dienen können. Beispielsweise finden wir solche Systeme in Straßennetzen, sozialen Netzwerken und im Internet. In diesem Kapitel steht die Analyse der Anzahl von Blättern in zufälligen Spannbäumen im Fokus. Ein Blatt in einem Graphen ist ein Knoten, der lediglich mit einem einzigen anderen Knoten verbunden ist. Unser Ziel ist es, ausgehend von einem r-regulären Graphen, die Wahrscheinlichkeit abzuschätzen, dass ein bestimmter Knoten ein Blatt ist. Hierzu werden wir den Parameter r nutzen, um diese Schranken zu bestimmen.

5.1 Oberer Schranke

Proposition 5.1.0.1 (Proposition 5)**.** Bezeichne mit $\mathcal{T}$ die gleichverteilte Zufallsvariable auf den Spannbäumen eines zusammenhängenden r-regulären Graphen $G = (V, E)$ mit $r > 3$. Sei v ein beliebiger Knoten von G, dann gilt:

$$\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) \leq \exp(-\frac{r-1}{2r})$$

[Ald90, Proposition 5]

Beweis. Sei $(X_j; j \geq 0)$ der Random Walk aus 4.1. Sei $v \in V$ beliebig, fest. Sei $N(v)$ wie in 2.1.1 die Menge der Nachbarn von v. Bezeichne mit α_i den Index j von X_j des i-ten Besuchs der Menge $N(v)$.

$$\begin{aligned} \alpha_1 &= \min\{j \geq 0 | X_j \in N(v)\} \\ \alpha_{i+1} &= \min\{j > \alpha_i | X_j \in N(v)\} \end{aligned} \tag{5.1}$$

Sei $\mathcal{T}_i = (V, E_i)$ der durch den Random Walk X_j bis zum index α_i induzierte Spannbaum, also der, der durch den Groundskeeper Algorithmus vom Random Walk $(X_j; j \leq \alpha_i)$ erzeugt wird. Bezeichne mit $\deg(v, \mathcal{T}_i)$ den Grad vom Knoten v im Graphen $\mathcal{T}_i$. Definiere:

$$D_i = \begin{cases} 0 & \text{falls } \deg(v, \mathcal{T}_i) = 0 \\ \deg(v, \mathcal{T}_i) - 1 & \text{sonst} \end{cases}$$

Bemerkung 5.1.0.1.

$$0 \leq D_i \leq r - 1$$

da G ein r-regulärer Graph ist und somit $\deg(v, \mathcal{T}_i) \leq r$ gilt.

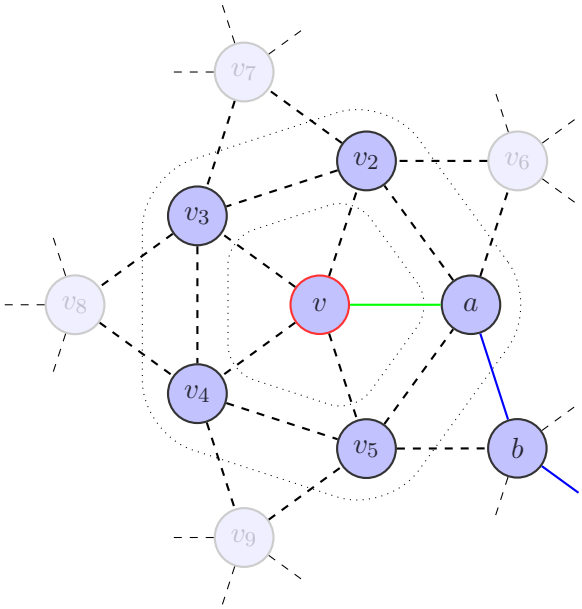


Abbildung 5.1: Random Walk auf dem 5-regulären Graphen mit möglichen Bäumen $\mathcal{T}_1$ in grün und blau

Da im Baum $\mathcal{T}_1$ nur ein Knoten aus $N(v)$ nämlich $a \coloneqq X_{\alpha_1}$ besucht wurde, kann a nur entweder von v oder einem Knoten $b \in V \setminus (N(v) \cup \{v\})$ erreicht worden sein. Im ersten Fall gilt $\{v, a\} \in E_1$ und der Baum T_1 besteht nur aus den Knoten $V = \{v, a\}$ und der Kantenmenge $E_1 = \{\{v, a\}\}$. Im zweiten Fall gilt $\{b, a\} \in E_1$.

$$\{v, a\} \in E_1 \Rightarrow \deg(v, \mathcal{T}_1) = 1 \Rightarrow D_1 = 0$$
$$\{b, a\} \in E_1 \Rightarrow \deg(v, \mathcal{T}_1) = 0 \Rightarrow D_1 = 0$$

Und somit gilt $D_1 = 0$.

Bemerkung 5.1.0.2.

$$E_i \subseteq E_{i+1} \Rightarrow D_i \leq D_{i+1} \tag{5.2}$$

wobei Gleichheit bei $E_i = E_{i+1}$ gilt.

Sei $i \geq 1$ und $X_{\alpha_{i+1}} = a \in N(v)$. Es gibt 3 Möglichkeiten, wie a erreicht werden kann:

1. a wird von v erreicht und $\{v, a\} \in E_{i+1} \setminus E_i \Rightarrow D_{i+1} = D_i + 1$
2. a wird von v erreicht und $\{v, a\} \in E_i \Rightarrow D_{i+1} = D_i$
3. a wird von einem Knoten $b \in V \setminus (N(v) \cup \{v\})$ erreicht. $\Rightarrow D_{i+1} = D_i$

Somit gilt:

Bemerkung 5.1.0.3.

$$D_{i+1} \in \{D_i, D_i + 1\}$$

Definition 5.1.1.

$$\Gamma(\alpha_i) \coloneqq |\{X_j \in N(v) | j \leq \alpha_i\}|$$

$\Gamma(\alpha_i)$ ist also die Menge der unterschiedlichen Knoten in $N(v)$, die bis zum Zeitpunkt α_i besucht wurden.

Bemerkung 5.1.0.4.

$$\Gamma(\alpha_i) \subseteq \Gamma(\alpha_{i+1})$$

In einem r-regulären Graphen gilt:

$$|N(v)| = r$$

somit

Falls v ein Blatt von $\mathcal{T}$ ist, dann gilt $\forall i \geq 1 : D_i = 0$ und wegen der Monotonie von 5.2 gilt:

$$\forall i \geq 1 : \mathbb{P}(D_i = 0) \geq \mathbb{P}(D_{i+1} = 0)$$

Somit gilt:

$$\forall i > r : \mathbb{P}(D_r = 0) \geq \mathbb{P}(D_i = 0)$$

Woraus

$$\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) \leq \mathbb{P}(D_r = 0) \tag{5.3}$$

folgt.

Lemma 5.1.1.

$$\begin{aligned}&\forall j \geq 1 \; \forall 1 \leq i < j : \\ &\mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_{j-i} = 0) = \mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_1 = 0)\end{aligned}$$

Beweis.
"$\geq$": folgt sofort aus

$$(D_j = 0, D_{j-1} = 0, \ldots, D_1 = 0) \Rightarrow (D_j = 0, D_{j-1} = 0, \ldots, D_{j-i} = 0)$$

"$\leq$": Sei $j \geq 1$ und $1 \leq i < j$. Wir nehmen an , dass $\mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_1 = 0) < \mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_{j-i} = 0)$ gilt. Es gilt:

$$\begin{aligned}&\mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_1 = 0) < \mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_{j-i} = 0) \\ \Leftrightarrow &\mathbb{P}(D_{j-i-1}, \ldots, D_1 = 0 | D_j = 0, \ldots, D_{j-i} = 0) \mathbb{P}(D_j, \ldots, D_{j-i} = 0) \\ &\mathbb{P}(D_j = 0, D_{j-1} = 0, \ldots, D_{j-i} = 0) \\ \Leftrightarrow &\mathbb{P}(D_{j-i-1}, \ldots, D_1 = 0 | D_j = 0, \ldots, D_{j-i} = 0) < 1 \\ \Leftrightarrow &1 < 1\end{aligned}$$

Widerspruch! □

Somit gilt:

$$\begin{aligned}\mathbb{P}(D_j = 0) &= \mathbb{P}(D_j = 0, D_{j-1} = 0) \\ &= \mathbb{P}(D_j = 0 | D_{j-1} = 0)\mathbb{P}(D_{j-1} = 0)\end{aligned}$$

und dadurch

$$\mathbb{P}(D_j = 0) = \prod_{i=1}^{j-1} \mathbb{P}(D_{i+1} = 0 | D_i = 0) \tag{5.4}$$

Wir betrachten nun die Wahrscheinlichkeit, dass $D_{i+1} = 0$ gegeben $D_i = 0$. Dazu ist es leichter die Gegenwahrscheinlichkeit $\mathbb{P}(D_{i+1} = 1 | D_i = 0)$ zu betrachten. In dieser Situation befindet sich der Random Walk an einem Knoten $X_{\alpha_i} \in N(v)$ und uns interessiert, mit welcher Wahrscheinlichkeit der Random Walk im nächsten Schritt v besucht und im übernächsten Schritt einen Knoten besucht, der noch nicht entdeckt wurde. Die Wahrscheinlichkeit, dass der Random Walk im nächsten Schritt v besucht ist $\frac{1}{r}$ da jeder Knoten r Nachbarn hat. Die Wahrscheinlichkeit, dass der Random Walk im übernächsten Schritt einen Knoten besucht, der noch nicht entdeckt wurde ist $1 - \Gamma(\alpha_i)/r$, da bereits $\Gamma(\alpha_i)$ Knoten um v entdeckt wurden. Es gilt also:

$$\mathbb{P}(D_{i+1} = 1 | D_i = 0) = \frac{1}{r}(1 - \frac{\Gamma(\alpha_i)}{r})$$

und daher

$$\mathbb{P}(D_{i+1} = 0 | D_i = 0) = 1 - \frac{1}{r}(1 - \frac{\Gamma(\alpha_i)}{r}) \tag{5.5}$$

Mit 5.4, 5.5 folgt für $2 \leq j \leq r$:

$$\begin{aligned}\mathbb{P}(D_j = 0) &= \prod_{i=1}^{j-1} \left(1 - \frac{1}{r}(1 - \frac{\Gamma(\alpha_i)}{r})\right) \\ &\leq \prod_{i=1}^{j-1} \left(1 - \frac{1}{r}(1 - \frac{i}{r})\right)\end{aligned} \tag{5.6}$$

wobei wir dabei genutzt haben, dass $\forall i \geq 1 : \Gamma(\alpha_i) \leq i$ gilt. Das sieht man leicht, da nach i besuchen der Menge $N(v)$ maximal i verschiedene Knoten aus dieser Menge besucht wurden. Mit 5.3 folgt:

$$\begin{aligned}\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) &\leq \mathbb{P}(D_r = 0) \\ &\leq \prod_{i=1}^{r-1} \left(1 - \frac{1}{r}(1 - \frac{i}{r})\right)\end{aligned}$$

Für den letzten Schritt benutzen wir die Tatsache, dass $\forall y \in \mathbb{R} : 1 - y \leq e^{-y}$ gilt. Sei

$y_i = 1/r(1 - i/r)$, dann gilt:

$$\begin{aligned}
\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) &\leq \prod_{i=1}^{r-1} \left(1 - \frac{1}{r}(1 - \frac{i}{r})\right) \\
&= \prod_{i=1}^{r-1} (1 - y_i) \\
&\leq \prod_{i=1}^{r-1} e^{-y_i} \\
&= \exp(-\sum_{i=1}^{r-1} y_i)
\end{aligned}$$

Wir berechnen nun $\sum_{i=1}^{r-1} y_i$:

$$\begin{aligned}
\sum_{i=1}^{r-1} y_i &= \frac{1}{r} \sum_{i=1}^{r-1} \left(1 - \frac{i}{r}\right) \\
&= \frac{r-1}{r} - \frac{1}{r^2} \sum_{i=1}^{r-1} i \\
&= \frac{r-1}{r} - \frac{1}{r^2} \frac{r(r-1)}{2} \\
&= \frac{r-1}{r} - \frac{r-1}{2r} \\
&= \frac{r-1}{2r}
\end{aligned}$$

Und somit gilt:

$$\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) \leq \exp(-\frac{r-1}{2r})$$

□

5.2 Untere Schranke

Wie der Graph in Abbildung 5.2 zeigt, kann es Knoten in einem Graphen geben, die nie ein Blatt in einem Spannbaum sein können. Aus diesem Grund können wir als untere Schranke für $\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T})$ nur 0 verwenden. Statt die Wahrscheinlichkeit für einen individuellen Knoten zu betrachten, schätzen wir die durchschnittliche Wahrscheinlichkeiten, für einen Knoten ein Blatt zu sein, ab.

Satz 5.2.1.

$$\text{ave}_{v \in G} \mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) \geq \alpha(r)$$

mit

$$\alpha(r) = \sum_{j=2}^{r-1} r^{-1}(1 - j/r)(1 - \prod_{i=1}^{j-1}(1 - r^{-1}(1 - i/r)))$$

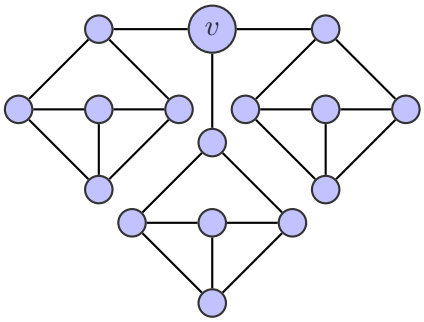


Abbildung 5.2: 3-regulärer Graph in dessen Spannbäumen v nie ein Blatt sein kann

Es bezeichnet

$$\text{ave}_{v\in G}\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) = \frac{1}{|V|}\sum_{v\in G}\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}).$$

Beweis. Wir bezeichnen mit $M = \min(i : D_i = 1)$ den ersten Index i in dem der Knoten v in dem durch den Random Walk definierten Baum $\mathcal{T}_i$ kein Blatt mehr ist.

Bemerkung 5.2.1.1. M ist wohldefiniert, da der Random Walk mit Wahrscheinlichkeit 1 jeden Knoten besucht.

Proposition 5.2.1.1. $\mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) = \mathbb{P}(M = \infty)$

Beweis. Da für den Fall dass v ein Blatt von $\mathcal{T}$ ist gilt, dass $\forall i \geq 1 : D_i = 0$ und somit $M = \min\{\} = \infty$. □

Mit $D_\infty = \deg(v, \mathcal{T}) - 1$ bezeichnen wir den Grenzwert von D_i für $i \to \infty$, also den Grad von v im fertigen Spannbaum $\mathcal{T}$ weniger 1. Wir können nun D_∞ ausdrücken als:

$$\begin{aligned} D_\infty &= \sum_{j=1}^{\infty}(D_{j+1} - D_j) \\ &= \mathbb{1}_{(M<\infty)} + \sum_{j=1}^{\infty}(D_{j+1} - D_j)\mathbb{1}_{(D_j>0)} \end{aligned}$$

Wir drücken den Erwartungswert für den grad von v im fertigen Spannbaum $\mathcal{T}$ durch D_∞ und M aus:

$$\begin{aligned} \mathbb{E}[D_\infty] &= \mathbb{E}[\mathbb{1}_{(M<\infty)}] + \mathbb{E}[\sum_{j=1}^{\infty}(D_{j+1} - D_j)\mathbb{1}_{(D_j>0)}] \\ &= \mathbb{P}(M < \infty) + \sum_{j=1}^{\infty}\mathbb{E}[(D_{j+1} - D_j)\mathbb{1}_{(D_j>0)}] \\ &= \mathbb{P}(M < \infty) + \sum_{j=1}^{\infty}\mathbb{E}[(D_{j+1} - D_j)]\mathbb{P}(D_j > 0) \end{aligned} \tag{5.7}$$

Den Erwartungswert $\mathbb{E}[(D_{j+1} - D_j)]$ können wir explizit anschreiben und für $1 \leq j \leq r$ abschätzen als:

$$\begin{aligned}\mathbb{E}[(D_{j+1} - D_j)] &= 0 \cdot \mathbb{P}(D_{j+1} - D_j = 0) + 1 \cdot \mathbb{P}(D_{j+1} - D_j = 1) \\ &= \mathbb{P}(D_{j+1} = D_j + 1) \\ &= \frac{1}{r}(1 - \frac{\Gamma(\xi_j)}{r}) \\ &\geq \frac{1}{r}(1 - \frac{j}{r})\end{aligned}$$

Somit folgt aus 5.7:

$$\mathbb{E}[D_\infty] \geq \mathbb{P}(M < \infty) + \sum_{j=1}^{r-1} \frac{1}{r}(1 - \frac{j}{r})\mathbb{P}(D_j > 0)$$

Mit Proposition 5.2.1.1 gilt:

$$\mathbb{E}[D_\infty] \geq \mathbb{P}(v \text{ ist kein Blatt von } \mathcal{T}) + \sum_{j=1}^{r-1} \frac{1}{r}(1 - \frac{j}{r})\mathbb{P}(D_j > 0) \tag{5.8}$$

$$\Leftrightarrow \quad \mathbb{E}[D_\infty] \geq 1 - \mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) + \sum_{j=1}^{r-1} \frac{1}{r}(1 - \frac{j}{r})\mathbb{P}(D_j > 0) \tag{5.9}$$

$$\Leftrightarrow \quad \mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) \geq 1 - \mathbb{E}[D_\infty] + \underbrace{\sum_{j=1}^{r-1} \frac{1}{r}(1 - \frac{j}{r})\mathbb{P}(D_j > 0)}_{\geq \alpha(r)} \tag{5.10}$$

Die Abschätzung für $\alpha(r)$ folgt aus 5.6:

$$\mathbb{P}(D_j > 0) = 1 - \mathbb{P}(D_j = 0) \geq 1 - \prod_{i=1}^{j-1} \left(1 - \frac{1}{r}(1 - \frac{i}{r})\right)$$

Wir können den Durchschnittlichen Erwartungswert $\text{ave}_v \mathbb{E}[D_\infty]$ berechnen:

Lemma 5.2.2.

$$\text{ave}_v \deg(v, \mathcal{T}) = \frac{1}{n} \sum_{v \in \mathcal{T}} \deg(v, \mathcal{T}) = \frac{1}{n} 2(n-1) = 2 - \frac{2}{n}$$

Somit folgt aus der Definition von D_∞:

$$\text{ave}_v \mathbb{E}[D_\infty] = 1 - \frac{2}{n}$$

Somit können wir bei 5.10 fortfahren und erhalten:

$$\begin{aligned}\text{ave}_v \mathbb{P}(v \text{ ist ein Blatt von } \mathcal{T}) &\geq 1 - (1 - \frac{2}{n}) + \sum_{j=1}^{r-1} \frac{1}{r}(1 - \frac{j}{r}) \left(1 - \prod_{i=1}^{j-1} \left(1 - \frac{1}{r}(1 - \frac{i}{r})\right)\right) \\ &= \frac{2}{n} + \alpha(r) \geq \alpha(r)\end{aligned}$$

□

5.3 Empirische Verteilung der Blätter

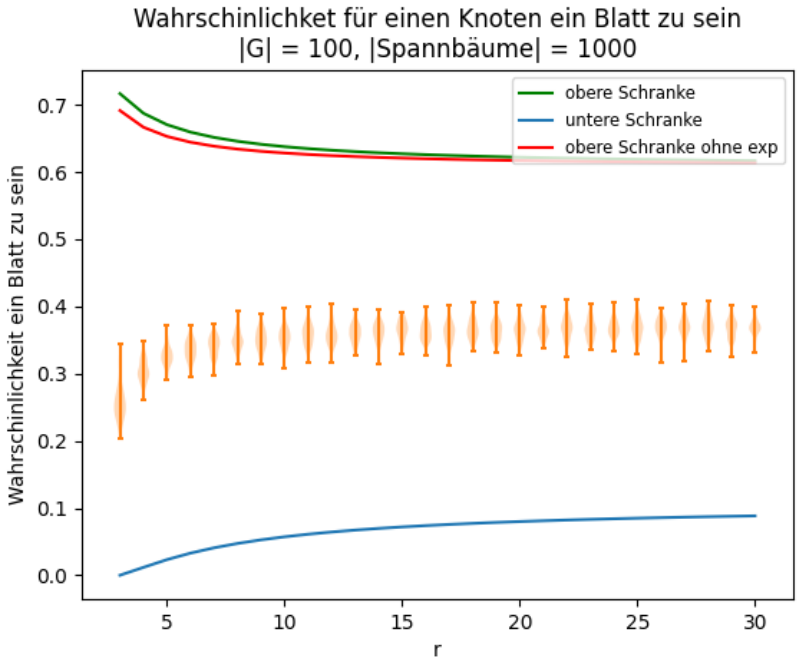


Abbildung 5.3: Wahrscheinlichkeit für einen Knoten in einem UST ein Blatt zu sein, in Abhängigkeit der Regularität der Graphen. Graph mit 100 Knoten, 1000 getestete UST pro Knoten

In Abbildung 5.3 ist die Wahrscheinlichkeit für einen Knoten in einem UST ein Blatt zu sein in Abhängigkeit der Regularität des Graphen mit 100 Knoten dargestellt. Jeder der 100 Knoten wurde mit 1000 zufälligen UST getestet, die mit Hilfe des Groundskeeper Algorithmus generiert wurden. TODO: Referenz auf Groundskeeper Algorithmus Die Wahrscheinlichkeiten erweisen sich als sehr stabil und liegen in etwas zwischen 0.3 und 0.4. Man sieht ebenfalls, das die Abschätzungen sehr großzügig waren, da die Schranken sehr weit weg liegen.

Literaturverzeichnis

[Aig15] Martin Aigner. *Problem und „Lösung“*, pages 3–16. Springer Fachmedien Wiesbaden, Wiesbaden, 2015.

[Ald90] David J Aldous. The random walk construction of uniform spanning trees and uniform labelled trees. *SIAM Journal on Discrete Mathematics*, 3(4):450–465, 1990.

[BB05] Vladimir Batagelj and Ulrik Brandes. Efficient generation of large random networks. *Phys. Rev. E*, 71:036113, Mar 2005.

[BHK20] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Random Walks and Markov Chains*, page 62–108. Cambridge University Press, 2020.

[Dan19] Will Dana. The groundskeeper's algorithm works. http://www-personal.umich.edu/~willdana/Electrees%20Notes.pdf, 2019. [Online; accessed 25-February-2022].

[Eis14] Friedrich Eisenbrand. Randomized algorithms. https://www.epfl.ch/labs/disopt/wp-content/uploads/2018/09/lec7.pdf, 2014. [Online; accessed 25-February-2022].

[Gil59] E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4):1141–1144, 1959.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[Kle13] Achim Klenke. *Wahrscheinlichkeitstheorie*. Springer Spektrum, 2013.

[MS05] David Meintrup and Stefan Schäffler. *Stochastik Theorie und Anwendungen*. Springer, 2005.

[NLKB11] Sadegh Nobari, Xuesong Lu, Panagiotis Karras, and Stéphane Bressan. Fast random graph generation. In *Proceedings of the 14th international conference on extending database technology*, pages 331–342, 2011.

[Pri18] Nicolas Privault. *Understanding markov chains examples and applications*. Springer Singapore, 2018.

[pyt] Python Documentation: random — Generate pseudo-random numbers. https://docs.python.org/3/library/random.html. Accessed: October 11, 2023.

Abbildungsverzeichnis