Universität Oldenburg

Fortgeschrittene Computerorientierte Physik

Verteilung der kürzesten Pfade in skalenfreien Graphen

Author:
Jan Kämper
Florian Börgel

Supervisor:
Alexander Hartmann

April 12, 2016

Contents

1 Problemstellung

Ziel dieses Projektes ist es, statistische Beobachtungen über die Verteilung von kürzesten Pfaden in skalenfreien Graphen sowie in ebenen Zufallsgraphen zu machen. In skalenfreien Graphen sind die Kanten pro Knoten nach einem Potenzgesetz verteilt:

$$P(k) \sim k^{-\alpha}$$

Die Erstellung von skalenfreien Graphen erfolgt durch spezielle Algorithmen. Der hier verwendete Algorithmus basiert auf dem Barabasi-Albert Model (siehe [1]) und nutzt die Methode Preferential Attachment. Dabei ist der Parameter m maßgeblich, der die Anzahl an Nachbarn eines neu hinzugefügten Knotens beschreibt.

Die Größe des Graphen wird über die Anzahl der Knoten n definiert. Um eine Aussage über die statistische Verteilung der kürzesten Pfade in verschiedenen Graphfamilien treffen zu können, müssen viele Simulationsläufe in Abhängigkeit der Parameter n und m durchgeführt werden, über die dann gemittelt wird. Zur Berechnung der kürzesten Wege wird auf den Floyd-Warshall Algorithmus zurückgegriffen, der in kubischer Laufzeit kürzeste Wege für alle Knotenpaare in einem ungerichteten Ausgangsgraphen liefert (siehe [2]). Experimente mit unterschiedlich parametrisierten Graphen wurden dann durch ein Fitting an Normalverteilung und Exponentialverteilung in Gnuplot ausgewertet.

In den Zusatzaufgaben wurde zum einen ein allgemeines preferential attachment umgesetzt, das zu einer veränderten Gradverteilung pro Knoten führt. Zum anderen wurde eine weitere Familie von Graphen betrachtet. Hierbei handelt es sich um ebene Zufallsgraphen in der $[0,1]^2$ Ebene, wobei Kanten zwischen zweien der gleichmäßig in der Ebene verteilten Knoten mit einer Wahrscheinlichkeit hinzugefügt werden, die kubisch in der euklidischen Distanz der Knoten abnimmt (hier: $f = 1, \alpha = 3$).

$$p_{ij} = f \cdot (1 + \sqrt{N \cdot \Pi} d_{ij}/\alpha)^{-\alpha}$$

Hier wurde analog zu den skalenfreien Graphen die Verteilung der kürzesten Weglängen untersucht, zusätzlich aber auch die mittlere Länge der kürzesten Wege in Abhängigkeit von der Größe des Graphen.

2 Softwarebeschreibung

Die wichtigsten Bestandteile des Programms werden hier kurz erläutert. Auf vollständige Methodensignaturen wird dabei verzichtet. Alle C-Metoden mit Ausnahme der Main-Funktionen sind in

der Datei *shortest_path_fragment.c* zu finden. Die Skripte und Gnuplot Funktionen befinden sich in den entsprechenden Ordnern.

2.1 Generierung der Graphen

Um die Problemstellung wie beschrieben bearbeiten zu können, werden zunächst Routinen benötigt, die Graphen des gewünschten Typs (skalenfreier Graph oder ebener Zufallsgraph) erzeugen. Die dafür benötigten Datenstrukturen für ungerichtete Graphen wurden aus der Vorlesung übernommen (Dateien *graph_lists.h, lists.h, lists.c*). Darin sind Graphen durch Adjazenzlisten mit einfach verketteten Listen realisisert. Für die Zusatzaufgabe wurde die Datenstruktur für Knoten um die x- und y-Koordinate ergänzt. Weitere wesentliche Graphoperationen (*gs_insert_edge, gs_create_graph, gs_edge_exists, gs_preferential_attachment*) konnten in der Datei *shortest_path_fragment.c* ebenfalls übernommen werden. Für die Erzeugung der ebenen Zufallsgraphen wurde die Methode *gs_create_planar_graph* geschrieben, welche n zufällig verteilte Knoten und Kanten entsprechend der Wahrscheinlichkeit $p_{ij} = f \cdot (1 + \frac{\sqrt{N\Pi} \cdot d_{ij}}{\alpha})^{-\alpha}$ erzeugt.

Bei der Erzeugung der skalenfreien Graphen wird nach dem Prinzip des preferential attachment vorgegangen. Knoten werden dabei sequentiell hinzugefügt und für jeden neuen Knoten m Nachbarn zufällig ausgesucht. Die Wahrscheinlichkeit einen existierenden Knoten dabei als Nachbarn zu wählen ist proportional zu dessen Grad (Anzahl an Kanten). Somit werden Knoten mit vielen Nachbarn bevorzugt, was letzlich zur gewünschten Gradverteilung im finalen Graphen führt. Um diese Auswahlwahrscheinlichkeit umzusetzen wird ein sogenanntes Pick-Array eingesetzt in dem jeder Knoten für jede seiner anliegenden Kanten genau einmal vorkommt:

```
max_pick = 2*m*g->num_nodes- m*(m+1);
pick = (int *) malloc(max_pick*sizeof(int));
num_pick=0;
for(n1=0; n1<m+1; n1++) /* start: complete subgraph of m+1 nodes */
    for(n2=n1+1; n2<m+1; n2++)
    {
        gs_insert_edge(g, n1, n2);
        pick[num_pick++] = n1;
        pick[num_pick++] = n2;
    }
for(n1=m+1; n1<g->num_nodes; n1++)  /* add other nodes */
{
    t=0;
    while(t<m)  /* insert m edges */
    {
        do
            n2 = (int) pick[(int) floor(drand48()*num_pick)];
```

```
                while(n2==n1);              /* chose pair of different nodes */
                if(!gs_edge_exists(g, n1, n2))
                {
                    gs_insert_edge(g, n1, n2);
                    pick[num_pick++] = n1;
                    pick[num_pick++] = n2;
                    t++;
                }
            }
        }
        free(pick);
```

2.2 Ausgabe

Zur Kontrolle der einzelnen Berechnungsschritte und abschließenden Ausgabe der Simulationsergebnisse wurden mehrere Funktionen implementiert, die Graphen, Matrizen oder Histogramme auf der Konsole oder in Dateien ausgeben. Die Methode *exportGraphDot* schreibt eine Ausgabedatei, welche von dem *GraphViz*-Tool *DOT* gelesen werden kann und durch die der Graph visualisiert wird. So konnten in frühen Stadien der Entwicklung bereits einfach Programmfehler erkannt und behoben werden. Zum gleichen Zweck dient *printedges*, die alle Kanten des Graphs in einfacher Form in eine Ausgabedatei schreibt. Um die berechnete Distanzmatrix mit allen kürzesten Weglängen im passenden Format auszugeben kann die Methode *printDistances* verwendet werden.

Die Ergebnisse der Simulationsläufe zur weiteren statistischen Analyse werden mit den Methoden *printHistogram* und *printHistogramNormed* in Dateien geschrieben. Die nicht-normierte Variante schreibt die einzelnen Einträge des Histograms zeilenweise in eine Datei. Die normierte Variante führt zusätzlich eine Normierung des Gesamtgewichts des Histograms auf 1 durch. Der Parameter *startindex* kann verwendet werdeno, um zum Beispiel die Histogram-Einträge mit Distanz 0 auszublenden.

2.3 Berechnung der kürzesten Wege

Als Schritt nach der Graph-Erzeugung müssen für den angelegten Graph die kürzesten Wege berechnet werden. D.h. für den gegebenen Graphen muss für jede Kombination von zwei Knoten der jeweilige kürzeste Abstand über das Netzwerk bestimmt werden. Das Ergebnis ist eine quadratisch symmetrische Matrix, die in jeder Zelle den kürzesten Abstand des Knotenpaars (Spalte und Zeile) enthält. Um überhaupt eine Aussage über die Distanz treffen zu können, muss den Kanten zunächst eine Wertigkeit bzw. Länge zugeschrieben werden. Im Falle der skalenfreien Graphen wurde überall der Abstand $d_{ij} = 1$ angenommen, sodass die Länge eines Weges der Anzahl an Kanten auf dem Weg entspricht. Im Falle der ebenen Graphen wurde die euklidische Distanz betrachtet. Für die

Berechnung der kürzesten Wege sollte laut Aufgabenstellung der Floyd-Warshall Algorithmus implementiert werden. Dieser ist in der Methode *gs_all_pair_shortest_paths* implementiert. Zunächst wird darin die Initialisierung der Distanzmatrix vorgenommen. Diese kann gewichtet (euklidisch) oder ungewichtet sein (skalenfreier Graph). Danach folgt die dreifach verschachtelte Schleife von Floyd-Wahrshall, welche die als Parameter vorhandene Distanzmatrix ausfüllt. Die drei entscheidenden verschachtelten Schleifen sind wenig komplex:

```
//preceding: initializations
for ( k=0; k<g->num_nodes;k++)
{
    for ( i=0; i<g->num_nodes;i++)
    {
        for ( j=0; j<g->num_nodes;j++)
        {
            if (dist[i][j] > dist[i][k]  + dist[k][j])
            {
                dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
}
```

Eine ausführliche Beschreibung des Algorithmus und Nachweis der Korrektheit ist in siehe [2] zu finden.

2.4 Aufbereitung der Simulationsdaten in Histogrammen

Wie bereits im vorherigen Kapitel beschrieben, müssen mehrere Durchläufe mit immer neu generierten Graphen durchgeführt werden, um eine statistische Aussage über die Verteilung der kürzesten Wege treffen zu können. Für das Programm bedeutet das, dass jeder Durchlauf und die dabei berechneten kürzesten Wege gespeichert und in ein Histogramm sortiert werden müssen. Für die Sortierung wird gezählt wie häufig eine berechnete Strecke innerhalb des Graphen vorkommt. Bei den skalenfreien Graphen sind alle Distanzen diskrete Werte (Kantengewichte 1) und damit jede Ganzzahl ein Histogram-Bin. Bei den ebenen Zufallsgraphen können beliebige Längen vorkommen. Somit wird die Anzahl an Bins mit der aus der Literatur bekannten Vorschrift $numBins = \sqrt{n}$ angenommen und die kürzesten Wege dementsprechend in gleichgroße Bins sortiert. Eine Alternative wäre die Regel von Sturge, siehe [3]. Auf beide Weisen wird ein Histogramm erstellt, welches die Verteilung der kürzesten Wege enthält. Die Methoden *fillHistogramDiscrete* und *fillHistogramContinuous* tun genau dies.

Im letzten Teil der Zusatzaufgabe soll anstelle der Verteilung aller kürzesten Wege die mittlere kürzeste Weglänge in Abhängigkeit der Größe n untersucht werden. Dies übernimmt die Funktion

computeMeanShortestPath, welche alle existierenden kürzesten Wege im gegebenen Graphen zählt und deren Längen mittelt.

2.5 Simulationssteuerung

Die Steuerung der Simulationsläufe erfolgt in den Methoden *runExperiments* (skalenfreie Graphen), *runExperimentsConstant* (skalenfreie Graphen mit allgemeinem preferential attachment), *runExperimentsPlanar* (ebene Zufallsgraphen) und *runMeanExperimentsPlanar* (mittlere Weglänge in ebenen Zufallsgraphen). Der Ablauf ist in allen Fällen gleich: Eine Schleife führt eine festgelegte Anzahl an Simulationsläufen durch. In jedem Lauf wird zunächst ein neuer Graph erzeugt, dann die kürzesten Wege berechnet und die Ergebnisse zwischengespeichert. Im Falle der Verteilungsuntersuchung wird zur Zwischenspeicherung das Histogram Array verwendet und über die Simulationsläufe hinweg hochgezählt. Im Falle der Mittelwertuntersuchung wird eine Double-Variable genutzt. Ausserdem gibt es hier noch eine äußere Schleife, die über unterschiedliche Graph-Größen iteriert. Nachdem alle Simulationsläufe beendet sind, erfolgt die Ausgabe der Ergebnisse in Dateiform. Die Main-Methode steht in der separaten Datei *shortest_path_main.c*. Hier werden die Nutzereingaben verarbeitet und entsprechend die gewünschten Simulationen durchgeführt anhand der Werte der Kommandozeilenparameter.

2.6 Automatisierung

Um die unterschiedlich parametrisierten Simulationen schnell neu anzustoßen (unterschiedliche Werte für die Parameter n und m) existieren die Skripte *simulationScalefree.scr* und *simulationPlanar.scr*. Diese rufen das kompilierte Programm mit allen benötigten Kommandozeilenparametern auf (n, ggf m, ggf k_0). Hier als Beispiel das Skript zur Untersuchung von skalenfreien Graphen:

```
#!/bin/bash

for N in 50 100 200 400
do
for M in 1 2
do
./shortest_path 1 $N $M
done
done
```

2.7 Auswertung mit Gnuplot

Basierend auf denen durch die Skripte erzeugten Dateien mit Endung .dat können Gnuplot Skripte verwendet werden, die die vorliegenden Daten graphisch darstellen und ein Fitting and die gewünschte

Verteilung durchführen. Diese Skripte befinden sich im Ordner *Plotscripts* und können in gnuplot mit dem *load* Befehl geladen werden.

3 Resultate

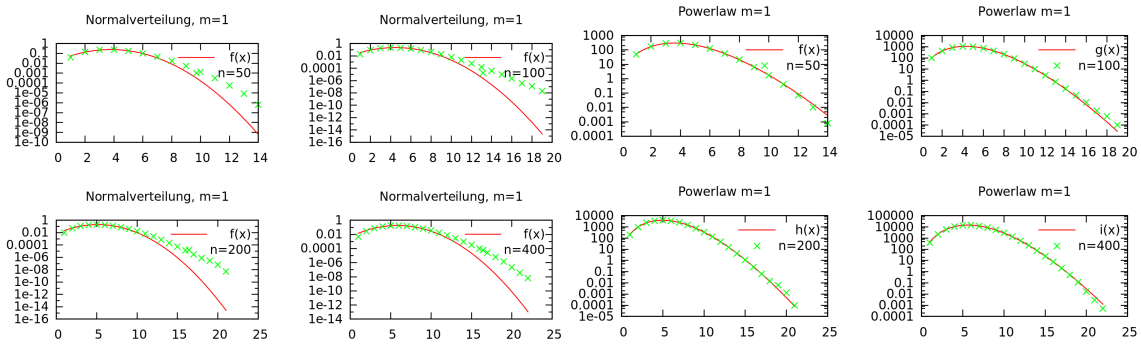
3.1 Skalenfreie Graphen: Normales preferential attachment

Für die skalenfreien Graphen wurde folgende Simulationen durchgeführt (Aufruf: $./simulationScalfree.scr$):

Größe n: $50, 100, 200, 400$

Parameter m: $1, 2$

Für $m = 1$ haben wir die Ergebnisse in Abbildung 1 erhalten. Die unterschiedliche Beschriftung der y-Achse ist der Tatsache geschuldet, dass für die Normalverteilung eine Normierung durchgeführt werden musste, was für das Potenzgesetz hier nicht notwendig war. Die Kurven sind aber in allen Abbildungen, die folgen, durch Skalierung identisch.



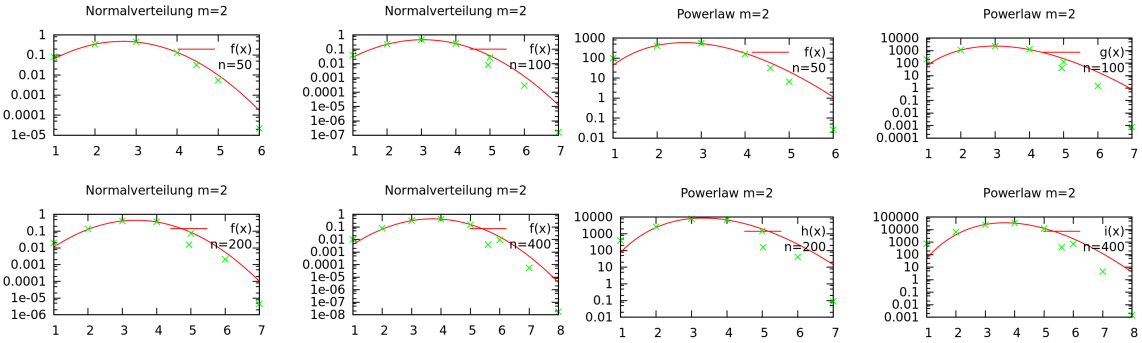
(a) Fitting an eine Normalverteilung

(b) Fitting an Potenzgesetz

Figure 1: Skalenfreie Graphen mit $m = 1$

Unabhängig von der Anzahl an Knoten scheint der Fit an das Potenzgesetz hier wesentlich besser zu sein. Dies ist im Falle $m = 2$ umgekehrt. Hier lässt sich feststellen, dass für ein Fitting an eine Normalverteilung geringere Abweichungen vorliegen. Die Ergebnisse sind in Abbildung 2 zu sehen.

Somit lässt sich also schlussfolgern, dass für $m = 1$ die Verteilung der kürzesten Wege im Graphen unabhängig von der Größe dem gleichen Gesetz folgt wie es für die Gradverteilung der Knoten gilt, nämlich dem Potenzgesetz. Ändert man den für das preferential attachment massgeblichen Parameter auf $m = 2$ so gilt dies nicht mehr, jetzt sind die kürzesten Weglängen im Graphen eher normalverteilt.



(a) Fitting an eine Normalverteilung (b) Fitting an Potenzgesetz

Figure 2: Skalenfreie Graphen mit $m = 2$

3.2 Skalenfreie Graphen: Allgemeines preferential attachment

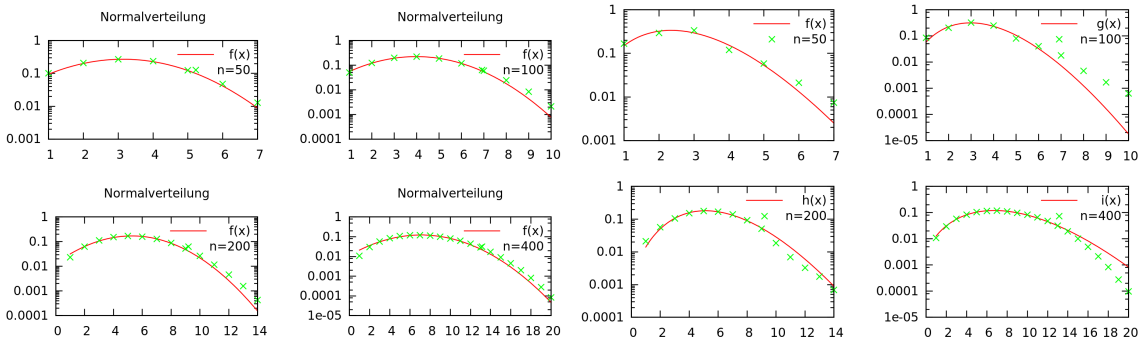
Flo Master

3.3 Ebene Zufallsgraphen

Für ebene Zufallsgraphen wurden die gleichen Simulationen wie für skalenfreie Graphen durchgeführt (Aufruf: *simulationPlanar.scr*). Der Parameter m spielt hier keine Rolle.

Größe n: $50, 100, 200, 400$

Es zeigt sich, dass auch hier eine Normalverteilung die bessere Approximation liefert (siehe Abbildung 3).



(a) Fitting an eine Normalverteilung (b) Fitting an Potenzgesetz

Figure 3: Ebene Zufallsgraphen

3.4 Mittlere Weglänge in ebenen Zufallsgraphen

Für ebene Zufallsgraphen wurde zusätzlich untersucht, wie sich die mittlere kürzeste Wegelänge in Abhängigkeit von der Anzahl der Knoten verhält. Es wurden also hier eine größere Menge an Graphgrößen untersucht:

Größe n: 10..1000, in 10er Schritten

Die mittlere Kürzeste-Weglänge konvergiert offenbar gegen einen Wert und der Fit an Potenzgesetze N^{β} scheint gut zu passen. Wir erhalten hier einen Wert von $B = -0.0741915$ mit einem Fehler von $+/- 0.001517$.

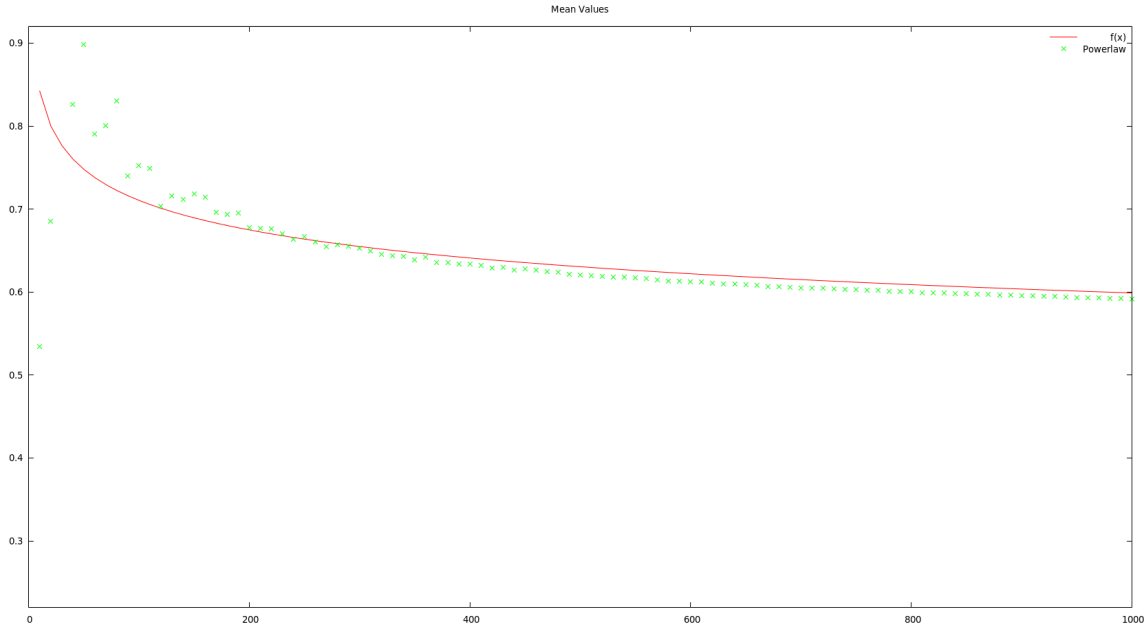


Figure 4: Fitting Power Law

4 Fazit

Lirum Larum Opsum

References

[1] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Mod. Phys.*, vol. 74, pp. 47–97, Jan 2002.

[2] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, pp. 345–, June 1962.

[3] H. A. Sturges, "The choice of a class interval," *Journal of the American Statistical Association*, vol. 21, no. 153, pp. 65–66, 1926.