

Universität Konstanz  
Fachbereich Informatik  
Prof. Dr. Ulrik Brandes/Christian Pich

Seminar 'Methoden der Netzwerkanalyse'  
Wintersemester 2005/2006

Ausarbeitung des Vortrags

# **ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs**

Hanna Kungl

basierend auf der gleichnamigen Arbeit von

*Christopher R. Palmer*

Computer Science Dept; Carnegie Mellon University Pittsburgh

*Phillip B. Gibbons*

Intel Research Pittsburgh

*Christos Faloutsos*

Computer Science Dept; Carnegie Mellon University Pittsburgh

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung und Definitionen</b>	<b>2</b>
<b>2</b>	<b>Exakte Berechnung</b>	<b>5</b>
2.1	Potenzieren der Adjazenzmatrix . . . . .	6
2.2	Breitensuche . . . . .	7
<b>3</b>	<b>Approximative Berechnung</b>	<b>7</b>
3.1	Probabilistische Zählmethode . . . . .	8
3.2	ANF-0 . . . . .	11
3.3	In-core ANF-0 . . . . .	13
3.4	ANF . . . . .	14
3.5	ANF-C . . . . .	15
<b>4</b>	<b>Experimentelle Auswertung</b>	<b>16</b>
4.1	Fehler . . . . .	16
4.2	Parameter . . . . .	16
4.3	Genauigkeit . . . . .	17
4.4	Geschwindigkeit und Skalierbarkeit . . . . .	18
<b>5</b>	<b>Anwendungen</b>	<b>20</b>
5.1	Knotenwichtigkeit . . . . .	20
5.2	Graphenähnlichkeit . . . . .	21
5.3	Teilgraphenähnlichkeit . . . . .	23

# 1 Einführung und Definitionen

Wenn man sehr große Graphen untersuchen möchte, stößt man schnell auf Grenzen, da viele Tools so konzipiert sind, daß entweder der gesamte Graph und zusätzlich benötigte Größen in den Speicher passen müssen oder daß die auf dem Graphen ausgeführten Operationen sehr teuer sind.

Das hier vorgestellte ANF ist ein sehr effizientes Tool, das zum ersten Mal erlaubt, daß riesige Graphen in einer relativ kurzen Zeit untersucht werden können. So sind die gewonnenen Erkenntnisse vor allem bei Graphen, die einem ständigen Wandel unterworfen sind, z.B. dem Internet, noch recht aktuell, wenn sie vorliegen.

ANF basiert auf einer schnellen, speichereffizienten und genauen Approximation der Nachbarschaftsfunktion, mit der viele wichtige Aspekte eines Graphen untersucht werden können.

Die Nachbarschaftsfunktion beschreibt das Wachstumsverhalten von Wegen zwischen Knotenpaaren in einem Graphen bei wachsender Weglänge und ist eine Art Verteilungsfunktion der kürzesten Wege zwischen Knotenpaaren.

Einige Fragen, die auf die Nachbarschaftsfunktion zurückgeführt werden können, lauten

1. Wie wichtig ist ein Knoten in Bezug auf Verbundenheit?
2. Haben zwei Graphen ähnliche Nachbarschaftsstruktur?
3. Wie sind zwei Teilgraphen innerhalb eines Graphen verbunden?

Was ist also die Nachbarschaftsfunktion?

Sei  $G = (V, E)$  ein gerichteter Graph mit  $n$  Knoten und  $m$  Kanten und sei  $dist(u, v)$  die Anzahl von Kanten auf dem kürzesten Weg von  $u$  nach  $v$ . Ungerichtete Graphen können dargestellt werden, indem man für eine existierende Kante zwei jeweils entgegengesetzt orientierte gerichtete Kanten einfügt.

## **Def 1**

Die individuelle Nachbarschaftsfunktion für  $u$  in  $h$  ist die Anzahl der Knoten mit Abstand  $h$  oder weniger von  $u$ .

$$IN(u, h) = |\{v : v \in V, dist(u, v) \leq h\}|$$

$IN(u, h)$  kann als ein Maß für die Verbundenheit eines Knotens im Graphen aufgefaßt werden.

Im Graphen aus Abbildung 1 sind z.B. die individuellen Nachbarschaftsfunktionen für

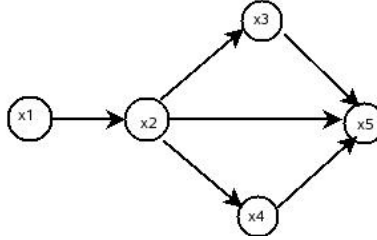


Abbildung 1: Beispielgraph 1

den Knoten  $x_1$  bei unterschiedlicher Distanz  $h$

$$IN(x_1, 0) = |\{v : v \in V, dist(x_1, v) \leq 0\}| = |\{x_1\}| = 1$$

$$IN(x_1, 1) = |\{v : v \in V, dist(x_1, v) \leq 1\}| = |\{x_1, x_2\}| = 2$$

$$IN(x_1, 2) = |\{v : v \in V, dist(x_1, v) \leq 2\}| = |\{x_1, x_2, x_3, x_4, x_5\}| = 5$$

Der Wert von  $IN(x_1, 3)$  ändert sich gegenüber dem von  $IN(x_1, 2)$  nicht mehr, da es zwar von  $x_1$  aus noch Wege der Länge 3 gibt, die Zielknoten (beides Mal  $x_5$ ) aber schon in der Menge von  $IN(x_1, 2)$  enthalten sind, d.h. die gefundenen Wege der Länge 3 sind keine kürzesten Wege zwischen den beiden Knoten, denn es existiert ein Weg der Länge 2, nämlich  $x_1x_2x_5$ .

### Def 2

Die Nachbarschaftsfunktion in  $h$  ist die Anzahl der Knotenpaare mit Abstand  $h$  oder weniger.

$$N(h) = |\{(u, v) : u \in V, v \in V, dist(u, v) \leq h\}| \text{ bzw. } N(h) = \sum_{u \in V} IN(u, h)$$

Die Nachbarschaftsfunktion für verschiedene Distanzen  $h$  für den Graphen aus Abbildung 1 lautet

$$\begin{aligned} N(0) &= |\{(u, v) : u \in V, v \in V, dist(u, v) \leq 0\}| \\ &= |\{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x_5, x_5)\}| = 5 \\ N(1) &= |\{(u, v) : u \in V, v \in V, dist(u, v) \leq 1\}| \\ &= |\{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x_5, x_5), \\ &\quad (x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_3, x_5), (x_4, x_5)\}| = 11 \\ N(2) &= |\{(u, v) : u \in V, v \in V, dist(u, v) \leq 2\}| \\ &= |\{(x_1, x_1), (x_2, x_2), (x_3, x_3), (x_4, x_4), (x_5, x_5), \\ &\quad (x_1, x_2), (x_2, x_3), (x_2, x_4), (x_2, x_5), (x_3, x_5), (x_4, x_5), \\ &\quad (x_1, x_3), (x_1, x_4), (x_1, x_5)\}| = 13 \end{aligned}$$

Der Wert von  $N(3)$  ändert sich gegenüber  $N(2)$  nicht mehr, da kein Knotenpaar existiert, für den der kürzeste verbindende Weg die Länge 3 hat.

Dieses Modell läßt sich so entweder nur auf einen Knoten oder den gesamten Graphen anwenden; oft sind allerdings Verbindungen zwischen zwei Teilgraphen von Interesse. Daher wird nun eine Verallgemeinerung der Nachbarschaftsfunktion auf Teilgraphen definiert.

Sei  $S$  eine Menge von Startknoten und  $C$  eine Menge von Endknoten.

**Def 3**

Die verallgemeinerte individuelle Nachbarschaftsfunktion für  $u$  in  $h$  bei gegebenem  $C$  ist die Anzahl der Knoten in  $C$ , die den Abstand  $h$  oder weniger von  $u$  haben.

$$IN^+(u, h, C) = |\{v : v \in C, dist(u, v) \leq h\}|$$

beachte:  $IN(u, h) = IN^+(u, h, V)$

Wählt man z.B. für den Graphen aus Abbildung 1  $u = x_1$  und  $C = \{x_4, x_5\}$ , so erhält man für verschiedene Distanzen  $h$

$$\begin{aligned} IN^+(x_1, 0, C) &= |\{v : v \in C, dist(u, v) \leq 0\}| = 0 \\ IN^+(x_1, 1, C) &= |\{v : v \in C, dist(u, v) \leq 1\}| = 0 \\ IN^+(x_1, 2, C) &= |\{v : v \in C, dist(u, v) \leq 2\}| = |\{x_4, x_5\}| = 2 \end{aligned}$$

Der Wert von  $IN^+(x_1, 3, C)$  ändert sich gegenüber dem von  $IN^+(x_1, 2, C)$  nicht mehr, da die einzigen Wege der Länge 3 von  $x_1$  aus, nämlich  $x_1x_2x_3x_5$  und  $x_1x_2x_4x_5$ , kein neues Knotenpaar beinhalten.

**Def 4**

Die verallgemeinerte Nachbarschaftsfunktion in  $h$  ist die Anzahl der Knotenpaare mit einem Knoten in  $S$ , den anderen in  $C$  mit Abstand  $h$  oder weniger voneinander.

$$N^+(h, S, C) = |\{(u, v) : u \in S, v \in C, dist(u, v) \leq h\}|$$

bzw.  $N^+(h, S, C) = \sum_{u \in S} IN^+(u, h, C)$

beachte:  $N(h) = N^+(h, V, V)$

Wählt man für den Graphen aus Abbildung 1  $S = \{x_1, x_2\}$  und wieder  $C = \{x_4, x_5\}$ , so erhält man für verschiedene Distanzen  $h$

$$\begin{aligned} N^+(0, S, C) &= |\{(u, v) : u \in S, v \in C, dist(u, v) \leq 0\}| = 0 \\ N^+(1, S, C) &= |\{(u, v) : u \in S, v \in C, dist(u, v) \leq 1\}| = |\{(x_2, x_4), (x_2, x_5)\}| = 2 \\ N^+(2, S, C) &= |\{(u, v) : u \in S, v \in C, dist(u, v) \leq 2\}| \\ &= |\{(x_2, x_4), (x_2, x_5), (x_1, x_4), (x_1, x_5)\}| = 4 \end{aligned}$$

Mit demselben Argument wie oben ändert sich der Wert von  $N^+(3, S, C)$  gegenüber dem von  $N^+(2, S, C)$  nicht mehr.

Dabei ist offensichtlich, daß sich die Werte der verschiedenen Funktionen nur für Werte  $0 \leq h \leq d = \text{diam}(G)$ ,  $\text{diam}(G)$  der Durchmesser des Graphen  $G$ , ändern, für  $h \geq d$  aber identisch sind mit dem von  $d$ .

Mit diesen Funktionen können verschiedene Graphenbeziehungen ausgedrückt werden:

1. Knotenwichtigkeit:  
Wie wichtig ist ein Knoten in Bezug auf Verbundenheit?  
Benutze dazu  $IN(u, h)$ , um die Knotenwichtigkeit von  $u$  bzgl. Verbundenheit im Graphen zu charakterisieren.
2. Graphenähnlichkeit:  
Haben die beiden Graphen  $G_1 = (V_1, E_1)$  und  $G_2 = (V_2, E_2)$  ähnliche Nachbarschaftsstruktur?  
Berechne dazu  $N(h)$  für beide Graphen für alle  $h$  und vergleiche dann die beiden Nachbarschaftsfunktionen in allen Punkten.
3. Teilgraphenähnlichkeit:  
Wie sind die beiden von  $V_1, V_2 \subset V$  induzierten Teilgraphen innerhalb des Graphen  $G$  verbunden?  
Vergleiche dazu  $N^+(h, V_1, V_1)$  mit  $N^+(h, V_2, V_2)$ .

## 2 Exakte Berechnung

Nun stellt sich die Frage, wie die Nachbarschaftsfunktion exakt berechnet werden kann. Denn falls diese exakten Berechnungen effizient sind, lohnt sich eine Approximation eventuell gar nicht; falls sie nicht effizient sind, geben sie einen Hinweis, wo und was verbessert werden kann.

Für  $h = 0$  und  $h = 1$  ist der exakte Wert auch ohne Berechnung schon bekannt:

$$\begin{aligned} N(0) &= |\{(u, v) : u \in V, v \in V, \text{dist}(u, v) \leq 0\}| = |\{(x_i, x_i) : 1 \leq i \leq n\}| = n \\ N(1) &= |\{(u, v) : u \in V, v \in V, \text{dist}(u, v) \leq 1\}| \\ &= |\{(x_i, x_i) : 1 \leq i \leq n; (x_i, x_j) : 1 \leq i, j \leq n, i \neq j, \\ &\quad \text{Kante } (x_i, x_j) \text{ existiert}\}| = n + m \end{aligned}$$

Was ist jetzt aber  $N(2)$ ?

$N(2) - N(1)$  sind die Wege der Länge 2, also ist das der neu hinzukommende Anteil, der

betrachtet werden muß. Allerdings muß man bei der Hinzunahme von Knotenpaaren vorsichtig sein, denn es ist wichtig zu wissen, welches der  $n^2$  möglichen Knotenpaare schon gezählt worden ist. Man kann ein Knotenpaar eventuell mehrmals finden, falls verschiedene Wege unterschiedlicher Länge zwischen ihnen existieren, aber nach der Definition von  $dist(u, v)$  zählt nur der kürzeste Weg.

Im folgenden werden zwei Ansätze zur exakten Berechnung der Nachbarschaftsfunktion vorgestellt.

## 2.1 Potenzieren der Adjazenzmatrix

Die Adjazenzmatrix  $A$  zu einem Graphen  $G$  ist eine quadratische Matrix, in der ein Eintrag  $(A)_{i,j}$  besteht, falls eine Kante zwischen den Knoten  $x_i$  und  $x_j$  existiert.

Beim Potenzieren von  $A$  mit  $h$  werden die Wege der Länge  $h$  zwischen den Knoten berechnet. Die Größe des Eintrags gibt an, wie viele Wege der Länge  $h$  zwischen den Knoten existieren. Uns interessiert also nicht die Größe, sondern allein die Existenz eines Eintrags.

Dabei muß wieder darauf geachtet werden, daß nicht zuviele Knotenpaare gezählt werden, da bei dieser Methode im Schritt  $h$  nicht bekannt ist, ob ein Knotenpaar, das durch einen Weg der Länge  $h$  verbunden ist, schon durch einen Weg der Länge  $h - 1$  oder kleiner verbunden ist. Deshalb müssen alle in Frage kommenden Knotenpaare vor der Hinzunahme erst mit den bereits gefundenen Knotenpaaren verglichen werden, um Mehrfachzählung zu verhindern und um sicherzustellen, daß das Ergebnis der Berechnung von  $N(h)$  korrekt ist.

Die Adjazenzmatrix zum Graphen in Abbildung 1 lautet

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

und die verschiedenen  $h$ -Potenzen dazu

$$A^0 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \quad A^1 = A, \quad A^2 = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix},$$

$$A^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \quad A^4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Für alle  $h \geq 4$  entsteht die Nullmatrix, da im Graphen keine Wege der Länge  $h \geq 4$  existieren.

Die Berechnung von Matrixmultiplikationen benötigt für den naiven Algorithmus  $\mathcal{O}(n^3)$  Laufzeit; der zur Zeit beste bekannte Algorithmus, der 1987 von Coppersmith und Winograd vorgestellt wurde, braucht asymptotisch nur  $\mathcal{O}(n^{2.38})$  Zeit. Allerdings wird hierbei der Speicherbedarf gegenüber dem naiven Algorithmus nicht gesenkt, er besteht ebenfalls in  $\mathcal{O}(n^2)$  Speicher, was bei sehr großen Graphen tunlichst zu vermeiden ist.

## 2.2 Breitensuche

Startet man eine Breitensuche im Knoten  $u$  und führt sie für  $h$  Ebenen aus, so erhält man  $IN(u, h)$ ; die Nachbarschaftsfunktion berechnet sich aus  $N(h) = \sum_{u \in V} IN(u, h)$ . Das bedeutet, daß man für jeden Knoten eine Breitensuche durchführen muß.

Diese Methode benötigt  $\mathcal{O}(nm)$  Laufzeit und  $\mathcal{O}(n + m)$  Speicher. Das Zugriffsverhalten auf die Speicherblöcke ist i.a. jedoch sehr teuer, da das Kanten-File für die hier betrachteten Graphen in der Regel sehr groß ist und somit extern gespeichert werden muß und die Kanten im Laufe der Breitensuche in einer wie zufälligen Reihenfolge betrachtet werden, also wahlfreier Zugriff ('random access') benötigt wird.

## 3 Approximative Berechnung

Das Ziel ist, sowohl  $N^+(h, S, C)$  als auch  $IN^+(h, S, C)$  zu approximieren. Diese Größen sind zur Beantwortung einiger interessanter Fragen nützlich und aus ihnen lassen sich zudem  $N(h)$  und  $IN(x, h)$  berechnen.

Definiere als  $\mathcal{M}(x, h)$  die Knotenmenge, die Abstand  $h$  oder weniger von  $x$  hat.

Von  $x$  aus erreicht man in  $h$  oder weniger Schritten diejenigen Knoten, die man von  $x$  aus in  $h-1$  oder weniger Schritten erreicht hat und zusätzlich die Knoten in  $\mathcal{M}(y, h-1)$ , falls die Kante  $(x, y)$  existiert.

Betrachtet man in dem Graphen aus Abbildung 1  $\mathcal{M}(x_1, 2)$ , so erhält man

$$\begin{aligned} \mathcal{M}(x_1, 2) &= \mathcal{M}(x_1, 1) \cup \mathcal{M}(x_2, 1) = \{x_1, x_2\} \cup \{x_2, x_3, x_4, x_5\} \\ &= \{x_1, x_2, x_2, x_3, x_4, x_5\} \end{aligned}$$

Die Berechnung dieser Knotenmenge wird in Algorithmus 1 dargestellt.

```

1  $\mathcal{M}(x, 0) = x$  for all  $x \in V$ 
2 for each distance  $h$  do
3    $\mathcal{M}(x, h) = \mathcal{M}(x, h - 1)$  for all  $x \in V$ 
4   for each edge  $(x, y)$  do
5      $\mathcal{M}(x, h) = \mathcal{M}(x, h) \cup \mathcal{M}(y, h - 1)$ 

```

**Algorithmus 1:** Knotenmenge mit Abstand  $h$  oder weniger von  $x$

Die Vereinigung der Mengen ist recht aufwendig, da beide zu vereinigenden Mengen auf der Suche nach mehrfach auftretenden Elementen durchlaufen werden müßten.

Für die Nachbarschaftsfunktion sind nur paarweise verschiedene Elemente von Interesse, daher muß die Anzahl verschiedener Elemente in  $\mathcal{M}(x, h)$  möglichst effizient berechnet werden.

Dies kann mit Hilfe eines Zugehörigkeitsbits realisiert werden. Dabei wird jedem Knoten ein Bitstring der Länge  $n$  zugeteilt, in dem für den Knoten  $i$  das  $i$ -te Bit gesetzt ist. Werden dann Mengen vereinigt, so geschieht dies über BITWEISE-ODER-Vergleiche. So können nur bisher nicht betrachtete Elemente in die Menge aufgenommen werden, da bereits betrachtete Elemente schon ihr Bit gesetzt haben.

Diese Methode benötigt  $\mathcal{O}(n^2)$  Speicher, da für  $n$  Knoten ein Bitstring der Länge  $n$  angelegt werden muß. Aber ein so hoher Speicherbedarf sollte bei großen Graphen vermieden werden.

Daher betrachten wir eine weitere Möglichkeit, wie die Anzahl verschiedener Elemente in  $\mathcal{M}(x, h)$  berechnet werden kann:

Da die Nachbarschaftsfunktion nur angenähert werden soll, genügt es, diese Anzahl zu approximieren; dies geschieht mit Hilfe der probabilistischen Zählmethode.

### 3.1 Probabilistische Zählmethode

Bei dieser Methode werden kürzere Bitstrings verwendet, nämlich Bitstrings der Länge  $\lceil \log_2(n) \rceil$ . Dies bedeutet, daß mehrere Elemente der Menge auf dieselbe binäre Darstellung abgebildet werden.

In jedem Bitstring wird genau ein Bit gesetzt mit Wahrscheinlichkeit  $P(\text{Bit } i = 1) = 0.5^{i+1}$ , wobei die Nummerierung der Stellen im Bitstring bei 0 beginnt. Diesem Ansatz liegt die Intuition zugrunde, daß man, wenn man Elemente betrachtet

und keines z.B. das Bit 1 gesetzt hat, mit hoher Wahrscheinlichkeit höchstens 4 Elemente gesehen hat.

Dann wird über den Logarithmus geschätzt und auf die tatsächliche Anzahl geschlossen: Der geschätzte Wert des Logarithmus der Anzahl liegt zwischen der Position des linkensten Nullbits und der des rechtensten Einsbits des Elementes, das entsteht, wenn man alle Elemente der betrachteten Menge vereinigt. Diese Näherung ist natürlich nicht sehr gut; daher wird die Approximation dadurch verbessert, daß man über die Werte einiger auf gleiche Weise erzeugter Elemente mittelt.

In einem Beispiel aus [3] wird eine Menge von 26692 Codezeilen, wovon 16405 verschiedenen sind, betrachtet. Bildet man diese Zeilen mittels einer Hashfunktion, die die obige Wahrscheinlichkeit zur Bitsetzung verwendet, auf einen Hashcode ab, der eine Länge von 24 Bits hat, und bildet die Vereinigungsmenge dieser Hashcodes durch BITWEISE-ODER-Vergleiche, so erhält man folgendes Element

```

111111111111101100000000.
  ↑           ↑  ↑           ↑
  0           13 15          23

```

Das linkeste Nullbit befindet sich an Stelle 13 und das rechteste Einsbit an Stelle 15 und es gilt

$$13 \leq \log_2 16405 \approx 14,002 \leq 15.$$

Sei  $b$  die erwartete Position des linkensten Nullbits (falls man das Eins-Element betrachtet, so gilt:  $b$  ist die Länge der Elemente),  $a$  die Anzahl verschiedener Elemente in der Menge und  $\varphi = 0,77351\dots$ . Dann läßt sich zeigen, daß  $b \approx \log_2 \varphi a$  gilt, d.h.

$$b \approx \log_2 \varphi a \quad \Leftrightarrow \quad 2^b \approx \varphi a \quad \Leftrightarrow \quad a \approx \frac{2^b}{\varphi}.$$

Zudem ist die Qualität dieser Abschätzung nach [3] beweisbar und die Vereinigung von Mengen erfolgt über BITWEISE-ODER, da ein Bit gesetzt ist, falls das zugehörige Element in der Menge ist. Dies bedeutet, daß das Schätzverfahren auch auf der ODER-Verbindung zweier Bitstrings wieder funktioniert.

Also wird  $\mathcal{M}(x, h)$  durch  $M(x, h)$  mit kürzeren Bitstrings der Länge  $\lceil \log_2(n) \rceil + r$ , die  $k$ -mal parallel aufgestellt werden, approximiert, wobei  $r \in \mathbb{N}$  klein und konstant ist und die Bitanzahl etwas steigert, in der Hoffnung, die Genauigkeit zu erhöhen. Also ist  $M(x, h)$  ein Bitstring der Länge  $k(\lceil \log_2(n) \rceil + r)$ . Diese verkürzten Bitstrings reichen aus, da man sowieso nur an der approximativen Berechnung der Anzahl verschiedener Elemente in der Menge interessiert ist.

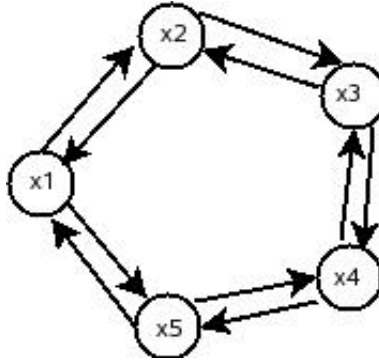


Abbildung 2: Beispielgraph 2

Bezeichne im folgenden  $\vee$  die BITWEISE-ODER-Operation, d.h.  $M(x, h-1) \vee M(y, h-1)$  sei die Vereinigung der Mengen  $M(x, h-1)$  und  $M(y, h-1)$  zu  $M(x, h)$  unter Benutzung des BITWEISE-ODER-Vergleiches.

Betrachtet man den Graph aus Abbildung 2, so ist die Knotenanzahl  $n = 5$  und somit  $\lceil \log_2(n) \rceil = 3$ . Wählt man  $r = 0$  und  $k = 3$ , so haben die Bitstrings für  $M(x, h)$  die Länge  $k(\lceil \log_2(n) \rceil + r) = 3(3 + 0) = 9$ .

Die Bitstrings für  $h = 0$  werden wie oben beschrieben gesetzt, dabei ist in jedem Teil-Bitstring der Länge  $\lceil \log_2(n) \rceil + r = 3$  ein Bit gesetzt mit  $P(\text{Bit } i = 1) = 0.5^{i+1}$  und somit erhält man die Spalte für  $M(x, 0)$  der nachfolgenden Tabelle.

$x$	$M(x, 0)$	$M(x, 1)$	$M(x, 2)$
$x_1$	100 100 001	110 110 101	110 111 101
$x_2$	010 100 100	110 101 101	110 111 101
$x_3$	100 001 100	110 101 100	110 111 101
$x_4$	100 100 100	100 111 100	110 111 101
$x_5$	100 010 100	100 110 101	110 111 101

Durch die Aufspaltung von  $M(x, h) = M(x, h-1) \vee M(y, h-1)$  bei vorhandener Kante  $(x, y)$  ergibt sich

$$\begin{aligned}
 M(x_1, 1) &= M(x_1, 0) \vee M(x_5, 0) \vee M(x_2, 0) \\
 &= 100\ 100\ 001 \vee 100\ 010\ 100 \vee 010\ 100\ 100 = 110\ 110\ 101 \\
 M(x_2, 1) &= M(x_2, 0) \vee M(x_1, 0) \vee M(x_3, 0) \\
 &= 010\ 100\ 100 \vee 100\ 100\ 001 \vee 100\ 001\ 100 = 110\ 101\ 101
 \end{aligned}$$

Genauso verfährt man mit den restlichen Knoten und erhält damit die Spalte für  $M(x, 1)$ .

Analog erhält man für die Spalte von  $M(x, 2)$

$$\begin{aligned}
M(x_1, 2) &= M(x_1, 1) \vee M(x_5, 1) \vee M(x_2, 1) \\
&= 110\ 110\ 101 \vee 100\ 110\ 101 \vee 110\ 101\ 101 = 110\ 111\ 101 \\
M(x_2, 2) &= M(x_2, 1) \vee M(x_1, 1) \vee M(x_3, 1) \\
&= 110\ 101\ 101 \vee 110\ 110\ 101 \vee 110\ 101\ 100 = 110\ 111\ 101
\end{aligned}$$

und ebenso für die verbleibenden Knoten  $x_3, x_4$  und  $x_5$ .

Bis jetzt können in den so aufgestellten Mengen Knotenpaare mehrfach vorkommen, aber wie viele unterschiedliche Knotenpaare sind jetzt in  $M(x, h)$  bzw. wie lautet die approximierte individuelle Nachbarschaftsfunktion  $\hat{IN}(x, h)$ ?

$x$	$\hat{IN}(x, 1)$	$\hat{IN}(x, 2)$
$x_1$	$b = \frac{2+2+1}{3} = \frac{5}{3}, \frac{2^b}{\varphi} \approx 4, 1$	$b = \frac{2+3+1}{3} = 2, \frac{2^b}{\varphi} \approx 5, 1$
$x_2$	$b = \frac{2+1+1}{3} = \frac{4}{3}, \frac{2^b}{\varphi} \approx 3, 25$	$b = \frac{2+3+1}{3} = 2, \frac{2^b}{\varphi} \approx 5, 1$
$x_3$	$b = \frac{2+1+1}{3} = \frac{4}{3}, \frac{2^b}{\varphi} \approx 3, 25$	$b = \frac{2+3+1}{3} = 2, \frac{2^b}{\varphi} \approx 5, 1$
$x_4$	$b = \frac{1+3+1}{3} = \frac{5}{3}, \frac{2^b}{\varphi} \approx 4, 1$	$b = \frac{2+3+1}{3} = 2, \frac{2^b}{\varphi} \approx 5, 1$
$x_5$	$b = \frac{1+2+1}{3} = \frac{4}{3}, \frac{2^b}{\varphi} \approx 3, 25$	$b = \frac{2+3+1}{3} = 2, \frac{2^b}{\varphi} \approx 5, 1$

Man sieht, daß selbst bei einem so kleinen Beispiel eine relativ genaue Abschätzung der individuellen Nachbarschaftsfunktion zustande kommt, denn die tatsächlichen Werte betragen offensichtlich

$$\begin{aligned}
IN(x_i, 1) &= |\{v : v \in V, dist(x_i, v) \leq 1\}| = |\{x_i, x_{i-1}, x_{i+1}\}| = 3 \\
&\text{bzw.} \\
IN(x_i, 2) &= |\{v : v \in V, dist(x_i, v) \leq 2\}| = |\{x_i, x_{i-1}, x_{i+1}, x_{i-2}, x_{i+2}\}| = 5 \\
&\text{für } i = 1, \dots, 5 \text{ und } x_{j(mod5)} \text{ in der Menge.}
\end{aligned}$$

Diese Approximation wird bei sehr großen Graphen, bei denen es v.a. um die Größenrelation geht, ungleich besser.

### 3.2 ANF-0

Mit der geleisteten Vorarbeit kann nun der Basis-ANF-Algorithmus ANF-0, der in Algorithmus 2 dargestellt ist, formuliert werden.

Dabei werden in Zeile 1 – 3 die Bitmasken wie in Abschnitt 3.1 beschrieben aufgestellt.

Die Zeile 4 beinhaltet die äußere Schleife, die für jede Distanz  $h$  durchlaufen wird. In Zeile 5–8 werden die Mengen  $M(x, h)$  nach der gefundenen Regel aufgestellt bzw. vereinigt und in den Zeilen 9 und 10 wird die approximierte individuelle Nachbarschaftsfunktion berechnet. Zum Schluß wird in Zeile 11 die approximierte Nachbarschaftsfunktion aus den erhaltenen approximierten individuellen Nachbarschaftsfunktionen aufsummiert.

**Input:** Graph  $G = (V, E)$   
**Output:** approximierte Nachbarschaftsfunktion  $\hat{N}(h)$

```

1  $\mathcal{M}(x, 0) = \{x\}$ 
2 for each  $x \in V$  do
3    $M(x, 0)$  ist Verbindung von  $k$  Bitmasken, jede mit einem Bit gesetzt mit
   |   Wahrscheinlichkeit  $P(\text{Bit } i) = 0.5^{i+1}$ 
4 for each  $h$  starting with 1 do
5   for each  $x \in V$  do
6      $M(x, h) = M(x, h - 1)$ 
7     for each  $(x, y)$  do
8        $M(x, h) = M(x, h)$  BITWISE-OR  $M(y, h - 1)$ 
9     for each  $x \in V$  do
10     $I\hat{N}(x, h) = \frac{2^b}{0.77351}$ 
11  $\hat{N}(h) = \sum_{x \in V} I\hat{N}(x, h)$ 

```

**Algorithmus 2:** ANF-0

Die **Nachteile** von ANF-0 bestehen zum einen darin, daß sehr viel Speicherplatz benötigt wird, da  $M(x, h)$  für jedes  $x$  und jedes  $h$  berechnet und gespeichert wird. Zum anderen wird die verallgemeinerte Nachbarschaftsfunktion  $N^+(h, S, C)$  nicht berechnet, obwohl sie für viele Anwendungen erforderlich ist.

Daher werden einige **Verbesserungen** vorgenommen:

Die Beobachtung, daß  $M(x, h)$  immer nur auf  $M(y, h - 1)$ , aber nie auf  $M(y, h - 2)$  zurückgreift, motiviert die Einführung von  $Mcur(x)$  anstelle von  $M(x, h)$  und  $Mlast(y)$  anstelle von  $M(y, h - 1)$ .

Außerdem wird in der Berechnung der Nachbarschaftsfunktion in Zeile 11 die Unterscheidung getroffen, ob der Knoten  $x$  in der Startmenge  $S$  ist oder nicht; es wird über  $x \in S$  statt über  $x \in V$  summiert. Dazu wird  $Mcur(x)$  um ein markierendes Bit erweitert, ob  $x \in S$  oder nicht.

Als drittes ändern die Endknoten in  $C$  den Start des Algorithmus in  $h = 0$  ab, denn es gilt  $\mathcal{M}(x, 0) = \emptyset$  für  $x \notin C$ , d.h.  $x \notin C$  wird mit der Bitmake 0 initialisiert.

### 3.3 In-core ANF-0

Mit diesen Verbesserungen kann der Algorithmus ANF-0 weiterentwickelt werden zum Algorithmus In-core ANF-0, der in Algorithmus 3 beschrieben ist.

**Input:** Graph  $G = (V, E)$   
**Output:** approximierte Nachbarschaftsfunktion  $\hat{N}(h)$

```

1 for each  $x \in V$  do
2   if  $x \in C$  then
3      $Mcur(x)$  ist Verbindung von  $k$  Bitmasken, jede mit einem Bit gesetzt
     mit Wahrscheinlichkeit  $P(\text{Bit } i) = 0.5^{i+1}$ 
4 for each  $h$  starting with 1 do
5   for each  $x$  do
6      $Mlast(x) = Mcur(x)$ 
7   for each  $(x, y)$  do
8      $Mcur(x) = Mcur(x)$  BITWISE-OR  $Mlast(y)$ 
9   for each  $x$  do
10     $\hat{IN}(x, h, C) = \frac{2^b}{0.77351}$ 
11  $\hat{N}^+(h, S, C) = \sum_{x \in S} \hat{IN}^+(x, h, C)$ 

```

**Algorithmus 3:** In-core ANF-0

Dieser Algorithmus In-core ANF-0 erfüllt nun eine Fehlergarantie für  $IN^+(x, h, C)$ , da die Methode des probabilistischen Zählens eine beweisbare Qualität besitzt, und somit auch für  $N^+(h, S, C)$ .

Die Laufzeit beträgt  $\mathcal{O}((n+m)d)$ ,  $d$  der Durchmesser des Graphen, da der Aufwand für die inneren Schleifen  $\mathcal{O}(n+m+n) = \mathcal{O}(n+m)$  und für die äußere Schleife  $\mathcal{O}(d)$  ist, also insgesamt  $\mathcal{O}((n+m)d)$ .

Es wird nur noch zusätzlicher Speicher für  $Mcur$  und  $Mlast$  benötigt, also ist der Speicherbedarf  $\mathcal{O}(n)$ .

Zudem ist der Algorithmus gut parallelisierbar, da das Knoten- und Kanten-File partitioniert und die Berechnungen von  $Mcur(x)$  für verschiedene  $x$  aufgeteilt werden kann. Erst nach jeder Iteration ist eine Synchronisation erforderlich.

Auch ist ein sequentieller Scan des Kanten-Files möglich.

Außerdem wird  $IN(x, h)$  berechnet, was bei vielen Fragen eine Rolle spielt.

Allerdings hat bis jetzt noch keine Anpassung an den verfügbaren Speicher stattgefunden.

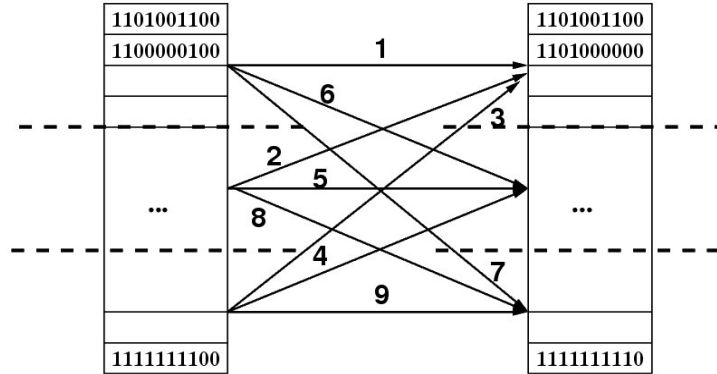


Abbildung 3: Partitionierung von  $Mlast(y)$ ,  $Mcur(x)$  und den Kanten  $(x, y)$

### 3.4 ANF

Der große **Nachteil** von In-core ANF-0 besteht darin, daß beim Sehen der Kante  $(x, y)$   $Mcur(x)$  gelesen und geschrieben und  $Mlast(y)$  gelesen wird. Falls die Tabellen von  $Mcur(x)$  und  $Mlast(y)$  sehr groß sind, mündet das in wie zufälligen Zugriff auf  $Mcur(x)$  und  $Mlast(y)$ .

Dies kann dadurch **verbessert** werden, daß die großen Bitmasken in  $Mcur(x)$  und  $Mlast(y)$  in  $b_1$  bzw.  $b_2$  Abschnitte eingeteilt werden, zusätzlich werden die Kanten auf  $b_1 \times b_2$  Buckets verteilt und sortiert; dann werden nur die jeweils zu bearbeitenden Buckets geladen. In der Abbildung 3 entspricht der linke Block  $Mlast(y)$ , aufgeteilt in  $b_2$  Sektionen, und der rechte Block  $Mcur(x)$ , aufgeteilt in  $b_1$  Sektionen. In diesem Beispiel wurde  $b_1 = b_2 = 3$  gewählt.

Wird nun das erste Bucket von  $Mcur(x)$  geladen, so folgt man den Nummerierungen der Kantenpartitionierung beim Laden der Buckets von  $Mlast(y)$ , d.h. zuerst sind die ersten Buckets von  $Mcur(x)$  und  $Mlast(y)$  im Speicher, danach wird in  $Mlast(y)$  das erste Bucket durch das zweite und schließlich durch das dritte ersetzt. Danach muß das zweite Bucket von  $Mcur(x)$  geladen werden, da das erste fertig bearbeitet ist. Da aber das dritte Bucket von  $Mlast(y)$  noch im Speicher ist, beginnt man diesmal damit und tauscht später das dritte gegen das zweite Bucket von  $Mlast(y)$  aus usw.

Dabei bemerkt man, daß jedes Bucket von  $Mcur(x)$  zwei I/O-Operationen benötigt, da gelesen und geschrieben wird, jedes Bucket von  $Mlast(y)$  jedoch nur eine I/O-Operation, da nur gelesen wird. Daher ist es sinnvoll, relativ große Partitionierungen von  $Mcur(x)$  und relativ kleine Partitionierungen von  $Mlast(y)$  zu bilden, d.h. wähle  $b_1$  so klein und

$b_2$  so groß, daß die entstehenden Buckets noch in den Speicher passen. Dies führt auf den Algorithmus für das verbesserte Update für  $Mcur(x)$ , der in Algorithmus 4 beschrieben wird.

```

1 for each bucket  $i$  of  $Mcur$  do
2   Load bucket  $i$  of  $Mcur$ 
3   for each bucket  $j$  of  $Mlast$  do
4     Load bucket  $j$  of  $Mlast$ 
5     for each  $(x, y)$  in bucket  $(i, j)$  do
6        $Mcur(x) = Mcur(x)$  OR  $Mlast(y)$ 
7   Write bucket  $i$  of  $Mcur$ 

```

**Algorithmus 4:** Update für  $Mcur$

Jetzt erfüllt der ANF-Algorithmus eine Fehlergarantie für  $N^+(h, S, C)$ , benötigt nur zusätzlichen Speicher für  $Mcur(x)$  und  $Mlast(y)$  und ist an den verfügbaren Speicher angepaßt. Er ist leicht parallelisierbar, besitzt ein gutes Zugriffsverhalten beim sequentiellen Scan des Kanten-Files und berechnet  $IN(x, h)$ .

### 3.5 ANF-C

Die Schnelligkeit von ANF kann nochmals gesteigert werden, denn betrachtet man die Bitmasken, die im Verlauf des Algorithmus auftreten, so stellt man fest, daß sie in der Regel im Laufe der Zeit immer mehr führende Einsen ansammeln, d.h. sie sind von der Form  $1111111110xxxxx$ . Diese führenden Einsen werden bisher im Algorithmus bei der BITWEISE-ODER-Operation jeweils einzeln verglichen, obwohl klar ist, daß immer eine Eins entsteht. Deshalb wird zur Führende-Einsen-Kompression ein Counter eingeführt, in dem die Anzahl führender Einsen gezählt wird, sodaß sie nicht mehr verglichen werden müssen.

Außerdem können die  $k$  parallelen Approximationen ineinander verschachtelt werden, da jede Teil-Bitmaske der Länge  $\lceil \log_2(n) \rceil + r$  viele führende Einsen besitzt. Dabei werden jeweils die  $i$ -ten Bits der  $k$  Bitmasken nacheinander aufgeschrieben für  $i = 0, \dots, \text{Länge der Elemente} - 1$ . Diese Methode wird Bitschieben genannt und überführt z.B. für  $k = 2$

11100 11010 nach 1111100100.

Somit wird der Anteil an führenden Einsen nochmals erhöht, die durch die Führende-Einsen-Kompression nicht mehr verglichen werden müssen.

## 4 Experimentelle Auswertung

Der ANF-Algorithmus hängt ab von dem Parameter  $r$ , der die Anzahl zusätzlicher Bits angibt, und dem Parameter  $k$ , der die Anzahl paralleler Approximationen bestimmt. Daher stellt sich die Frage, wie diese beiden Größen Einfluß auf den Verlauf des Algorithmus nehmen.

Zum anderen interessiert, ob ANF schneller und genauer als bereits bestehende Approximationen ist und ob ANF zu großen Graphen skaliert.

### 4.1 Fehler

Der Fehler, der benötigt wird, um die Qualität der approximierten Nachbarschaftsfunktion  $\hat{N}(h)$  abzuschätzen, läßt sich auf die übliche Weise definieren.

Der **relative Fehler** zwischen der exakten und der mit ANF approximierten Nachbarschaftsfunktion, der bezüglich einer festen Distanz  $h$  gemessen wird, lautet

$$rel(N(h), \hat{N}(h)) = \frac{|N(h) - \hat{N}(h)|}{N(h)}.$$

Der **gesamte Fehler** zwischen der exakten und der mit ANF approximierten Nachbarschaftsfunktion, der über alle Distanzen  $h$  gemessen wird, besteht in

$$e(N, \hat{N}) = \sqrt{\frac{\sum_{h=2}^d rel(N(h), \hat{N}(h))^2}{d-1}}.$$

Dabei läuft die Summe erst ab Distanz  $h = 2$ , da die exakten Werte für  $h = 0$  und  $h = 1$  ja bekannt sind, und  $d$  bezeichnet wieder den Durchmesser des Graphen.

### 4.2 Parameter

Der Parameter  $k$  bestimmt die Anzahl paralleler Approximationen und ist somit ein Kompromiß zwischen Zeit und Genauigkeit, denn je mehr parallele Approximationen durchgeführt werden, umso genauer wird zwar das Ergebnis, aber es dauert auch viel länger. Die Autoren geben an, daß der ANF-Algorithmus für  $k = 64$  gute Resultate liefert.

Der Parameter  $r$  gibt die Anzahl zusätzlicher Bits beim Aufstellen der Bitmasken an. Um einen guten Wert für  $r$  zu finden, muß der gesamte Fehler  $e(N, \hat{N})$  für verschiedene Werte von  $r$  und verschiedene Datenmengen berechnet werden. Die Autoren schlagen  $r = 7$  vor, da dann der gesamte Fehler relativ klein sei.

### 4.3 Genauigkeit

Um die Genauigkeit zu messen, muß ANF mit den anderen Verfahren zur Approximation der Nachbarschaftsfunktion verglichen werden. Deshalb werden die einzigen beiden Ansätze, die sich überhaupt ernsthaft mit deren Näherung befassen, vorgestellt.

Die **RI-Approximation** basiert auf einem ähnlichen Ansatz wie ANF, benutzt allerdings ein anderes Zählverfahren, um die Anzahl unterschiedlicher Elemente in einer Menge zu finden.

**Sampling** ist ein Verfahren, das (für einen vorher festgelegten Prozentsatz an Knoten) für zufällig ausgewählte Knoten die exakte Nachbarschaftsfunktion unter Benutzung der Breitensuche berechnet.

Die Tests wurden auf sieben Graphen durchgeführt, wovon drei reell und vier konstruiert sind:

*cornell*: reeller gerichteter Graph, crawl der Cornell-Website

*cora*: reeller gerichteter Zitierungsgraph

*router*: reeller ungerichteter Graph, der Routerwege darstellt

*cycle*: ungerichteter Kreis

*grid*: ungerichtetes Gitter

*uniform*: Graph mit zufälligen ungerichteten Kanten

*80-20*: speziell konstruierter ungerichteter Graph

Die **Genauigkeit von ANF gegenüber RI** mit  $r = 7$  und  $k = 64$  wird in Abbildung 4 dargestellt. Es wird deutlich, daß der relative ANF-Fehler ziemlich unabhängig von der zugrunde liegenden Datenmenge ist, der relative RI-Fehler jedoch stark von den Daten abhängt.

ANF ist zudem recht genau und besitzt für  $k = 64$  weniger als 7% Fehler.

Weitere Testergebnisse für verschiedene Werte von  $k$  sind in der folgenden Tabelle zusammengestellt:

$k$	<i>ANF</i> – Fehler	<i>RI</i> – Fehler
32	10%	27%
64	7%	14%
128	5%	12%

Die **Genauigkeit von ANF gegenüber Sampling** ist relativ schwierig darzustellen, da Sampling erhebliche Speicheranforderungen hat, da das Kanten-File bei der Breitensuche komplett in den Speicher passen muß. Außerdem berechnet Sampling keine individuelle Nachbarschaftsfunktion und die Qualität ist stark graphenabhängig, weswegen

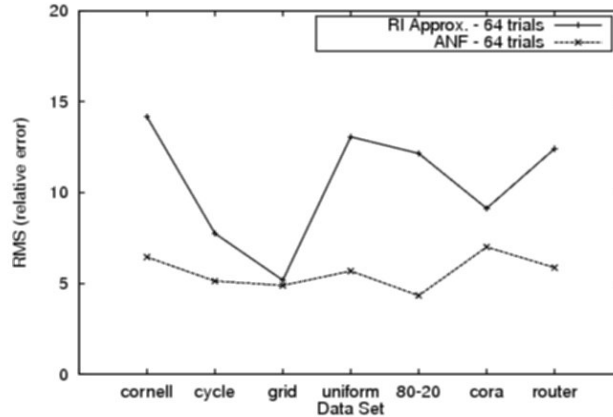


Abbildung 4: Genauigkeit von ANF gegenüber RI

man keine Fehlerschranke angeben kann. Es wurde ein spezieller Graph konstruiert, auf dem Sampling total versagte, ANF aber gute Ergebnisse lieferte.

#### 4.4 Geschwindigkeit und Skalierbarkeit

Der Fairness wegen wurden alle Berechnungen der verschiedenen Algorithmen auf denselben Rechnern mit denselben Ressourcen durchgeführt.

Die **Geschwindigkeit von ANF gegenüber RI** mit  $r = 7$  und  $k = 64$  wird in Abbildung 5 präsentiert. Man sieht deutlich, daß ANF entweder gleich schnell oder schneller als RI auf allen Test-Datenmengen arbeitet.

Für die Werte  $k = 32$  bzw.  $k = 128$  ergibt sich das gleiche Bild, nur daß sich die Laufzeit halbiert bzw. verdoppelt.

Die **Geschwindigkeit von ANF gegenüber Sampling** darzustellen ist nicht sinnvoll, da Sampling -wie wir gesehen haben- sehr viele Probleme bereitet.

Für die **Skalierbarkeit** wurde ein Graph mit zufällig erzeugten Knoten und Kanten mit Kanten-Knoten-Verhältnis  $8 : 1$  generiert und alle Algorithmen (die verschiedenen ANF-Varianten, RI und Sampling) wurden darauf getestet. Dabei wurden alle Parameter der Algorithmen so gewählt, daß sie die gleiche Laufzeit auf dem Startgraphen haben. Erst im Laufe der Zeit bei Erhöhung der Knoten- und Kantenanzahl machen sich die Stärken bzw. Schwächen der verschiedenen Algorithmen bemerkbar.

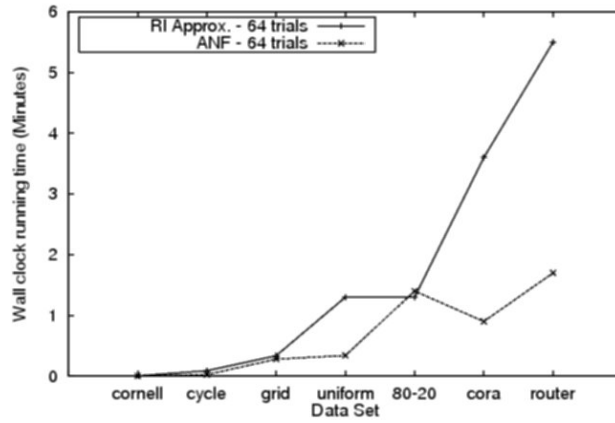


Abbildung 5: Geschwindigkeit von ANF gegenüber RI

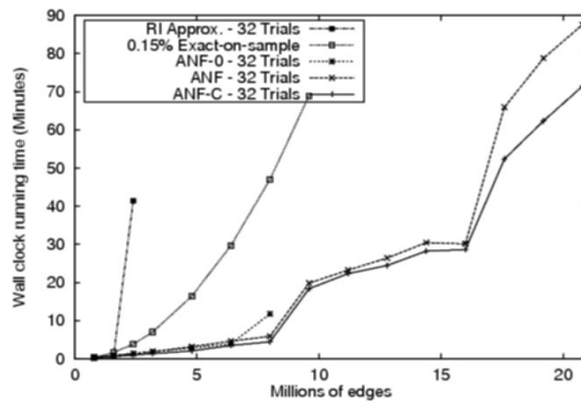


Abbildung 6: Skalierbarkeit der verschiedenen betrachteten Algorithmen

In Abbildung 6 wird dies verdeutlicht.

Dabei sieht man deutlich, daß die RI-Approximation verschwenderisch mit ihren Ressourcen umgeht und bei einer Kantenanzahl von ca. 2 Millionen abgebrochen werden mußte.

Sobald das Kanten-File nicht mehr in den Speicher paßt (bei ca. 7 Millionen Kanten), muß auch Sampling abgebrochen werden. Außerdem skaliert Sampling nicht linear sondern eher exponentiell.

Auch der Basis-ANF-Algorithmus ANF-0, der verschwenderisch mit Speicherplatz umgeht, muß zu diesem Zeitpunkt beendet werden.

Lediglich ANF und die schnellere Variante ANF-C können auf sehr großen Graphen eingesetzt werden. Beide skalieren linear, allerdings mit verschiedenen Steigungen bei verschiedenen Graphengrößen. Der nächste gravierende Einschnitt ist bei einer Kantenanzahl von ca. 16 Millionen; ab diesem Zeitpunkt passen die beiden Tabellen von  $Mcur(x)$  und  $Mlast(y)$  nicht mehr komplett in den Speicher.

Die Geschwindigkeitssteigerung von ANF-C gegenüber ANF kann v.a. auf dem anschließenden Teilstück gesehen werden.

Abschließend kann gesagt werden, daß ANF ein Tool für die relativ genaue Berechnung der Nachbarschaftsfunktion ist, das sehr gut zu großen Graphen skaliert und sehr viel schneller als die exakte Berechnung sein kann.

## 5 Anwendungen

Mit dem herausgearbeiteten ANF-Algorithmus ist man nun in der Lage, die eingangs gestellten Fragen zu beantworten.

### 5.1 Knotenwichtigkeit

Um z.B. den besten Eröffnungszug bei Tic-Tac-Toe herauszufinden, bei dem der Startspieler  $X$  gewinnt, wird so vorgegangen:

- Definiere den zugrunde liegenden gerichteten Graphen als  $G = (V, E)$ , wobei  $V$  die Menge aller möglichen Spielfeldbelegungen und  $E$  die Menge aller möglichen Übergänge zwischen diesen Spielfeldbelegungen darstellt.
- Sei  $S$  die Menge der Eröffnungszüge, in denen  $X$  beginnt und  $C$  die Menge der Endzüge, in denen  $X$  gewinnt und  $O$  verliert.
- Der beste Eröffnungszug ist derjenige, für den die verallgemeinerte individuelle Nachbarschaftsfunktion  $IN^+(h, S, C)$  am höchsten ist.

Die Ergebnisse sind in der folgenden Tabelle festgehalten, wobei die eingetragenen Werte denen von  $IN^+(h, S, C)$  bzgl. des Feldes entsprechen.

2, 50	2, 37	2, 50
2, 36	2, 57	2, 36
2, 50	2, 36	2, 51

Dabei kommt die wenig überraschende Erkenntnis zustande, daß es am günstigsten ist, im Mittelfeld zu starten gefolgt von den Eckfeldern.

Das Maß der individuellen Nachbarschaftsfunktion für Knotenwichtigkeit kann mit der **Closeness-Zentralität** in Verbindung gebracht werden.

Für  $d$  den Durchmesser des Graphen ist  $IN(x, d)$  die Anzahl aller kürzesten  $(x, t)$ -Wege, wobei  $t \in V$  ist, d.h.  $IN(x, d)$  ist ein Maß für die Verbundenheit eines Knotens  $x$  im Graphen  $G = (V, E)$  und  $IN(x, d) \geq 1$ .

Die Closeness-Zentralität eines Knotens  $x$  im Graphen  $G = (V, E)$  ist mit  $d_G(x, t) = \text{kürzeste Länge des } (x, t) - \text{Weges}$  definiert als

$$c_C(G)_x = \frac{1}{\sum_{t \in V} d_G(x, t)}$$

und ist somit ebenfalls ein Maß für die Verbundenheit eines Knotens  $x$  im Graphen  $G = (V, E)$ .

Die Closeness-Zentralität kann allerdings auch mit Hilfe der individuellen Nachbarschaftsfunktion ausgedrückt werden:

$$c_C(G)_x = \left( \sum_{h=1}^d h \cdot IN(x, h) \right)^{-1} = \left( \sum_{t \in V} dist(x, t) \right)^{-1}.$$

So wird deutlich, daß sich die Closeness-Zentralität als eine Art inverser 'individueller Nachbarschaftsfunktions-Zentralität' auffassen läßt.

## 5.2 Graphenähnlichkeit

Will man zwei komplette Graphen vergleichen, bedeutet dies, daß die Nachbarschaftsfunktionen beider Graphen für jede Distanz  $h$  berechnet werden müssen und daß diese dann in jedem  $h$  verglichen werden müssen. Das bedeutet bei großen Graphen jedoch, daß zwei Funktionen auf einen riesigen Definitionsbereich zu vergleichen sind.

Aus diesem Grund wird der **Hop-Exponent**  $\mathcal{H}$  bzw. der **individuelle Hop-Eponent**  $\mathcal{H}_x$  eingeführt, der das Vergleichen erleichtert.

Viele reelle Graphen genügen einem Potenzgesetz  $N(h) \propto h^{\mathcal{H}}$ , und falls sie das tun,

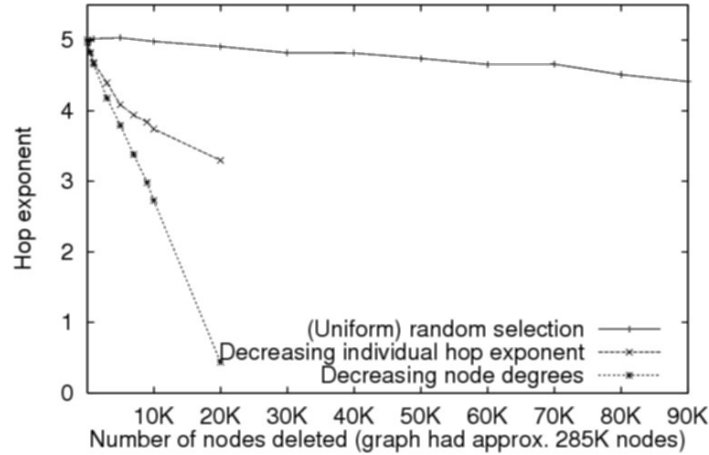


Abbildung 7: Anzahl verbundener Routerpaare vs. Anzahl gelöschter Router

so besitzt ihre Nachbarschaftsfunktion  $N(h)$  einen linearen Teil mit Steigung  $\mathcal{H}$  in der log-log-Darstellung, d.h. den Hop-Exponenten  $\mathcal{H}$  kann man aus der log-log-Darstellung 'ablesen'. Der Hop-Exponent entspricht der 'Dimensionalität' des Graphen, z.B. hat ein Kreis  $\mathcal{H} = 1$  und ein Gitter  $\mathcal{H} = 2$ . Dies bedeutet aber auch, daß Graphen mit unterschiedlichem Hop-Exponent nicht ähnlich sein können.

Damit ist es nur noch nötig, die Hop-Exponenten der Graphen zu vergleichen, welche auf einem etwas kleineren Graphen berechnet werden können:

Sei  $d_{\text{eff}}$  der effektive Durchmesser des Graphen, d.h.  $d_{\text{eff}}$  ist das kleinste  $h$ , sodaß 90% der Knotenpaare enthalten sind. Berechne dann  $N(d_{\text{eff}})$  und finde mit der Methode der kleinsten Fehlerquadrate die Ausgleichsgerade in der log-log-Darstellung von  $N(d_{\text{eff}})$ . Diese Ausgleichsgerade besitzt die Steigung  $\mathcal{H}$  und ist ein Maß für den Zuwachs der Nachbarschaftsfunktion. Entsprechend kann das gleiche für  $\mathcal{H}_x$  bzgl. des Knotens  $x$  durchgeführt werden.

Damit ist man jetzt in der Lage, die Frage, wie robust das Internet gegen Routerausfälle ist, schnell zu beantworten. In Abbildung 7 wird deutlich, daß der Ausfall zufälliger Router die Verbundenheit nicht sehr stark beeinträchtigt; wählt man die Router aber in absteigender  $\mathcal{H}_x$ -Reihenfolge oder in absteigender Knotengrad-Reihenfolge, so sind die Auswirkungen auf die Verbundenheit verheerend.

### 5.3 Teilgraphenähnlichkeit

Schauspieler und Filme können als Knoten des Graphen  $G = (V, E)$  dargestellt werden, indem eine ungerichtete Kante zwischen ihnen gesetzt wird, falls der Schauspieler im Film einen Auftritt hat. Ein anderer Graph kann ähnlich definiert werden, in ihm werden die Filme in Genres aufgeteilt. D.h. Filme und Filmgenres sind Knoten und eine ungerichtete Kante entsteht, wenn ein Film einem gewissen Genre angehört. Betrachtet man diese beiden Graphen als einen, so besteht die Knotenmenge aus Schauspielern, Filmen und Filmgenres.

Will man nun herausfinden, in welchen unterschiedlichen Filmgenres ein Schauspieler auftritt, bedeutet das, daß man die von den Filmen eines Genres induzierten Teilgraphen betrachten und zu jedem Teilgraphen den Hop-Exponenten bestimmen muß. Danach werden die Hop-Exponenten, die sich um weniger als 0.1 unterscheiden, zusammengefaßt und die zugehörigen Genres bilden einen Cluster. So erhält man Informationen, welche Filmgenres 'spezialisierte' Schauspieler besitzen und zwischen welchen Filmgenres die Schauspieler wechseln. Z.B. besteht ein Cluster aus den Genres 'Horror, Abenteuer, Krimi, Thriller, Action', was bedeutet, daß Schauspieler in der Regel zwischen diesen Filmgenres wechseln; andererseits besteht z.B. der Cluster 'Comedy' aus nur einem Genre, da die Schauspieler hierauf 'spezialisiert' sind.

## Literatur

- [1] C. R. Palmer, P. B. Gibbons, C. Faloutsos: ANF: A Fast and Scalable Tool for Data Mining in Massive Graphs, *KDD*, Seiten 81-90, 2002.
- [2] C. R. Palmer, P. B. Gibbons, C. Faloutsos: Fast Approximation of the 'Neighbourhood' Function for Massive Graphs, *Manuskript*, 2001.
- [3] P. Flajolet, G. N. Martin: Probabilistic Counting Algorithms for Data Base Applications, *Journal of Computer and System Sciences*, 31:182-209, 1985.
- [4] <http://www.inf.uni-konstanz.de/algo/lehre/ss05/mna/skript/zentralitaeten.pdf>, Stand 17.2.2006.