Praktikum zur Clusterung von Graphen

Daniel Karch, Hanno Kersting
Betreuer: Marco Gaertler

20. September 2007

Zusammenfassung

Thema dieser Ausarbeitung sind zwei Algorithmen zur Clusterung von Graphen, die das Zentralitätsmaß *Betweenness* verwenden. Diese werden vorgestellt und anhand der Testergebnisse auf verschiedenen Instanzen von Graphen miteinander verglichen. Motiviert durch diese Ergebnisse ergeben sich für die Algorithmen unterschiedliche Einsatzmöglichkeiten. Zuletzt werden weitere Möglichkeiten zur Optimierung und Forschung angeregt.

Inhaltsverzeichnis

1 Einleitung **3**
1.1 Frühere Ansätze . . . 3
1.2 Ziel des Praktikums . . . 3
1.3 Der Praktikumsverlauf/Gliederung . . . 3

2 Grundlagen **4**
2.1 Begriffsklärung . . . 4
2.2 Was macht eine gute Clusterung aus? . . . 4
2.3 Das Betweenness-Zentralitätsmaß . . . 5
2.4 Zusammenhang zwischen guten Clusterungen und der Betweenness . . . 5

3 Hauptteil **6**
3.1 Implementierung . . . 6
3.1.1 Der Programmablauf . . . 6
3.2 Der Versuchsaufbau . . . 7
3.2.1 Erzeugung der zu testenden Graphen . . . 7
3.2.2 Die Bewertungsfunktion . . . 8
3.2.3 Algorithmus1 - Mehrmalige Betweennessberechnung . . . 8
3.2.4 Algorithmus2 - Einmalige Betweennessberechnug . . . 9
3.3 Die Funktionsweise der Algorithmen am Beispiel . . . 10
3.4 Ergebnisse . . . 11
3.5 Optimierungsmöglichkeiten . . . 13

4 Schluss **14**
4.1 Zusammenfassung . . . 14
4.2 Rekapitulation . . . 14

Literatur **14**

1 Einleitung

In den unterschiedlichsten Anwendungsgebieten tritt oft das Problem auf, eine Menge von Daten so in disjunkte Teilmengen zu zerlegen, dass die Daten innerhalb dieser Teilmengen gewisse Eigenschaften gemein haben, während Daten aus unterschiedlichen Teilmengen bezüglich dieser Eigenschaften keine oder nur marginale Gemeinsamkeiten aufweisen. Diese Gruppen (engl. **Cluster**) zu finden, lässt sich als graphentheoretisches Problem auffassen, indem man die Daten als Knoten und die Beziehungen zwischen den Daten als Kanten zwischen den jeweiligen Knoten auffasst.

1.1 Frühere Ansätze

Bisherige Ansätze zum Clustern von Graphen verwenden unter anderem *maximale Flüsse* [5] oder *Distance-k Cliques* [4]. Die Verwendung von *Betweenness* wurde von Newman und Girvan vorgeschlagen [6].

1.2 Ziel des Praktikums

Im Verlauf des Praktikums war es unsere Aufgabe, verschiedene Algorithmen zur Clusterung von Graphen zu implementieren und zu untersuchen. Neben der Qualität der Ergebnisse lag das Hauptaugenmerk vor allem auf der Optimierung der Laufzeit.

Wir werden im Folgenden zwei Ansätze zur Clusterung von Graphen vorstellen, die auf dem Zentralitätsmaß der **Betweenness** basieren. Die *Betweenness* misst die Wichtigkeit eines Knotens oder einer Kante hinsichtlich des Vorkommens auf kürzesten Wegen im Graphen.

1.3 Der Praktikumsverlauf/Gliederung

Die erste Phase des Praktikums bestand aus der Einarbeitung in die vorgegebenen Programmbibliotheken und dem Einlesen in die Literatur zum Thema „Betweenness“. Anschließend folgte die Implementierung diverser Graphenalgorithmen zur Berechnung der Betweenness sowie eines Programmgerüstes, das Clusterungen erstellt und bewertet. Darauf aufbauend haben wir Verfahren zur Clusterung von Graphen entwickelt, getestet und verglichen.

Der nachfolgende Teil der Ausarbeitung behandelt zwei Algorithmen, die sich im Laufe des Praktikums als besonders praktikabel und interessant herausgestellt haben.

2 Grundlagen

2.1 Begriffsklärung

Im Folgenden definieren wir Begriffe, die für das Verständnis des nachfolgenden Textes notwendig sind. Wir setzen jedoch einige Begriffe als bekannt voraus, so z.B. die $\mathcal{O}$-Notation zur Angabe der Laufzeiten von Algorithmen, den Begriff des Graphen sowie grundlegende Graphenalgorithmen wie die Breitensuche.

Graphen bezeichnen wir mit G, G' usw., Knoten mit $v_0, v_1, \ldots$ und Kanten mit $e_0, e_1, \ldots$. Wir setzen außerdem $|E| =: m$ und $|V| =: n$.

Sei im Folgenden der ungewichtete und ungerichtete Graph G mit $V = \{v_0, \ldots, v_{n-1}\}$ und $E = \{e_0, \ldots, e_{m-1}\}$ gegeben.

Definition 1 *Eine Folge von Kanten $e_{i_1}, \ldots, e_{i_r}$ mit $e_{i_j} := \{v_{i_{j-1}}, v_{i_j}\}$ nennen wir einen **Weg** in G von v_{i_0} nach v_{i_r}. Die Anzahl der Kanten in der Folge nennen wir die **Länge** des Weges.*

Definition 2 *Ein Weg in G von v_i nach v_j heißt **kürzester Weg**, wenn er unter allen Wegen zwischen diesen beiden Knoten minimale Länge hat.*

Definition 3 *Eine Partitionierung der Knotenmenge V in die Teilmengen $C_1, \ldots, C_k$ nennt man eine **Clusterung** von G. Die $C_i, i \in \{1, \ldots, k\}$ heißen **Cluster**.*

Definition 4 *Eine Kante, die zwischen Knoten des gleichen Clusters verläuft, nennen wir **Intraclusterkante**. Kanten, die zwischen unterschiedlichen Clustern verlaufen heißen **Interclusterkanten**.*

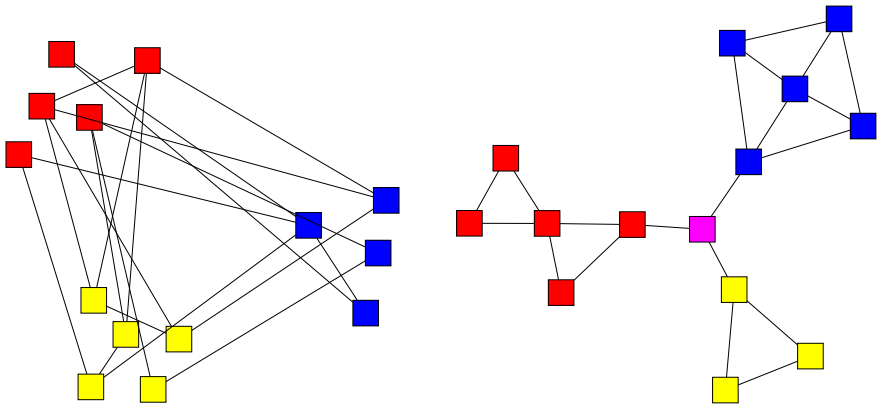
Unter dem **Clustern** eines Graphen verstehen wir die Einteilung der Knotenmenge in verschiedene Cluster. Wenn im Folgenden von einem gut geclusterten Graphen gesprochen wird, ist damit ein Graph gemeint, der eine Clusterung aufweist, die unseren Ansprüchen genügt.

2.2 Was macht eine gute Clusterung aus?

Um Algorithmen zur Clusterung von Graphen zu entwickeln, muss zunächst geklärt werden, was wir unter einer guten Clusterung verstehen.

Prinzipiell lässt sich jede Partitionierung der Knotenmenge als Clusterung auffassen. Uns geht es jedoch darum, bestimmte strukturelle Eigenschaften des Graphen zu finden und durch die Clusterung auszudrücken. Eine gute Clusterung zeichnet sich dadurch aus, dass die einzelnen Cluster eine relativ hohe Kantendichte aufweisen, während zwischen ihnen nur wenige Kanten verlaufen.

Um gute Algorithmen zur Clusterung zu entwerfen, benötigen wir also eine Möglichkeit, für jede Kante effizient zu entscheiden, ob sie innerhalb eines Clusters liegt oder ob sie zwischen zwei Clustern verläuft.



(a) schlechte Clusterung (b) gute Clusterung

Abbildung 1: Der gleiche Graph mit zwei unterschiedlichen Clusterungen

2.3 Das Betweenness-Zentralitätsmaß

Eine Möglichkeit, die Kanten eines Graphen vergleichen zu können, stellt die Verwendung des Zentralitätsmaßes *Betweenness* dar. Dieses Zentralitätsmaß ordnet jeder Kante einen festen, deterministischen Wert zu, der angibt, auf wie vielen kürzesten Wegen im Graphen die Kante liegt.

Definition 5 *Wir bezeichnen mit $\sigma_G(s,t)$ die Anzahl der kürzesten Wege von s nach t in G. $\sigma_G(s,t|e)$ sei die Anzahl der kürzesten Wege von s nach t über e.*

Dann ist die ***Kantenbetweenness*** *von e definiert als*

$$c_B(G)_e = \sum_{s,t \in V} \frac{\sigma_G(s,t|e)}{\sigma_G(s,t)}$$

Nach [1] lässt sich die Kantenbetweenness für alle Kanten in G in $\mathcal{O}(nm)$ berechnen, indem man von jedem Knoten eine modifizierte Breitensuche startet.

2.4 Zusammenhang zwischen guten Clusterungen und der Betweenness

Um sich den Zusammenhang zwischen der Betweenness und der Clusterung von Graphen zu veranschaulichen, betrachte man den Beispielgraphen in Abbildung 2, in dem eine eindeutige Clusterung vorliegt und bereits hervorgehoben ist.

Für jedes Paar $(v,w) \in V_1 \times V_2$ muss ein kürzester Weg zwischen beiden Knoten eine Interclusterkante enthalten. Also haben Interclusterkanten tendenziell eine höhere Betweenness als Intraclusterkanten. Die Einteilung der Kantenmenge in Inter- und Intraclusterkanten ermöglicht dann, gute Clusterungen in Graphen zu finden.

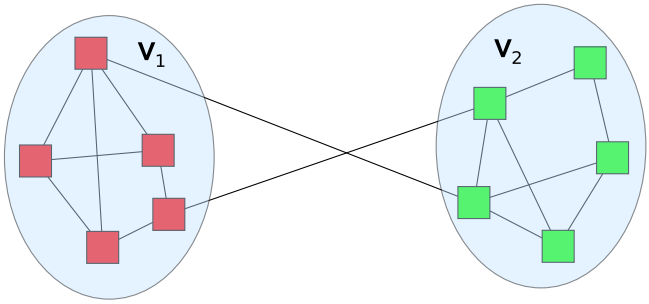


Abbildung 2: Ein Graph, für den eine mögliche Clusterung hervorgehoben ist

3 Hauptteil

3.1 Implementierung

3.1.1 Der Programmablauf

Im Ablauf des Programms werden unterschiedliche Clusterungen erzeugt und anhand einer Bewertungsfunktion verglichen. Am Ende wird die Clusterung zurückgegeben, die im Programmverlauf die beste Bewertung erhalten hat.

Ausgehend vom ursprünglichen Graphen werden sukzessive Kanten gelöscht. So entstehen nach und nach neue Zusammenhangskomponenten, bis der Graph keine Kanten mehr enthält. In jedem Bewertungsschritt werden diese Zusammenhangskomponenten als Cluster aufgefasst und die dadurch induzierte Clusterung bewertet.

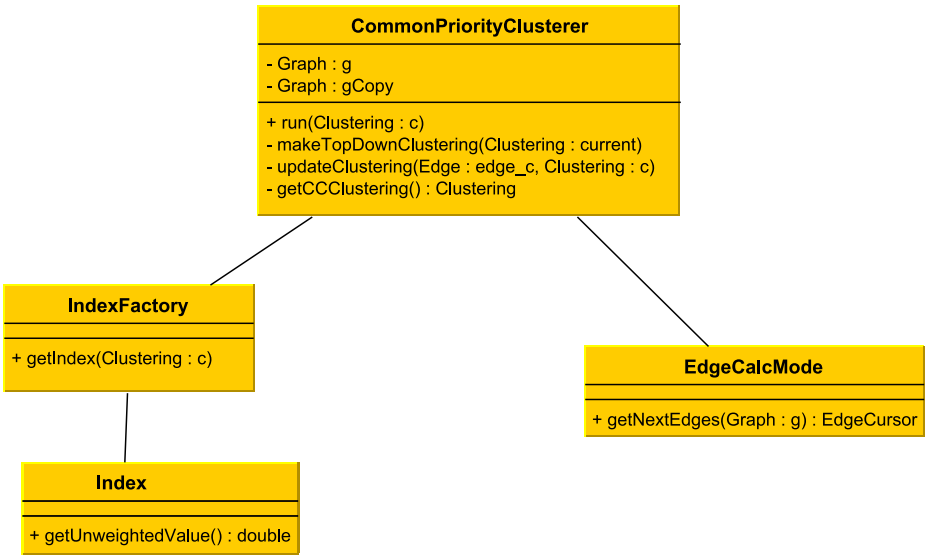


Abbildung 3: Das UML-Diagramm zu unserer Implementierung

Die Klasse **CommonPriorityClusterer** stellt die Schnittstelle zum Anwender dar. Dieser Klasse wird ein Clustering übergeben, das im Programmverlauf verändert wird. Die Klasse **EdgeCalcMode** liefert die Kanten, die im jeweiligen Durchgang gelöscht werden. Das so entstehende Clustering wird an eine **IndexFactory** übergeben und mittels einer Bewertungsfunktion (gekapselt in der Klasse **Index**) bewertet. Der *CommonPriorityClusterer* merkt sich die bisher am besten bewertete Clusterung.

3.2 Der Versuchsaufbau

3.2.1 Erzeugung der zu testenden Graphen

Zur Erzeugung der Graphen verwendeten wir den Graphengenerator *Gaussian Random Generator*. Ausgehend von den Parametern Knotenanzahl n, Wahrscheinlichkeit für Intraclusterkanten p_{in} und für Interclusterkanten p_{out} erzeugt dieser „zufällige“ Graphen.

Der Generator wählt zuerst zufällig ein $k \in [\log n, \sqrt{n}]$ und erzeugt anschließend k Cluster. Die Knotenanzahl eines jeden Clusters wird durch eine Gaußverteilung um $\frac{n}{k}$ bestimmt. Für jedes Knotenpaar (u, v) innerhalb eines Clusters wird mit der Wahrscheinlichkeit p_{in} eine Kante in den Graphen eingefügt, zwischen zwei Knoten aus verschiedenen Clustern mit der Wahrscheinlichkeit p_{out}. Nähere Informationen finden sich in [3].

Für uns waren nur Graphen von Interesse, die eine deutliche Clusterung aufweisen. Das heißt, dass wir nur Graphen untersucht haben, für die p_{in} deutlich größer ist als p_{out}.

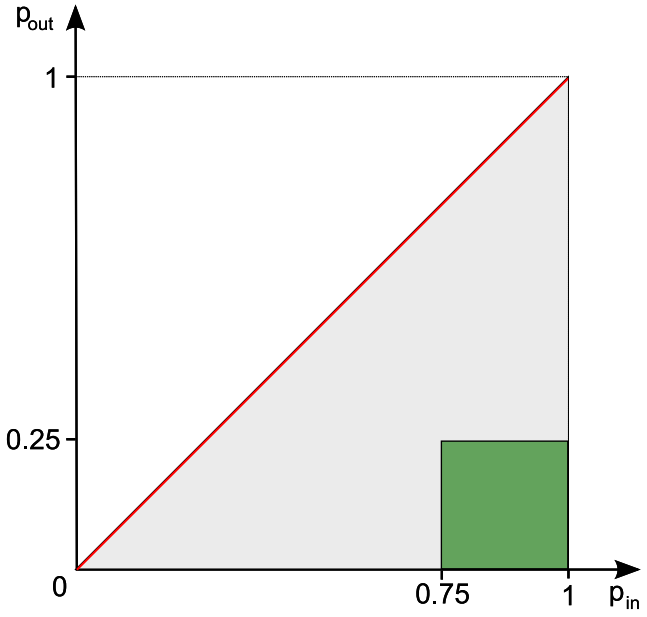


Abbildung 4: Die für uns relevanten Werte liegen unterhalb der Winkelhalbierenden.

Für unsere Versuche wählten wir $0,75 \leq p_{in} \leq 1$ und $0 \leq p_{out} \leq 0,25$ (siehe Abb. 4).

3.2.2 Die Bewertungsfunktion

Als Bewertungsfunktion haben wir das Qualitätsmaß *Modularity* verwendet, das uns durch die verwendeten Bibliotheken zur Verfügung stand. Informationen zu diesem Qualitätsmaß finden sich in [2].

3.2.3 Algorithmus1 - Mehrmalige Betweennessberechnung

Unser erster Ansatz berechnet vor jedem Kantenlöschvorgang die Betweenness im aktuellen Graphen. Durch die Löschvorgänge zerfällt der Graph in Zusammenhangskomponenten, die im weiteren Verlauf als voneinander unabhängige Subgraphen betrachtet werden. So werden nach und nach quasi rekursiv immer feinere Clusterungen erzeugt.

Algorithmus 1 – Mehrmalige Betweennessberechnung

Eingabe: Graph G
Ausgabe: Clustering c_{best}

```
1  val_{c_best} ← −∞
2  E' ← E
3  solange E' ≠ ∅ tue
4  |  Berechnen der Betweenness für alle e ∈ E'
5  |  b_max ← max_{e∈E'}{c_B(G')_e} ;        /* G' := (V, E'); c_B(G')_e: Betweenness von e */
6  |  B ← {e ∈ E' | c_B(G)_e = b_max}
   |  /* Die Kanten mit maximaler Betweenness werden entfernt. */
7  |  E' ← E' \ B
8  |  c ← durch E' induzierte Clusterung von G
   |  /* wert(c): Aufruf der Bewertungsfunktion */
9  |  wenn wert(c) < val_{c_best} dann
10 |  |  c_best ← c
11 |  |  val_{c_best} ← wert(c)
12 |  Ende
13 Ende
```

Die *while*-Schleife wird höchstens m-mal durchlaufen, denn in jedem Durchlauf wird mindestens eine Kante entfernt. Aus 2.3 wissen wir, dass sich die Betweenness für alle Kanten in $\mathcal{O}(nm)$ berechnen lässt.

Im Verlaufe des Algorithmus wird jede Kante entfernt, der Entfernungsschritt trägt demnach zur Gesamtlaufzeit $\mathcal{O}(m)$ bei und wird also von der Betweennessberechnung dominiert. Jede neu entstandene Clusterung wird mittels der Bewertungsfunktion (*Modularity*) bewertet, was nach [2] in $\mathcal{O}(n)$ geschieht. Daraus ergibt sich eine Laufzeit von $\mathcal{O}(nm)$ pro Schleifendurchlauf, was zu einer Gesamtlaufzeit von $\mathcal{O}(nm^2)$ führt.

3.2.4 Algorithmus2 - Einmalige Betweennessberechnug

Im Gegensatz zum ersten Algorithmus findet hier lediglich eine einmalige Betweennessberechnung statt. Diese Betweennesswerte repräsentieren die ursprüngliche Struktur des Graphen, für den eine Clusterung gefunden werden soll. Es findet eine einmalige Sortierung statt, so dass leicht die Kanten mit dem nächsthöheren Wert gefunden werden können.

Algorithmus 2 – Einmalige Betweennessberechnung

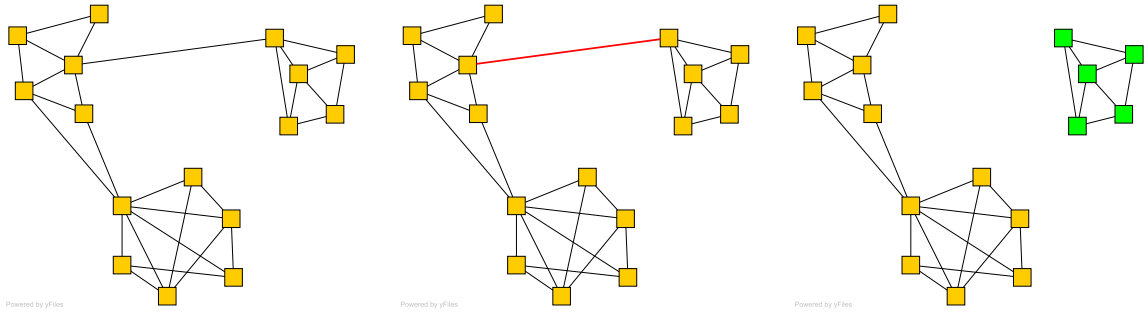
Eingabe: Graph G
Ausgabe: Clustering c_{best}

```
1  val_{c_best} ← −∞
2  E' ← E
3  Berechnen der Betweenness für alle e ∈ E'
4  Sortieren von E' nach Betweennesswert
5  solange E' ≠ ∅ tue
6  |  b_max ← max_{e∈E'}{c_B(G)_e} ;          /* c_B(G)_e: Betweenness von e */
7  |  B ← {e ∈ E'|c_B(G)_e = b_max}
   |  /* Die Kanten mit maximaler Betweenness werden entfernt. */
8  |  E' ← E' \ B
9  |  c ← durch E' induzierte Clusterung von G
   |  /* wert(c): Aufruf der Bewertungsfunktion */
10 |  wenn wert(c) < val_{c_best} dann
11 |  |  c_best ← c
12 |  |  val_{c_best} ← wert(c)
13 |  Ende
14 Ende
```

Die Betweenness wird nur einmal in $\mathcal{O}(nm)$ berechnet, die Sortierung geschieht in $\mathcal{O}(m \log m)$. Die *while*-Schleife wird wieder höchstens m-mal durchlaufen. Der Schleifenrumpf wird durch die Bewertung der neu entstandenen Clusterung in $\mathcal{O}(n)$ dominiert. Daraus ergibt sich eine Laufzeit von $\mathcal{O}(n)$ pro Schleifendurchlauf. Zusammen mit der initialen Betweennessberechnung resultiert daraus eine Gesamtlaufzeit von $\mathcal{O}(nm+m*n) = \mathcal{O}(nm)$.

Durch die lediglich einmalige Berechnung der Betweenness lässt sich die Laufzeit also um den Faktor m verbessern.

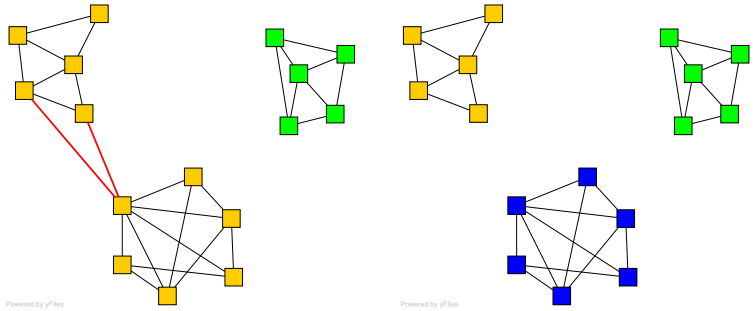
3.3 Die Funktionsweise der Algorithmen am Beispiel



(a) Der Eingabegraph

(b) Die Kante mit der höchsten Betweenness wird ausgewählt.

(c) Nach dem Löschen der Kante ergeben sich neue Zusammenhangskomponenten.



(d) Unter den verbleibenden Kanten werden wieder die mit der höchsten Betweenness gefunden.

(e) Nach einigen Schritten wird so eine gute Clusterung erreicht.

Abbildung 5: Der Algorithmus an einem Beispielgraphen

3.4 Ergebnisse

In allen Tests war $n = 100$ fest gewählt. Für die Tests wurden Graphen erzeugt mit $p_{in} \in \{0.75, 0.80, \ldots, 1\}$ und $p_{out} \in \{0, 0.05, \ldots, 0.25\}$. So ergaben sich 36 verschiedene zu testende Kombinationen. Die Tests wurden hinreichend oft wiederholt, so dass sich repräsentative Werte ergaben.

Zuerst wollten wir Graphen betrachten, die schon eine starke Clusterung aufweisen. Es war von Interesse, welcher Algorithmus diese Struktur besser erkennt bzw. ob es überhaupt merkliche qualitative Unterschiede in den Ergebnissen gibt. Dazu haben wir p_{out} fest auf 0.05 gesetzt und p_{in} variieren lassen (siehe Abb. 6). Bei dieser geringen Wahrscheinlichkeit für Interclusterkanten sind Graphen mit deutlich unterscheidbaren Clustern zu erwarten. Erstaunlicherweise lieferte der schnellere Algorithmus 2 bessere Ergebnisse. Abgesehen von

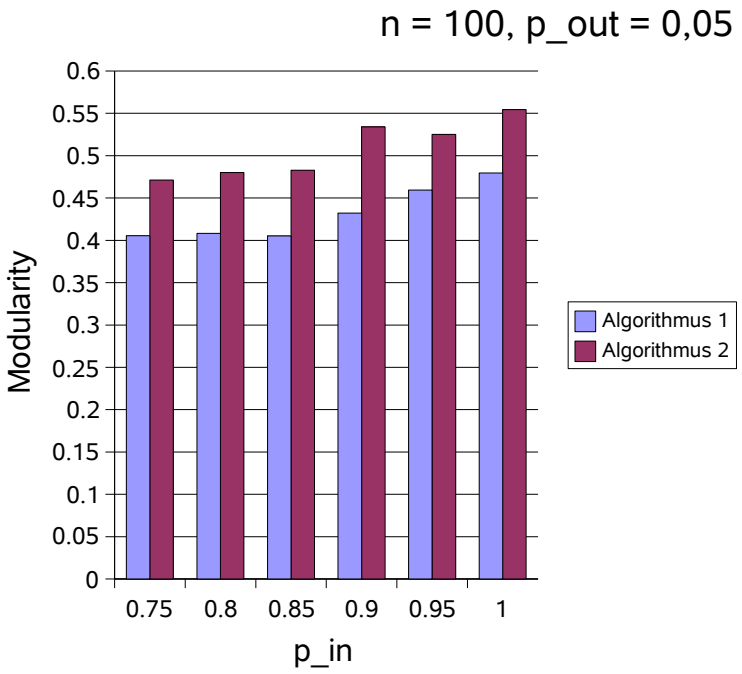


Abbildung 6: Testreihe an Graphen mit geringer Wahrscheinlichkeit für Interclusterkanten

der Laufzeit erzielte jedoch auch Algorithmus 1 gute Ergebnisse.

Nun interessierte uns, ob sich diese Ergebnisse auch bei Graphen mit weniger deutlichen Clusterungen wiederholen. Wir wählten dafür $p_{out} = 0.25$ fest und ließen wieder p_{in} variieren (siehe Abb. 7).

Hier ergab sich ein umgekehrtes Bild: Im Gegensatz zu unserer ersten Testreihe war der schnelle Algorithmus 2 hier deutlich unterlegen, seine gefundenen Clusterungen waren aber immer noch sehr akzeptabel.

Die Qualität der Ergebnisse scheint also stark vom Parameter p_{out} abzuhängen. Das

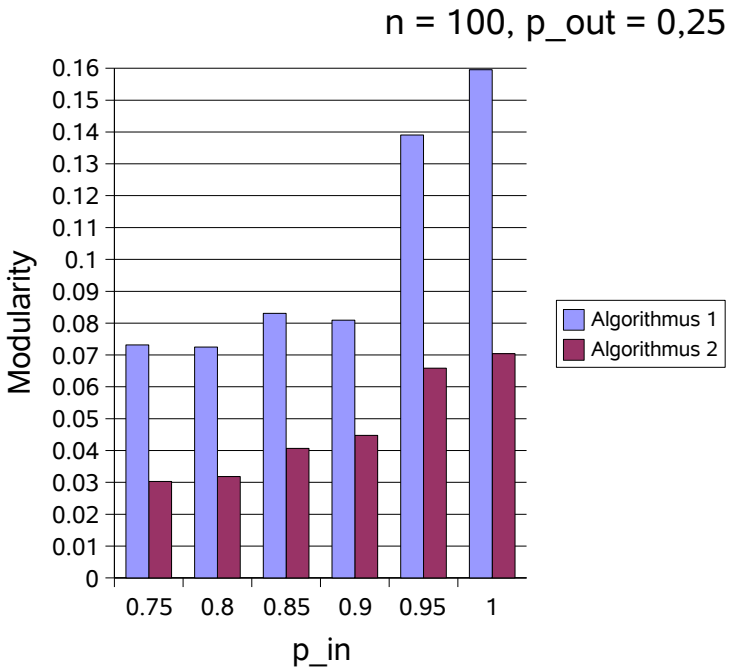


Abbildung 7: Testreihe an Graphen mit relativ hoher Wahrscheinlichkeit für Interclusterkanten

zeigt sich auch in Abbildung 8. Hier wurde p_{in} fest auf 0.9 eingestellt, während p_{out} diesmal variierte.

Hier sieht man, dass Algorithmus 2 nur für kleine p_{out}-Werte bessere Ergebnisse liefert. Insgesamt hängt die Qualität beider Algorithmen stärker von p_{out} als von p_{in} ab.

Als Ergebnis unserer Versuchsreihen lässt sich also festhalten, dass beide Algorithmen auf gut geclusterten Graphen zuverlässig arbeiten. Testet man Graphen, bei denen man eine starke Clusterung erwartet, sollte man aufgrund der deutlich besseren Laufzeit Algorithmus 2 den Vorzug geben. Auch bei anderen Graphen sind seine Ergebnisse oft gut genug. Der Mehraufwand durch seinen Einsatz fällt durch die m-fache Laufzeit des ersten Algorithmus kaum ins Gewicht.

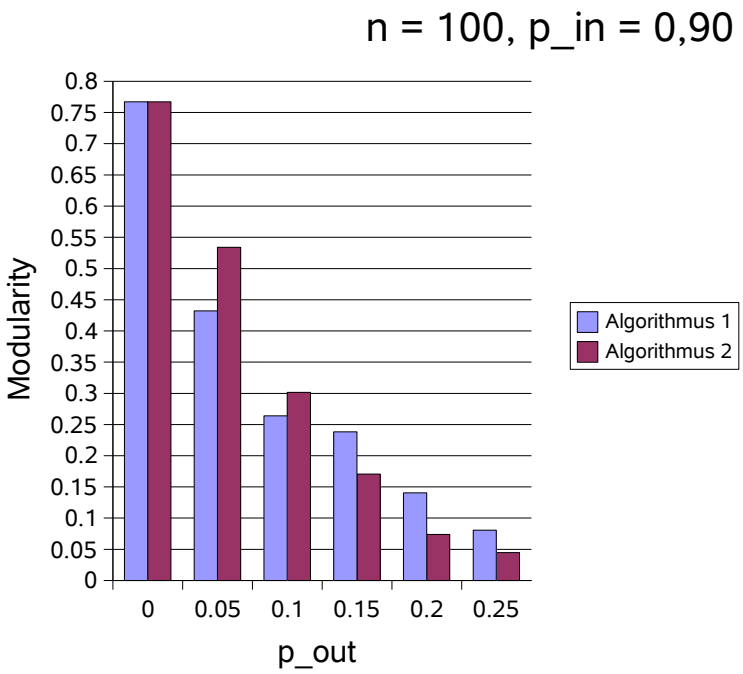


Abbildung 8: Testreihe an Graphen mit festem p_{in} und variiendem p_{out}

3.5 Optimierungsmöglichkeiten

Wir haben gesehen, dass die Laufzeit beider Algorithmen von der Betweennessberechnung dominiert wird. Eine Möglichkeit zu ihrer Optimierung stellt die Verwendung eines approximierten Verfahrens dar, bei dem nur bei einer Teilmenge von V die Breitensuche gestartet wird.

In den vorgestellten Algorithmen werden stets alle Kanten entfernt, bis der Graph nur noch einelementige Cluster enthält. Sehr kleine Cluster sind meist unerwünscht. Man könnte den Algorithmus vorzeitig abbrechen, z.B. wenn die Anzahl der Cluster eine Obergrenze erreicht hat oder eine vorher festgelegte Anzahl Kanten gelöscht wurde.

Da nach jedem Löschvorgang eine Bewertung (und bei Algorithmus 1 eine erneute Betweennessberechnung) stattfindet, können mehr Kanten auf einmal gelöscht werden, so dass der Graph schneller in Zusammenhangskomponenten zerfällt.

Diese Optimierungsmöglichkeiten haben in unseren Versuchen nur bei deutlichem Qualitätsverlust zu einer spürbaren Verbesserung der Laufzeit geführt. Hier besteht noch Bedarf an weiteren Versuchsreihen.

4 Schluss

4.1 Zusammenfassung

Im Zuge des Praktikums wurden zwei Algorithmen zur Clusterung von Graphen untersucht, die das Zentralitätsmaß der *Betweenness* nutzen. Die Algorithmen wurden implementiert und hinsichtlich ihrer Laufzeit sowie ihrer Eignung für verschiedene Familien von Graphen verglichen.

Das Ergebnis unserer Versuche war, dass beide Algorithmen auf den getesteten Szenarien zufriedenstellende Ergebnisse liefern. Aufgrund der beträchtlichen Laufzeitunterschiede ist Algorithmus 2, der nur zu Beginn einmalig die Kantenbetweenness berechnet, trotz eventuellem Qualitätsverlust oft vorzuziehen. Gerade für größere Graphen ist der erste Algorithmus nicht mehr geeignet und nur noch Algorithmus 2 praktikabel.

4.2 Rekapitulation

Da wir uns im Rahmen des Praktikums zum ersten Mal mit der Thematik des Graphenclusterns befasst haben, benötigten wir viel Zeit für die Einarbeitung. Auch die Arbeit mit den zur Verfügung gestellten Bibliotheken und Programmen bedurfte einiger Zeit. Daher fehlte am Ende die Zeit für ausführlichere Testläufe und das Implementieren der diversen Optimierungsmöglichkeiten. Hier gibt es sicher noch Raum für Verbesserungen.

Literatur

[1] U. Brandes. A faster algorithm for betweenness centrality, 2001.

[2] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On Finding Graph Clusterings with Maximum Modularity. In *Proceedings of the 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, 2007. to appear.

[3] Daniel Delling, Marco Gaertler, and Dorothea Wagner. Generating significant graph clusterings, 2006.

[4] Jubin Edachery, Arunabha Sen, and Franz-Josef Brandenburg. Graph clustering using distance-k cliques. In *Graph Drawing*, pages 98–106, 1999.

[5] Gary Flake, Robert Tarjan, and Kostas Tsioutsiouliklis. Clustering methods based on minimum-cut trees, 2004.

[6] Michelle Girvan and Mark E. J. Newman. Community structure in social and biological networks. Proceedings of the National Academy of Science 99, 2002.