AGT Summary

May 29, 2024

Contents

1 Netzwerke und Zentralität **2**

1.1 Charakterisierung der wichtigsten Ecke 2

1.2 Berechnung der Zentralitätsmaße 2

1.3 Random Walks auf Graphen 3

1.4 Eigenwert Zentralität 4

1.5 PageRank 5

2 Clustering **5**

2.1 Berechnung des Clustering Koeffizienten 6

2.2 Gomory-Hu Clustering 7

2.3 Berechnung des Gomory-Hu Baums 8

2.4 ratio cuts 8

2.5 Modularity 10

2.6 Louvain-Algorithmus 11

2.7 Label Propagation Clustering 12

2.8 Correlation Clustering 13

2.9 Cluster-Metrik 13

3 Streaming **14**

3.1 HyperLogLog 14

1 Netzwerke und Zentralität

1.1 Charakterisierung der wichtigsten Ecke

Hierfür gibt es mehrere Möglichkeiten

- größter Einfluss
- wichtig für Informationsfluss

Die Wichtigkeit wird mit einem Zentralitätsmaß gemessen.

Definition 1.1.1. Zentralitätsmaße sind sehr unterschiedlich. Es muss nur erfüllt sein, dass bei einem Sterngraphen das Zentrum das größte Zentralitätsmaß erhält. Möglich sind Bewertungen nach

1. dem Maximalgrad (*degree centrality*)
2. der durchschnittlichen Entfernung zu anderen Ecken (*closeness centrality*) (bzw. der Kehrwert davon)
3. der Anzahl der Komponenten, die mit dieser Ecke verbunden sind (*betweenness centrality*). Dafür sei $\sigma_{s,t}$ die Anzahl der kürzesten $s-t$-Wege. $\sigma_{s,t}(v)$ für $v \neq s,t$ ist dann die Anzahl der kürzesten $s-t$-Wege, die durch v gehen. Damit gilt

$$betweenness(v) = \sum_{s,t \in V()G \setminus \{v\}} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$

1.2 Berechnung der Zentralitätsmaße

Wir führen nur die Berechnung der betweenness ein. Die anderen beide Maße sind sehr einfach.
Der Algorithmus zur Berechnung von $\sigma_{s,t}$ ist an Dijkstra angelehnt. Beginnend mit s wird die Anzahl der Nachbarn von s bestimmt. Anschließend die Anzahl der Knoten mit Abstand 2 usw. Um die Komplexität der Algorithmen zu bestimmen, werden im Folgenden einige Annahmen getroffen:

1. Kotenadjazenz kann in $\mathcal{O}(1)$ bestimmt werden
2. Kanteninzidenz kann in $\mathcal{O}(1)$ bestimmt werden
3. die Nachbarschaft eines Knoten wird in $\mathcal{O}(1)$ pro Knoten bestimmt
4. die zu einem Knoten inzidenten Kanten können in $\mathcal{O}(1)$ pro Kante bestimmt werden

5. alle elementaren Operationen (z.B. Kante löschen) in $\mathcal{O}(1)$.

Auf diese Weise kann man leicht sehen, dass die Laufzeit zur Berechnung von $\sigma_{s,t}$ für alle s, t in $\mathcal{O}(n \cdot m)$ implementiert werden kann. Wir nehmen nun an, $\sigma_{s,t}$ sei bekannt und wir definieren

$$\rho_s(v) = \sum_{t \neq v} \frac{\sigma_{s,t}(v)}{\sigma_{s,t}}$$

Kennt man nun alle $\rho_s(v)$, dann ist

$$betweenness(v) = \frac{1}{2} \sum_{s \neq v} \rho_s(v)$$

Lemma 1.2.1. *Sei v ein Knoten mit Distanz mindestens $d \geq 1$ zu s und sei L die Menge der Knoten mit Distanz $d+1$ zu s. Dann ist*

$$\rho_s(v) = \sum_{w \in L \cap N(v)} \frac{\sigma_{s,v}}{\sigma_{s,w}} (1 + \rho_s(w)$$

Mit dieser Überlegung lässt sich ein Algorithmus finden, der die *betweenness* jedes Knotens in $\mathcal{O}(mn)$ berechnet.

1.3 Random Walks auf Graphen

Wir wählen zunächst einen Startknoten v_0 bezüglich einer Wahrscheinlichkeitsverteilung $\pi^{(0)}$. Anschließend wird mit Gleichverteilung ein zufälliger Nachbar v_1 von v_0 gezogen usw. Wenn man sich nun die Frage stellt, was die Wahrscheinlichkeit ist, dass der erste gezogenen Knoten (d.h. v_1) gleich dem Knoten u ist, dann entspricht das der Wahrscheinlichkeit, dass u ein Nachbar von v_0 ist mal der Wahrscheinlichkeit, dass anschließend u gezogen wird. Da der zweite Schritt gleichverteilt ist, ergibt sich

$$\pi_u(1) = \sum_{v \in N(u)} \pi_v^{(0)} \cdot \frac{1}{d(v)}$$

Das wird geschrieben als Transition Matrix mit

$$T_{uv} = \begin{cases} \frac{1}{d(v)}, & \text{wenn } uv \in E \\ 0, & \text{sonst} \end{cases}$$

Schreibt man die Wahrscheinlichkeitsverteilung $\pi^{(0)}$ einfach als Vektor, dessen Komponenten zu 1 addieren, ergibt sich

$$\pi^{(n+1)} = T\pi^{(n)}$$

für $n \geq 0$. Um zu überprüfen, ob ein bestimmter Knoten im Random Walk jemals besucht wird, muss die Grenzwertverteilung bestimmt werden

$$\pi^* = \lim_{k\to\infty} T^k \pi^{(0)}$$

Existiert π^*, dann ist π^k eine Cauchy-Folge und man kann leicht sehen, dass $T\pi^* = \pi^*$. Dann ist π^* also ein Eigenvektor zum Eigenwert 1 von T.

Theorem 1.3.1 (Perron-Frobenius)**.** *Sei $A \in \mathbb{R}^{n\times n}$ sodass $\exists k \in \mathbb{N}$ mit $A^k_{ij} > 0$ für alle $i, j \in [n]$. Dann gibt es einen eindeutigen Eigenwert λ^* mit größtem Betrag. Wenn $\lambda^* > 0$ gibt es einen positiven Eigenvektor v^* zu λ^* und alle anderen Eigenvektoren zu λ^* sind Vielfache von v^*. Ist außerdem $\lambda^* = 1$, dann konvergiert $v^{(k+1)} = Av^{(k)}$ gegen ein Vielfaches von v^* für alle positiven Startvektoren $v^{(0)} > 0$.*

Damit kann man sich überzeugen, dass T Eigenwert 1 hat und dass das der größte Eigenwert ist. Außerdem erfüllt T die Eigenschaft aus obigem Theorem, wenn G zusammenhängend und nicht bipartit ist.

1.4 Eigenwert Zentralität

Für einen Knoten v verwenden wir wieder die Matrix T und nehmen $\pi^* > 0$ als den Eigenvektor zum Eigenwert 1 mit $||\pi^*|| = 1$. Der Eintrag π^*_v ist dann die Eigenwert Zentralität von v.
Das Problem dieses Zentralitätsbegriffs ist, dass man den Eigenwert recht leicht erraten kann. Betrachte dazu

$$\overline{\pi}_v = \frac{d(v)}{2\,|E|}\ \forall v \in V$$

Es ist leicht zu sehen, dass dieser Vektor ein Eigenvektor zum Eigenwert 1 ist. Es folgt also, dass wir einen neuen Begriff haben, der aber sehr ähnlich zur *degree centrality* ist. In gerichteten Graphen ist der Begriff ein wenig hilfreicher. Der wichtigste Unterschied ist die modifizierte Matrix T mit

$$T_{vu} = \begin{cases} \frac{1}{d^+(u)}, & \text{wenn es eine Kante von } u \text{ nach } v \text{ gibt} \\ 0, & \text{sonst} \end{cases}$$

Das größere Problem sind Senken (d.h. Knoten v mit ausgehendem Grad $d^+(v) = 0$). Das kann gelöst werden, indem der Prozess neugestartet wird (d.h. eine Kante zu jedem anderen Knoten eingeführt wird).

1.5 PageRank

PageRank ist der Suchalgorithmus von Google. Er funktioniert in den folgenden Schritte, die sehr ähnlich zum Eigenwertzentralität sind

1. Wähle unter Gleichverteilung einen Startknoten
2. Mit Wahrscheinlichkeit $1-\alpha$ (α konstant) wähle einen Nachbarn und gehe dorthin.
3. Mit Wahrscheinlichkeit α wähle einen neuen Startknoten.

Die Transitionsmatrix definiert nun eine andere Matrix

$$P = (1-\alpha)T + \frac{\alpha}{n}J$$

wobei J die Matrix mit nur 1 Einträgen ist. Es ergibt sich der Prozess

$$\pi^{(k+1)} = P\pi^{(k)}$$

Da P positiv ist, ergibt Theorem 1.3.1 die Existenz der Grenzwertverteilung p_v^*. Es gilt $\text{PageRank}(v) = \pi_v^*$. Da der erste Teil von P *sparse* ist, kann die Iteration relativ effizient durchgeführt werden.

2 Clustering

Wie führen zunächst den Begriff des Clustering-Koeffizienten ein. Sei $v \in V$. dann ist

$$C(v) = \frac{|E_G[N(v)]|}{\binom{|N(v)|}{2}}$$

Der durchschnittliche Clustering-Koeffizient ist dann

$$C(G) = \frac{1}{|V|}\sum_{v\in V} C(v)$$

Ein Zufallsgraph mit Kantenwahrscheinlichkeit p hat im Erwartungswert eine Kantendichte $\frac{|E|}{\binom{n}{2}}$ von ungefähr p. Der Clustering-Koeffizient ist ebenso ungefähr p.

2.1 Berechnung des Clustering Koeffizienten

Es ist leicht zu sehen, dass man den Clustering Koeffizienten eines einzelnen Knotens v in $\mathcal{O}(d(v)^2)$ berechnen kann. Um den durchschnittlichen Wert zu bestimmen genügt daher eine Laufzeit von $\mathcal{O}(\sum_{v \in V(G)} d(v)^2)$. Die Summe lässt sich nach oben abschätzen durch $2mn$ wodurch die Laufzeit bei $\mathcal{O}(2mn)$ liegt.
Ist $d(v)$ klein, so ist der Algorithmus sehr effizient, aber ist $d(v) >> \sqrt{m}$ so lässt sich eine Verbesserung erzielen, indem für jede Kante uw überprüft wird, ob u, v, w ein Dreieck bilden. Kombiniert man diese beiden Überlegungen zu einem Algorithmus mittels einer Fallunterscheidung, so erhält man einen Algorithmus zur Berechnung des durchschnittlichen Clustering Koeffizienten mit einer Laufzeit von $\mathcal{O}(m^{\frac{3}{2}})$.
Es gibt außerdem einen randomisierten Ansatz für die Schätzung des durchschnittlichen Clustering Koeffizienten auf Graphen mit Minimalgrad mindestens 2. Hierfür wird zunächst eine Konstante $k \in \mathbb{N}$ festgelegt. Anschließend werden nacheinander k Knoten $v_1, ..., v_k$ zufällig gezogen und aus $N(v_i)$ werden jeweils zwei Nachbarn u_i, w_i zufällig gezogen. Es wird gezählt, wie viele dieser Nachbarn der k Knoten mit v_i ein Dreieck aufspannen und diese Anzahl anschließend durch k geteilt.

Theorem 2.1.1. *Sei $\varepsilon > 0, \delta > 0$ und $k = \lceil \ln\left(\frac{2}{\delta}\right) / (2\varepsilon^2) \rceil$. Dann hat der Algorithmus eine Laufzeit von $\mathcal{O}(\ln\left(\frac{1}{\delta}\right) / \varepsilon^2 \cdot \ln n)$ und mit Wahrscheinlichkeit mindestens $1 - \delta$ unterscheidet sich der berechnete Wert um maximal ε vom tatsächlichen Wert.*

Sei G ein Graph mit $V(G) = U \dot{\cup} W$. Wir schreiben

$$\partial_G = \{uw \in E(G) : u \in U, w \in W\}$$

Für $A \subset V$ ist $\partial_G(A)$ die Anzahl der Kanten zwischen A und $v \setminus A := B$. Ist w eine Funktion, die jeder Kante ein Gewicht zuordnet, definiere

$$w(F) = \sum_{e \in F} w(e)$$

für alle $F \subset E$.

Definition 2.1.2. Die *expansion* von A ist

$$\frac{w(\partial_G(A))}{\min\{|A|, |B|\}}$$

Der *ratio cut* von A ist

$$\frac{w(\partial_G(A))}{|A|\,|B|}$$

2.2 Gomory-Hu Clustering

Definition 2.2.1. Sei G ein Graph und w die Kantengewichte. Der Gomory-Hu-Baum T für G ist ein Graph mit

- $V(T) = V(G)$
- beim Löschen einer Kante uv aus dem Baum entstehen zwei Komponenten $T_{uv}(u)$ und $T_{uv}(v)$. Für alle $uv \in E(T)$ soll gelten

$$w(\partial_G(T_{uv}(u))) = \min_{X \subseteq V(G): v \notin X \ni u} w(\partial_G(X))$$

Wir definieren darauf aufbauend

$$\lambda(s,t) = \min_{X \subseteq V(G): t \notin X \ni s} w(\partial_G(X))$$

Lemma 2.2.2. *Sei G ein Graph mit Kantengewichten w und T ein Gomory-Hu-Baum für G, w. Seien $s, t \in V(G), s \neq t$, sei P der st-Weg in T und $uv \in E(P)$ mit*

$$\min_{ab \in E(P)} w(\partial_G(T_{ab}(a))) = w(\partial_G(T_{uv}(u)))$$

Dann ist $w(\partial_G T_{uv}(u)) = \lambda(s,t)$.

Theorem 2.2.3. *Für alle G, w existiert ein Gomory-Hu-Baum. Ein solcher Baum kann in $\mathcal{O}(n\tau)$ berechnet werden. τ ist dabei die Laufzeit um einen gewichtsminimalen $s-t$ Schnitt für beliebige s, t zu finden.*

Mit diesem Konzept kann nun ein Clustering in den folgenden Schritten gefunden werden

1. füge einen universellen Knoten t mit Kantengewichten α zurück
2. berechne den G-H-Baum T
3. gib die Komponenten von $T - t$ als Cluster zurück

Lemma 2.2.4. *Wir arbeiten auf einem Graphen G mit Gewichten w. Mit dem G-H-clustering erreichen wir ein Cluster $C \subseteq V(G)$. Dann gilt*

$$\frac{w(\partial_G C)}{|V \setminus C|} \leq \alpha$$

Lemma 2.2.5. *Teilt man in der Situation von oben das Cluster C noch weiter in Q, P, dann gilt*

$$\frac{w(\partial_G(P,Q))}{\min\{|P|,|Q|\}}$$

2.3 Berechnung des Gomory-Hu Baums

Lemma 2.3.1. *Die Gewichte eines Cuts sind submodular. Für alle $U, W \subseteq V$ gilt*

$$w(\partial U) + w(\partial W) \geq w(\partial(W \cap U)) + w(\partial(U \cup W))$$

Lemma 2.3.2. *Seien $s, t \in V(G)$ und sei für $X \subseteq V$ $\partial_G X$ ein minimaler $s-t$-Cut. Nun wird X zu einem neuen Knoten v_x kontrahiert. Wobei mehrfache Kanten als eine Kante mit der Summe der Gewichte eingeführt wird. Sei $p, q \in V \setminus X$ und $U \subseteq V \setminus X$ sodass $\partial_{G/X}(U \cup v_x)$ ein minimaler $p-q$-Cut om G/X ist. Dann ist $\partial_G(U \cup X)$ ein minimaler $p-q$-Cut in G.*

Definition 2.3.3 (Teil GH Baum)**.** Sei $R \subseteq V(G)$. Dann ist ein Baum $T = (R, F)$ mit einer Partition $(C_r)_{r \in R}$ von R ein GH Baum für R, wenn

$$\forall uv \in F : \partial_G \left(\bigcup_{r \in V(T_{uv}(u))} C_r \right)$$

Ist $R = V(G)$, dann entspricht der GH-Baum für R dem GH-Baum für G.

Mit dieser Überlegung lässt sich der GH-Baum von G rekursiv aufbauen.

2.4 ratio cuts

Die Idee ist, dass für eine gegebenes $k \in \mathbb{N}$ eine Partition $C_1, ..., C_k$ berechnet wird, sodass

$$\sum_{i=1}^{k} w(\partial C_i)$$

minimal ist. Damit nicht nur isolierte Knoten geclustert werden, soll der *ratio cut* minimiert werden:

$$\min_{C_1,...,C_k} \sum_{i=1}^{k} \frac{w(\partial C_i)}{|C_i|}$$

Da dieses Problem aber NP-schwer ist, soll stattdessen eine Annäherung gefunden werden.

Definition 2.4.1. Wie immer ist $G = (V, E)$ und w eine Gewichtsfunktion auf den

Kanten. Die Laplace Matrix $L \in \mathbb{R}^{V \times V}$ von G ist

$$L_{u,v} = \begin{cases} -w_{uv}, & \text{if } uv \in E \\ \sum_{e \in \delta(u)} w(e), & \text{if } u = v \\ 0, & \text{sonst} \end{cases}$$

Man kann schreiben $L = B^T \cdot D \cdot B$ wobei $D \in \mathbb{R}^{E \times E}$ im Feld (e, e) das Gewicht $w(e)$ hat und 0 sonst. B ist aus $\mathbb{R}^{E \times V}$. Für B geben wir allen Kanten aus G eine Richtung vor. In der Spalte $e = (u, v)$ steht 1 in Spalte v, -1 in Spalte u, wenn (v, u) eine Kante ist und 0 sonst.

Wir können nun $D^{1/2}$ definieren als jede Matrix die Wurzel genau die Einträge der Matrix D hat. Diese Definition ist daher sinnvoll, da D eine Diagonalmatrix ist. Es folgt $L = (D^{1/2}B)^T(D^{1/2}B)$ und

$$x^T L x = ||D^{1/2}Bx||_2^2 = \sum_{uv \in E} w_{uv}(x_u - x_v)^2$$

Lemma 2.4.2. *Seien G und w wie immer. Sei L die Laplace Matrix von G. Dann*

1. *L ist symmetrisch und positiv semi-definit*
2. *kleinster Eigenwert ist 0*
3. *ist $\mathcal{C}$ die Menge der Komponenten von G. Dann ist χ_C, $C \in \mathcal{C}$ orthogonale Eigenbasis des Eigenwerts 0.*

Zurück zu ratio cuts. Sei $A \subsetneq V$ mit $A \neq \varnothing$. Wähle

$$\mathbb{R}^V \ni z = \sqrt{\frac{|\overline{A}|}{|A|}}\chi_A - \sqrt{\frac{|A|}{|\overline{A}|}}\chi_{\overline{A}}$$

woraus folgt

$$z^T L z = |V| \sum_{uv \in \partial A} w_{uv}\left(\frac{1}{|A|} + \frac{1}{|\overline{A}|}\right) = \underbrace{|V| \left(\frac{w(\partial A)}{|A|} + \frac{w(\partial A)}{|\overline{A}|}\right)}_{\text{ratio-cut Zielfunktion}}$$

Die ratio-cut Zielfunktion zu minimieren ist also obige Matrix-Vektor-Multiplikation über alle solchen z^A zu minimieren. Wir beobachten zunächst

1. $||z^A||_2^2 = |V|$ für alle A

2. $\chi^T z^A = 0$

Da alle z^A diese Eigenschaften erfüllen, können diese als Nebenbedingung zur Optimierungsproblem ergänzt werden, ohne dass der Lösungsraum verändert wird. Minimiert man $x^T Lx$ nun über alle x mit obigen Nebenbedingungen aber streicht die Bedingung, dass $x = z^A$ für ein A, so erhält man ein relaxiertes Problem, das potentiell leichter zu lösen ist. Sei nun x^* eine Lösung für das relaxierte Problem, dann setze

$$A = \{v \in V : x_v^* \geq 0\}$$

Theorem 2.4.3 (Rayleigh-Koeffizienten)**.** *Sei $B \in \mathbb{R}^{n \times n}$ symmetrisch. Seien $\lambda_1 \leq \ldots \lambda_n$ Eigenwerte mit Vielfachheiten. Seien $v^{(1)}, ..., v^{(n)}$ eine orthogonale Basis von Eigenvektoren, sodass $v^{(i)}$ Eigenvektor zum Eigenwert λ_i ist. Es gilt*

$$\lambda_{i+1} = \min_{x \perp v^{(1)}, ..., v^{(i)}} \frac{x^T Bx}{x^T x}$$

Ein Vektor x minimiert diesen Term genau dann, wenn er ein Eigenvektor zum Eigenwert λ_{i+1} ist.

Ziel ist es nun, das ratio-cut Annäherungsverfahren von oben auf k Cluster zu verallgemeinern. Das geschieht wie folgt

1. bilde die Laplace'sche
2. berechne die Eigenvektoren $x^{(1)}, ..., x^{(k)}$ zu den k kleinsten Eigenwerten (ohne Vielfachheit). Es ist keine Einschränkung anzunehmen, dass die $x^{(i)}$ paarweise orthogonal und normiert sind.
3. sie $X \in \mathbb{R}^{V \times k}$ die Matrix gebildet aus obigen Eigenvektoren als Spalten. Sei $r^{(v)}$ die Zeile zum Knoten v
4. wende ein Clustering-Verfahren auf die Zeilen $r^{(v)}$ an
5. gebe die Cluster aus

2.5 Modularity

Hierbei handelt es sich um eine Zieflunktion für Graphen-Clusterings. Ist $\mathcal{C}$ ein Clustering, so soll $q(\mathcal{C})$ maximiert werden. Für einen ungewichteten Graphen ist

$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left(\frac{|E[C]|}{m} - \left(\sum_{v \in C} \frac{d(v)}{2m} \right)^2 \right)$$

wobei

$$E[C] = \{e = (uv) : u, v \in C\}$$

und im gewichteten Graphen

$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \left(\frac{w(E[C])}{m} - \left(\sum_{v \in C} \frac{w(\delta(v))}{2m} \right)^2 \right)$$

Motivation

- Wenn die modularity maximiert wird, so liefert das eine optimale Anzahl an Clustern
- Für einen Graphen G mit m Kante, kann ein Zufallsgraph R mit m Kanten betrachtet werden. Ist die Dichte $(\frac{|E[G]|}{\binom{|C|}{2}})$ eines Clusters in G höher als in R, so ist das ein gutes Indiz

Ziel: konstruiere einen Zufallsgraphen R auf $V(G)$, sodass die Grade jedes Knotens gleich bleiben.
Die Idee ist leicht. In R wird jeder Knoten von G übernommen aber die Kanten werden nur zur Hälfte eingesetzt. Das heißt jede Kante hat einen Anfangsknoten (damit der Grad jedes Knotens gleich bleibt) aber noch keinen Endknoten. Dann werden die Halbkanten (genannt stubs) zufällig verbunden. Bemerke, dass der resultierende Graph ein Multigraph wird. Da wir $\sum_{C \in \mathcal{C}} \frac{|E_G(C)|}{m}$ mit $\sum_{C \in \mathcal{C}} \frac{|E_R(C)|}{m}$ vergleichen wollen, müssen wir den Erwartungswert von $|E_R(C)|$ bestimmen. Es gilt

$$\mathbb{E}[|E_R(C)|] = \sum_{u,v \in C} \mathbb{E}[\text{Anzahl der Kante zwischen } u \text{ und } v] = \sum_{u,v \in C} \underbrace{\frac{d(u)d(v)}{2m-1}}_{u \neq v} + \underbrace{\frac{(d(u)-1)d(u)}{4m-2}}_{u=v}$$

Die Anzahl der Schleifen ist sehr klein. Deswegen wird dieser Term abgeschätzt mit $\sum_{u,v \in C} \frac{d(u)d(v)}{4m} = \frac{1}{4m} \sum_{u \in C} \sum_{v \in C} d(v)d(u) = \frac{1}{4m} \left(\sum_{v \in C} d(v) \right)^2$. Auf diese Weise kann die Modularity mit der eines Zufallsgraphen verglichen werden.

2.6 Louvain-Algorithmus

Hierbei handelt es sich um ein Verfahren Clusterings mit maximaler Modularity zu bestimmen. Der Algorithmus funktioniert auf gewichteten Graphen.

1. Input sind ein Graph G und ein initiales Clustering $\mathcal{C} = \{\{v\} : v \in V\}$.

2. fixiere einen Knoten v aus dem Cluster C_v und schreibe

$$q(\mathcal{C}) = \sum_{C \in \mathcal{C}} \underbrace{\left(\frac{|E[C]|}{m} - \left(\sum_{u \in C} \frac{d(u)}{2m} \right)^2 \right)}_{:= p(C)}$$

3. definiere $\Delta^v p(C_v) = p(C_v \setminus \{v\}) - p(C_v)$ und $\Delta^v p(C) = p(C(\cup\{v\})) - p(C)$.

4. in der ersten Phase werden nun Knoten so lange zwischen jeweils zwei Clustern getauscht, wie eine Verbesserung möglich ist

5. in der zweiten Phase wird jedes Cluster, das in der ersten Phase berechnet wurde zu einem Knoten kontrahiert. Der dabei entstandene Graph wird dann in die erste Phase eingegeben und so weiter

2.7 Label Propagation Clustering

1. nummeriere die Knoten in zufälliger Reihenfolge

2. für alle Knoten v, ändere die Nummer zu der häufigsten Nummer in $\delta(v)$

3. wiederhole bis Konvergenz

Es stellt sich nun die Frage, was die Zielfunktion sein soll. Sei

$$l : V \to \{1, ..., k\}$$

die Funktion der initialen Nummerierung der Knoten. Die Zielfunktion sollte dann

$$\mathcal{H}(l) = \sum_{uv \in E} \delta(l(u), l(v))$$

sein wobei δ hier das Kronecker-Delta sei. Diese Funktion wird vom Algorithmus automatisch maximiert. Problem: diese Funktion wird maximiert, wenn jeder Knoten das selbe Label hat, das heißt, wenn es nur ein Cluster mit allen Knoten gibt. Deshalb wird die Zielfunktion leicht abgeändert zu

$$\bar{\mathcal{H}}(l) = \sum_{uv \in E} \delta(l(u), l(v)) - C \sum_{\alpha} \left| l^{-1}(\alpha) \right|^2$$

2.8 Correlation Clustering

Ein Graph, der in mehrere disjunkte Cliquen zerfällt, sollte offensichtlich in diese Cliquen geclustert werden. Seinen die Cluster eines Graphen nummeriert mit $C_1, ..., C_k$, dann führt das zu der Zielfunktion

$$\min_{C_1,...,C_k} \sum_{i=1}^{k} \left(\binom{|C_i|}{2} - |E[C_i]| \right) + \sum_{i=1}^{k} \sum_{j=1, j\neq i}^{k} |\partial(C_i, C_j)|$$

In der Theorie ist dieses Verfahren sehr interessant, allerdings normalerweise nicht praktisch umsetzbar. Deswegen wird das nicht weiter verfolgt.

2.9 Cluster-Metrik

Wenn die verschiedenen Algorithmen miteinander verglichen werden, ist ein Test-Graph nötig. Solche Testfälle werden meist so aufgebaut, dass ein objektiv bestes Clustering existiert (*ground truth*). Dann muss verglichen werden, welcher Algorithmus ein Clustering produziert, dass am nächsten an dieses Clustering heranreicht. Dafür sind Clustering-Metriken nötig.

Rand Distanz: Seien A, B zwei Clusterings. Wir nennen $u, v \in V$ eine Unstimmigkeit, wenn $\exists a \in A : u, v \in a$ aber $\nexists b \in B : u, v \in b$ oder umgekehrt. $d(A, B)$ als die Anzahl der Unstimmigkeiten ist eine Metrik. Diese Metrik ist leicht zu berechnen, denn

$$d(A, B) = \sum_{a\in A} \binom{|a|}{2} + \sum_{b\in B} \binom{|b|}{2} - 2 \sum_{a\in A, b\in B} \binom{|a \cap b|}{2}$$

Entropie: Sei V der Größe n und seien A, B, C drei Clusterings. Diese werden in folgender Weise als Zufallsvariablen modelliert: wähle ein $v \in V$ mit gleichmäßiger Wahrscheinlichkeit. Dann ist

$$X_A : V \to A, v \mapsto a \ni v$$

eine Zufallsvariable wobei

$$\mathbb{P}[X_A = a] = \frac{|a|}{n}$$

Es sei dann weiter

$$H(A) = H(X_A) = -\sum_{a\in A} \frac{|a|}{n} \log_2 \left(\frac{|a|}{n} \right)$$

Die Entropie des Clusterings. Weiter ist

$$H(A, B) = -\sum_{a\in A, b\in B} \frac{|a \cap b|}{n} \log_2 \left(\frac{|a \cap b|}{n} \right)$$

und $VI(A,B) = 2H(A,B) - H(A) - H(B)$ die *variation of information*. Diese ist dann eine Metrik.

3 Streaming

3.1 HyperLogLog

Es wird eine Reihe an Zahlen eingelesen, die nicht in den Speicher passt. Wie kann die Anzahl der paarweise verschiedenen Zahlen festgestellt werden?
Die Zahlenreihe $a_1, ...a_n$ wird beim Einlesen in Binärdarstellung umgewandelt. Die Binärzahlen werden anschließend mit einer zufälligen aber deterministischen Transformation in eine andere Binärzahl konvertiert. Bei jeder transformierten Zahl werden die letzten Stellen betrachtet und die Anzahl der 0en am Ende gezählt. Die maximale Zahl R von 0en am Ende einer Zahl wird gespeichert. Ausgegeben wird 2^R.

Theorem 3.1.1. *Sei F_0 die Anzahl der paarweise verschiedenen Elemente eines Streams von m Zahlen und sei Y der Output des Algorithmus ($Y = 2^R$). Jedes a des Streams liegt in $\{1, ..., n\}$. Es wird $\mathcal{O}(\log n)$ Platz gebraucht und für alle Integer $c > 2$ ist*

$$\mathbb{P}\left[\frac{1}{c} \leq \frac{Y}{F_0} \leq c\right] \geq 1 - \frac{3}{c}$$