Ausarbeitung zu dem Artikel: Hierarchical, Parameter-Free Community Discovery Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and Philip S. Yu, 2008

Jens Kirch

25. Juli 2009

Inhaltsverzeichnis

Abbildungsverzeichnis

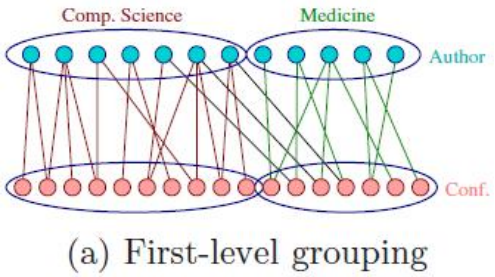
1 Motivation

In dieser Ausarbeitung wird der Artikel „Hierarchical, Parameter-Free Community Discovery“[13], der 2008 von den Autoren Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, und Philip S. Yu veröffentlicht wurde, behandelt.

In der Praxis trifft man häufig auf große Bipartite Graphen. Diese treten unter anderem in folgenden Bereichen auf:

1. Information Retrieval
2. Collaborative Filtering
3. Social Networks

Das erste Anwendungsgebiet *Information Retrieval* bildet in der Form einen bipartiten Graph, dass eine Anzahl an Dokumenten vorliegt und es verschiedene Suchanfragen mit unterschiedlichen Ausdrücken geben kann, so dass der Graph auch als *Dokument-Ausdruck-Graph* bezeichnet werden kann. In dem Gebiet des Collaborative Filtering werden Produkte zu Personen zugeordnet, so dass ein *Person-Produkt-Graph* entsteht. Als drittes Beispiel sind die *Social Networks* zu nennen bei denen Personen Gruppen zugeordnet werden.
Um die Vorgehensweise des vorgestellten Algorithmus besser zu verstehen, wird ein Beispiel benutzt. Gegeben ist eine Menge von Autoren und eine Menge von Konferenzen. Es existiert eine Kante zwischen Autor und Konferenz, wenn der betreffende Autor einen Vortrag auf dieser Konferenz gehalten hat. Zusätzlich, wie an Abbildung 1 zu erkennen, wird angenommen, dass die Autoren und Konferenzen nur aus den Bereichen der Informatik/*Computer Science* bzw. Medizin/*medicine* stammen.

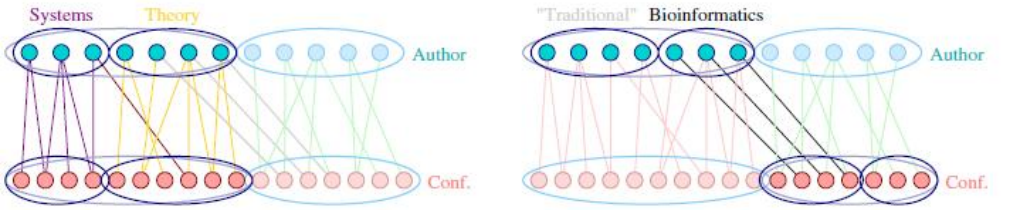


(a) First-level grouping

Abbildung 1: Bipartiter Graph - Autoren und Konferenzen

Untersucht werden sollen Bipartite Graphen zum Beispiel unter dem Gesichtspunkt von nützlichen Mustern. Das heißt das Augenmerk liegt auf der

Entdeckung von Communities. Wie sind aber Communities in diesem Fall definiert?



(b) Second-level grouping (c) Alternative second-level grouping

Abbildung 2: Mögliche Subgraphen

Anhand des Schaubilds 2 lassen sich verschiedene Unterteilungen des Graphen in Communities erkennen. Communties sind also Subgraphen des Ursprungsgraphen. In Abschnitt 3 werden diese aber vollständig definiert. Wie lassen sich diese Subgraphen nun finden und an welcher Stelle wird gesucht? Vorige Algorithmen suchten diese Muster vor allem an zwei Stellen.

1. global, Muster werden nur im gesamten Graph untersucht
2. lokal, Muster werden nur in Subgraphen der Blatt-Ebene untersucht

Das heißt entweder wurden Muster gesucht, die für den gesamten Graphen gelten oder Muster, die nur für die Blätter gelten. Der Ansatz des Artikels ist nun, die Muster auf allen Ebenen des Graphen zu entdecken und darzustellen. Dabei soll der Nutzer über den Graph navigieren können ohne Parameter eingeben zu müssen.

2 Einordnung

Um den Algorithmus besser einordnen zu können, werden im folgenden alternative Ansätze zum Thema *Data Mining* diskutiert. Große Ähnlichkeit zum vorliegenden Ansatz haben die Themen *Collaborative Filtering*[6] und *Finden überlappender Communities in sozialen Netzwerken*[4], da diese drei zum Bereich des *Community-Mining* gehören. Als Grundlage für diese drei Ansätze kann der Artikel *Ontologies are us: A unified model of social networks and semantics*[11] herangezogen werden, in dem die Grundlagen des *Community-Mining* erklärt werden. Dabei hat der Ansatz des *Collaborative Filtering* die Einschränkung, dass hier nur die globale bzw. lokale Ähnlichkeit betrachtet wird und ‚wie oben beschrieben, die Zwischenebenen außer Acht

gelassen werden. Das Thema *Finden überlappender Communities in sozialen Netzwerken* beschäfftigt sich mit der Auffindung von überlappenden Communities, die in bei dem in dieser Ausarbeitung vorgestellten Algorithmus nicht beachtet werden. Hier werden diese sogar durch eine Einschränkung ausdrücklich nicht berechnet.
Verwandt mit dem Themenbereich des *Community-Mining* ist die Subgruppen Entdeckung. Auch in dem Artikel *Tight optimisticestimates for fast subgroup discovery*[7] geht es um eine schnelle und effektive Entdeckung von Subgruppen. Dies wird durch die Benutzung einer Qualittätsfunktion realisiert.
Die untersuchten Subgruppen werden auch als häufige Muster bezeichnet. Kapitel 5.2 aus dem Grundlagenbuch *Data Mining - Concepts and Techniques*[5] und die Artikel *Komprimierte Muster*[1], *Erweiterung auf Zeichenketten in verteilten Datenbanken*Zeichenketten und *Erweiterung auf Sequenzen*[3] beschäftigen sich mit dem *Frequent Set Mining*. Eben mit häufig auftretenden Mustern. Bei dem vorliegenden Artikel werden häufig auftretende Muster in Graphen untersucht. Dies ist eine weitere Anwendung des *Frequent Set Mining* ebenso wie die beiden letzten Artikel eine Erweiterung auf Zeichenketten in verteilten Datenbanken bzw. auf Sequenzen darstellen. Das Thema zur Komprimierung der Muster findet sich auch in diesem Artikel[13] wieder, da wie später noch beschrieben die *Minimum Description Length* die möglichst effektive Kodierung einer Datenmenge umschreibt, so dass eine Komprimierung der Datenmenge stattfindet.
Die Themen zum Bereich der *Top-Down Induction of Decision Trees* haben in so fern Zusammenhänge mit dem vorgestellten Thema als, dass auch bei diesen *frequent item sets* in einer einer besonderen Art von Graph, nämlich Bäume, gesucht werden. Die Grundlagen sind im *Handbuch der künstlichen Intelligenz* unter Kapitel 14 zu finden. Allerdings wurde bei dem vorliegenden Artikel[13] keine Überlegungen zur Anwendung auf verteilte Datenbanken getroffen. Im Gegensatz dazu ist dieser Bereich für *Decision Trees* gut untersucht, so dass es zwei verschiedene Möglichkeiten mit *Cascade RSVM in peer-to-peer networks*[10] und *Distributed decision tree induction in peer-to-peer systems*[9] existieren, um die Suche nach *frequent item sets* in verteilten Datenbanken zu lösen.
Desweiteren existieren noch die *Support-Vector-Machines*, kurz *SVM*, die sich ebenfalls mit der Mustererkennung befassen. Die Grundlagen lassen sich anhand des Tutorials *A tutorial on support vector machines for pattern recognition*[2] gut nachvollziehen. Hier werden Muster mithilfe von Trennungsebenen abgegrenzt. Dies kann durch lineare Trennungsebenen geschehen. Meist reichen diese aber nicht aus und es werden Hyperebenen höherer Dimension herangezogen. Um diese Vorgehensweise noch zu optimieren können wie in

dem Artikel *Fast training of support vector machines using sequential minimaloptimization*[12] noch schnellere Algorithmen benutz werden. Zusätzlich können die *SVM* noch, wie in dem Artikel *Large Margin Methods for Structured and Interdependent Output Variables*[8]aufgezeigt, auf strukturelle Ausgaben erweitert werden. Abschließend können die gefundenen Muster mithilfe von strukturellen *SVM´s* noch mit Labeln versehen werden. Dieses Vorgehen wird im Paper *Exact and approximate inferencefor annotating graphs with structural SVMs*[14] beschrieben.

3 Grundlagen

Um auf Graphen arbeiten zu können müssen diese in eine Form gebracht werden in der diese effizient bearbeitet werden können. Dazu werden diese kodiert. Um die Kodierung der Graphen verständlicher zu machen, werden im folgenden die Grundlagen an Münzwürfen verdeutlicht. Die Anzahl der benötigten Bits für die Kodierung wird *Description Length* genannt.

3.1 Minimum Description Length

3.1.1 Kodierung

Grundlegend ist die Fragestellung die folgende: „Wie kann eine Datenmenge kodiert werden?“. Zusätzlich soll dieses Vorhaben möglichst effizient durchgeführt werden.
Um eine Datenmenge zu kodieren benötigen wir als erstes ein Modell das diese Datenmenge beschreibt. Da aber eine große Anzahl an Modellen unterschiedlicher Komplexität existiert, muss eine Auswahl stattfinden. Wie oben erwähnt sollte das ausgewählte Modell möglichst effektiv die Datenmenge kodieren. Daher sollten häufig auftretende Muster in der Datenmenge möglichst kurz kodiert werden. Daraus folgt dann auch eine möglichst große Komprimierung der Daten. Dieses Vorgehen lässt sich am einfachsten an einem Beispiel erläutern.

Betrachten wir ein einfaches Beispiel von Münzwürfen. Hierbei werden die Möglichkeiten betrachtet, die bei zwei aufeinander folgenden Münzwerfen auftreten können. In der nachfolgenden Tabelle sind die Möglichkeiten mit dem zugehörigen Code aufgezeigt.

Wurf	1xKopf1xZahl	2xZahl	2xKopf
Kodierung	1	10	11

Dabei wird die zur Hälfte auftretende Möglichkeit von *1x Kopf, 1xZahl* mit dem kürzesten Code von *1* kodiert. Die Wahrscheinlichkeit für *2xZahl* und *2xKopf* ist nur halb so groß. Daher wird an diesen Stellen ein längerer Code benutzt. Wie lässt sich nun eine Formel für die Codelänge definieren?

Die Codelänge einer Möglichkeit wird durch $l_1 = -log_2 P(z_i)$ definiert. Daraus folgt die durchschnittliche Codelänge mit Länge $\geq -\sum Pr(z_i) log_2(Pr(z_i))$. Es gilt Gleichheit, wenn $P(z_i) = A^{-l_i}$ ist. Dabei ist A, die Anzahl der verwendeten Zeichen. Für das oben genannte Beispiel würde ebenfalls die Gleichheit gelten. Dies lässt sich an folgender Gleichung ersehen.

$P(2K) = 1/4 = 2^{-2} = A^{-l_i}, A = 2, l_i = 2$

Nachfolgend wird die Shannon-Entropie H, die die Länge auf die gleiche Art berechnet, verwendet.

3.1.2 Vergleich von zwei Modellen

Nun wurde die Kodierung definiert. Aber wie sieht die Kodierung in der Praxis aus? Dies wird wieder anhand eines einfachem Beispiel von Münzwürfen behandelt.
Gegeben eine binäre Sequenz $A := [a(1), a(2), ..., a(n)]$ von n Münzwürfen. Sei $M^{(1)}$ ein einfaches Modell, das die Anzahl $h, h \leq n$ an Kopfwürfen spezifiziert. Zur Vereinfachung der Berechnung werden wir die Kosten wie folgt definieren.

Definition 1 : Codelength and description complexity 1 *Gegeben A eine Adjazenzmatrix, $M^{(1)}$ ein Modell. $C(A|M^{(1)})$ beschreibt die Code-Länge für A und $C(M^{(1)})$ beschreibt die Modell-Beschreibungskomplexität. Insgesamt ist dann $C(A, M^{(1)}) := C(A|M^{(1)}) + C(M^{(1)})$*

Um die Effektivität bei der Kodierung abzudecken wird das Prinzip der *Minimum Description Length* verwendet. Das heißt es wird stets das Modell benutzt, welches am wenigsten Bits zur Kodierung braucht.
Aus der obigen Definition folgend kann die Kodierung der Datenmenge mit $C(A|M^{(1)}) := nH(h/n)$ beschrieben werden. Dabei wird der Faktor n verwendet, da es n Münzwürfe sind. Der Anteil der Kopfwürfe lässt sich durch $C(M^{(1)}) := log^* n + \lceil log(n+1) \rceil$ beschreiben. Insgesamt ist die Codelänge

$C(A, M^{(1)}) := nH(h/n) + log^*n + \lceil log(n+1) \rceil$.
Um den Unterschied bei den verschiedenen Modellen zu betrachten wird ein zweites, komplexeres Modell herangezogen. Dieses teilt die Sequenz in zwei Teile $n_1 \geq 1$ und $n_2 = n - n_1$. Hier brauchen wir die Anzahl der Kopfwürfe für den ersten Teil, $h1 \leq n_1$, und die Anzahl für den zweiten, $h2 \leq n_2$. Die Datenmenge kann in diesem Fall durch $C(A|M^{(2)}) := n_1 H(h_1/n_1) + n_2 H(h_2/n_2)$ Bits kodiert werden. Dann kann das Modell $M^{(2)} \equiv \{h_1/n_1, h_2/n_2\}$ mit $C(M^{(2)}) := log^*n + \lceil logn \rceil + \lceil log(n-n_1) \rceil + \lceil log(n_1+1) \rceil + \lceil log(n_2+1) \rceil$ Bits beschrieben werden. Insgesamt kann die Codelänge für die komplexere Variante durch diese Gleichung $C(A, M^{(2)}) := n_1 H(h_1/n_1) + n_2 H(h_2/n_2) + log^*n + \lceil logn \rceil + \lceil log(n-n_1) \rceil + \lceil log(n_1+1) \rceil + \lceil log(n_2+1) \rceil$ beschrieben werden.

Im weiteren folgt der Vergleich der beiden Modelle. Gegeben zwei binäre Sequenzen $A_1 := \{0, 1, 0, 1, ..., 0, 1\}$ und $A_2 := \{0, ..., 0, 1, ..., 1\}$ mit jeweils 16 Münzwürfen. Nach Benutzung der zuvor definierten Formeln können folgende Werte für die erste Sequenz berechnet werden. $C(A_1, M^{(1)}) \approx 16 + 15 = 31$ und $C(A_1, M^{(2)}) \approx 15 + 19 = 34$
Aus den Werten lässt sich ersehen, dass das erste Modell weniger Bits zur Kodierung braucht. Die erste Sequenz wird zwar besser von dem zweiten Modell kodiert, aber durch die Komplexität werden mehr Bits gebraucht, so dass das Modell $M^{(1)}$ insgesamt besser komprimiert.
Für die zweite Sequenz lässt sich aus den Werten ersehen, dass das zweite Modell weniger Bits verbraucht. $C(A_2, M^{(1)}) \approx 16 + 15 = 31$ und $C(A_2, M^{(2)}) \approx 0 + 24 = 24$.
Aus den Ergebnissen lässt sich ersehen, dass es kein Modell gibt, welche jede beliebige Datenmenge minimal kodiert. Um eine möglichst effektive Kodierung zu realisieren, muss eine Auswahl aus der Menge der Modelle getroffen werden.
Nachfolgend wird die Kodierung für Graphen erläutert.

3.2 Bipartite Graphen und Kodierung

3.2.1 Bipartite Graphen

Es wurde die Kodierung anhand von Modellen aufgezeigt. Es ist in diesem Anwendungsfall nur ein Graph gegeben, keine Modelle. Wie lässt sich die Codelänge für Graphen definieren? Auch hier ist die Kodierung von Mustern, anstatt der Kodierung von einzelnen Knoten, im Zentrum der Aufmerksamkeit. Diese Muster bilden dann die gesuchten Communities. Bei der Zerlegung in Subgraphen/Communities sind drei wichtige Eigenschaften zu berücksichtigen.

- zusammenhängend, d.h. die Subgraphen sollen möglichst viele Kanten innerhalb besitzen und zu anderen Subgraphen möglichst wenige
- flexibel, die Struktur soll möglichst wenige Einschränkungen haben, aber dennoch gut automatisierbar sein
- progressiv, die Zerlegung soll in hierarchischer Weise nach und nach den Graph unterteilen

Zunächst müssen Bipartite Graphen und deren Subgraphen definiert werden.

Definition 2 : Bipartite graph and subgraph 1 *Gegeben sei ein bipartiter Graph* $G \equiv (I, J, A)$ *mit* $I := \{1, 2, ..., m\}$ *Quellknoten,* $J := \{1, 2, ..., n\}$ *Zielknoten und* $A := [a(i, j)]$ *einer binären m x n - Adjazenzmatrix.* $G' \equiv (I', J', A')$ *ist ein Subgraph von G, wenn* $I' \subseteq I, J' \subseteq J$ *und* $A' := [a(i', j')]$ *für alle* $i' \in I'$ *und* $j' \in J'$

Zusätzlich gibt es eine weitere Einschränkung bei der Zerlegung.

Definition 3 : Subgraph partitioning 1 *Sei* $G \equiv (I, J, A)$. *Der Graph wird so partitioniert, dass die Vereinigung der Subgraphen*$\{G_1, G_2, ..., G_T\}$ *wieder G entspricht*

3.2.2 Kodierung

Wie zuvor beschrieben werden wir den Graphen hierarchisch durchlaufen. Dazu wird der *top-down*-Ansatz verwendet. Bei jeder Anwendung wird eine m x n - Matrix, A, übermittelt. Diese beinhaltet entweder den gesamten Graph(auf der ersten Ebene) oder einen Teilgraphen. Bei der Kodierung sind nun zwei Fälle zu berücksichtigen.

- 1. Der Basis-Fall, definiert das Random-Graph-Modell
- 2. Der rekursive Fall, definiert das partitionierte Graph-Modell

3.2.3 Der Basis-Fall

Bei dieser Kodierungsart wird angenommen, dass in dem betreffenden Graph keine Muster enthalten sind und eine Kante mit Wahrscheinlichkeit $p(A)$ gewählt wird. $p(A)$ spezifiert die Dichtefunktion. Aus diesen Annahmen folgt, dass sich der Graph mit aus der Definition 4 ersehbaren Anzahl von Bits kodieren lässt.

Definition 4 : Random graph model 1 $C_0(A) := \lceil log(|A|+1) \rceil + \lceil |A|H(p(A)) \rceil$ *Bits*

Der erste Teil der Gleichung stellt die Kodierung der Dichtefunktion dar. Der zweite Teil die Kodierung der Kanten.

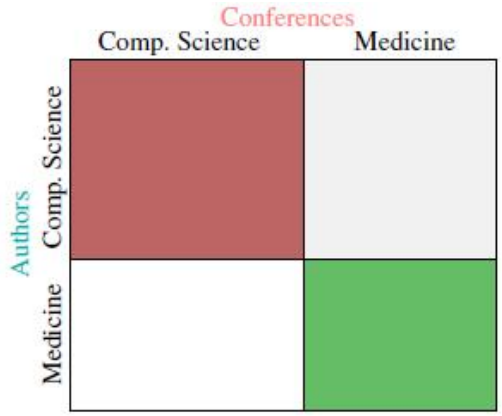
3.3 Der rekursive Fall

Den interessanteren Fall bezüglich der gegebenen Aufgabenstellung stellt der rekursive Fall dar. In dieser Situation wird von Muster im Graphen ausgegangen und es findet eine Partitionierung in Subgraphen statt. Das heißt es wird versucht Gruppen von Knoten zu finden. Da der Graph als Adjazenzmatrix vorliegt, wird die Partitionierung durch eine Zerlegung in Kacheln erreicht. Auch hier müssen drei Einschränkungen bei der Zerlegung beachtet werden:

- 1. exklusiv, es darf keine Überlappung bei der Zerlegung in Kacheln vorkommen
- 2. komplett, die komplette Adjazenzmatrix muss von den Kacheln überdeckt werden
- 3. hierarisch, die Zerlegung in Kacheln findet rekursiv statt

Allgemein erhält man so $T = k \cdot l$ Subgraph-Kacheln. Wobei k die Anzahl der Quellknotengruppen I_p beschreibt, mit $1 \leq p \leq k$ und l die Anzahl der Zielknotengruppen J_q beschreibt, mit $1 \leq q \leq l$. Die zugehörigen Submatrizen sind dann durch $A_{p,q} := [a(I_p, J_q)]$ für $1 \leq p \leq k$ und $1 \leq q \leq l$, definiert. Insgesamt folgt daraus, dass die Zerlegung einem Schachbrettmuster folgt. An dem folgenden Schaubild lässt sich die Struktur anschaulich erkennen.
Dabei ist I_1 die Menge der „computer science researcher“, I_2 die Menge der „medical researcher“. Die Konferenzen J_1 und J_2 sind analog definiert. Da die Schachbrett-Struktur auch als Kartesisches Produkt der Quell- und Zielknotenpaare aufgefasst werden kann, ergeben sich die folgenden Subgraphen:

- $G_1 = (I_1, J_1, A_{1,1})$
- $G_2 = (I_1, J_2, A_{1,2})$
- $G_3 = (I_2, J_1, A_{2,1})$
- $G_4 = (I_2, J_2, A_{2,2})$



(a) First level

Abbildung 3: Schachbrettmuster

Für die Übermittlung von k bzw. l werden $\lceil logm \rceil$ Bits bzw. $\lceil logn \rceil$ Bits benötigt. Für die Partitionierung von $I_1, ..., I_k$, d.h. der m Quellknoten in k Quellknotengruppen brauchen wir $\lceil log\binom{m}{m_1 ... m_k} \rceil$ Bits. Für die Partionierung der n Zielknoten $\lceil log\binom{n}{n_1 ... n_l} \rceil$ Bits. Unter Benutzung der Stirling-Formel bekommen wir die folgende Abschätzung $log\binom{m}{m_1 ... m_k} \approx mH(\frac{m_1}{m},...,\frac{m_k}{m})$. Durch diese Abschätzung lässt sich die Verbindung zu den Grundlagen der Kodierung erkennen. Abschließend muss die Kodierung der $k \cdot l$-Submatrizen rekursiv aufgerufen werden. Insgesamt folgt hieraus die folgende Kostengleichung:

Definition 5 : Partitioned graph 1 $C_1(A) := \lceil logm \rceil + \lceil logn \rceil + \lceil log\binom{m}{m_1 ... m_k} \rceil + \lceil log\binom{n}{n_1 ... n_l} \rceil + \sum_{p=1}^{k} \sum_{q=1}^{l} C(A_{p,q})$

Da eine rekursive Komponente in einer Kostengleichung zu viel Rechenaufwand bedeuten würde, wird dieser Teil der Gleichung durch eine Heuristik ausgetauscht. In der Praxis ist diese Heuristik trotzdem sehr gut und kommt der ursprünglichen Berechnung sehr nahe. Für den Algorithmus wird daher diese Gleichung verwendet:

Definition 6: Partitioned graph(heuristical) 1 $C_1'(A) := \lceil logm \rceil + \lceil logn \rceil + \lceil log\binom{m}{m_1 ... m_k} \rceil + \lceil log\binom{n}{n_1 ... n_l} \rceil + \sum_{p=1}^{k} \sum_{q=1}^{l} C_0(A_{p,q})$

Gegeben ein Graph mit (I, J, A), können die Kosten für die Kodierung folgendermaßen berechnet werden:

Definition 7 : Total hierarchical codelength 1 $C(A) := 1 + min\{C_0(A), C_1'(A)\}$

Im weiteren wird der Begriff der Communities für diese Anwendung definiert.

3.4 Context-specific Cluster-Tree(CCT)

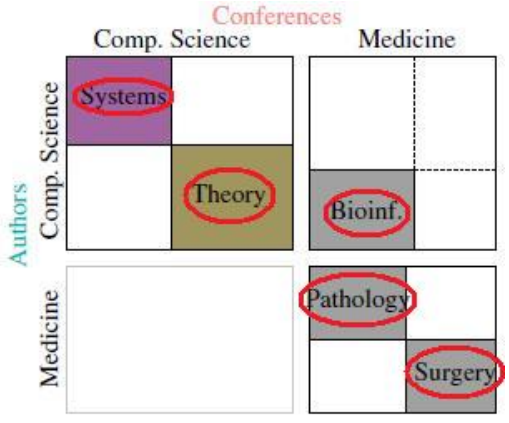
In diesem Abschnitt wird der Begriff Kontext definiert, der die gesuchten Communities beschreibt. Daraus folgend wird die Definition der gesuchten Struktur des *Cluster-specific Cluster Trees* erläutert.

3.4.1 Kontext

Da nun ein Kostenmaß für den Vergleich der Modelle in Graphen vorliegt, wird die detaillierte Definition der gesuchten Communities notwendig.

Definition 8: Context 1 *Gegeben Mengen von Paaren von Quell- und Zielknoten* (I_i, J_i)*, ein Kontext von* (I_i, J_i) *ist jedes Paar* (I_c, J_c) *für das gilt:* $I_i \subseteq I_c$ *und* $J_i \subseteq J_c$

Anschaulich bildet ein Kontext einen Aspekt der gegebenen Datenmenge ab. Daraus folgt, dass verschiedene Kontexte auch verschiedene Aspekte derselben Datenmenge abbilden. In Abbildung 4 sind mögliche Kontexte rot eingerahmt.



(b) Second level

Abbildung 4: Mögliche Kontexte

Dazu zählt zum Beispiel der Kontext der *Bioinformatik* im Schnittbereich der Autoren der Informatik, die auf Medizin-Konferenzen einen Vortrag gehalten haben.
Weiterhin ist noch eine Definition für den *minimalen hierarchischen Kontext* notwendig.

Definition 9: Minimal hierarchical context 1 *Der minimale hierarische Kontext in einer Menge von Kontexten ist der Kontext* (I_{mc}, J_{mc}) *für den gilt, dass kein anderer Kontext* (I_c, J_c) *existiert mit* $I_{mc} \subseteq I_c$ *und* $I_{mc} \subseteq J_c$

3.4.2 CCT

Mit dem Begriff des Kontextes und der vorhergehenden Zerlegung in Subgraphen lässt sich nun die gesuchte Struktur zur Darstellung der gefundenen Communities definieren.

Definition 10 : Context-specific Cluster Tree 1 *Die Menge aller Subgraphen der progressiven, hierarischen Zerlegung bildet den Context-specific Cluster Tree.*

Zur besseren Veranschaulichung der Definition zeigt der *CCT* des laufenden Beispiels, die Kontexte auf den unterschiedlichen hierarchischen Ebenen. Dieser ist in Abbildung 5 aufgetragen.

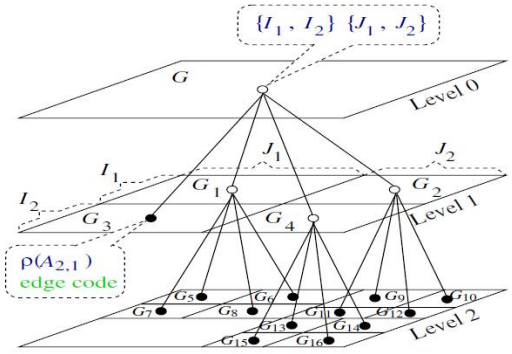


Abbildung 5: Vollständiger Context-specific Cluster Tree

Im nächsten Abschnitt wird der Algorithmus zur Berechnung des *Cluster-specific Cluster Tree* vorgestellt und die einzelnen Bestandteile erläutert. Zusätzlich wird die Laufzeit des Algorithmus vorgestellt.

4 Der Algorithmus

Zuerst wird eine kurze anschauliche Erklärung des Algorithmus folgen. Im Anschluss daran wird der Algorithmus und dessen Bestandteile im Detail erklärt.

Die Suchstrategie des Algorithmus ist ein *top-down*-Ansatz. Als erster Schritt wird das Schachbrettmuster des *parititionierten Graph-Modell* für den gegebenen Graphen ermittelt und dann mit den Kosten für das *Random-Graph-Modell* verglichen. Wenn das partitionierte Modell besser bezüglich der Kosten sein sollte, wird der Algorithmus rekursiv auf die ermittelten Subgraphen angewendent. Wenn das *Random-Graph-Modell* effizienter sein sollte, stoppt der Algorithmus. Insgesamt konvergiert der Algorithmus in den lokalen Minima der verschiedenen Subgraphen.

4.1 Bestandteile des Algorithmus

4.1.1 Hierarchical

Algorithm HIERARCHICAL:

1. Try SPLIT to find the best partitioned graph model.
2. Compare its codelength $C_1'(A)$ with that of the random graph model, $C_0(A)$.
3. If the partitioned graph model is better then, for each subgraph $(\mathcal{I}_p, \mathcal{J}_q, A_{p,q})$, for all $1 \leq p \leq k$ and $1 \leq q \leq \ell$, apply HIERARCHICAL recursively.

Abbildung 6: Algorithmus zur Auffindung des CCT´s

Dem Teilalgorithmus wird ein Graph übergeben und auf diesen wird der Algorithmus *Split* angewendet. Dieser soll das *partitionierte Graph-Modell* finden. Nach der Rückgabe von *Split* wird die Codelänge des ermittelten Modells mit der des *Random-Graph-Modell* verglichen. Wenn das partitionierte Modell kürzer sein sollte, wende *Hierarchical* auf die ermittelten Subgraphen an, ansonsten stoppe. An dem Schaubild 7 lässt sich auf dem Level 1 erkennen, dass für drei Subgraphen das partitionierte Modell besser ist und für den Subgraphen *G3* das *Random-Graph-Modell* die bessere Kodierung bedingt.

4.1.2 Split

Der Algorithmus *Split* startet mit einer Quell und einer Zielpartition. Der Algorithmus bestimmt Anzahl der Quell- und Zielpartitionen. Dazu werden als erstes die Quellpartitionen betrachtet, um diese dann zu erhöhen. Daher

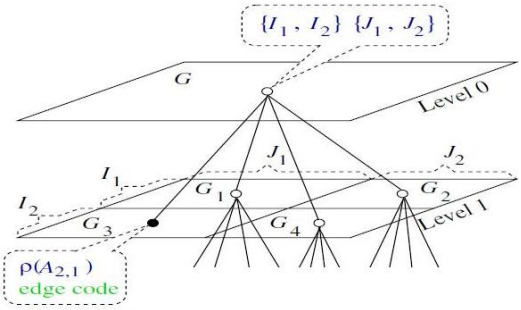


Abbildung 7: Ausschnitt des CCT´s

Algorithm SPLIT:

Start with $k^0 = \ell^0 = 1$ and at each iteration τ:

1. Try to increase the number of source partitions, holding the number of destination partitions fixed. We choose to split the source partition p^* with maximum per-node entropy, i.e.,
$$p^* := \arg\max_{1 \le p \le k} \sum_{1 \le q \le \ell} |A_{p,q}| H(\rho(A_{p,q}))/m_p.$$
Increase the number of row partitions, $k^{\tau+1} = k^\tau + 1$ and construct a partitioning $\{\mathcal{I}_1^{(\tau+1)}, \ldots, \mathcal{I}_{k^{\tau+1}}^{(\tau+1)}\}$ by moving each node i of the partition $\mathcal{I}_{p^*}^{(\tau)}$ that will be split into the new source partition $\mathcal{I}_{k^{\tau+1}}^{(\tau+1)}$, if and only if this decreases the per-node entropy of the p^*-th partition.
2. Apply algorithm SHUFFLE with initial state $\{\mathcal{I}_p^{(\tau+1)} \mid 1 \le p \le k^{\tau+1}\}$ and $\{\mathcal{J}_p^{(\tau)} \mid 1 \le p \le \ell^\tau\}$, to find better assignments of nodes into partitions.
3. If there is no decrease in total cost, stop and return $(k, \ell) = (k^\tau, \ell^\tau)$ with corresponding partitions. Otherwise, set $\tau \leftarrow \tau + 1$ and continue.

4–6. Similar to steps 1–3, but trying to increase destination partitions instead.

Abbildung 8: Algorithmus zur Bestimmung der Anzahl der Quell- und Zielpartitionen

werden die Zielpartitionen fest gehalten. Die folgenden Schritte werden noch einmal für die Zielpartitionen wiederholt und die Quellpartitionen werden nicht verändert.
Als erster Schritt wird die Partition p^* bestimmt, die den maximalen Entropiewert und daher das größte Optimierungspotenzial besitzt. Dies wird anhand der folgenden Gleichung berechnet:

$$p^* := argmax_{1 \le p \le k} \sum_{1 \le q \le l} \frac{|A_{p,q}| H(p(A_{p,q}))}{m_p}$$

Hier wird für jede Quellpartition die Anzahl der Elemente der Matrix mit der Entropie der Dichtefunktion multipliziert und dies durch die Dimension der Quellpartitionen geteilt.
Im weiteren wird die Anzahl der Quellpartitionen um 1 erhöht, so dass eine neue Partition konstruiert werden kann. Dazu wird geprüft, welche Knoten,

wenn sie in die neu konstruierte Partition verschoben werden, eine Senkung der Entropie der ausgewählten Partition p^* bedingt.
Im zweiten Schritt des Algorithmus wird die Menge der Partitionen dem Algorithmus *Shuffle* übergeben. Dieser soll die Zuteilung der Knoten zu den Partitionen verbessern.
Im dritten Schritt wird überprüft, ob die Kosten während des Algorithmus gesenkt werden konnte. Wenn dies der Fall war, wird eine weitere Iteration ausgeführt, ansonsten wird gestoppt.
Diese drei Schritte werden dann für die Zielpartitionen entsprechend wiederholt.

4.1.3 Shuffle

Algorithm SHUFFLE:
Start with an arbitrary partitioning of the matrix A into k source partitions $\mathcal{I}_p^{(0)}$ and ℓ column partitions $\mathcal{J}_q^{(0)}$. Subsequently, at each iteration t perform the following steps:

1. For this step, we will hold destination partitions, i.e., $\mathcal{J}_q^{(t)}$, for all $1 \leq q \leq \ell$, fixed. We start with $\mathcal{I}_p^{(t+1)} := \mathcal{I}_p^{(t)}$ for all $1 \leq p \leq k$. Then, we consider each source node $i, 1 \leq i \leq n$ and move it into the p^*-th partition $\mathcal{I}_{p^*}^{(t+1)}$ so that the choice maximizes the "surrogate cost gain" $C_1'(A)$ of Equation (4).
2. Similar to step 1, but swapping destination nodes instead to find new partitions $\mathcal{J}_q^{(t+2)}$ for $1 \leq q \leq \ell$.
3. If there is no decrease in surrogate cost $C_1'(A)$, stop. Otherwise, set $t \leftarrow t + 2$, go to step 1, and iterate.

Abbildung 9: Algorithmus zur Verbesserung Zuteilung der Knoten

In diesem Teilalgorithmus wird versucht die Zuteilung der Knoten zu den Partitionen zu ändern, um somit eine Verminderung der Kosten zu bewirken.
Auch hier werden wieder zuerst die Quellpartitionen betrachtet und die Zielpartitionen fest gehalten. Als ersten Schritt werden die Partitionen in der vorliegenden Iteration kopiert,um darauf arbeiten zu können. Dann wird jeder Knoten betrachtet und untersucht, ob der Knoten in eine Partition verschoben werden kann, so dass die Kosten gesenkt werden. Sollte es mehrere Partitionen geben, wird die Partition ausgewählt bei der der Gewinn maximal ist.
Im zweiten Schritt wird dieser Vorgang für die Zielpartitionen wiederholt.
Konnten die Kosten nicht verringert werden, stoppt der Algorithmus, ansonsten wird eine weitere Iteration des Algorithmus ausgeführt.

4.2 Kosten

In diesem Teilabschnitt werden die Laufzeitkosten des Algorithmus betrachtet.
Der Teilalgorithmus *Shuffle* hat eine lineare Laufzeit bezüglich der Anzahl der Kanten und Iterationen. *Shuffle* wird von *Split* aufgerufen und zwar für jede Teilung in eine neue Partition. Daraus resultiert das *Shuffle* im *worst-case*-Fall $2(k+l+1)$ mal aufgerufen werden kann. Auf der obersten Ebene des Algorithmus wird *Hierarchical* ausgeführt. Dieser Teilalgorithmus ist ebenfalls linear, da für jede Rekursion die Anzahl der Kanten von allen Partitionen zusammengenommen höchstens so groß ist wie die Anzahl der Kanten des Ursprungsgraph.
Insgesamt folgt daraus, dass der Gesamtalgorithmus proportional zu der Anzahl der Kanten, der Tiefe des Baums und der Anzahl der Partitionen ist. Somit verfügt der Algorithmus über eine lineare Laufzeit und ist sehr effizient bei der Konstruktion des *Context-specific Cluster Tree*.

5 Fazit

Der vorgestellte Algorithmus ist ein Algorithmus zur Entdeckung von Communites auf allen Ebenen des Graphs. Das heißt es ist nun möglich hierarchisch den Graph nach möglichen Communities zu durchsuchen. Zusätzlich zu dieser Neuerung ist der vorgestellte Algorithmus sehr effizient, da selbst sehr große Graphen in linearer Zeit durchlaufen werden können.
Ein weiterer Vorteil ist, dass Nutzer keine Parameter für die Suche definieren müssen und alle durch den vollständigen Context-specific Cluster Tree ermittelten Communities einsehen können.
Zukünftige Arbeiten in diesem Bereich könnten sich mit der Parallelisierung des Aufbaus des Context-specific Cluster-Tree beschäftigen, da eine parallele Bearbeitung die Ausführungszeit nochmal deutlich beschleunigen könnte. Ebenso könnte die Anwendung auf verteilte Syteme für bestimmte Szenarien von Nutzen sein.
Insgesamt ist der Algorithmus eine gute Möglichkeit um Communitites in Bipartiten Graphen zu finden.

Literatur

[1] J. V. Arno Siebes and M. van Leeuwen. Item sets thatcompress. In *Procs. SIAM Int. Conference on Data Mining*, 2006.

[2] C. Burges. A tutorial on support vector machines for pattern recognition. In *Data Mining and Knowledge Discovery*, volume 2(2), pages 121 – 167, 1998.

[3] T. C. Chedy Raissi and P. Poncelet. Mining conjunctive sequentialpatterns. In *Data Mining and Knowledge Discovery*, volume 17(1), pages 77 – 93, 2009.

[4] S. Gregory. A fast algorithm to find overlapping communities in networks. In *Walter Daelemands, Bart Goethals, and Katharina Morik, editors,Procs. ECML PKDD*, pages 408 – 423, 2008.

[5] J. Han and M. Kamber. Data mining - concepts and tech-niques. In *Morgan Kaufmann*, volume 2 edititon, 2006.

[6] R. S. Heng Luo, Changyong Niu and C. Ullrich. A collaborativefiltering framework based on both local user similarity and globaluser similarity. In *Machine Learning*, volume 72(3), pages 231 – 245, 2008.

[7] S. R. Henrik Grosskreutz and S. Wrobel. Tight optimisticestimates for fast subgroup discovery. In *Walter Daelemans, Bart Goethals,and Katharina Morik, editors, Procs. ECML PKDD*, pages 440 – 456, 2008.

[8] T. H. a. A. Ioannis Tsochantaridis, Thorsten Joachims. Large margin methods for structured and interdependentoutput variables. In *Journal of Machine Learning Research*, volume 6, pages 1453 – 1484, 2005.

[9] K. L. Kamalika Das, Kanishka Bhaduri and H. Kargupta. Distributedidentification of top-l inner product elements and its applicationin a peer-to-peer network. In *IEEE Transactions on Knowledge and DataEngineering*, 2008.

[10] C. G. Kanishka Bhaduri, Ran Wolff and H. Kargupta. Distributeddecision tree induction in peer-to-peer systems. In *Statistical Analysisand Data Mining Journal*, volume 1(2), pages 85 – 103, 2008.

[11] P. Mika. Ontologies are us: A unified model of social networks and semantics. In *Procs. ISWC*, 2005.

[12] J. Platt. Fast training of support vector machines using sequential minimaloptimization. In *B. Scholkopf, C. Burges, and A. Smola, editors, Advancesin Kernel Methods - Support Vector Learning*, page chapter 12, 1999.

[13] C. F. Spiros Papadimitriou, Jimeng Sun and P. S. Yu. Hierarchical, parameter-free community discovery. In *Walter Daelemans,Bart Goethals, and Katharina Morik, editors, Procs. ECML PKDD*, 2008.

[14] U. B. Thoralf Klein and T. Scheffer. Exact and approximate inferencefor annotating graphs with structural svms. In *Walter Daelemans,Bart Goethals, and Katharina Morik, editors, Procs. ECML PKDD*, pages 611 – 623, 2008.