Referat: Hierarchical, Parameter-Free Community Discovery

Spiros Papadimitriou, Jimeng Sun, Christos Faloutsos, and Philip S. Yu, 2008

Seminar: Aktuelle Arbeiten des Data Mining

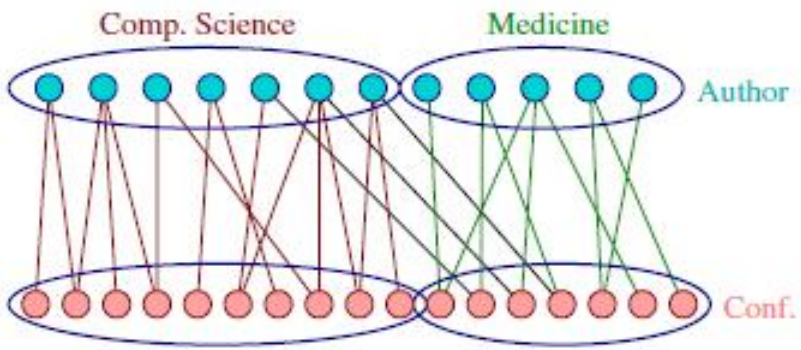
Ein Vortrag von Jens Kirch

14.07.2009

Gliederung

Problemstellung 1/2

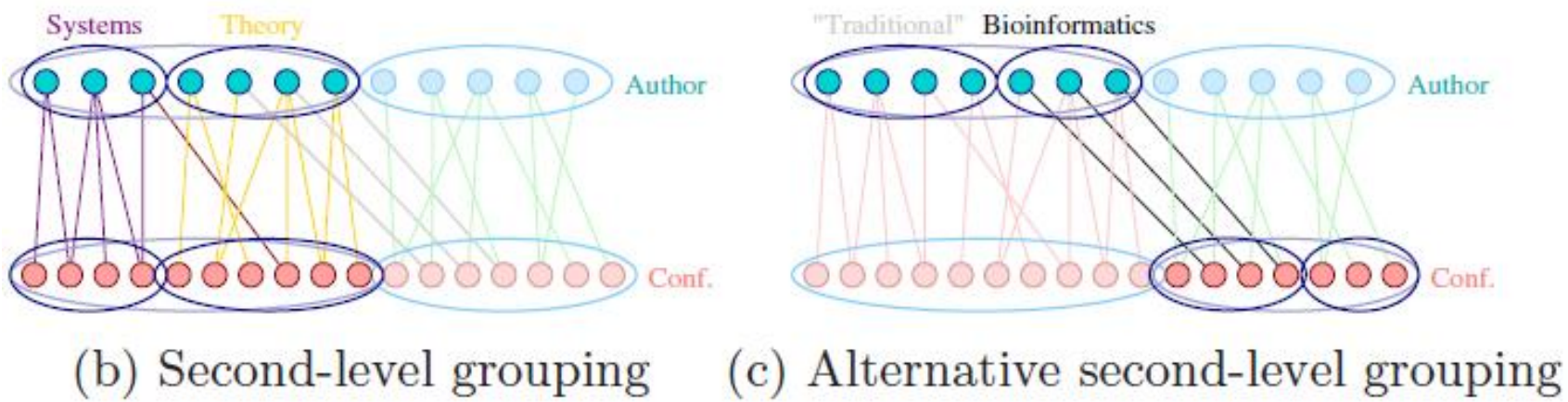
- Wir wollen bipartite Graphen untersuchen, da diese häufig anzutreffen sind z.b. in den Bereichen:
  - Information Retrieval(Dokument-Ausdruck-Graphen)
  - Collaborative Filtering(Person-Produkt-Graph)
  - Social Networks



(a) First-level grouping

Problemstellung 2/2

- Augenmerk liegt hier auf der Entdeckung von Communities
- Aber wie sind Communities definiert?



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

- Communities sind Subgraphen des Ursprungsgraphen!

Neuer Ansatz

- Aufgabe: Finde nützliche Muster innerhalb des Graphen
- Vorher: nur 2 Sichtweisen
  1. global, Muster werden nur im gesamten Graph untersucht
  2. lokal, Muster werden nur in Subgraphen untersucht
- Nun: Finden der Muster auf allen Ebenen möglich
- Nutzer kann über Graph navigieren ohne Parameter anzugeben

Minimum Description Length

- Wie kodiert man nun eine Datenmenge und dabei noch möglichst effektiv?
  - Um eine gegebene Menge von Daten zu kodieren benötigen wir ein Modell
  - Problem: Welches Modell beschreibt die Menge effektiv?
  - Lösung: Das Modell, das die Daten am besten komprimiert
  - Faustregel: Kodiere häufig auftretende Muster möglichst kurz

Beispiel: Münzwürfe

- Betrachten wir das Beispiel von Münzwürfen
- Möglichkeiten bei zwei Würfen

Wurf	1xKopf1xZahl	2xZahl	2xKopf
Kodierung	1	10	11

- $P(1K1Z) = 1/2$
- $P(2K) = P(2Z) = 1/4$

Codelänge und Entropien

- Wähle Codelänge $l_1 = -log_2 P(z_i)$
- Daraus folgt die durchschnittliche Codelänge mit Länge $\geq -\sum Pr(z_i) log_2(Pr(z_i))$
- Es gilt Gleichheit, wenn $P(z_i) = A^{-l_i}$, A Anzahl der verwendeten Zeichen
- Beispiel: $P(2K) = 1/4 = 2^{-2} = A^{-l_i}, A = 2, l_i = 2$
- Nachfolgend wird die Shannon-Entropie H, die die Länge auf die gleiche Art berechnet, verwendet

Ein einfaches Modell - Beispiel 1/4

- Sei $A := [a(1), a(2), ..., a(n)]$ eine Binäre Sequenz von n Münzwürfen
- Sei $M^{(1)}$ ein einfaches Modell, das die Anzahl $h, h \leq n$ an Kopfwürfen spezifiziert
- Wieviele Bits brauchen wir, um dieses Modell und die Sequenz zu beschreiben?

Description Length

Definition 1 : Codelength and description complexity

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 $C(A, M^{(1)}) := C(A|M^{(1)}) + C(M^{(1)})$

- Das Prinzip der Minimum Description Length besagt,dass aus der Menge der möglichen Modelle immer das Modell ausgewählt wird, welches am wenigsten Bits verbraucht

w

Modell im Detail - Beispiel 2/4

- Sei nun $M^{(1)} \equiv \{h/n\}$ gegeben, so kann die Datenmenge A mit $C(A|M^{(1)}) := nH(h/n)$ Bits codiert werden
- Zusätzlich muss der Anteil an Kopfwürfen h/n übermittelt werden
- Dafür werden $C(M^{(1)}) := log^*n + \lceil log(n+1) \rceil$ Bits benötigt
- Die Code-Länge ist dann insgesamt:
  $C(A, M^{(1)}) := nH(h/n) + log^*n + \lceil log(n+1) \rceil$

Ein komplexeres Modell - Beispiel 3/4

- Sei $M^{(2)}$ ein komplexeres Modell, das die Sequenz in zwei Teile aufteilt mit $n_1 \geq 1$ und $n_2 = n - n_1$
- Sei $h1, h2$ die Anzahl der Kopfwürfe in den jeweiligen Teilen
- A kann 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
- Die Code-Länge ist dann insgesamt : $C(A, M^{(2)}) := n_1 H(h_1/n_1) + n_2 H(h_2/n_2) + log^* n + \lceil logn \rceil + \lceil log(n - n_1) \rceil + \lceil log(n_1 + 1) \rceil + \lceil log(n_2 + 1) \rceil$

Vergleich der Modelle - Beispiel 4/4

- Sei nun $A_1 := \{0, 1, 0, 1, ..., 0, 1\}$ und $A_2 := \{0, ..., 0, 1, ..., 1\}$ Wir betrachten nun die beiden Modelle $M^{(1)}, M^{(2)}$
- Nach einsetzen in die Formeln bekommen wir für A_1:
- $C(A_1, M^{(1)}) \approx 16 + 15 = 31$ und $C(A_1, M^{(2)}) \approx 15 + 19 = 34$
- und für A_2:
- $C(A_2, M^{(1)}) \approx 16 + 15 = 31$ und $C(A_2, M^{(2)}) \approx 0 + 24 = 24$

Kodierung von Graphen?

- Problem: Keine Modelle gegeben. Gegeben ist ein Graph
- Wie ist nun die Code-Länge bei Graphen definiert?
- Nicht jeden Knoten einzeln beschreiben, sondern sinnvolle Muster finden
- Finde Subgraphen(Communities) im Graphen!
- Wichtige Eigenschaften dabei: zusammenhängend, flexibel und progressiv

Bipartite Graphen und Subgraphen

Definition 2 : Bipartite graph and subgraph

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* $j' \in J'$

Definition 3 : Subgraph partitioning

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*

Hierarchisches Enkodieren

- Um das Problem der Codierung zu lösen, werden wir den gegeben Graph rekursiv mit dem top-down Ansatz traversieren
- Gegeben ist eine m x n-Matrix A. Dabei kann A den kompletten Graph beinhalten oder nur einen Teilgraph.
- Nun sind zwei Fälle zu berücksichtigen:
  - 1. Der Basis-Fall(Random-Graph-Modell)
  - 2. Der rekursive Fall(partitioniertes-Graph-Modell)

Der Basis-Fall

- Wir nehmen an in dem Graph sind keine Muster enthalten.
- Die Daten können durch die Annahme, dass jede Kante mit Wahrscheinlichkeit $p(A)$ gewählt wird, kodiert werden
- Dabei ist $p(A)$ die Dichtefunktion der Kanten.

Definition 4 : Random graph model

$C_0(A) := \lceil log(|A|+1) \rceil + \lceil |A| H(p(A)) \rceil$ *Bits*

Der rekursive Fall

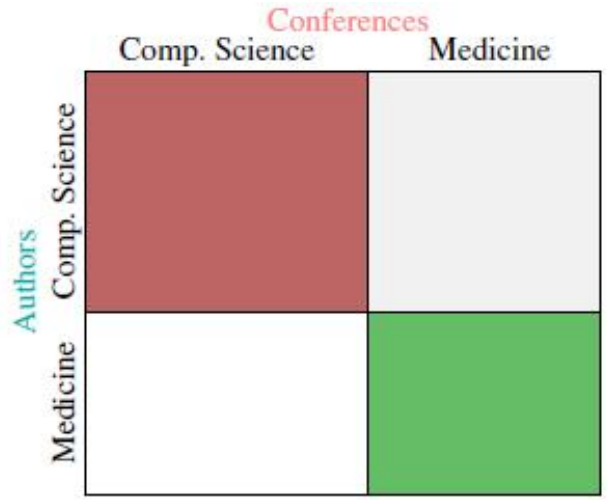
- Wir versuchen Gruppen von Knoten zu finden, die den Graph partitionieren.
- Für die gegebene Matrix kann dies durch Zerlegung in Kacheln erreicht werden.
- Die Zerlegung soll mit folgende Einschränkungen durchgeführt werden:
  - 1. exklusiv
  - 2. komplett
  - 3. hierarisch

Der rekursive Fall

- Aus diesen Einschränkungen folgt, dass die Kachelung einem Schachbrettmuster entsprechen soll.
- Allgemein ist dies eine Zerlegung in $T = k \cdot l$ Subgraph-Kacheln.
- Dabei ist k die Anzahl der Quellknotengruppen I_p mit $1 \leq p \leq k$ und l die Anzahl der Zielknotengruppen J_q mit $1 \leq q \leq l$
- Die zugehörigen Submatrizen sind dann $A_{p,q} := [a(I_p, J_q)]$ für $1 \leq p \leq k$ und $1 \leq q \leq l$

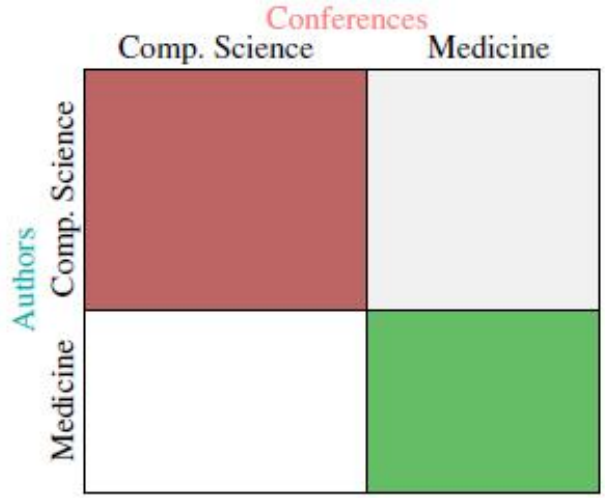
Beispiel: Schachbrettmuster

- I_1 = Menge der „computer science researcher“, I_2 = Menge der „medical researcher“. Analog für Konferenzen mit J_1 und J_2
- $G_1 = (I_1, J_1, A_{1,1})$, $G_2 = (I_1, J_2, A_{1,2})$, $G_3 = (I_2, J_1, A_{2,1})$ und $G_4 = (I_2, J_2, A_{2,2})$
- Die Schachbrett-Struktur kann dann als Kartesisches Produkt der Quell- und Zielknotenpaare aufgefasst werden.

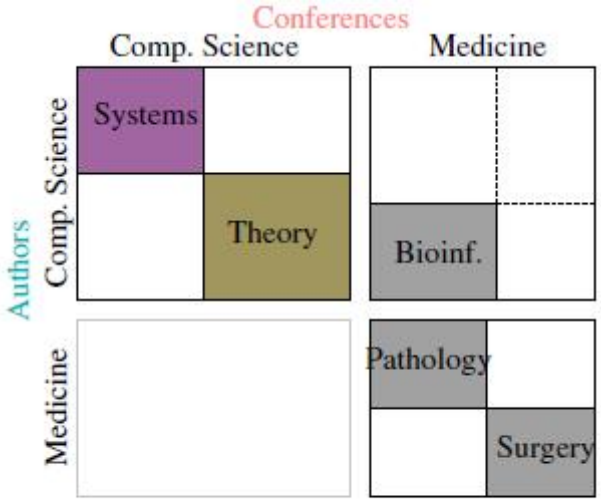


(a) First level

Beispiel: Schachbrettmuster



(a) First level



(b) Second level

Der rekursive Fall - Kosten des partionierten Graph-Modells

- Wir brauchen $\lceil logm \rceil$ Bits um k und $\lceil logn \rceil$ Bits um l zu versenden
- Für die Partitionierung von $l_1, ..., l_k$, 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 $\lceil log \binom{n}{n_1 ... n_l} \rceil$
- Unter Benutzung der Stirling-Formel bekommen wir $log \binom{m}{m_1 ... m_k} \approx mH(\frac{m_1}{m}, ..., \frac{m_k}{m})$
- Abschließend muss die Kodierung der $k \cdot l$-Submatrizen rekursiv aufgerufen werden

Problem der Kosten

Definition 5 : Partitioned graph

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

- Aber: Die Berechnung des partitionierten Graph-Modell ist so zu teuer
- Daher wird eine Heuristik benutzt:

Definition 6: Partitioned graph(heuristical)

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

Total hierarchical Codelength

- Gegeben eine hierarchische Zerlegung, die Kosten für das übermitteln von (I, J, A) sind:

Definition 7 : Total hierarchical codelength

$C(A) := 1 + min\{C_0(A), C_1'(A)\}$

Kontext

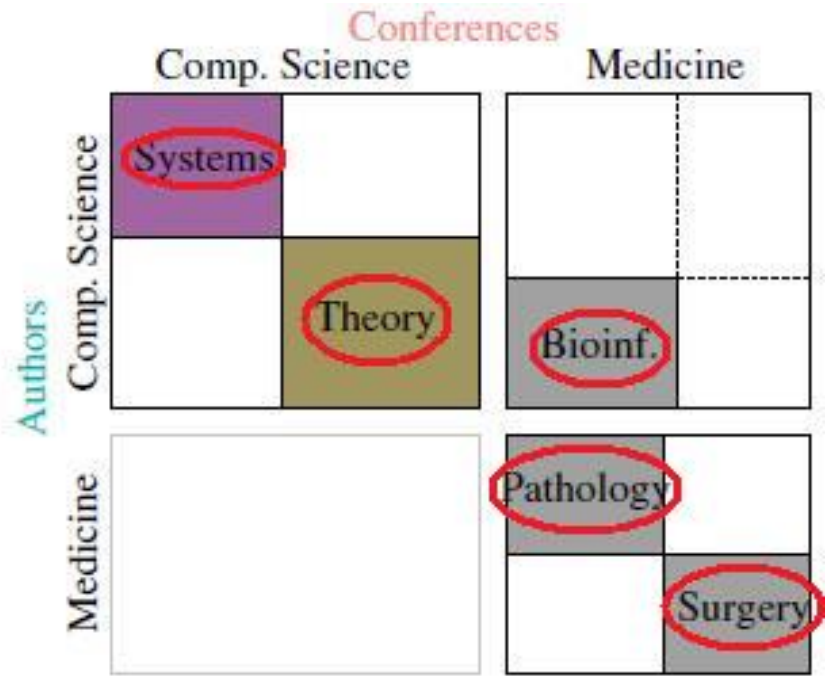
- Nun haben wir ein Kostenmaß zum Vergleich der Modelle bei Graphen definiert
- Was wollten wir im Graph finden? Communities! Wie sind diese definiert?

Definition 8: Context

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* $J_i \subseteq J_c$

- Ein Kontext bildet einen Aspekt der Daten ab. Verschiedene Kontexte bilden verschiedene Aspekte derselben Datenmenge ab.

Beispiel: Kontext



(b) Second level

Context-specific Cluster Tree(CCT)

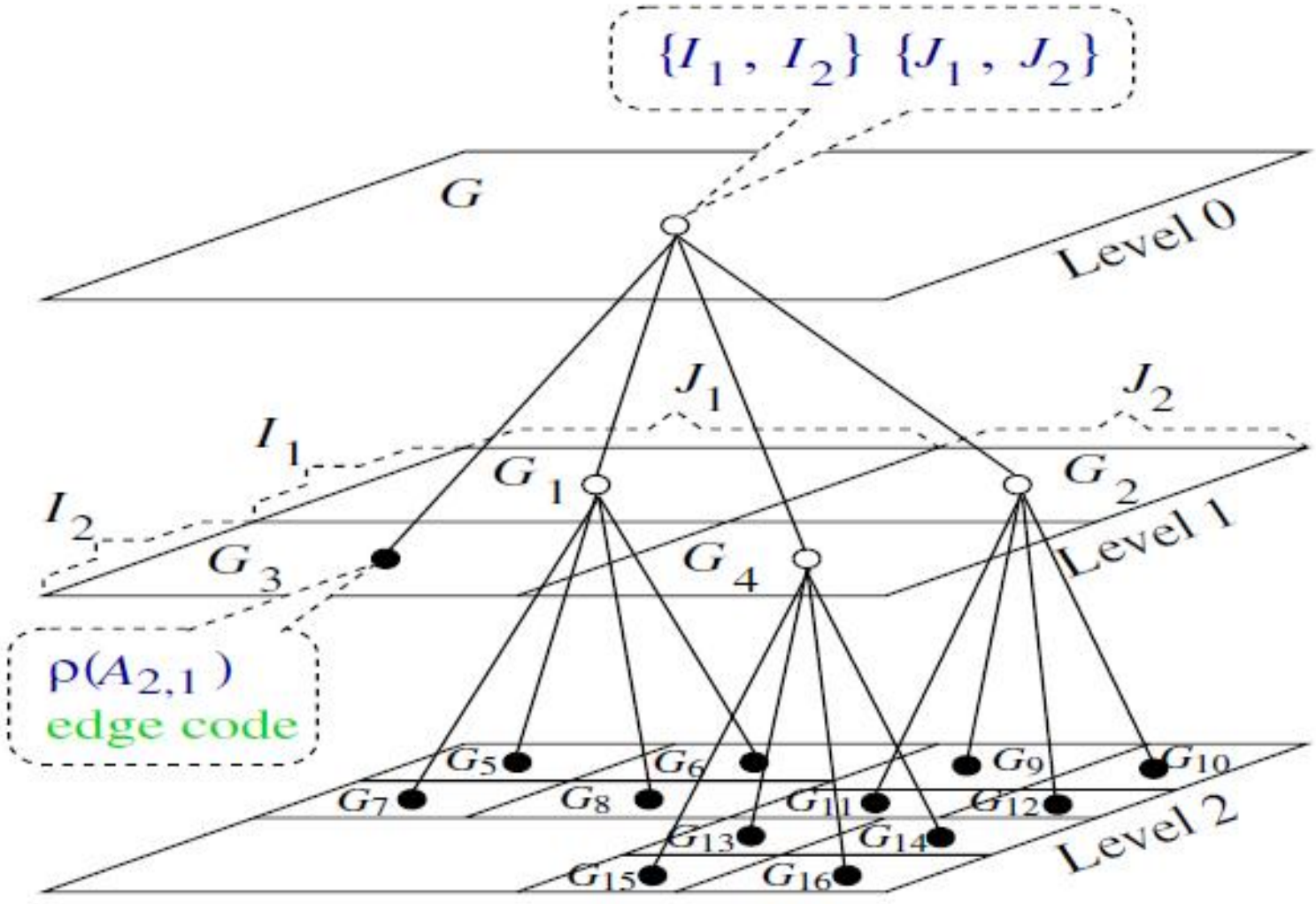
Definition 9: Minimal hierarchical context

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* $I_{mc} \subseteq J_c$

Definition 10 : Context-specific Cluster Tree

Die Menge aller Subgraphen der progressiven, hierarischen Zerlegung bildet den Context-specific Cluster Tree.

Beispiel: Context-specific Cluster Tree



Fahrplan

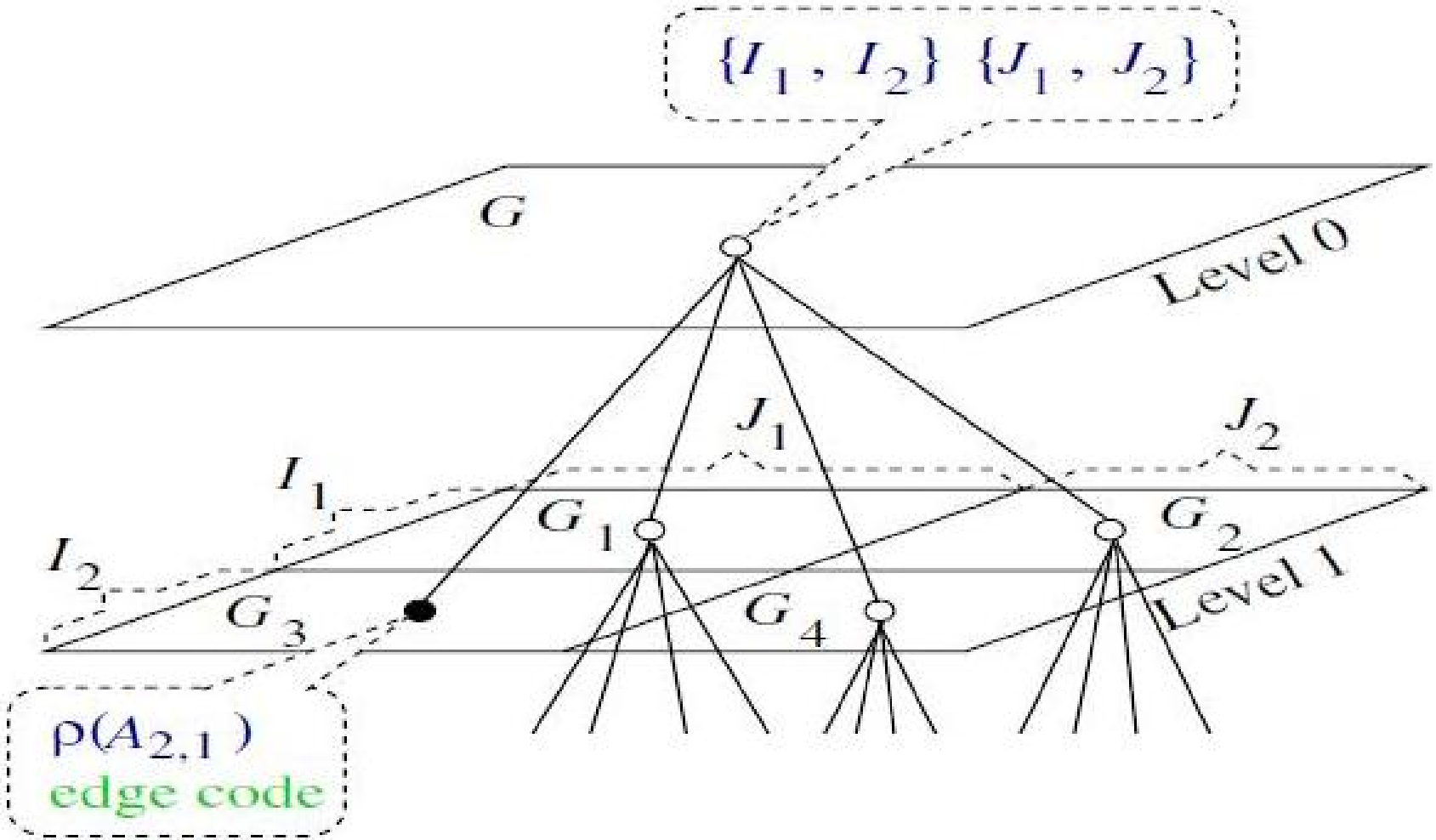
Suchstrategie für Graphen

- top-down-Ansatz
- Schachbrettmuster(Kacheln) finden für gegebenen Graph
- Prüfen welches Modell besser
- Wenn Partitioniertes besser: Rekursives anwenden des Algorithmus für die verschiedenen Kacheln
- Wenn Random-Graph-Modell besser: Stop
- Insgesamt konvergiert der Algorithmus in lokalem Minimum der verschiedenen Subgraphen

Beschreibung des Algorithmus - Hierarchical

- Benutze Algorithmus *SPLIT* bei gegebenem (Sub-)Graph und finde das partitionierte Graph-Modell
- Vergleiche die Codelänge des gefundenen partitionierten Modells mit dem Random-Graph-Modell
- Wenn partitioniertes Modell besser, wende rekursiv *Hierarchical* auf alle gefundenen Subgraphen an

Beispiel: Hierarchical



Beschreibung des Algorithmus - Split - 1/2

- Starte mit $k^0 = l^0 = 1$, Iteration $\tau = 0$

1.
   - Vorgehen: Erhöhe Anzahl der Quellpartitionen und teste, ob Knoten verschoben werden können(Anzahl der Zielpartitionen fest)
     - Wähle Quellpartition p^* mit maximaler Entropie

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

     - Erhöhe die Anzahl der Quellpartionen k um 1 für den nächsten Durchgang, $k^{\tau+1} = k^\tau + 1$,
     - konstruiere neue Partitionierung $\{l_1^{\tau+1}, ..., l_{k^\tau+1}^{\tau+1}\}$
     - Verschiebe Knoten aus der ausgewählten Partition p^*, wenn die Entnahme des Knotens eine Senkung der Entropie der jeweiligen Partition bedingt

Beschreibung des Algorithmus - Split - 2/2

1. Anzahl der Quellpartitionen erhöhen, Knoten verschieben, wenn Kosten verringert werden
2.
   - Benutze Algorithmus *Shuffle* mit $\{I_p^{\tau+1} | 1 \leq p \leq k^{\tau+1}\}$ und $\{I_p^{(\tau)} | 1 \leq p \leq l^{\tau}\}$
   - bessere Zuordnung der Knoten in die Partitionen
3. Wenn die Kosten nicht sinken, Stop und gebe $(k, l) = (k^{\tau}, l^{\tau})$ mit den zugehörigen Partitionen zurück, sonst setze $\tau = \tau + 1$
4. wiederhole Schritte 1-3 entsprechend für Zielpartitionen

Beschreibung des Algorithmus - Shuffle

- Starte k Quell-Partitionen $I_p^{(0)}$ und l Ziel-Partitionen $J_q^{(0)}$
- Bei jeder Iteration t führe folgende Schritte aus:

1.
   - Halte Zielpartitionen, $J_p^{(t)}$,für alle $1 \leq q \leq l$, fest
   - Kopiere bestehende Partitionierung, $I_p^{(t+1)} := I_p^{(t)}$, für alle $1 \leq p \leq k$
   - Versuche zu tauschen: Betrachte jeden Knoten $i, 1 \leq i \leq n$
   - Verschiebe Knoten in die Partition $I_{p^+}^{(t+1)}$, die den größten Gewinn bzgl. $C_1'(A)$ bedingt
2. Entsprechend für die Zielknoten, versuche neue Partitionen $J_q^{(t+2)}$, für $1 \leq q \leq l$, zu erstellen
3. Wenn die Kosten nicht minimiert wurden, stoppe, ansonsten setze $t = t + 2$

Kosten des Algorithmus

- *Shuffle*: linear zur Anzahl der Kanten und Iterationen
- *Split*: ruft *Shuffle* auf für jede Teilung, worst-case $2(k + l + 1)$ Teilungen
- *Hierarchical*: Pro Rekursion: Anzahl der Kanten, für alle Partitionen $\leq$ Anzahl der Kanten des Ursprungsgraph
- *Insgesamt*: Laufzeit proportional zu, Anzahl der Kanten, Tiefe des Baums und Anzahl der Partitionen

Fazit

- Hierarchisches Traversieren des Graphen möglich
- Finden von Communities auf allen Ebenen möglich
- Keine Angabe von Parametern nötig
- Sehr effizient, auch bei großen Graphen
- Zukunftsaussicht: Parallelisierung des Aufbaus des CCT´s