Behandelte Arbeit:

Ioannis Giotis, Venkatesan Guruswami:

Correlation Clustering with a fixed Number of Clusters

Theory of Computing, Volume 2 (2006), pp. 249-266

Jan Rochel (*jan.rochel@stud.uka.de*)

6. Februar 2007

1 Einleitung

In ihrer Arbeit befassen sich Giotis und Guruswami mit dem Zerteilen (engl. *to cluster*) von Graphen auf der Grundlage von *qualitativer* Information, bei der die Anzahl der Cluster im Ergebnis durch eine vorgegebene Konstante begrenzt wird. Die Erkenntnisse über diese Problemstellung waren bisher gering, obwohl verwandte Probleme schon eingehend studiert worden waren. Anwendungsgebiete sind unter anderem in der Bioinformatik die Analyse von Gengruppen, die K.I.-Forschung oder Data-Mining.

Die Problemstellung ist gegeben durch eine endliche Menge von Elementen, zu denen lediglich *qualitative* Information bezüglich ihrer *Ähnlichkeit*, *Zusammengehörigkeit* oder sonstigen Beziehung zueinander gegeben ist, also nur die Aussage ob dies für zwei der Elemente der Fall ist, oder nicht. Die Elemente sollen nun so in eine vorgegebenen Anzahl von Gruppen eingeteilt werden, dass diese Vorgaben möglichst gut berücksichtigt werden.

1.1 Darstellung des Problems

Diese Problemstellung lässt sich nun graphentheoretisch formulieren. Dazu betrachten wir die n Elemente als Knoten eines Graphen und die gegebene Information über die Beziehung zweier Elemente zueinander als seine Kanten. Hierbei sollen zwei Knoten mit einer Kante verbunden werden, die mit „+" beschriftet ist, wenn sie als ähnlich klassifiziert wurden. Ansonsten sollen sie mit einer Kante verbunden werden, die mit „−" beschriftet ist. So sprechen wir von *„+"-Kanten* und *„−"-Kanten* und erhalten einen vollständigen Graphen. Die Zusammenfassung der Elemente zu Gruppen nennen wir eine *k-Zerteilung* (engl. *k-clustering*) des Graphen in eine Menge von k *Cluster*. Gesucht ist nun diejenige k-Zerteilung des Graphen in eine vorgegebene Anzahl von k Cluster, so dass zwei Knoten möglichst dem selben Cluster zugeordnet werden, wenn sie mit einer „+"-Kante verbunden sind, ansonsten (wenn sie mit einer „−"-Kante verbunden sind) verschiedenen Clustern.

Für eine präzise Spezifikation des Problems bedarf es noch eines Maßes für die Güte einer solchen Zerteilung, das quantifiziert, wie gut die Zerteilung der Beschriftung der Kanten

entspricht. Wir verfolgen zwei komplementäre Ansätze: Wir suchen eine k-Zerteilung, bei der die Anzahl der

1. Übereinstimmungen maximal ist. *Übereinstimmungen* (engl. *agreements*) nennen wir hierbei Kanten, die der Zerteilung entsprechen, also „+"-Kanten, die zwei Knoten verbinden, die im selben Cluster liegen, und „−"-Kanten zwischen zwei Knoten, die in verschiedenen Clustern liegen.

2. Unstimmigkeiten minimal ist. Umgekehrt heißen hier *Unstimmigkeiten* (engl. *disagreements*) „+"-Kanten zwischen zwei Clustern und „−"-Kanten innerhalb eines Clusters, also Kanten, die *nicht* der Zerteilung entsprechen.

Die zugehörigen Probleme heißen entsprechend MaxAgree[k] und MinDisagree[k].

1.2 Formale Darstellung des Problems

Formal können wir das Problem folgendermaßen beschreiben: Gegeben sei ein vollständiger, ungerichteter Graph G mit n Knoten V und den Kanten $E : V \times V \to \{+, -\}$. Ein k-Tupel $M \equiv (M_1, \ldots, M_k)$ heißt k-Zerteilung, wenn jeder Knoten $v \in V$ genau in einem M_i vorkommt, also wenn die Bedingung $\hat{c}_k(G, M_1, \ldots, M_k)$ erfüllt ist, mit
$\hat{c}_k(G, M_1, \ldots, M_k) \equiv (\cup_{i=1}^{k} M_i = V) \wedge (\forall i, j \in \{1, \ldots, k\}, i \neq j : M_i \cap M_j = \emptyset)$.

Dann können wir die Menge aller k-Zerteilungen von G definieren als die Menge aller k-Tupel über der Knotenmenge V, die obige Bedingung erfüllen:
$C_k(G) = \{M \in V^k | \hat{c}_k(G, M)\}$

Nun definieren wir die beiden oben erwähnten Maße für die Güte einer Zerteilung als die Anzahl der durch die k-Zerteilung M induzierten Übereinstimmungen $agree(G, M)$ bzw. Unstimmigkeiten $disagree(G, M)$:
$agree(G, M_1, .., M_k) = |\{\{u, v\}) \in V^2 | E(u, v) = + \Leftrightarrow \exists i \in \{1, \ldots, k\} : u, v \in M_i\}|$
$disagree(G, M_1, \ldots, M_k) = |\{\{u, v\} \in V^2 | E(u, v) = + \nLeftrightarrow \exists i \in \{1, \ldots, k\} : u, v \in M_i\}|$

Jetzt können wir die nach dem jeweiligen Kriterium optimale k-Zerteilung definieren als:
MaxAgree[k]$(G) = argmax_{c \in C_k(G)} agree(G, c)$
MinDisagree[k]$(G) = argmin_{c \in C_k(G)} disagree(G, c)$

Im Anhang ist ausgehend von dieser formalen Definition ein einfacher Algorithmus implementiert. Entsprechend der Anzahl der möglichen k-Zerteilungen eines Graphen $|C_k(G)| = k^n$ hat der Algorithmus die Laufzeit $O(k^n)$.

1.3 Anmerkungen

Die Gegenstücke zu MaxAgree[k] und MinDisagree[k] ohne Beschränkung der Anzahl der Cluster im Ergebnis heißen MaxAgree und MinDisagree. Die Probleme wurden schon eingehend untersucht. Es ist bekannt, dass MinDisagree APX-schwer ist [2]. Für MaxAgree ist ein polynomialer Approximationsalgorithmus (engl. *PTAS, polynomial time approximation scheme*) verfügbar [3]. Diese Ergebnisse lassen sich jedoch nicht auf MaxAgree[k] und MinDisagree[k] übertragen, da es sich hierbei um andere Probleme handelt, die mit anderen Algorithmus gelöst werden müssen. Jedenfalls ist keine effiziente Transformation bekannt, mit der effektiv Ergebnisse so transformiert werden können, dass sie für das jeweilige Gegenstück eine brauchbare Lösung darstellen.

Es ist einfach zu zeigen, dass MaxAgree[k] und MinDisagree[k] identisch sind. Man muss sich nur klar machen, dass bei gegebener Zerteilung jede Kante des Graphen entweder eine Übereinstimmung oder eine Unstimmigkeit darstellt. So werden bei einer Maximierung der Anzahl Übereinstimmigkeiten gerade die Anzahl der Unstimmigkeiten minimiert. Das gilt auch für MaxAgree und MinDisagree.

Der Grund, in beiden Fällen zwei veschiedene Formulierungen für ein und das selbe Problem beizubehalten, ist, dass die Probleme wegen ihrer Komplexität approximativ gelöst werden müssen. Ein approximatives Herangehen ergibt eine Lösung, die in der Qualität (hier die Anzahl der Übereinstimmung/Unstimmigkeiten) vom Optimum abweicht. Dieser Fehler ist bei guten Approximationen relativ klein. Lösungen für die genannten Probleme zeichnen sich aber durch eine große (maximale) Anzahl von Übereinstimmungen und eine kleine (minimale) Anzahl von Unstimmigkeiten aus. Deshalb kann der Fehler einer solchen Lösung, im Verhältnis zu der (großen) Anzahl von Übereinstimmungen gering sein und gleichzeitig groß, im Verhältnis zu der (kleinen) Anzahl an Unstimmigkeiten. Deshalb benötigt man jeweils einen eigenen Approximationsalgorithmus, je nachdem, ob man eine gute Näherungslösung für die Maximierung der Übereinstimmung oder für die Minimierung der Unstimmigkeiten anstrebt.

2 Ergebnisse

Das Hauptergebnis dieser Arbeit sind jeweils ein polynomiales Approximationsschema (PTAS) für MaxAgree[k] und MinDisagree[k]. Außerdem wird ein einfacher Beweis dafür erbracht, dass die beiden (identischen) Probleme NP-schwer sind. Zwar war ein Beweis dafür vorher schon bekannt [4], nur war dieser viel schwieriger nachzuvollziehen als der hier vorgestellte Reduktionsbeweis. Dass trotz der APX-Schwere von MinDisagree ein PTAS für MinDisagree[k] existiert, mag verwunderlich erscheinen, zeigt aber, dass es sich um wirklich verschiedene Probleme handelt.

Weiter wird das Problem für allgemeinen Graphen diskutiert, also wenn im Graphen auch unbeschriftete Kanten zugelassen sind.

2.1 Komplexitätsbetrachtung

Satz: MinDisagree[2] ist NP-schwer
Beweis: Reduktion von GraphMinBisection

GraphMinBisection bezeichnet das Zerteilen eines ungerichteten, vollständigen Graphen mit einer geraden Anzahl n von Knoten in zwei gleich große Teile, so, dass die Anzahl der Kanten, die beide Teile verbinden, minimal ist. Es ist bekannt, dass GraphMinBisection NP-schwer ist [5].

Die **polynomielle Transformation** erfolgt in den Schritten:

- Ergänze jeden Knoten $v \in V$ mit n zusätzlichen Knoten. Wir nennen diese $n+1$ Knoten eine *Gruppe* V_v. Verbinde alle Knoten einer Gruppe untereinander.

- Beschrifte alle Kanten mit „+"und verbinde jedes unverbundene Knotenpaar mit einer „−"-Kante.

- Wende auf den entstandenen Graphen G' MinDisagree[2] an.

- Entferne alle hinzugefügten Knoten aus der gewonnenen Zerteilung.

Wir zeigen nun, dass die über die Transformation gewonnene Zerteilung eine gültige Lösung für GraphMinBisection ist. Wir stellen zunächst fest, dass MinDisagree[2] G' in zwei gleich große Cluster zerteilt, so, dass die Anzahl der Kanten, die Knoten aus verschiedenen Clustern verbinden, minimal ist. Dann muss nur noch bewiesen werden, dass alle Knoten einer Gruppe im selben Cluster liegen und beide Cluster gleich groß sind.

Dazu nehmen wir an, dass die gewonnene Zerteilung $W \equiv (W_1, W_2)$ diesen Kriterien nicht entspricht, dass also eine Gruppe V_v existiert,

- die entweder aus dem größeren Cluster stammt,

- oder deren Knoten nicht alle im selben Cluster sind.

Es kann nun gezeigt werden, dass die Anzahl der Unstimmigkeiten durch eine feste Vorschrift reduziert werden kann, dass eine solche Zerteilung also nicht optimal ist und somit nicht Ergebnis von MinDisagree[2] sein kann. Somit wäre bewiesen, dass die gewonnene Zerteilung eine gültige Lösung für GraphMinBisection darstellt.

Dazu verschieben wir dieser Regel folgend die Gruppe V_v vollständig in denjenigen Cluster, sodass die beiden Cluster sich in ihrer Größe annähern und somit die Differenz der Größen der Cluster minimal wird. Wir stellen nun fest, dass sich die Zerteilung bezüglich der geforderten Eigenschaften (gleiche Größe (klar), minimale Anzahl von Verbindungen) verbessert hat:

Da V_v nun vollständig in einem Cluster liegt lässt sich durch eine einfache (aber uninteressante) Abschätzung zeigen, dass die Anzahl der induzierten Unstimmigkeiten abgenommen hat, basierend auf der Tatsache, dass die Zerteilung wegen der Größe der Gruppen durch die große Anzahl von Unstimmigkeiten dominiert wird, die induziert werden durch

- „+"-Kanten zwischen den Knoten einer Gruppe im ursprünglichen Graphen

- „−"-Kanten zwischen den Knoten einer Gruppe und allen anderen Elementen im selben Cluster.

Satz: MinDisagree[k] ist NP-schwer
Beweis: Reduktion von MinDisagree[2]

Die **polynomielle Transformation erfolgt** ähnlich wie oben. Um eine Probleminstanz für MinDisagree[2] mit MinDisagree[k] zu lösen fügt man zum Graphen $k-2$ Gruppen mit jeweils $n+1$ Knoten hinzu. Es ist leicht zu zeigen, dass bei der Anwendung von MinDisagree[k] jede Gruppe einem seperaten Cluster zugeordnet wird und die ursprünglichen Knoten des Graphen entsprechend MinDisagree[2] zerteilt werden. Entfernt man nun die zusätzlichen $n-2$ Cluster aus der Zerteilung, erhalten wir ein gültiges Ergebnis für MinDisagree[2].

Satz: MaxAgree[k] ist NP-schwer
Beweis: MaxAgree[k] = MinDisagree[k]

Abbildung 1: PTAS-Algorithmus für MaxAgree$[k]$

---

1. Teile die Knotenmenge V in m etwa gleich große Teile $(V^1, \ldots, V^m)$, mit $m = \lceil \frac{4}{\epsilon} \rceil$.
2. Ziehe für $i = 1, \ldots, m$ jeweils eine zufällige Teilmenge S^i aus $V \setminus V^i$ der Größe $r = \frac{32^2}{2\epsilon^2} \log \frac{64mk}{\epsilon\delta}$.
3. Speichere eine beliebige Zerteilung von G.
4. Für jede Zerteilung aller Teilmengen S^i in $(S^i_1, \ldots, S^i_k)$ tue:
   (a) Für $i = 1, \ldots, m$ tue:
      i. Für alle Knoten $v \in V^i$ tue:
         A. Für $j = 1, \ldots, k$ sei $\beta_j(v) =$ „die durch v induzierte Anzahl von Übereinstimmungen, falls v in Cluster j platziert würde"
         B. Platziere v in Cluster $argmax_j \beta_j(v)$
   (b) Falls die aktuelle Zerteilung mehr Übereinstimmungen auf G induziert, als die momentan gespeicherte, speichere die aktuelle.
5. Gib die gespeicherte Zerteilung aus.

2.2 PTAS für MaxAgree$[k]$

Der Input, auf dem der Algorithmus operiert, setzt sich wie folgt zusammen:
Ein Graph $G = (V, E)$ mit n Knoten V und Kanten $E : V \times V \to \{+, -\}$; eine Konstante $k \in N$ mit $k \geq 2$; ein Güteparameter $\epsilon > 0$; ein Probabilitätsparameter $0 < \delta < 1$.

Der Algorithmus gibt eine k-Zerteilung von G aus, die mit einer Wahrscheinlichkeit größer als $1-\delta$ in der Anzahl der Übereinstimmungen höchstens um $\epsilon n^2/2$ von der bestmöglichen k-Zerteilung für G abweicht.

Die Laufzeit ergibt sich zu $n \cdot k^{O(\epsilon^{-2} \log(\frac{k}{\epsilon\delta}))}$.

2.2.1 Beschreibung des Algorithmus

Das Lösen des Problems in Polynomialzeit basiert darauf, dass nur für eine kleine Teilmenge von V alle möglichen k-Zerteilungen untersucht werden. Diese k-Zerteilungen dienen dann als Bewertungsmaßtab für die Zuordnung der restlichen Knoten $v \in V$ in die Cluster $1, \ldots, k$.

Dazu wird die Knotenmenge V des Graphen G in $m = \lceil \frac{4}{\epsilon} \rceil$ gleich große Teile $V^1, \ldots, V^m$ unterteilt. Für alle $i = 1, \ldots, m$ ziehen wir eine zufällige Teilmenge S^i aus $V \setminus V^i$ mit der Größe $r = \frac{32^2}{2\epsilon^2} \log \frac{64mk}{\epsilon\delta}$.

Nun wird für jede mögliche k-Zerteilung aller Teilmengen S^i eine Scheife durchlaufen, in der jedes $v \in V$ demjenigen Cluster zugeordnet wird, für den die Anzahl der Übereinstimmungen, die durch Kanten, die ihn mit den Knoten der entsprechenden Teilmenge verbindet, induziert wird, maximal ist.

In einem solchen Schleifendurchlauf ist also eine Menge von k-Zerteilungen $\{(S_1^i, \ldots, S_k^i) | i = 1, \ldots, m\}$ der Teilmengen $S^1, \ldots, S^m$ gegeben. Wir untersuchen nacheinander alle $v \in V$. Für ein solches $v \in V^i$ betrachten wir die zugehörige k-Zerteilung $(S_1^i, \ldots, S_k^i)$ von V^i. Wir ordnen nun v genau dem Cluster $j \in \{1, \ldots, k\}$ zu, für den $|\{u \in S^i | E(u, v) = + \Leftrightarrow u \in S_j^i\}|$ maximal ist. In jedem Schleifendurchlauf erhalten wir so eine k-Zerteilung des gesamten Graphen, die auf den k-Zerteilungen der Teilmengen $S^1, \ldots, S^m$ basiert, die in diesem Schleifendurchlauf ausprobiert werden.

Zuletzt wird diejenige der erhaltenen k-Zerteilungen des gesamten Graphen ausgegeben, die die meisten Übereinstimmungen induziert.

2.2.2 Performance-Analyse

Der Beweis für der o.g. Gütegarantien für das Ergebnis erfolgt über eine Abschätzung mit dem Ergebnis, dass die Anzahl der Übereinstimmungen, die bei der Abarbeitung einer Gruppe V^i gegenüber der optimalen k-Zerteilung ,,verloren gehen" können mit hoher Wahrscheinlichkeit klein ist.

Hierbei wird argumentiert, dass ja alle möglichen Zerteilungen der Teilmengen $S^1, \ldots, S^m$ betrachtet werden, also auch diejenigen, bei denen die Elemente $\cup_{i=1}^m S^i$ der Teilmengen gerade den selben Clustern zugeordnet wurden, wie bei der optimalen Zerteilung. Über eine probabilistische Abschätzung mit Hilfe von Chernoff- und Markov-Ungleichungen wird gezeigt, dass diese betrachtete k-Zerteilung eine gute Bewertungsgrundlage darstellt und mit hoher Wahrscheinlichkeit fast alle Knoten richtig zugeordnet wurden.

2.3 PTAS für MinDisagree$[k]$

Als Input für den Algorithmus ist vorgesehen:
Ein Graph $G = (V, E)$ mit n Kanten V und Knoten $E : V \times V \to \{+, -\}$; eine Konstante $k \in N$ mit $k \geq 2$; ein Güteparameter $\epsilon > 0$; eine weitere Konstante $c_1 > 0$, als brauchbarer Wert wird $c_1 = \frac{1}{20}$ vorgeschlagen.

Sei γ so definiert, dass die optimale k-Zerteilung von G γn^2 Unstimmigkeiten auf G induziert, dann besteht das Ergebnis aus einer k-Zerteilung, die höchstens $\gamma n^2(1+\epsilon/3)(1+\epsilon/4+8k^4\gamma/c_1^2)$ Unstimmigkeiten auf G induziert.

Die Laufzeit ist $n^{O(\frac{k^4}{\epsilon^2})} \cdot \log n$

2.3.1 Beschreibung

Der vorgestellte Algorithmus basiert auf der Erkenntnis, dass eine gute Näherungslösung für MaxAgree$[k]$ auch eine gute Näherungslösung für MinDisagree$[k]$ darstellt, betrachtet man *ausschließlich die großen Cluster* im Ergebnis. Knoten in großen Clustern zeichnen sich nämlich dadurch aus, dass sie tendenziell mit vielen Knoten im selben Cluster mit einer ,,+"-Kante verbunden sind. Somit ist es für diese Knoten höchst unwahrscheinlich, dass sie bei einer optimalen k-Zerteilung einem anderen Cluster zugordnet wären, weil diese ,,+"-Kanten dann zu einer entsprechend großen Anzahl an Unstimmigkeiten führen würden.

So steht am Anfang des Algorithmus für MinDisagree$[k]$ ein Abschnitt, der prinzipiell

dem oben vorgestellten Algorithmus für MaxAgree[k] entspricht, bis auf die Tatsache, dass der Graph nicht in Gruppen eingeteilt wird, sondern als Ganzes behandelt wird. Es wird also wiederum eine kleine Teilmenge S von V gezogen, für die jede mögliche k-Zerteilung betrachtet wird und auf Basis jeder dieser Zerteilungen der ganze Graph zerteilt wird.

Wir betrachten nun die s kleinsten der erhaltenen Cluster mit weniger als $\frac{n}{2k}$ Elementen, als diejenigen Cluster, die potenziell im erheblichen Umfang Knoten enthalten, die falsch zugeordnet wurden. Nun zerteilen wir die Vereinigung dieser Cluster in einem rekursiven Aufruf des Algorithmus erneut, diesmal in s Cluster und mit strengerer Gütegarantie $\epsilon/3$. Die Zusammenführung der größeren Cluster mit den s neuen Clustern stellt dann die Lösung für einen Schleifendurchlauf dar. Als Ergebnis wird dann wiederum die k-Zerteilung des Graphen ausgegeben, für die die Anzahl der Unstimmigkeiten minimal ist.

2.3.2 Performance-Analyse

Der Beweis der Schranken begründet sich wieder auf der Tatsache, dass in einem der Schleifendurchläufe die Teilmenge S so zerteilt wurde, wie das bei der optimalen k-Zerteilung von G der Fall wäre. Über eine aufwändige und schwer vermittelbare Abschätzung kommt man zu dem Ergebnis, dass die Anzahl der falsch zugeordneten Knoten, die aus großen Clustern stammen, zusammen mit denen aus den kleinen Clustern mit hoher Wahrscheinlichkeit hinreichend gering ist.

2.4 Komplexität auf allgemeinen Graphen

Erweitert man die Problemstellung dahingehend, dass über zwei Knoten keine Aussage darüber getroffen werden kann, ob diese zusammenhängend/ähnlich oder unähnlich/getrennt sind, so erhalten wir einen allgemeinen Graphen.

2.4.1 MaxAgree[k]

Satz: Auf allgemeinen Graphen existiert für alle $k \geq 3$ ein Faktor-0.7666- Approximationsalgorithmus für MaxAgree[k] (und ein Faktor-0.878- Approximationsalgorithmus für $k = 2$).

Die Erkenntnis basiert auf einer Anpassung des MaxCut-Problems, für das entsprechende Algorithmen bekannt sind.

2.4.2 MinDisagree[k]

Satz: Auf allgemeinen Graphen existiert für kein $k \geq 3$ ein Approximationsalgorithmus mit konstantem Faktor MinDisagree[k]. Für $k = 2$ existiert ein $O(\sqrt{\log n})$- Approximationsalgorithmus.

Beweis: MinDisagree[2] ist reduzierbar auf Min2CNFDeletion für das ein $O(\sqrt{\log n})$-Approximationsalgorithmus verfügbar ist. Für $k \geq 3$ kann der Speziallfall, in dem der Graph keine „+"-Kanten enthält, durch eine Reduktion von k-Coloring gelöst werden. Daraus folgt, dass es für eine solche gegebene Instanz von MinDisagree[k] NP-schwer

ist, ob die Anzahl der Unstimmigkeit bei der optimalen k-Zerteilung größer 0 ist oder nicht.

2.5 Zusammenfassung

Die Ergebnisse von Giotis und Guruswami stellen zumindest theoretisch eine erhebliche Verbesserung der Situation beim Clusten von Graphen dar. Denn der Effizienzgewinn ist für viele Anwendungsfälle attraktiv, selbst wenn dafür eine Beschränkung der Cluster-Anzahl auf einen konstanten Wert in Kauf genommen werden muss.

Jedoch darf bezweifelt werden, ob der vorgestellte Algorithmus in seiner dargestellten Form so anwendbar ist. Denn die zufällige Teilmenge des Graphen, die im PTAS für MaxAgree$[k]$ verwendet wird, ist mit seiner definierten Kardinalität von $r = \frac{32^2}{2\epsilon^2} \log \frac{64mk}{\epsilon\delta}$ auch für gnädig gewählte Werte für m, k und ϵ so groß, dass der Umfang des Graphen den machbaren Rahmen jedes Anwendungsfalles sprengen würde.

Für eine Validierung der Ergebnisse in der Praxis steht eine Anpassung des Algorithmus und dessen Implementierung also noch aus.

3 Anhang

Es folgt der Quelltext dreier Haskell-98-Module, die eine Implementierung eines $O(k^n)$-Algorithmus zur Lösung des MaxAgree[k]/MinDisagree[k]-Problems beinhalten.

```
module Main where

import Clustering (maxAgree, minDisagree)
import Graph (graph)

main :: IO ()
main = optClusterRandomGraph 8 3

optClusterRandomGraph :: Int -> Int -> IO ()
optClusterRandomGraph n k = do
  g <- graph n -- Generate a random graph g with n vertices
  putStrLn $ show g -- Output graph g as an n*n correlation matrix
  putStrLn $ show $ maxAgree k g -- Output clustering calculated by maxAgree
  putStrLn $ show $ minDisagree k g -- and by minDisagree
  -- If the implementation is correct both clusterings should be equal
  -- Clustering is shown as an array of Tuples.
  -- Read a tuple (i,j) as an assignment of vertex i into cluster j
```

```
module Clustering where

import Data.Array
import JanniLib (maximize, minimize, combine)
import Graph (Graph, Vertex, connected, graphBounds)

type Cluster = Int

type Clustering = Array Vertex Cluster

-- Maximize the number of induced agreements on graph g over the set of all
-- possible k-clusterings of g.
maxAgree :: Int -> Graph -> Clustering
maxAgree k g = maximize (agreements g) (clusterings k $ graphBounds g)

-- Minimize the number of induced disagreements on graph g over the set of all
-- possible k-clusterings of g.
minDisagree :: Int -> Graph -> Clustering
minDisagree k g = minimize (disagreements g) (clusterings k $ graphBounds g)

-- Number of agreements induced by clustering c on graph g
agreements :: Graph -> Clustering -> Int
agreements g c = length $ filter id $ agreements' (assocs c) where
  agreements' [] = []
  agreements' (x:xs) = map (check x) xs ++ agreements' xs
  check :: (Vertex,Cluster) -> (Vertex,Cluster) -> Bool
  check (v1,c1) (v2,c2) = connected g v1 v2 == (c1 == c2)

-- Number of disagreements induced by clustering c on graph g
disagreements :: Graph -> Clustering -> Int
disagreements g c = length $ filter id $ disagreements' (assocs c) where
  disagreements' [] = []
  disagreements' (x:xs) = map (check x) xs ++ disagreements' xs

  check :: (Vertex,Cluster) -> (Vertex,Cluster) -> Bool
  check (v1,c1) (v2,c2) = connected g v1 v2 /= (c1 == c2)


type Assignment = (Cluster,Vertex) -- Assign vertex to a cluster

-- All possible k-clusterings for a graph with vertices bounded by vs
clusterings :: Int -> (Vertex,Vertex) -> [Clustering]
clusterings k vs = map (array vs) assignments where
  assignments :: [[Assignment]]
  assignments = combine [ [(i,j) | j <- [1..k]] | i <- range vs]
```

```
-- This is a short excerpt from the collection of useful functions JanniLib
-- version 0.4

module JanniLib where

import qualified System.Random as SR (randomRIO)
import qualified Control.Monad as CM (liftM)

-- | Humans start counting from one, not zero. I am a human!
-- | I index a list xs with [1 .. length xs]
(!) :: [a] -> Int -> a
xs ! i = xs !! (i - 1)

randomElem :: [a] -> IO a
randomElem xs = CM.liftM (xs!) (SR.randomRIO (1, length xs))

-- | combine [[1,2],[3,4],[5]] = [[1,3,5],[1,4,5],[2,3,5],[2,4,5]]
combine :: [[a]] -> [[a]]
combine = combineBy (:)

combineBy :: (a -> [a] -> [a]) -> [[a]] -> [[a]]
combineBy f [] = []
combineBy f [xs] = [ [x] | x <- xs ]
combineBy f (xs:xss) = [ f x xs | x <- xs, xs <- combineBy f xss ]

-- Find the optimum out of a list of elements.
-- c: Compare two weightings, returning True if the first value is better.
-- w: Weigh a value
optimum :: (w -> w -> Bool) -> (a -> w) -> [a] -> a
optimum _ _ [    ] = error "optimum _ _ []"
optimum c w (x:xs) = select (w x,x) (zip (map w xs) xs) where
  select (w,x) [] = x
  select (w,x) ((v,y):zs) = if c w v then select (w,x) zs else select (v,y) zs

-- Find the maximum
maximize :: Ord w => (a -> w) -> [a] -> a
maximize = optimum (>=)

-- Find the minimum
minimize :: Ord w => (a -> w) -> [a] -> a
minimize = optimum (<=)
```

4 Literatur

1. Ioannis Giotis, Venkatesan Guruswami: Correlation Clustering with a fixed Number of Clusters, *Theory of Computing*, Volume 2 (2006), pp. 249-266

2. M. Charikar, V. Guruswami, A. Wirth: Clustering with qualitative information, *Journal of Computer and System Sciences*, 71(3):360-383, October 2005.

3. N. Bansal, A. Blum, S. Chawla: Corralation clustering, *Machine Learning, Special Issue on Clustering*, 56:89-113, 2004

4. R. Shamir, R. Sharan, D. Tsur: Cluster graph modification problems, *Proc. of 28th Workshop on Graph Theory (WG)*, pp 379-90, 2002

5. W. Fernandez de la Vega, Marek Karpinski, Claire Kenýon: A Polynomial Time Approximation Scheme for Metric MIN-BISECTION, *Electronic Colloqium on Computational Complexity*, Report No. 41 (2002)