FAKULTÄT FÜR MATHEMATIK
UNIVERSITÄT KARLSRUHE

Optimisation of Clustering Algorithms for the Identification of Customer Profiles from Shopping Cart Data

Optimierung von Clusterverfahren zur Identifikation von Käuferprofilen aus Warenkorbdaten

Diplomarbeit von:
Stefanie Nagel
Matrikel Nummer: 1190189

Karlsruhe, October 17, 2008

Ausgeführt unter der Leitung von

Referentin/Betreuerin:
Prof. Dr. Dorothea Wagner
Fakultät für Informatik
Universität Karlsruhe

Koreferent:
Prof. Dr. Willy Dörfler
Fakultät für Mathematik
Universität Karlsruhe

betreuender Mitarbeiter:
Robert Görke
Fakultät für Informatik
Universität Karlsruhe

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Karlsruhe, February 25, 2009

Stefanie Nagel

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last few years the Internet has become more and more important, not only for business communication but also for many private households. Current news, information on science, research and education, event calendars, current service information (weather, traffic), entertainment, sports and cultural news are available for everyone. Emails are one of the most important communication channels, home banking has become a common and accepted service, news groups, chats, online communities, online games and partnership agencies established the important role of the Internet in private life. Financial and technical barriers of Internet accesses have been overcome. Flatrates and user friendly applications have made the Internet to a widely used instrument. A study of the ARD/ ZDF-Medienkommission has analysed this trend (cf. Table 1.1).

Development of the Internet usage in Germany 1997 to 2008 (cf. [15])												
	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008
in %	6.5	10.4	17.7	28.6	38.8	44.1	53.5	55.3	57.9	59.5	62.7	65.8
in Mio	4.1	6.6	11.2	18.3	24.8	28.3	34.4	35.7	37.5	38.6	40.8	42.7
increase in %	-	61.0	68.0	64.0	36.0	14.0	22.0	4.0	5.0	3.0	6.0	5.0

Table 1.1: Occasional usage of the Internet in Germany.

Online shopping as well has become an appreciated feature of the Internet. That is why *online shops*[1] are one of the major fields, which have to be adjusted to the new user's behaviour. In order to consult the customer in an online shop, EPOQ GmbH in Karlsruhe has developed a software called *EPOQ Recommendation Service*, shortly EPOQ RS, which is able to recommend products with the possibly biggest interest for every single customer. It is based on the real-time analysis of the customer's search- and sale-behaviour. Click streams, event data such as moving a product into the shopping cart or buy a product, product information or, if available, customer's personal data are the fields of consideration. The reaction of the customer to the recommendations is taken into account, EPOQ RS learns what the *right* recommendations are (reinforcement learning). Thus the service optimises itself.

Nevertheless it is desirable to exploit the given information about the customer's behaviour in a more efficient way. Therefore the number of products is reduced down to small product groups that are interesting for the current user. The restriction of the recommendations to these product groups

[1]An online shop is a virtual point of sale, it corresponds to a store in real life.

could improve the suitability of these proposals for the user. An approach to realise this is described in this diploma thesis.

Chapter 2

Motivation

Due to the wide availability of huge amounts of data and the need for turning such data into useful information and knowledge, the knowledge discovery process from data is very important nowadays. This knowledge discovery process consists of seven steps (cf. [21]):

1. **Data cleaning**
   Noise and inconsistent data is removed.

2. **Data integration**
   Data of multiple data sources can be combined.

3. **Data selection**
   The selected data depends on the analysis task.

4. **Data transformation**
   Data is transformed or consolidated into appropriate forms.

5. **Data mining**
   An essential process where intelligent methods are applied in order to extract data patterns.

6. **Pattern evaluation**
   The truly interesting patterns, which represent knowledge, are identified based on some interestingness measures.

7. **Knowledge presentation**
   Visualisation and knowledge representation techniques are used to present the results to the user.

The first four steps are preprocessing steps, where the data is prepared for the data mining step. In Figure 2.1 the primitives for specifying a data mining task are shown.

- The **task-relevant data** specifies the portions of the database, respectively the set of data in which the user is interested in.

- The **kind of knowledge** describes the data mining function to be performed. This includes characterisation, discrimination, association or correlation analysis, classification, prediction, clustering, outlier analysis or evolution analysis.

- The **background knowledge** about the domain, which is analysed, is useful to guide the knowledge discovery process and to evaluate the patterns found. *Concept hierarchies* are one form of background knowledge, which allow the mining of data at multiple levels of abstraction. A concept hierarchy for the attribute age could be that all are devided into youth $(20, \ldots, 39)$, middleaged $(40, \ldots, 59)$ and senior $(60, \ldots, 89)$. Here the most general abstraction level is denoted by all. The user's beliefs about the relationships in the data are another form of background knowledge.

- The **interestingness measures and thresholds** for pattern evaluation support the guiding of the mining process and the evaluation of the discovered patterns after the mining process. Different kinds of knowledge have different interestingness measures. This measure for association rules include *support* and *confidence* and rules below these thresholds are considered as uninteresting.

- The expected **representation for visualising** the discovered patterns describe the form of the display. This includes rules, tables, charts, graphs, decision trees and cubes.

A number of different data repositories can serve as data sources, such as *relational databases*, *data warehouses*, *transactional databases*, *advanced databases systems*, *flat files*, *data streams* and the *World Wide Web*. The data repository, which is the data source of this diploma thesis, is a transactional database. That means every record represents a transaction. A transaction typically includes a unique transaction identity number (TID) and a list of the items making up the transaction. An example is shown in Table 2.1.

TID	list of item IDs
100	f, a, c, d, g, i, m, p
200	a, b, c, f, l, m, o
300	b, f, h, j, o
400	b, c, k, s, p
500	a, f, c, e, l, p, m, n

Table 2.1: Example of a transactional database: every transaction identity number (TID) is listed with the corresponding item identities (IDs).

2.1 Finding Frequent Itemsets with FP-Growth

Frequent itemset mining allows for the discovery of interesting correlation relationships among huge amounts of transaction records. One typical example is the market basket analysis. In the following the *FP-Growth* (frequent-pattern growth) approach is described (cf. [22]). The knowledge type to be mined with the help of the detected frequent itemsets of FP-Growth is association and correlation analysis. The interestingness measures are certainty (confidence) and utility (support). Additionally a correlation measure is used in order to filter out uninteresting association rules. In this way the interestingness measures are augmented and correlation rules are created. Thus the form in which the results are to be displayed are rules (cf. Figure 2.1). A frequent itemset is a set of items, which occurs at least as frequently as a predetermined minimum support ξ. The FP-Growth approach can be described with the help of Algorithm 1, 2 and 3.

The FP-tree is constructed according to Algorithm 1 and is based on a transactional database DB (line 1). The supportcount of each item is defined as the number of occurences of this item

Task-relevant data
Database or data warehouse name
Database tables or data warehouse cubes
Conditions for data selection
Relevant attributes or dimensions
Data grouping citeria

Knowledge type to be mined
Characterisation
Discrimination
Association/ correlation
Classification/ prediction
Clustering

Background knowledge
Concept hierarchies
User beliefs about relationships in the data

Pattern interestingness measures
Simplicity
Certainty (e. g. confidence)
Utility (e. g. support)
Novelty

Visualisation of discovered patterns
Rules, tables, reports, charts, graphs,
decision trees and cubes
Drill-down and roll-up

Figure 2.1: Primitives for specifying a data mining task (based on [21]).

in DB (line 2, 3). In line 4 F is restricted to the items with a support greater than the minimum support threshold ξ. The choice of ξ is more or less arbitrary. Afterwards F is sorted according to the descending order of the supportcounts (line 5) and this is stored as the $FList$. After creating the root of the FP-tree (line 6), each transaction in DB is regarded (line 7 - line 10): after sorting the transaction (line 8), the sorted frequent-item list of the transaction is inserted into the FP-tree (line 10).

Algorithm 1: FP-TREE CONSTRUCTION

```
    Input: A transaction database $DB$ and a minimum support threshold $\xi$.
    Output: FP-tree, the frequent-pattern tree of $DB$.
 1  $F \leftarrow$ set of all items;
 2  foreach item $p \in F$ do
 3      supportcount($p$) = number of occurrences of $p$ in DB;
 4  $F \leftarrow \{p \in F | \text{supportcount}(p) \geq \xi\}$;
 5  $FList \leftarrow$ Sort $F$ by supportcount in descending order;
 6  Create the root of an FP-tree, $T$, labelled as "null".
 7  foreach transaction Trans $\in DB$ do
 8      Sort Trans $\cap F$ according to the order of the $FList$.
 9      The sorted frequent-item list of Trans is denoted by $[p|P]$, $p$ is the first element, and $P$ the
        rest of the list.
10      insertTree($[p|P], T$);
```

This insertion of a frequent-item list $[p|P]$ into the FP-tree is described in Algorithm 2. There are two cases: if T has already a child named p (line 1, 2), then the count of this vertex has to be increased by 1. Otherwise a new vertex is created and the count is set to 1 (line 3 - 5). The parent link directs to the parent vertex T. Vertices with the same item-name are linked in sequence via vertex links. Following the vertex link structure of a frequent item p permits the detection of all possible patterns containing only frequent items and p (cf. [22], Property 3.1). Thus this structure has to be updated after creating a new vertex (line 5). This is repeated until the rest of the frequent-item list P is empty (line 6, 7).

Algorithm 2: INSERTTREE($[p|P], T$)

```
    Input: A sorted frequent-item list of a transaction and the vertex $T$.
    Output: The FP-tree, in which the frequent-item list $[p|P]$ is inserted.
 1  if $T$ has a child $N$ and $N$.item − name = $p$.item − name then
 2      Increment $N$'s count by 1.
 3  else
 4      Create a new vertex $N$ with an initialised count of 1.
 5      Its parent link links to $T$ and the vertex link structure is updated.
 6  if $P \neq \emptyset$ then
 7      insertTree($P, N$);
```

In order to understand Algorithm 3, the pattern generation out of an FP-tree, which only consists of a single path S, has to be explained (cf. [22], Lemma 3.2). The frequent patterns of such an FP-

Algorithm 3: FP-Growth

Input: A database DB, represented by an FP-tree, which was constructed according to Algorithm 1, and a minimum support threshold ξ.

Output: The complete set of frequent patterns.

```
1  Call FP-Growth(FP-tree, $null$).
   // the procedure is called for the entire FP-tree
2  FP-Growth($Tree, \alpha$){
3  if $Tree$ contains a single prefix path then
4      Let $P$ be the single prefix-path part of $Tree$.
5      Let $Q$ be the multipath part with the top branching vertex replaced by a $null$ root.
6      foreach subset $\beta$ of vertices in the path $P$ do
7          Generate pattern $\beta \cup \alpha$ with support = minimum support of vertices in $\beta$.
8      Let $freqPatternSet(P)$ be the set of patterns so generated.
9  else
10     Let $Q$ be $Tree$.
11 foreach item $p \in Q$ do
12     Generate pattern $\beta = p \cup \alpha$ with support = $p$.support.
13     Construct $\beta$'s conditional pattern-base and then $\beta$'s conditional FP-tree $Tree_\beta$.
14     if $Tree_\beta \neq \emptyset$ then
15         Call FP-Growth($Tree_\beta, \beta$).
16     Let $freqPatternSet(Q)$ be the set of patterns so generated.
17 return
   $(freqPatternSet(P) \cup freqPatternSet(Q) \cup (freqPatternSet(P) \times freqPatternSet(Q)))$
18 }
```

tree are all combinations of the subpaths of S. The support is given by the minimum support of the items in this subpath. That is the reason why Algorithm 3 is divided into two parts: line 3 - line 8 and line 11 - line 16. The idea is to subdivide the tree into a single prefix-path part P and a multipath part Q. An example is shown in Figure 2.2.

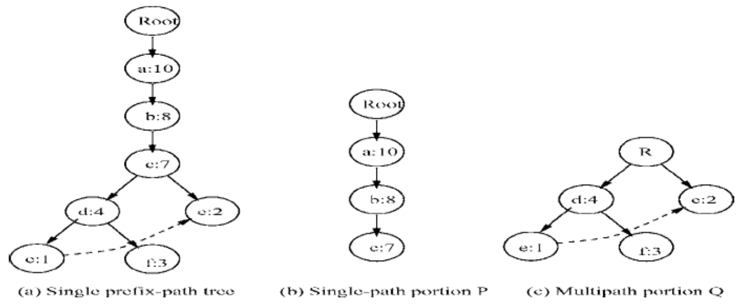


Figure 2.2: Mining an FP-tree with a single prefix path (taken from [22]). The dotted lines are the vertex links.

In line 1 of Algorithm 3 the procedure `FP-Growth` is called for the entire FP-tree and an empty set of frequent patterns. The two parts P and Q are detected in line 4 and line 5. The multipath part Q can be viewed as an independent FP-tree. The frequent patterns of the single prefix path P are generated according to Lemma 3.2 in [22] as mentioned above (line 6 - line 8). If there does not exist a single prefix path P, Q is regarded as the whole tree (line 10). Due to the prefix path property (cf. [22], Property 3.2) the frequent patterns in Q with suffix p can be obtained by accumulating only the prefix subpaths of p. The count of every vertex in the prefix path should be the same as the count of p in this path (line 12). In line 13 the conditional pattern-base and the conditional FP-tree are generated. The procedure is called recursively until the conditional FP-tree is empty (line 15). The set of all frequent patterns is given by the cross-product of the frequent patterns generated from P and Q (denoted by $freqPatternSet(P) \times freqPatternSet(Q)$) conjoint with the frequent patterns from P and from Q (line 17). The cross-product means that each frequent item is the union of one frequent itemset from P and one from Q and the support is the minimum support of these two.

The FP-Growth approach is visualised with the example in Table 2.1. The ordered frequent item list is $FList = \{f, c, a, b, m, p, l, o, d, e, g, h, i, j, k, n, s\}$. The minimum support threshold $\xi = 3$ is chosen. The frequent items of the transactions are sorted according to this $FList$ (cf. Table 2.2). The FP-tree is generated as described in Algorithm 1 and the result is visualised in Figure 2.3.

TID	list of item IDs
100	f, c, a, m, p
200	f, c, a, b, m
300	f, b
400	c, b, p
500	f, c, a, m, p

Table 2.2: The ordered frequent items of the example in Table 2.1.

On the basis of this FP-tree the frequent patterns are mined. The frequent patterns of vertex p are considered as an example. The vertex link of vertex p is denoted by the dotted line. All patterns, which only contain frequent items and include vertex p, can be obtained by following this vertex link. The immediate frequent pattern of vertex p is (p:3) and it has two different paths: $\langle f{:}4, c{:}3, a{:}3, m{:}2, p{:}2\rangle$ and $\langle c{:}1, b{:}1, p{:}1\rangle$. These paths imply that the string "(f, c, a, m, p)" appears

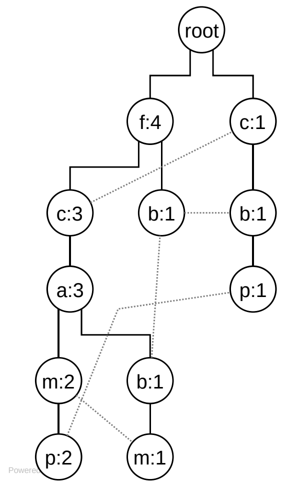


Figure 2.3: The FP-tree of the example in Table 2.1.

twice, whereas the string "(c, b, p)" appears only once. The prefix paths of p, $\{(fcam{:}2), (cb{:}1)\}$, form p's conditional pattern-base, the subpattern-base under the condition, that p exists. Afterwards an FP-tree is constructed on this conditional pattern-base (the conditional FP-tree). In the case of p this conditional FP-tree has only one branch (c:3), which is shown in Figure 2.4.

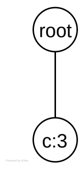


Figure 2.4: The conditional FP-tree of vertex p in the example of Table 2.1.

Thus only one frequent pattern (cp:3) is derived (due to the prefix path property, Property 3.2 in [22]). No further frequent patterns associated with p can be obtained. This procedure is made for every vertex, beginning with the item, which has the least support (item p) and ending with the item, which has the most support (item f). Vertices, which have already been considered, do not have to be reconsidered in further steps (that means during the analysis of vertices with a higher support). In Table 2.3 the results for the example given in Table 2.1 are shown.

item	conditional pattern-base	conditional FP-tree	frequent itemsets involving the current item
p	$\{(fcam : 2), (cb : 1)\}$	$\{(c : 3)\}\|p$	$\{(cp{:}3)\}$
m	$\{(fca : 2), (fcab : 1)\}$	$\{(f : 3, c : 3, a : 3)\}\|m$	$\{(m{:}3),\ (am{:}3),\ (cm{:}3),\ (fm{:}3),\ (cam{:}3),\ (fam{:}3),\ (fcam{:}3),\ (fcm{:}3)\}$
b	$\{(fca : 1), (f : 1), (c : 1)\}$	$\emptyset$	$\emptyset$
a	$\{(fc : 3)\}$	$\{(f : 3, c : 3)\}\|a$	$\{(fa{:}3),\ (ca{:}3),\ (fca{:}3)\}$
c	$\{(f : 3)\}$	$\{(f : 3)\}\|c$	$\{(fc{:}3)\}$
f	$\emptyset$	$\emptyset$	$\emptyset$

Table 2.3: The result of FP-Growth for the example in Table 2.1.

2.2 Generating Association Rules

Once frequent itemsets are obtained from transactions in a database, strong *association rules* (these satisfy both minimum support and minimum confidence) can be generated straightforward. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be the set of all possible items. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset \mathcal{I}$, $B \subset \mathcal{I}$ and $A \cap B = \emptyset$. The support $\sup(A \Rightarrow B)$ is the percentage of transactions that contains **both** A and B ($A \cup B$[1]), while the confidence *conf* is the percentage of transactions containing A, that also contains B. The equations are the following (cf. [21])

$$\begin{aligned} \sup(A \Rightarrow B) &= P(A \cup B) \\ &= \frac{\text{supportcount}(A \cup B)}{\text{total number of transactions}} \end{aligned} \tag{2.1}$$

$$\begin{aligned} \text{conf}(A \Rightarrow B) &= P(B|A) \\ &= \frac{\text{supportcount}(A \cup B)}{\text{supportcount}(A)} \quad . \end{aligned} \tag{2.2}$$

The supportcount denotes the absolute number of transactions containing the according set. Association rules can be generated out of frequent itemsets in the following way:

- For each frequent itemset l, all nonempty subsets of l are generated. These subsets are frequent itemsets themselves.
- For every nonempty subset r of l the rule $r \Rightarrow (l - r)$ is generated if $\text{conf}(r \Rightarrow (l - r)) = \frac{\text{supportcount}(r \cup (l-r))}{\text{supportcount}(r)} = \frac{\text{supportcount}(l)}{\text{supportcount}(r)} \geq \min_{\text{conf}}$. The minimum confidence threshold is $\min_{\text{conf}}$.

Since the rules are generated from frequent itemsets, each automatically satisfies minimum support.

2.3 Improving Association Rules

Association rules can be improved by expanding them to correlation rules. A *correlation rule* has the following form (cf. [21])

$$A \Rightarrow B[\text{support, confidence, correlation}] \quad . \tag{2.3}$$

In addition to support and confidence a correlation rule also takes the correlation between itemsets A and B into account. There are many different correlation measures. One of these is the *lift*. The itemsets A and B are independent, if $P(A \cup B) = P(A) \cdot P(B)$, otherwise the itemsets are *dependent* and *correlated*. This can also be extended to more than two itemsets. The lift between the occurrence of A and B can be calculated in the following form

$$\text{lift}(A, B) = \frac{P(A \cup B)}{P(A) \cdot P(B)} \quad . \tag{2.4}$$

The occurrence of A is *negatively correlated with* the occurrence of B if $\text{lift}(A, B) < 1$. The two itemsets are *positively correlated with* each other if $\text{lift}(A, B) > 1$, that means the occurrence of one itemset influences the occurrence of the other in a positive way. If $\text{lift}(A, B) = 1$ the two itemsets are

[1]The set $A \cup B$ contains every item in A and every item in B. It is not "A or B".

independent and thus *uncorrelated*. The lift in general describes the degree to which the occurrence of one itemset "lifts" the occurrence of another itemset.

Another correlation measure is the *cosine* measure. It is a kind of harmonised lift measure and has the the null-invariance property. The two formulas are very similar, except that in the formula of the cosine measure the square root on the product of the probabilities of A and B is taken. Thus the total number of transactions does not influence the cosine measure, it is only affected by the supports of A, B and $A \cup B$. Let T denote the total number of transactions. The formula of cosine is given by

$$\begin{aligned}
\text{cosine}(A,B) &= \frac{P(A \cup B)}{\sqrt{P(A) \cdot P(B)}} && (2.5)\\
&= \frac{\frac{\text{supportcount}(A \cup B)}{T}}{\sqrt{\frac{\text{supportcount}(A)}{T} \cdot \frac{\text{supportcount}(B)}{T}}} && (2.6)\\
&= \frac{\text{supportcount}(A \cup B)}{\sqrt{\text{supportcount}(A) \cdot \text{supportcount}(B)}} \; . && (2.7)
\end{aligned}$$

In case of large data sets, the null-invariance property is an enormous advantage, because large data sets typically have many null-transactions (i. e. transactions that do not contain any of the currently regarded itemsets A and B). Null-invariant measures are not influenced by null-transactions. However it is wise to use additional measures such as the lift measure, if the results are not conclusive.

An example is given in Table 2.4. Let the data sets D_0, D_1 and D_2 represent the transactions with respect to the purchase of the itemsets A and B. The union $A \cup B$ means that both frequent itemsets A and B are bought together, while the union $\overline{A} \cup B$ means that only B is bought, $A \cup \overline{B}$ means that only A is bought and $\overline{A} \cup \overline{B}$ means that neither A nor B are bought.

Data Set	$A \cup B$	$\overline{A} \cup B$	$A \cup \overline{B}$	$\overline{A} \cup \overline{B}$	cosine(A,B)	lift(A,B)
D_0	4 000	3 500	2 000	0	0.6	0.84
D_1	4 000	3 500	2 000	500	0.6	0.89
D_2	4 000	3 500	2 000	10 000	0.6	1.73

Table 2.4: Comparison of two correlation measures.

2.4 Possibilities for Improving This Procedure

An interesting advancement of the FP-Growth approach in Chapter 2.1 is an improvement in order to avoid the arbitrary choice of the minimum support threshold ξ. In a large graph this choice is a hard challenge. If ξ is chosen relatively large, the number of frequent itemsets is small and consequently only a few association rules can be generated. These rules solely contain items, which occur often. Most of these rules are *obvious*. That means they concern items, which are already well known and studied because of their important role in the product portfolio. If ξ is chosen to be relatively small, the number of association rules is very large. These rules can be reduced with the help of a minimum confidence threshold. But if ξ is too small, it could happen that many of the rules are not *relevant*, at least for a certain item. The choice of ξ is independent of the item, for which a rule should be generated. Considering an item with a very small support possibly leads to no association rule, since this item is not part of any frequent itemset with minimum support ξ.

A possibility to improve the choice of ξ is the dependence of ξ on the item i, which is considered. This assures the appearance of association rules, which include i. If furthermore an *environment* of i can be detected, the rules can be reduced to the items of this environment. The items of the environment of i are more linked to i than to the rest of the items.

In this diploma thesis several clustering algorithms are introduced. With the help of these, environments of items can be detected. This is a preprocessing step of the FP-Growth approach and could enrich the background knowledge. Either the minimum support ξ is set to the minimum support of the items in the detected environment of i

$$\xi = \min_{j \in \text{environment}(i)} (\text{supportcount}(j))$$

or ξ is set dependent on the maximal support in the environment of i, for example

$$\xi = \max\{0.4 \cdot \max_{j \in \text{environment}(i)} (\text{supportcount}(j)), \text{supportcount}(i)\} \ .$$

In both cases ξ is at least as large as supportcount(i).

Chapter 3

Graph Modelling

In order to detect customer profiles with the help of clustering algorithms, the most important issue at the beginning is creating a model of the data in a proper manner. Usually there is much information within such data and the challenge is to create a graph which closely models the relevant facts.

In the first step this graph $G = (V, E, w)$ is initialised as an undirected weighted graph. The vertices V represent the products and an edge $e \in E$ between two products represents the fact that two products have been bought together. The edge weights w shall model the probability that the two linked products have been bought together in the past, and thus the expected probability of a combined future purchase of them. The choice of an undirected graph in contrast to a directed graph was made because of the symmetry of this relationship.

Let R be a binary relation over $V \times V$ which stands for "are bought together". Then the following holds

$$\forall k, l \in V \ : (k, l) \in R \Rightarrow (l, k) \in R \ . \tag{3.1}$$

Because of the relation "are bought together", the input data is reduced to the subset of products which have been bought together with at least one other product. Hence products which have been bought exclusively (not in combination with any other product) are filtered out in advance.

3.1 Building the Weights

In a first step edge weights are built as conditional probabilities. Let K be the event that product k is bought and L the event that product l is bought. The weight w_{kl} of the edge which links the nodes k and l can be expressed as follows:

$$w_{kl} = p(K \cap L | K \cup L) \ . \tag{3.2}$$

The intersection $K \cap L$ in this context means that the two products are bought by one customer while the union $K \cup L$ means that at least one of the products is bought by a customer. Note that this usage of "$\cap$" and "$\cup$" is essentially contrary from Chapter 2.2 and Chapter 2.3.

TID	list of products
100	f, a, c, d, g, i, m, p
200	a, b, c, f, l, m, o
300	b, f, h, j, o

Table 3.1: Example for building the weights of G.

An example is given with the help of the transactional database in Table 3.1. The conditional probability of the combined purchase of product a and product f, given one of these products is bought by a customer, is $w_{af} = \frac{2}{2+3-2} = \frac{2}{3}$. The probability $w_{ac} = \frac{2}{2+2-2} = 1$ signifies that product a and product c have always been bought together.
In order to formalise this, some definitions are made. Let $P = \{1, \dots, N\}$ be the set of all products and $C = \{1, \dots, M\}$ the set of all customers. The event B_{kc} expresses the following:

$$B_{kc} = \{k \in P, c \in C : \textit{ product } k \textit{ was bought by customer } c\} \ . \tag{3.3}$$

The indicator function $1_{B_{kc}}$ can be interpreted as follows

$$1_{B_{kc}} = \begin{cases} 1 & \text{if product k was bought by customer c} \\ 0 & \text{else.} \end{cases} \tag{3.4}$$

In the same way the events C_{kc}, V_{kc} and CR_{kc} are defined:
$C_{kc} = \{k \in P, c \in C : \textit{ product } k \textit{ has been put in the cart by customer } c\}$,
$V_{kc} = \{k \in P, c \in C : \textit{ product } k \textit{ has been viewed by customer } c\}$,
$CR_{kc} = \{k \in P, c \in C : \textit{ product } k \textit{ has been removed from the cart by customer } c\}$.

The indicator functions $1_{C_{kc}}$, $1_{V_{kc}}$ and $1_{CR_{kc}}$ indicate whether the event takes place or not (cf. $1_{B_{kc}}$ in equation (3.4)). With these definitions the weight w_{kl} in equation (3.2) can be written as

$$\begin{aligned} w_{kl} = p(K \cap L | K \cup L) &= \frac{p((K \cap L) \cap (K \cup L))}{p(K \cup L)} \\ &= \frac{p(K \cap L)}{p(K \cup L)} \\ &= \frac{\sum_{c \in C} (1_{B_{kc}} \cdot 1_{B_{lc}})}{\sum_{c \in C} (1_{B_{kc}} + 1_{B_{lc}} - 1_{B_{kc}} \cdot 1_{B_{lc}})} \ . \end{aligned} \tag{3.5}$$

In addition to this the events C_{kc}, V_{kc}, CR_{kc} and their indicator functions could be integrated into the graph modelling process. This is a second step and against the background of selling products the vertices would not change (i.e. only products which are bought at least once with another product would be considered). The graph, which is generated on the basis of the buy events, is called G_{buy}.

3.2 Community

Let $G = (V, E)$ be a graph. The adjacency matrix of G is $A_{i,j}$. The graph $G' = (V', E' = E\big|_{V'})$ is a subgraph of G. The *degree* $\deg(i)$ of a vertex i can be expressed as $\deg(i) = \sum_j A_{i,j}$. The number of edges which connect vertex i to vertices within the subgraph G' is represented by $\deg^{\text{in}}_{G'}(i) = \sum_{j \in G'} A_{i,j}$ and the number of edges which connect vertex i to vertices which are not in G' is represented by $\deg^{\text{out}}_{G'}(i) = \sum_{j \notin G'} A_{i,j}$. If vertex i belongs to the subgraph G' the equation $\deg_{G'}(i) = \deg^{\text{in}}_{G'}(i) + \deg^{\text{out}}_{G'}(i)$ holds.

Unfortunately there exists no standard definition of a *community* (i. e. *cluster*). Nevertheless a first impression what a community of a vertex *should be*, is given.

General definition In [32] the trial is made to give such a definition (based on [27]). A community is a subgraph of vertices which are more densely connected to each other than to the rest of the graph. But this is a very general and imprecise definition. It is not clear *how dense* a subgraph must be to be defined as a community. But this definition gives a first impression what a community should be.

Communities in a strong and a weak sense Another possible definition is given in [31]. In contrast to the general definition above, this paper formalises the density of a community. The subgraph G' is a community in a strong sense if

$$\deg_{G'}^{\text{in}}(i) > \deg_{G'}^{\text{out}}(i),\ \forall i \in G' \ . \tag{3.6}$$

This equation (3.6) says that each vertex of a community in a strong sense has more connections (number of edges) within the community than outside the community.
The subgraph G' is a community in a weak sense if

$$\sum_{i \in G'} \deg_{G'}^{\text{in}}(i) > \sum_{i \in G'} \deg_{G'}^{\text{out}}(i) \ . \tag{3.7}$$

Equation (3.7) requires less than equation (3.6). The sum of all vertex degrees within the community has to be larger than the sum of all vertex degrees towards the rest of the graph. That means that vertex i with a high $\deg_{G'}^{\text{in}}(i)$ can balance a small $\deg_{G'}^{\text{in}}(j)$ of vertex j. This is not possible in equation (3.6). Every community in a strong sense is a community in a weak sense as well. The converse is not true.

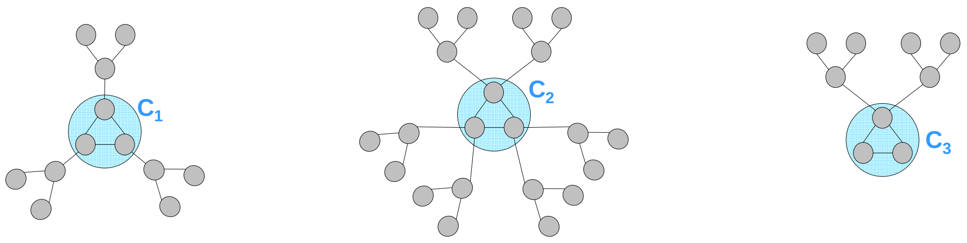


Figure 3.1: C_1 and C_3 are weak communities, C_2 is not a weak community.

The definition of a community in a weak sense is *weaker* than it should be. Figure 3.1 confirms this. While C_1 is a community in a weak sense, C_2 is not. This different classification is not intuitive. The internal edges are counted twice in equation (3.7). A more convincing and restrictive condition would remedy this as follows:

$$\frac{1}{2} \sum_{i \in G'} \deg_{G'}^{\text{in}}(i) > \sum_{i \in G'} \deg_{G'}^{\text{out}}(i) \ . \tag{3.8}$$

With this definition neither C_1 nor C_2, but C_3 in Figure 3.1 are communities in a weak sense.

Web Community The definition of a community depends a lot on the context in which it is used. In [17] a *web community* is defined as a collection of web pages, in which each web page has more hyperlinks to web pages within the collection than to the rest of the network. This shows that the general definition above can be applied to several areas.

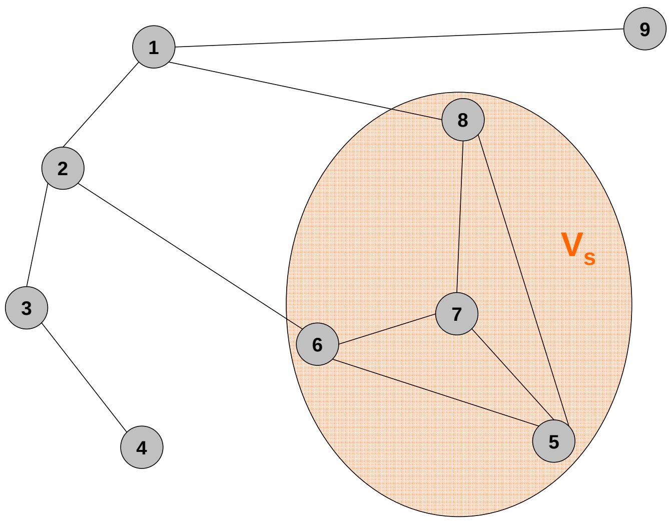


Figure 3.2: Example of an LS set.

LS-set Let the inclusion $L \subset V_s \subset V$ hold. The set V_s is an LS-set if each proper subset L has more edges to $V_s \backslash L$ than to vertices outside V_s (that means $V \backslash V_s$) (cf. [23]). In the example given in Figure 3.2 the set of vertices is $V = \{1, 2, \dots, 9\}$ and we choose $V_s = \{5, 6, 7, 8\}$. For all possible subsets L of V_s ($L = \emptyset, V_s$ is not allowed) the number of edges to vertices inside V_s is larger than the number of edges to vertices outside V_s as can be seen in Table 3.2. Thus V_s is an LS-set.

id	edges to vertices inside V_s	edges to vertices outside V_s
$\{5\}$	3	0
$\{6\}$	2	1
$\{7\}$	3	0
$\{8\}$	2	1
$\{5,6\}$	3	1
$\{5,7\}$	4	0
$\{5,8\}$	3	1
$\{6,7\}$	3	1
$\{6,8\}$	4	2
$\{7,8\}$	3	1
$\{5,6,7\}$	2	1
$\{5,6,8\}$	3	2
$\{5,7,8\}$	2	1
$\{6,7,8\}$	3	2

Table 3.2: All subsets of V_s fulfill the necessary condition of an LS-set.

The definition of an "LS-set" requires even more than the definition of a "community in a strong sense". To show the difference the example of Figure 3.2 is reconsidered. The condition required by a community in a strong sense $\deg^{\text{in}}_{V_s}(i) > \deg^{\text{out}}_{V_s}(i),\ \forall i \in V_s$, is satisfied by an LS-set. While in an LS-set *every* subset must have more edges to vertices inside the set than to vertices outside, in a community in a strong sense only the subsets which consist of one vertex must fulfill this condition (that means: LS-set $\Rightarrow$ community in a strong sense). In Figure 3.2 V_s is an LS-set and a community in a strong sense as well, whereas the set $\{1,2,3,4,9\}$ is only a community in a strong sense, since every vertex in this set has more edges to vertices in this set than to vertices outside this set.

3.3 The Number of Clusterings

In general many clustering techniques are computationally intractable. The number of possible divisions of n vertices into g communities is given by the Stirling number of the second kind $S_n^{(g)}$, which has the following form

$$S_n^{(g)} = \sum_{r=1}^{g} (-1)^{g-r} \frac{r^n}{r!(g-r)!} \ . \tag{3.9}$$

The sum $\sum_{g=1}^{n} S_n^{(g)}$ describes the total number of possibilities to devide n vertices into communities (whereas the size of the communities can range between 1 and n and overlapping communities are not considered). Because of

$$S_n^{(1)} + S_n^{(2)} = 1 + 2^{n-1} - 1 = 2^{n-1} \quad \forall n > 1 \ ,$$

this sum increases at least exponentially in n. Therefore it is not feasible to search for an optimal clustering over all possible divisions of the graph in any naive fashion.

Chapter 4

Quality Indices

In order to measure the quality of clusterings several quality functions are introduced in this chapter. These functions express to what extent a clustering realises the paradigm of intra-cluster density versus inter-cluster sparsity.

Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a clustering of the graph $G = (V, E, w)$, where C_i, $i \in \{1, \ldots, k\}$, are non-empty subsets of V. The set of all possible clusterings is $\mathcal{A}(G)$. Most of the quality indices can be written in the following form (cf. [18])

$$\operatorname{index}(\mathcal{C}) := \frac{f(\mathcal{C}) + g(\mathcal{C})}{\max\{f(\mathcal{C}') + g(\mathcal{C}') : \mathcal{C}' \in \mathcal{A}(G)\}} \quad . \tag{4.1}$$

The two functions $f, g : \mathcal{A}(G) \to \mathbb{R}_0^+$ quantify the density inside the clusters (f) and the sparsity between the clusters (g). This index is made under the assumption that there exists a clustering so that the denominator is not 0.

4.1 Coverage

The quality index named *coverage* describes to what extent the edges of the graph G are *intra-cluster edges*. The endpoints of an intra-cluster edge are assigned to the same cluster. The set of all intra-cluster edges is denoted by $E(\mathcal{C})$. The fraction of the number of intra-cluster edges $|E(\mathcal{C})|$ and the total number of edges $|E|$ gives the coverage

$$\operatorname{cov}(\mathcal{C}) = \frac{|E(\mathcal{C})|}{|E|} \quad . \tag{4.2}$$

Thus the function, which quantifies the density inside the clusters is $f(\mathcal{C}) = |E(\mathcal{C})|$ and the function, which quantifies the sparsity between the clusters is $g = 0$. The range of coverage is the unit interval and as described in [13] this index will be large if the ratio of the intra-cluster edges is high. This quality function can serve as a comparison instrument. If the input graph remains the same and several clustering algorithms are applied, this index indicates which algorithm detects the community structure in the best way. But in the trivial case of a community which includes the whole graph this index is 1. Thus it is no stand-alone solution.

Let m be the sum of all weights($m = \sum_{\{v,w\} \in E} w(\{v, w\})$). If a weighted graph is considered, the formula of this quality index has the following form

$$\operatorname{cov}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\} \in E(\mathcal{C})} w(\{v, w\})}{m} \quad . \tag{4.3}$$

4.2 Performance

The quality function *performance* also involves missing *inter-cluster edges*. The endpoints of inter-cluster edges are assigned to different clusters. The number of possible undirected edges in the graph $G = (V, E)$ is given by $|V||V-1|\frac{1}{2}$. The sum of the number of intra-cluster edges $|E(\mathcal{C})|$ and the number of missing inter-cluster edges $m_{\text{missing inter}}(\mathcal{C}) = \frac{1}{2}\sum_{\{v,w\}\in V\times V}[\{v,w\} \notin E]\cdot[v \in C_i, w \in C_j, i \neq j]$ is divided by the number of possible undirected edges

$$\text{perf}(\mathcal{C}) = \frac{|E(\mathcal{C})| + m_{\text{missing inter}}(\mathcal{C})}{|V||V-1|\frac{1}{2}} \quad . \tag{4.4}$$

The notation $\{v,w\} \in V \times V$ is equivalent to $v, w \in V$ and implies that every undirected edge is mentioned twice. The functions f and g in equation (4.1) are given by $f = |E(\mathcal{C})|$ and $g = m_{\text{missing inter}}(\mathcal{C})$ in case of performance.

The difficulty of the weighted case in the context of performance is the weight of a missing inter-cluster edge. The average weight of all edge weights $\bar{w}$ can be taken

$$\bar{w} = \frac{\sum_{\{v,w\}\in E} w(\{v,w\})}{|E|} \quad . \tag{4.5}$$

The sum of all weights is m. The resulting performance formula looks like this

$$\text{perf}_{\text{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E(\mathcal{C})} w(\{v,w\}) + m_{\text{missing inter}} \cdot \bar{w}}{m + (|V||V-1|\frac{1}{2} - |E|)\cdot \bar{w}} \quad . \tag{4.6}$$

Another possibility is to judge a missing inter-cluster edge with the maximum occurring weight $w_{\max}$, while an existing inter-cluster edge $\{t,u\}$ could be judged with $w_{\max} - w(\{t,u\})$. This takes into account that the inter-cluster edge $\{t,u\}$ is a *missing* inter-cluster edge to *some* degree.

4.3 Modularity

In case of *modularity* the formula cannot be presented as in (4.1). The idea behind this quality index is another. A combination of vertices is a community if the number of edges which links these vertices to each other is greater than the number of edges which would be expected between these vertices when they were set randomly.

$$\begin{aligned} \text{mod}(\mathcal{C}) &= \text{cov}(\mathcal{C}) - E(\text{cov}(\mathcal{C})) \\ &= \frac{|E(\mathcal{C})|}{|E|} - \frac{1}{4|E|^2}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\deg(v)\right)^2 \end{aligned} \tag{4.7}$$

The optimisation of this index supports finding communities in large networks (cf. [32]) because of the ability of this index to measure community structure in quite a good manner. The experience shows that a value above about 0.3 is a reference to significant community structure in a network (cf. [13]).

In the weighted case the formula of modularity can be modified. Let s_v be the *strength* of vertex v, m the sum of all weights and let $e \sim v$ be the notation for "edge e is incident to vertex v". The strength s_v of vertex v can be written as $s_v = \sum_{e\sim v} w(e)$. The modularity $\text{mod}_{\text{w}}(\mathcal{C})$ can now be expressed as

$$\mathrm{mod_w}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E(\mathcal{C})} w(\{v,w\})}{m} - \sum_{C_k\in\mathcal{C}} \left(\frac{\sum_{v\in C_k} s_v}{2m} \right)^2 \quad . \tag{4.8}$$

Detecting overlapping communities requires the extension of the definitions of the quality indices for the non-overlapping case given above. These extensions should be generalisations of the indices for the non-overlapping case.

4.4 Extending Coverage

Let $\mathcal{C}(v)$ be the set of communities to which v belongs. Then the indicator function

$$1_{\mathrm{intra}}(\{v,w\}) = 1_{\{\sum_{C_k\in\mathcal{C}}\left(1_{\{C_k\in\mathcal{C}(v)\}}1_{\{C_k\in\mathcal{C}(w)\}}\right)\geq 1\}}$$

describes whether the edge is covered completely by at least one community or not. A new definition of intra-cluster edge is possible. If the indicator function of an edge is 1 it is an intra-cluster edge. The extension of coverage, if overlapping communities are allowed, has the following form

$$\mathrm{cov}^{\mathrm{ov}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\})}{|E|} \quad . \tag{4.9}$$

This *overlapping* definition can be modified for the weighted case as well (m is again the weight of all edges)

$$\mathrm{cov}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} \left(1_{\mathrm{intra}}(\{v,w\}) \cdot w(\{v,w\})\right)}{m} \quad . \tag{4.10}$$

4.5 Extending Performance

The definition of intra-cluster edges is made as in the extension of coverage above. The definition of inter-cluster edges can be made in a similar way

$$1_{\mathrm{inter}}(\{v,w\}) = 1_{\{\sum_{C_k\in\mathcal{C}}\left(1_{\{C_k\in\mathcal{C}(v)\}}1_{\{C_k\in\mathcal{C}(w)\}}\right)=0\}} \quad .$$

An edge $\{v,w\}\in E$ is an inter-cluster edge if it is not covered by a community ($\mathcal{C}(v)\cap\mathcal{C}(w)=\emptyset$), that means the indicator function $1_{\mathrm{inter}}(\{v,w\})$ of the edge is 1.

The number $m_{\text{missing inter}}$ of missing inter-cluster edges can be derived by the following consideration. The number of possible inter-cluster edges $m_{\text{inter max}}$ is

$$m_{\text{inter max}} = \frac{1}{2} \sum_{\{v,w\}\in V\times V} 1_{\mathrm{inter}}(\{v,w\})$$

and the number of (undirected) inter-cluster edges m_{inter} is

$$m_{\mathrm{inter}} = \sum_{\{v,w\}\in E} 1_{\mathrm{inter}}(\{v,w\}) \quad .$$

The extension of performance, if overlapping communities are allowed, has the following form

$$\begin{aligned}
\operatorname{perf}^{\mathrm{ov}}(\mathcal{C}) &= \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{missing inter}}}{|V||V-1|\frac{1}{2}} \\
&= \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{inter max}} - m_{\text{inter}}}{|V||V-1|\frac{1}{2}} \\
&= \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + \frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\mathrm{inter}}(\{v,w\}) - \sum_{\{v,w\}\in E} 1_{\mathrm{inter}}(\{v,w\})}{|V||V-1|\frac{1}{2}}
\end{aligned} \tag{4.11}$$

A formula for the weighted and overlapping case is derived in the same way as in the definition of $\operatorname{perf}_{\mathrm{w}}$ (the non-overlapping but weighted case). Let m again be the sum of all weights.

$$\begin{aligned}
\operatorname{perf}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C}) &= \frac{\sum_{\{v,w\}\in E} \left(1_{\mathrm{intra}}(\{v,w\}) \cdot w(\{v,w\})\right)}{m + (|V||V-1|\frac{1}{2} - |E|)\cdot \bar{w}} \\
&+ \frac{\left(\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\mathrm{inter}}(\{v,w\}) - \sum_{\{v,w\}\in E} 1_{\mathrm{inter}}(\{v,w\})\right)\cdot \bar{w}}{m + (|V||V-1|\frac{1}{2} - |E|)\cdot \bar{w}}
\end{aligned} \tag{4.12}$$

4.6 Extending Modularity

In [32] a modularity definition for directed graphs with overlapping communities ($Q_{\mathrm{ov,dir}}$) is given

$$Q_{\mathrm{ov,dir}} = \frac{1}{|E|}\sum_{C_k\in\mathcal{C}}\sum_{v,w\in V}\left[\beta_{(v,w),C_k}\cdot A_{vw} - \frac{\beta^{\mathrm{out}}_{(v,w),C_k}\cdot \deg^{\mathrm{out}}(v)\cdot \beta^{\mathrm{in}}_{(v,w),C_k}\cdot \deg^{\mathrm{in}}(w)}{|E|}\right] . \tag{4.13}$$

In order to get a formula for an undirected graph with overlapping communities this formula is modified. The set of overlapping communities is $\mathcal{C}$. For every vertex $v \in V$ there exists a vector of *"belonging factors"* $[\alpha_{v,C_1},\ \alpha_{v,C_2},\ \ldots,\ \alpha_{v,C_{|\mathcal{C}|}}]$ which expresses to what extent this vertex belongs to the communities $C_1, C_2, \ldots, C_{|\mathcal{C}|}$. Without loss of generality the following conventions can be introduced

$$0 \le \alpha_{v,C_k} \le 1 \ \forall v \in V,\ \forall C_k \in \mathcal{C} \tag{4.14}$$

and

$$\sum_{k=1}^{|\mathcal{C}|} \alpha_{v,C_k} = 1\ . \tag{4.15}$$

In the same way "belonging factors" for the edges are introduced. The directed edge which starts at vertex v and ends in vertex w is (v,w) and $\beta_{(v,w)}$ is the corresponding "belonging factor". It has the following form

$$\beta_{(v,w),C_k} = \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) \tag{4.16}$$

and can be chosen like this

$$\beta_{(v,w),C_k} = \alpha_{v,C_k} \cdot \alpha_{w,C_k} \quad . \tag{4.17}$$

Another possible alternative is $\beta_{(v,w),C_k} = \max\{\alpha_{v,C_k}, \alpha_{w,C_k}\}$. The adjacency matrix A_{vw} of the graph $G = (V, E, w)$ has the following form

$$A_{vw} = \begin{cases} 1 & \text{if v and w are connected} \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

The in-degree of a vertex v is represented by $\deg^{\text{in}}(v)$ and the out-degree by $\deg^{\text{out}}(v)$.

$$\begin{aligned} \deg^{\text{in}}(v) &= |\{(w,v) \in E\}| & (4.19) \\ \deg_v^{\text{out}}(v) &= |\{(v,w) \in E\}| & (4.20) \end{aligned}$$

In case of a directed graph $\beta^{\text{out}}_{(v,w),C_k}$ and $\beta^{\text{in}}_{(v,w),C_k}$ define the expected "belonging factor" of any possible edge (v, w) starting from the vertex v into community C_k or ending in the vertex v into community C_k.

$$\begin{aligned} \beta^{\text{out}}_{(v,w),C_k} &= \frac{\sum_{w \in V} \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k})}{|V|} & (4.21) \\ \beta^{\text{in}}_{(v,w),C_k} &= \frac{\sum_{v \in V} \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k})}{|V|} & (4.22) \end{aligned}$$

In case of an undirected graph each (undirected) edge can be regarded as bidirected and the degree $\deg(v)$ of a vertex v describes the number of edges which are adjacent to v. And so the equations $\deg^{\text{out}}(v) \cdot \deg^{\text{in}}(w) = \deg^{\text{in}}(v) \cdot \deg^{\text{out}}(w) = \deg(v) \cdot \deg(w)$ and $A_{vw} = A_{wv}$ hold. The product $\beta^{\text{out}}_{(v,w),C_k} \cdot \beta^{\text{in}}_{(v,w),C_k}$ in (4.13) can be rewritten for the case of an undirected graph

$$\begin{aligned} \beta^{\text{out}}_{(v,w),C_k} \cdot \beta^{\text{in}}_{(v,w),C_k} = \underbrace{\beta^{\text{in}}_{(w,v),C_k}}_{\overset{\text{undirected graph}}{=} \beta^{\text{in}}_{(v,w),C_k}} \cdot \beta^{\text{in}}_{(v,w),C_k} &= (\beta^{\text{in}}_{(v,w),C_k})^2 \\ &= (\beta^{\text{in}}_{(w,v),C_k})^2 \\ &= (\beta^{\text{out}}_{(v,w),C_k})^2 \\ &= (\beta^{\text{out}}_{(w,v),C_k})^2 \quad . \end{aligned} \tag{4.23}$$

If an undirected edge is considered to be bidirected it is counted twice and so $|E|$ in (4.13) has to be replaced by $2|E|$. Finally the following equation of modularity, if overlapping communities are allowed, is derived

$$Q_{\text{ov,undir}} = \frac{1}{2|E|} \sum_{C_k \in \mathcal{C}} \sum_{v,w \in V} \left[\beta_{(v,w),C_k} \cdot A_{vw} - \frac{\deg(v) \cdot (\beta^{\text{in}}_{(v,w),C_k})^2 \cdot \deg(w)}{2|E|} \right] \quad . \tag{4.24}$$

In general the formula of $Q_{\text{ov,undir}}$ depends on the choice of $\mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k})$ in equation (4.16). Regarding the example in Figure 4.1(a) shows that neither the formula of $Q_{\text{ov,undir}}$ with $\mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \alpha_{v,C_k} \cdot \alpha_{w,C_k}$ nor $Q_{\text{ov,undir}}$ with $\mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \max\{\alpha_{v,C_k}, \alpha_{w,C_k}\}$ is a generalisation of the formula of modularity for the non-overlapping case ($\text{mod}(\mathcal{C})$). If communities do not overlap, an edge is either an intra-cluster edge or an inter-cluster edge, thus $\beta^{\text{in}}_{(v,w),C_k}$ should be 1 or 0 in these cases [1]. Regarding $\beta^{\text{in}}_{(v,1),C_1}$ and $\beta^{\text{in}}_{(v,1),C_2}$ for vertex 1 shows that this condition is not fulfilled. The vectors of belonging factors for the vertices 1, 2, 3, 4 are given by $[1, 0]$ and for the vertices 5, 6, 7, 8 by $[0, 1]$.

$$\beta^{\text{in}}_{(v,1),C_1} = \begin{cases} \frac{1\cdot1+1\cdot1+1\cdot1+1\cdot1+1\cdot0+1\cdot0+1\cdot0+1\cdot0}{8} = \frac{1}{2} \\ \text{if } \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \alpha_{v,C_k} \cdot \alpha_{w,C_k} \\ \\ \frac{\max\{1,1\}+\max\{1,1\}+\max\{1,1\}+\max\{1,1\}+\max\{1,0\}+\max\{1,0\}+\max\{1,0\}+\max\{1,0\}}{8} = 1 \\ \text{if } \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \max\{\alpha_{v,C_k}, \alpha_{w,C_k}\} \end{cases} \tag{4.25}$$

$$\beta^{\text{in}}_{(v,1),C_2} = \begin{cases} \frac{0\cdot0+0\cdot0+0\cdot0+0\cdot0+0\cdot1+0\cdot1+0\cdot1+0\cdot1}{8} = 0 \\ \text{if } \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \alpha_{v,C_k} \cdot \alpha_{w,C_k} \\ \\ \frac{\max\{0,0\}+\max\{0,0\}+\max\{0,0\}+\max\{0,0\}+\max\{0,1\}+\max\{0,1\}+\max\{0,1\}+\max\{0,1\}}{8} = \frac{1}{2} \\ \text{if } \mathcal{F}(\alpha_{v,C_k}, \alpha_{w,C_k}) = \max\{\alpha_{v,C_k}, \alpha_{w,C_k}\} \end{cases} \tag{4.26}$$

Consequently the formula in (4.24) has to be optimised. Therefore we introduce the *commonness* $c_{v,w}$. This measure gives the probability that an (undirected) edge $\{v, w\}$ is an intra-cluster edge. Let $\mathcal{C}(v)$ be the set of communities to which v belongs, and let $\mathcal{C}(v \cdot w)$ be the set of communities to which v *and* w belong, then the commonness of v and w is given by

$$c_{v,w} = \frac{|\mathcal{C}(v \cdot w)|}{|\mathcal{C}(v)| \cdot |\mathcal{C}(w)|} \ . \tag{4.27}$$

This definition is meaningful as can be seen with the help of an example. Suppose $|\mathcal{C}(v)| = 2$ and $|\mathcal{C}(w)| = 2$. If the two clusters C_1 and C_2, which contain v also contain w, the commonness $c_{v,w} = \frac{2}{2\cdot2} = \frac{1}{2}$ is derived. This is intuitive because the edge $\{v, w\}$ can be interpreted in four ways: starting in C_1 and ending in C_1, starting in C_1 and ending in C_2, starting in C_2 and ending in C_1, starting in C_2 and ending in C_2. Two of the interpretations indicate that $\{v, w\}$ is an intra-cluster edge, therefore $c_{v,w} = \frac{1}{2}$ is intuitive.

In case of non-overlapping communities it is $c_{v,w} = 1$ for intra-cluster edges and $c_{v,w} = 0$ for inter-cluster edges. The commonness permits to give a measure how *intra-* and how *inter-cluster* an edge is in the overlapping case.

The probability that an edge is an inter-cluster edge is given by $\overline{c_{v,w}} = \frac{|\mathcal{C}(v)|\cdot|\mathcal{C}(w)|-|\mathcal{C}(v\cdot w)|}{|\mathcal{C}(v)|\cdot|\mathcal{C}(w)|} = 1 - c_{v,w}$. This leads to a more differentiated consideration of intra- and inter-cluster edges in the overlapping case than the consideration derived with the definitions of $1_{\text{intra}}(\{v, w\})$ and $1_{\text{inter}}(\{v, w\})$ in chapter 4.4 and 4.5. The resulting formula for the modularity for overlapping communities is

$$\text{mod}^{\text{ov}}(\mathcal{C}) = \frac{1}{2|E|} \sum_{C_k \in \mathcal{C}} \sum_{\{v,w\} \in V \times V} \beta_{\{v,w\},C_k} \cdot A_{vw} - \frac{1}{4|E|^2} \sum_{\{v,w\} \in V \times V} c_{v,w} \cdot \deg(v) \cdot \deg(w) \tag{4.28}$$

[1] Remember that modularity in the non-overlapping and unweighted case can be expressed as

$$\text{mod}(\mathcal{C}) = \frac{1}{2|E|} \sum_{i,j \in V} \left[A_{i,j} - \frac{\deg(i) \cdot \deg(j)}{2|E|} \right] \delta(c_i, c_j)$$

with $\delta(c_i, c_j) = 1$ if i and j belong to the same community and $\delta(c_i, c_j) = 0$ otherwise.

Let $s_v = \sum_{e \sim v} w(e)$ be the strength of vertex v. The *overlapping* definition can be extended for a weighted graph as well. Let m be the sum of all edge weights, $m = \sum_{\{v,w\} \in E} w(\{v,w\})$. The resulting formula for the weighted case, if overlapping communities are allowed, has the following form

$$\mathrm{mod}_{\mathrm{w}}^{\mathrm{ov}}(\mathcal{C}) = \frac{1}{2m} \sum_{C_k \in \mathcal{C}} \sum_{\{v,w\} \in V \times V} \beta_{\{v,w\},C_k} \cdot w(\{v,w\}) \cdot A_{vw} - \frac{1}{4m^2} \sum_{\{v,w\} \in V \times V} c_{v,w} \cdot s_v \cdot s_w \qquad (4.29)$$

Consequently we have derived a true generalisation for modularity regarding both the overlapping and the weighted case. The introduced commonness $c_{v,w}$ can also be applied for coverage and performance.

Coverage including the "commonness" The coverage for the overlapping case (cf equations (4.9), (4.10)) are kind of *optimistic*. Comparing these values with the first summand of the modularity formulas $\mathrm{mod}^{\mathrm{ov}}(\mathcal{C})$ and $\mathrm{mod}_{\mathrm{w}}^{\mathrm{ov}}(\mathcal{C})$ in equation (4.28) and (4.29) shows the difference of regarding an edge either as intra-cluster edge or as inter-cluster edge with

$$\mathrm{cov}^{\mathrm{ov}}(\mathcal{C}) = \frac{\sum_{\{v,w\} \in E} 1_{\mathrm{intra}}(\{v,w\})}{|E|}$$

or considering the commonness $c_{v,w}$ of the endpoints of an edge with

$$\frac{1}{2|E|} \sum_{C_k \in \mathcal{C}} \sum_{\{v,w\} \in V \times V} \beta_{\{v,w\},C_k} \cdot A_{vw} \quad .$$

The commonness of the endpoints of an edge is implied in $\sum_{C_k \in \mathcal{C}} \sum_{\{v,w\} \in V \times V} \beta_{\{v,w\},C_k} \cdot A_{vw}$, since $\beta_{\{v,w\},C_k}$ is chosen as $\beta_{\{v,w\},C_k} = \alpha_{v,C_k} \cdot \alpha_{w,C_k} = \frac{1}{\mathcal{C}(v)} \cdot \frac{1}{\mathcal{C}(w)}$, which is nothing else than $\frac{c_{v,w}}{|\mathcal{C}(v \cdot w)|}$. Consequently the following holds

$$\frac{1}{2|E|} \sum_{C_k \in \mathcal{C}} \sum_{\{v,w\} \in V \times V} \beta_{\{v,w\},C_k} \cdot A_{vw} = \frac{\sum_{\{v,w\} \in E} c_{v,w}}{|E|} = \mathrm{cov}_{\mathrm{alternative}}^{\mathrm{ov}}(\mathcal{C}) \quad . \qquad (4.30)$$

An alternative formula for $\mathrm{cov}_{\mathrm{w}}^{\mathrm{ov}}(\mathcal{C})$ can be obtained in the same way

$$\mathrm{cov}_{\mathrm{w,alternative}}^{\mathrm{ov}}(\mathcal{C}) = \frac{\sum_{\{v,w\} \in E} \left(c_{v,w} \cdot w(\{v,w\})\right)}{m} \quad . \qquad (4.31)$$

Performance including the "commonness" The performance formula for the overlapping case, which integrates the commonness $c_{v,w}$, has the following form:

$$\begin{aligned} \mathrm{perf}_{\mathrm{alternative}}^{\mathrm{ov}}(\mathcal{C}) &= \frac{\sum_{\{v,w\} \in E} c_{v,w}}{|V||V-1|\frac{1}{2}} \\ &+ \frac{\frac{1}{2}\sum_{\{v,w\} \in V \times V} 1_{\mathrm{inter}}(\{v,w\}) - \sum_{\{v,w\} \in E} 1_{\mathrm{inter}}(\{v,w\})}{|V||V-1|\frac{1}{2}} \\ &+ \frac{\sum_{\{v,w\} \in E} (1 - \overline{c_{v,w}})}{|V||V-1|\frac{1}{2}} \quad . \end{aligned} \qquad (4.32)$$

The judgement of a missing inter-cluster edge with $w_{\max}$ and a inter-cluster edge $\{v,w\}$ with $w_{\max} - w(\{v,w\})$ in the weighted and overlapping case has been discussed before. Integrating the commonness additionally would result in the following performance formula

$$\begin{aligned} \operatorname{perf}^{\text{ov}}_{\text{w,alternative}}(\mathcal{C}) &= \frac{\sum_{\{v,w\}\in E} (c_{v,w} \cdot w(\{v,w\}))}{(|V||V-1|\frac{1}{2}) \cdot w_{\max}} \\ &+ \frac{\left(\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\text{inter}}(\{v,w\}) - \sum_{\{v,w\}\in E} 1_{\text{inter}}(\{v,w\})\right) \cdot w_{\max}}{(|V||V-1|\frac{1}{2}) \cdot w_{\max}} \\ &+ \frac{\sum_{\{v,w\}\in E} (\overline{c_{v,w}} \cdot (w_{\max} - w(\{v,w\})))}{(|V||V-1|\frac{1}{2}) \cdot w_{\max}} \quad . \end{aligned} \tag{4.33}$$

4.7 An Example

Within this section the quality indices for the example in Figure 4.1(a) - Figure 4.1(d) are considered. The colours of the vertices indicate to which community they belong. C_1 is the green coloured community and C_2 is the orange coloured community. If a vertex is green and orange at the same time, it belongs to both of the communities.

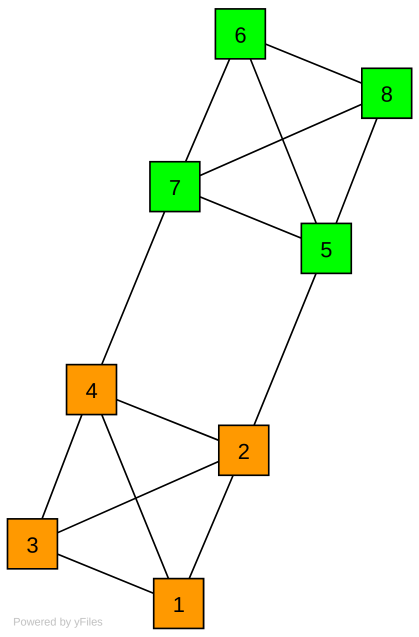
Figure 4.1(a) This is the most *classical* case. The graph is regarded as unweighted and C_1 and C_2 do not overlap. The definition of the required formulas $\operatorname{cov}(\mathcal{C})$, $\operatorname{perf}(\mathcal{C})$ and $\operatorname{mod}(\mathcal{C})$ are given in (4.2), (4.4) and (4.7).

Figure 4.1(b) In this figure the weighted graph is considered. The communities C_1 and C_2 do not overlap. The definitions $\operatorname{cov}_{\text{w}}(\mathcal{C})$, $\operatorname{perf}_{\text{w}}(\mathcal{C})$ and $\operatorname{mod}_{\text{w}}(\mathcal{C})$ are given in (4.3), (4.6) and (4.8).

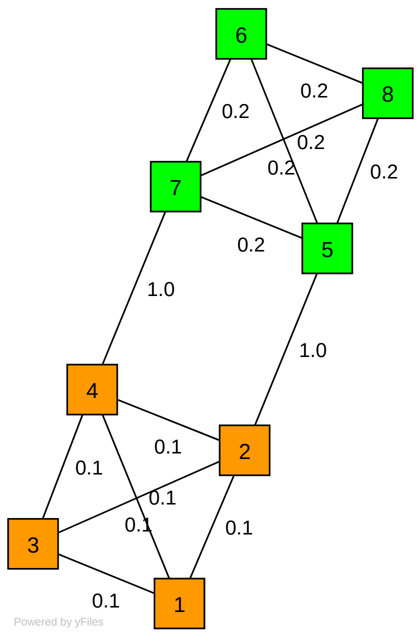
Figure 4.1(c) Within this figure, the vertices 2, 4, 5 and 7 belong to both, C_1 and C_2, whereas the graph is regarded as unweighted. The formulas of $\operatorname{cov}^{\text{ov}}(\mathcal{C})$, $\operatorname{perf}^{\text{ov}}(\mathcal{C})$ and $\operatorname{mod}^{\text{ov}}(\mathcal{C})$ are given in (4.9), (4.11) and (4.28).

Figure 4.1(d) As in Figure 4.1(c) the communities C_1 and C_2 overlap. This time the graph is regarded as weighted and the definitions $\operatorname{cov}^{\text{ov}}_{\text{w}}(\mathcal{C})$, $\operatorname{perf}^{\text{ov}}_{\text{w}}(\mathcal{C})$ and $\operatorname{mod}^{\text{ov}}_{\text{w}}(\mathcal{C})$ are given in (4.10), (4.12) and (4.29).

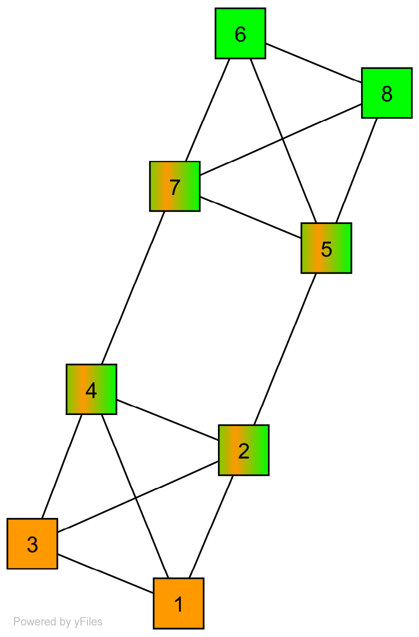
Table 4.1 shows the quality indices for the examples described above. The calculations of the quality indices for the examples in Figure 4.1(a) and Figure 4.1(b) show that the overlapping formulas are *real* generalisations of the non-overlapping formulas. In Table 4.2 a general overview for all discussed cases is given.



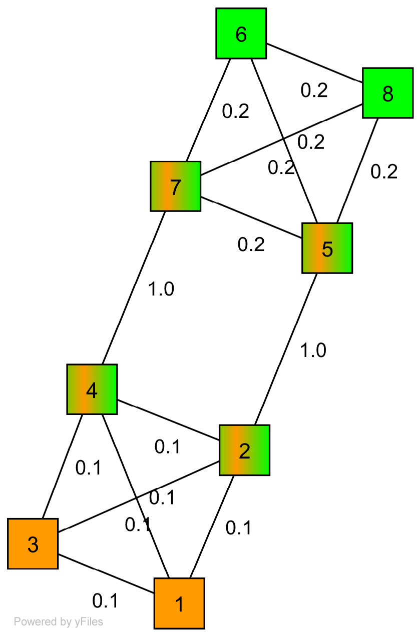
(a) unweighted, no overlaps



(b) weighted, no overlaps



(c) unweighted, overlaps



(d) weighted, overlaps

Figure 4.1: Different clusterings for the unweighted and weighted case of an example graph.

Quality indices in comparison				
quality index	unweighted no overlaps Figure 4.1(a)	weighted no overlaps Figure 4.1(b)	unweighted overlaps Figure 4.1(c)	weighted overlaps Figure 4.1(d)
$\text{mod}(\mathcal{C})$	0.3571			
$\text{cov}(\mathcal{C})$	0.8571			
$\text{perf}(\mathcal{C})$	0.9286			
$\text{mod}_\text{w}(\mathcal{C})$		-0.0388		
$\text{cov}_\text{w}(\mathcal{C})$		0.4737		
$\text{perf}_\text{w}(\mathcal{C})$		0.7368		
$\text{mod}^\text{ov}(\mathcal{C})$	0.3571		0.0714	
$\text{cov}^\text{ov}(\mathcal{C})$	0.8571		1.0	
$\text{perf}^\text{ov}(\mathcal{C})$	0.9286		0.6429	
$\text{mod}^\text{ov}_\text{w}(\mathcal{C})$		-0.0388		0.0364
$\text{cov}^\text{ov}_\text{w}(\mathcal{C})$		0.4737		1.0
$\text{perf}^\text{ov}_\text{w}(\mathcal{C})$		0.7368		0.6429

Table 4.1: Coverage, performance and modularity for the example graphs in comparison.

Formulas of quality indices	
case	formula
unweighted no overlaps (Figure 4.1(a))	$\mathrm{cov}(\mathcal{C}) = \frac{\lvert E(\mathcal{C})\rvert}{\lvert E\rvert}$ (4.2) $\mathrm{perf}(\mathcal{C}) = \frac{\lvert E(\mathcal{C})\rvert+m_{\text{missing inter}}(\mathcal{C})}{\lvert V\rvert\lvert V-1\rvert\frac{1}{2}}$ (4.4) $\mathrm{mod}(\mathcal{C}) = \frac{\lvert E(\mathcal{C})\rvert}{\lvert E\rvert} - \frac{1}{4\lvert E\rvert^2}\sum_{C\in\mathcal{C}}\left(\sum_{v\in C}\deg(v)\right)^2$ (4.7)
weighted no overlaps (Figure 4.1(b))	$\mathrm{cov}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E(\mathcal{C})} w(\{v,w\})}{m}$ (4.3) $\mathrm{perf}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E(\mathcal{C})} w(\{v,w\})+m_{\text{missing inter}}\cdot\bar{w}}{m+(\lvert V\rvert\lvert V-1\rvert\frac{1}{2}-\lvert E\rvert)\cdot\bar{w}}$ (4.6) $\mathrm{mod}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E(\mathcal{C})} w(\{v,w\})}{m} - \sum_{C_k\in\mathcal{C}}\left(\frac{\sum_{v\in C_k} s_v}{2m}\right)^2$ (4.8)
unweighted overlaps (Figure 4.1(c))	$\mathrm{cov}^{\mathrm{ov}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} 1_{\text{intra}}(\{v,w\})}{\lvert E\rvert}$ (4.9) $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} 1_{\text{intra}}(\{v,w\})+\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\text{inter}}(\{v,w\})-\sum_{\{v,w\}\in E} 1_{\text{inter}}(\{v,w\})}{\lvert V\rvert\lvert V-1\rvert\frac{1}{2}}$ (4.11) $\mathrm{mod}^{\mathrm{ov}}(\mathcal{C}) = \frac{1}{2\lvert E\rvert}\sum_{C_k\in\mathcal{C}}\sum_{\{v,w\}\in V\times V}\beta_{\{v,w\},C_k}\cdot A_{vw}$ $-\frac{1}{4\lvert E\rvert^2}\sum_{\{v,w\}\in V\times V} c_{v,w}\cdot\deg(v)\cdot\deg(w)$ (4.28)
weighted overlaps (Figure 4.1(d))	$\mathrm{cov}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E}(1_{\text{intra}}(\{v,w\})\cdot w(\{v,w\}))}{m}$ (4.10) $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E}(1_{\text{intra}}(\{v,w\})\cdot w(\{v,w\}))}{m+(\lvert V\rvert\lvert V-1\rvert\frac{1}{2}-\lvert E\rvert)\cdot\bar{w}}$ $+\frac{\left(\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\text{inter}}(\{v,w\})-\sum_{\{v,w\}\in E} 1_{\text{inter}}(\{v,w\})\right)\cdot\bar{w}}{m+(\lvert V\rvert\lvert V-1\rvert\frac{1}{2}-\lvert E\rvert)\cdot\bar{w}}$ (4.12) $\mathrm{mod}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C}) = \frac{1}{2m}\sum_{C_k\in\mathcal{C}}\sum_{\{v,w\}\in V\times V}\beta_{\{v,w\},C_k}\cdot w(\{v,w\})\cdot A_{vw}$ $-\frac{1}{4m^2}\sum_{\{v,w\}\in V\times V} c_{v,w}\cdot s_v\cdot s_w$ (4.29)

Table 4.2: Formulas of coverage, performance and modularity.

Alternative formulas of quality indices	
case	formula
unweighted overlaps (Figure 4.1(c))	$\mathrm{cov}^{\mathrm{ov}}_{\mathrm{alternative}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} c_{v,w}}{\vert E\vert}$ (4.30) $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{alternative}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E} c_{v,w}}{\vert V\vert\vert V-1\vert\frac{1}{2}} + \frac{\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\mathrm{inter}}(\{v,w\}) - \sum_{\{v,w\}\in E} 1_{\mathrm{inter}}(\{v,w\})}{\vert V\vert\vert V-1\vert\frac{1}{2}} + \frac{\sum_{\{v,w\}\in E}(1-\overline{c_{v,w}})}{\vert V\vert\vert V-1\vert\frac{1}{2}}$ (4.32)
weighted overlaps (Figure 4.1(d))	$\mathrm{cov}^{\mathrm{ov}}_{\mathrm{w,alternative}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E}(c_{v,w}\cdot w(\{v,w\}))}{m}$ (4.31) $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{w,alternative}}(\mathcal{C}) = \frac{\sum_{\{v,w\}\in E}(c_{v,w}\cdot w(\{v,w\}))}{(\vert V\vert\vert V-1\vert\frac{1}{2})\cdot w_{\max}} + \frac{\left(\frac{1}{2}\sum_{\{v,w\}\in V\times V} 1_{\mathrm{inter}}(\{v,w\}) - \sum_{\{v,w\}\in E} 1_{\mathrm{inter}}(\{v,w\})\right)\cdot w_{\max}}{(\vert V\vert\vert V-1\vert\frac{1}{2})\cdot w_{\max}} + \frac{\sum_{\{v,w\}\in E}(\overline{c_{v,w}}\cdot(w_{\max}-w(\{v,w\})))}{(\vert V\vert\vert V-1\vert\frac{1}{2})\cdot w_{\max}}$ (4.33)

Table 4.3: Alternative formulas for coverage and performance.

Chapter 5

Network Analysis

In general the structural properties of an abstract graph are methodological determinants in many application areas. One of these areas is clustering. The structure of the graph G_{buy} (cf. Chapter 3.1) could influence for example the running time of a clustering algorithm or the number of detected communities. Therefore a network analysis is made to get a detailed view of some properties of G_{buy}.

5.1 Visualisation of the Graph

The TGF ("Trivial Graph Format") is a graph format, which includes primarily the structure of the graph. That means any information concerning an embedding, the visualisation or additional attributes is not included and a memory-efficient storage is possible. An example of this format is shown below.

```
1 "FirstProduct"
2 "SecondProduct"
3 "ThirdProduct"
4 "FourthProduct"
5 "FifthProduct"
#
1 3 0.2
2 3 0.267
2 4 0.462
2 5 0.287
3 4 0.21
4 5 0.53
```

It is one of the possible input formats for the graph editor yEd[1], which is available as a free download. In Appendix A a visualisation of the graph G_{buy}, which is derived with yEd, is given. The chosen layout is called *organic classic* and is especially appropriate to represent complex undirected graphs. Application areas for this layout algorithm are bioinformatics, enterprise networking, knowledge representation, systems management and World Wide Web visualisation. It is based on the force directed layout paradigm. The vertices are considered to be physical objects with repulsive forces to one another, like protons or electrons. The edges are also considered to have physical properties and act as metal springs, which are attached to the vertices. The layout algorithm simulates the physical forces between the vertices and according to these the positions of the vertices are

[1]The license conditions are published at http://www.yworks.com/products/yed/license.html.

rearranged. As a result the sum of the forces emitted by the vertices and the edges reaches a (local) minimum.

The visualisation in Appendix A (Figure A.1) shows that the graph consists of several components. A more detailed overview of the components is given in Table 5.1. Obviously this graph is *disconnected*, that means the graph is not composed of only one component. This is an important first observation.

number of components	number of vertices in these components
72	2
12	3
3	4
1	5
1	6
1	9
1	5638

Table 5.1: Overview of the components of G_{buy}.

In Figure A.2 in Appendix A an enlargement of the visualisation of Figure A.1 is given. With this point of view the meaning of the modelled weights gets more clear. Within the left component the product in the middle could have been bought three times, while the two neighbour products have been bought once and the two products, which are on the top and the button have been bought twice. The edge weight $\frac{1}{2}$ ($\frac{1}{3}$) means that the two products, which build the endpoints of this edge, have been bought together once, while one of the products has been bought twice (three times).

5.2 Degree Distribution

In Figure 5.1 the degree distribution of the vertices is considered as a next step (it has been derived with the help of **JUNG**[2]). The average degree is 30.6284, this means the graph is *relatively dense.* The logarithmic trend of the degree distribution can be approximated by a power-law function. In general a degree distribution p, which follows a power-law, has the following form (cf. [11])

$$p(k) = ck^{-\delta} \quad \delta > 0,\ c > 0\ . \tag{5.1}$$

In case of G_{buy} the formula $p(k) = 2000 \cdot k^{-1.3}$ gives such an approximation of the degree distribution (pink straight *line* in Figure 5.1).

Examples for graphs, which have a power-law degree distribution include (cf. [11]) the actor collaboration graph ($\delta \approx 2.3$), the power grid of the United States of America ($\delta \approx 4$) and the Internet ($\delta \approx 2.2$).

5.3 Betweenness Centrality

The *centrality* of a vertex is in general a measure of the structural importance of the vertex. The betweenness centrality in particular descibes the number of "times" that a vertex has to go through a given vertex to reach any other vertex by the shortest path. In G_{buy} this centrality measure

[2]This software library is freely provided under the BSD open-source license.

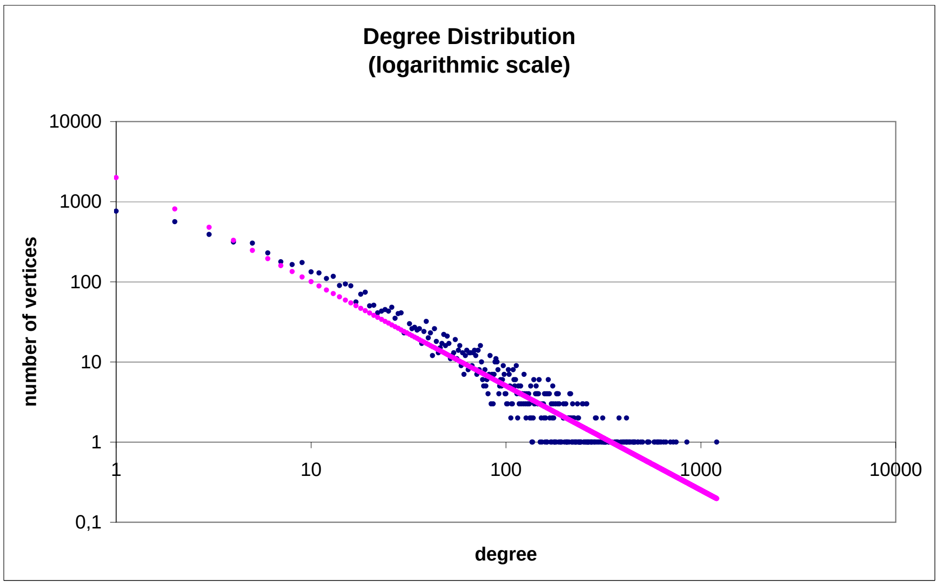


Figure 5.1: Degree distribution of the vertices of G_{buy}.

indicates weather a vertex has a favoured role and thus is central or not. In [9] a fast algorithm for betweenness centrality is introduced. The running time is $O(nm)$ and $O(nm+n^2 \log n)$ on unweighted and weighted graphs. A general formula of the betweenness centrality is

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \, . \tag{5.2}$$

This centrality index is based on shortest paths, as described before. Let $\sigma_{st} = \sigma_{ts}$ be the number of shortest paths from vertex s to vertex t, $\sigma_{ss} = 1$ by convention. Let $\sigma_{st}(v)$ denote the number of shortest paths from s to t, which pass vertex $v \in V$. In Figure 5.2 the distribution of the relative betweenness centralities is given. The computation has been realised with the help of **JUNG**[3]. The relative betweenness centrality of the most important vertex v_1 is $0,0763$. The big difference to the betweenness centrality of the second important vertex v_2 $(0,0272)$ clarifies the exclusive importance of the top-central vertex v_1. The distribution confirms the intuition that there exists a huge number of vertices, which are similar important. This is one of the challenges, which have to be faced by a clustering algorithm.

The betweenness centrality has been introduced regarding social networks. In this context a vertex with a high value holds a favoured position in the network and is at the same time a single point of failure.

The betweenness centrality in the context of G_{buy} can only be transferred to the interpretation concerning the favoured role of a vertex. In general a vertex v with a high betweenness centrality could either be a vertex, which represents a product purchased with a huge number of other products (”universal product”) or v could represent a product, which has been bought with only a few other products (comparable to a single point of failure). In the second case v would not be important

[3]This software library is freely provided under the BSD open-source license.

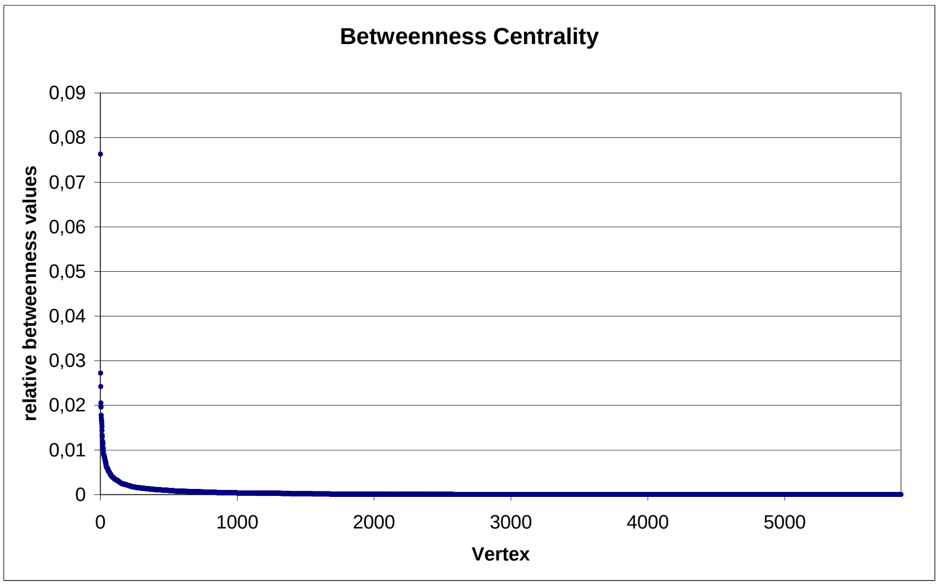


Figure 5.2: Betweenness centrality of the vertices of G_{buy}.

in relation to the purchase with the majority of the products. However the visualisation of G_{buy} in Appendix A shows one component, which contains the majority of the vertices. There is no bottleneck and thus a vertex of G_{buy} with high betweenness centrality represents a universal product.

5.4 Clustering Coefficients

The clustering coefficient $c(v)$ of a vertex v represents the probability that two neighbours of v are also neighbours of each other. In the context of G_{buy} this means that the products, which are represented by vertices with a high clustering coefficient, are strongly connected with *combinations* of their neighbours. The average of $c(v)$ over all vertices is regarded as the clustering coefficient of the whole graph. The coefficient can be expressed in terms of triangles and tripels. A *triangle* $\triangle = \{V_\triangle, E_\triangle\}$ is a completely connected subgraph of G_{buy} with three vertices. The number of triangles of a vertex v is involved in $\lambda(v) = |\{\triangle | v \in V_\triangle\}|$. A *triple* is a subgraph of G_{buy} with three vertices and two edges. It is a *triple at vertex* v, if both edges have v as an endpoint. The number of triples of v can be formulated as

$$\tau(v) = \binom{deg(v)}{2} = \frac{deg(v)^2 - deg(v)}{2} \,. \tag{5.3}$$

The clustering coefficient $c(v)$ is defined for $\tau(v) \neq 0$ as

$$c(v) = \frac{\lambda(v)}{\tau(v)} \,. \tag{5.4}$$

A discussion concerning the clustering coefficient is described in [11]. As can be seen in Figure 5.3 about 2000 vertices have a clustering coefficient $c(v) = 1$ and almost 3000 vertices derive a value

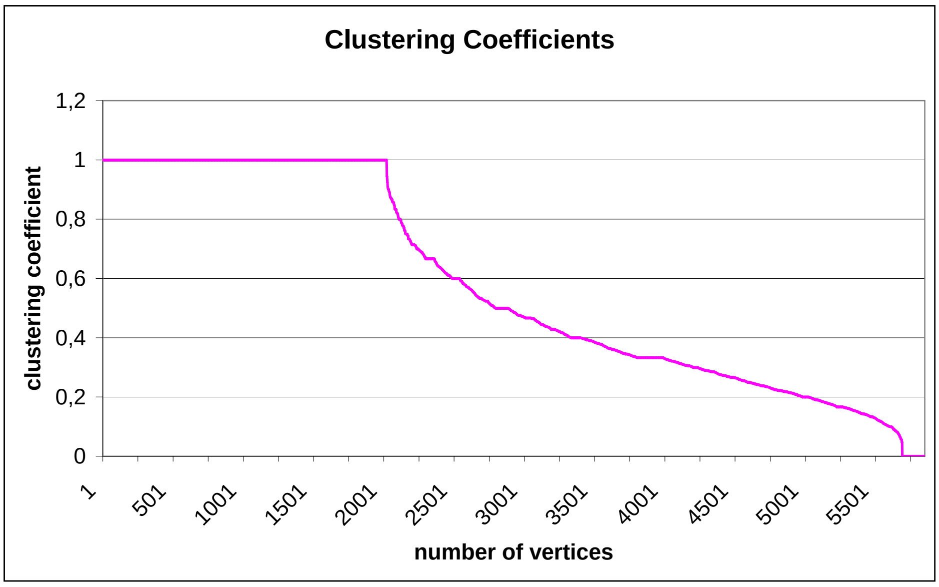


Figure 5.3: Clustering coefficients of the vertices of G_{buy}.

greater than 0.5. These values are calculated with the help of **JUNG**[4]. For vertices with $deg(v) = 1$ (761 vertices of G_{buy} have degree 1), $c(v)$ is defined to be 1. Nevertheless these clustering coefficients indicate the enormous density of some parts of G_{buy}. The clustering coefficient of the graph is $0,5773$, thus the probability that two neighbours of a vertex are also neighbours of each other is more than one half.

The Internet at the Autonomous System (AS) level is a famous graph, which has also been analysed with respect to the clustering coefficient (cf. [5]). The clustering coefficient of the AS graph depends on the period, which is analysed. The graph of January 2002 has a clustering coefficient of 0.45, the graph of January 2006 possesses a clustering coefficient of 0.38 and in July 2007 the clustering coefficient of the AS graph has been 0.33. These values are all below the clustering coefficient of G_{buy}. This clarifies the significant relatedness of the vertices, which represent the products to purchase.

In conclusion, the degree distribution and the clustering coefficients indicate the enormous density of G_{buy}. The centrality betweenness additionally shows the central role of a few vertices. These properties of G_{buy} are a result of the graph modelling process. The products, which have been bought together by a customer, are represented by connected vertices. The degree distribution shows, that a few products are connected to a huge number of other products. These products are the "universal products" and probably the top sellers as well. The betweenness centrality emphasises the central role of these few products, which are bought with the combination of very many other products. One of these products could be a battery. This product can be essential for a large quantity of products. The clustering coefficients show, that the probability that two neighbours of a vertex are also neighbours of each other, is high. That means that the combination of more than two products

[4]This software library is freely provided under the BSD open-source license.

often occurs. An example could be the purchase of a game console with batteries and a game.

5.5 k-Core Analysis

The k-core decomposition is an appropriate method of analysis in order to analyse a graph with a hierarchical structure of importance. Moreover a running time of $O(\max(m, n))$ can be attained (m is the number of edges, n the number of vertices) for this analysis (cf. Algorithm 4). Since the possible maximum of m is $m_{\max} = \frac{n \cdot (n-1)}{2}$, $O(\max(m, n))$ is at most $O(n^2)$. Nevertheless it can be applied to large *sparse* graphs. Both is relevant in case of G_{buy}.

A *k-core* (or *"core of the order k"*) is, according to [4], a subgraph $H_k = (W, E|W)$ of the graph G_{buy} with the following properties: it is induced by the set $W \subseteq V$, which is defined as $W := \{v \in V | deg_H(v) \geq k\}$, and H_k is the maximum subgraph with this property. In other words each vertex $v \in H_k$ has at least k neighbours in the same core H_k.

The core with the maximum k is called *main* core and the *core number* of vertex v is the highest order of a core that includes v. In order to clarify this definition, an example is given in Figure 5.4. It can be seen, that cores are nested, $i < j \;\Rightarrow\; H_j \subseteq H_i$, and furthermore not necessarily connected subgraphs.

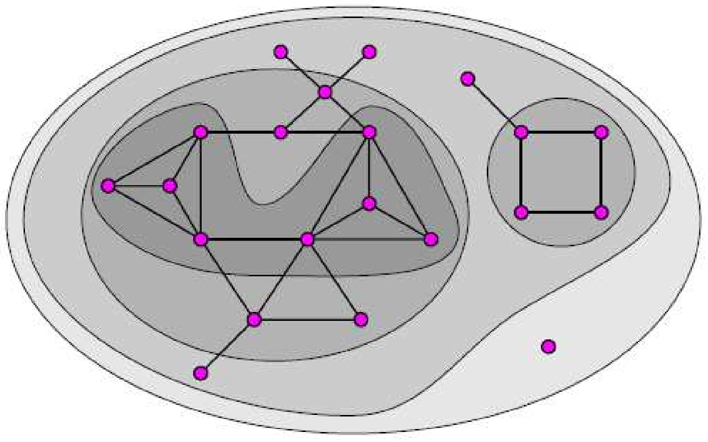


Figure 5.4: Example of 0, 1, 2 and 3 core, taken from [4].

The algorithm for determining k-cores, which is introduced in [4], is based on the property:

> If from a given graph $G = (V, E)$ we recursively delete all vertices, and edges incident with them, of degree less than k, the remaining graph is the k-core.

The corresponding pseudocode is given in Algorithm 4. In line 6 - line 9 the deletion of vertex v and all edges, which are incident with this vertex, is described.
The k-core decomposition is realised with the tool **LaNet-vi**[5]. The result for the undirected graph G_{buy} is shown in Figure 5.5. In [2] an interpretation guide is given. A vertex v has *shell index k*, if it is contained in the k-core, but not in the $k+1$-core. The vertices of a shell are drawn with the same

[5]The images are licensed under a Creative Commons License (http://creativecommons.org/licenses/by-nc/2.0/). Authors: Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, Alessandro Vespignani.

Algorithm 4: k-CORE DECOMPOSITION

```
Input: Graph $G = (V, E)$ represented by a list of neighbours.
Output: Table core with the core number for each vertex.
1 foreach vertex $v \in V$ do
2     Compute the degree of $v$.
3 Order the set of vertices $V$ in increasing order of their degrees.
4 foreach vertex $v \in V$ in the order do
5     core[$v$] := degree[$v$] ;
6     foreach $u \in$ neighbours($v$) do
7         if degree[$u$] > degree[$v$] then
8             degree[$u$] := degree[$u$] − 1 ;
9             Reorder $V$ accordingly.
```

colour. On the right these shell indices are listed from $k_{\max} = 73$ down to $k_{\min}$. The size of each vertex is proportional to the original degree of this vertex. The vertex sizes with the corresponding degree are listed on the left. The diameter of each k-shell depends on the index k and is proportional to $k_{\max} - k$. The concentric layout is a result of the trivial order relation of the shell indices.

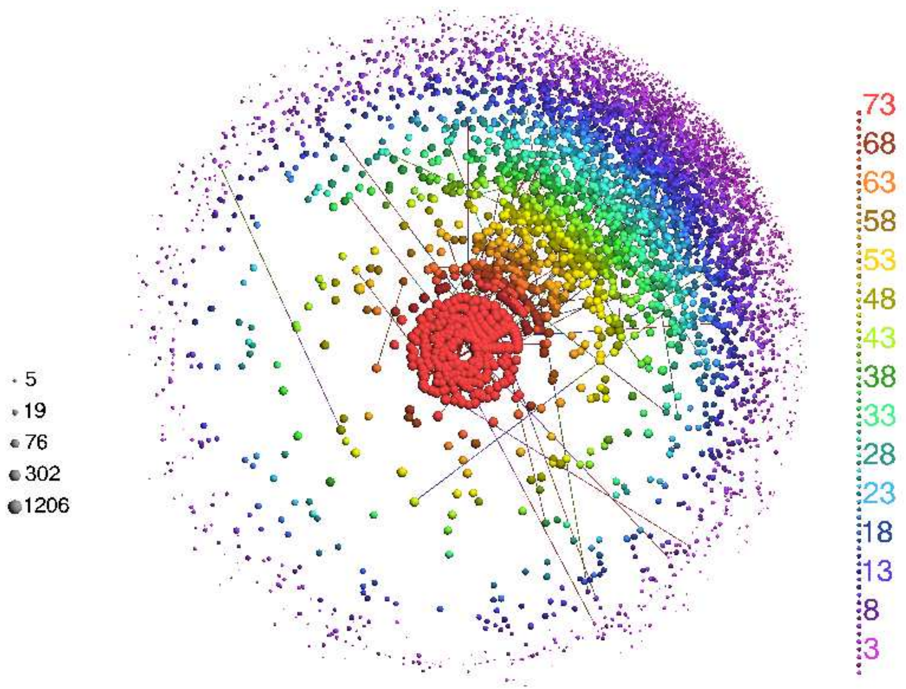


Figure 5.5: k-cores of the connected components of G_{buy} (LaNet-vi).

Shells width: The vertices of a shell are arranged around the corresponding diameter. Vertices, which have neighbours of higher shell indices are closer to the center and vice versa. The width of a shell is the maximum possible distance between two vertices of this shell. Regarding Figure 5.5 shows that the shells of G_{buy} are wide and overlap each other.

Shell clusters: The angular distribution of the vertices depends on the connectivity of these vertices in the graph. Vertices of the same k-shell, which are connected to each other, are positioned close to each other. In this way shell clusters are derived and it is easy to see, whether a shell consists of a large connected component or of several components. All the shells consist of one large component and several isolated vertices (these vertices are isolated with respect to the other vertices of the same shell).

Degree-Coreness Correlation: Another issue to analyse is the correlation between the degree of the vertices and the shell index. Figure 5.5 shows that this correlation exists within G_{buy}. Due to the proportionality of the vertex size to the vertex degree, the correlation is represented by large vertices in the center and smaller vertices in more external areas. That means the central vertices are high-degree hubs of the graph. Many real communication networks with a hierarchical structure possess this correlation. Examples are the Internet at the Autonomous System level or the World Wide Air-transportation network.

Edges: Only a randomly chosen fraction of the edges is drawn. On the one hand the visualisation is clearer, on the other hand information of the graph structure gets "lost".

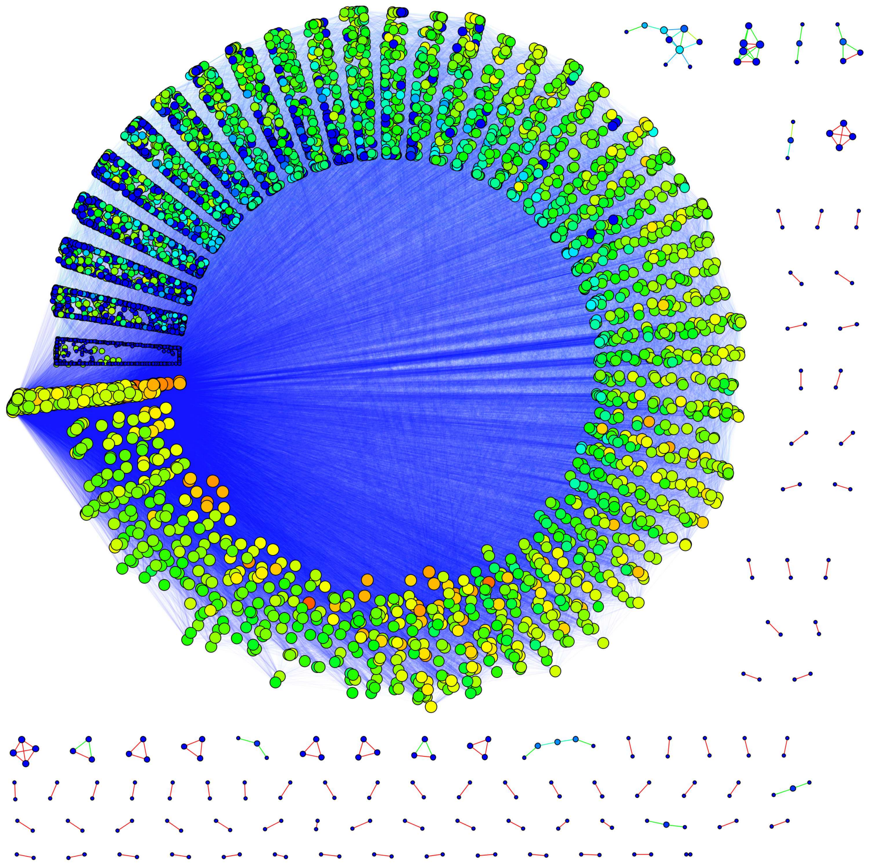


Figure 5.6: k-cores of G_{buy} (LunarVis).

Another k-core analysis tool, which allows for a more structurally oriented interpretation, is **LunarVis** (cf. [20]). The visualisation is given in Figure 5.6. Since 73 is a high number of shells, the layout of an annulus is chosen. This layout supports the readability of hierarchies, it allows a detailed view into the shells' interior and it offers a large area for drawing the edges of the graph. Thus the connectivity of the shells can easily be regarded. The area of the vertices is proportional to the degree of the vertices. Vertices with a high (low) betweenness are coloured red (blue). The lower shells (on the upper left side of the annulus) and the maximum shell (right below the lower shells) are well interconnected. The shell widths and lengths are comparable to each other, that means the number of vertices and edges does not differ considerably from each other regarding different shells. Both degree and betweenness of the vertices increase when the shell number increases. This means the universal products (cf. Chapter 5.3) are in the higher shells. The edges with a large (small) weight are coloured red (turquoise). Large edge weights only occur within the shells. The shells are primarily connected to the maximum shell (the core) and these inter-shell edges possess low weights. Although the number of shells is high the presence of a hierarchy is obvious.

Chapter 6

Clustering Algorithms

In this chapter we will describe several clustering algorithms. We selected these out of a huge number of existing clustering algorithms due to recent research trends. Some of these algorithms are wide spread others feature a remarkable running time but all of them meet the requirements of the considered application area.

6.1 CPM

6.1.1 The Idea

CPM stands for the Clique Percolation Method. Especially the requirement of detecting overlapping communities in large networks can be realised by this method. Within this context communities can be interpreted as units which consist of vertices that are more densely connected to each other than to the rest of the network (cf. [14]). A few further expressions are introduced in [14]:

- A **k-clique** is a subgraph of k vertices which are all connected with each other.
- Two k-cliques are **k-clique adjacent** if they have k-1 vertices in common.
- The combination of a sequence of adjacent k-cliques is called a **k-clique chain**.
- The k-cliques which are members of a k-clique chain are **k-clique-connected**.
- A **k-clique percolation cluster (or component)** is the union of all k-cliques that are k-clique-connected to a certain k-clique.

In the first step any k-clique of the original graph can be selected. In the second step this k-clique is *rolled* to an adjacent k-clique by changing only one vertex and keeping the other k-1 vertices as before. Then this k-clique itself is rolled to another adjacent k-clique. If there is no possibility left to reach unvisited vertices by rolling through adjacent k-cliques a k-clique percolation cluster is found.

In other words clusters which are generated by this method are subgraphs that can be completely explored but cannot be left by rolling k-cliques through these subgraphs. In Figure 6.1 three clusters are derived by rolling a 3-clique through the graph, the red cluster consists of four, the blue cluster of one and the orange cluster of two 3-cliques. At the end the two violet vertices belong to more than one community at the same time.

The structure of a graph influences the sizes of the communities, which are detected by the CPM, extremely. For the case of a random network an analytic expression, at which a *giant component* (made of k-cliques) appears, can be provided (cf. [14]). A giant component is a community, which

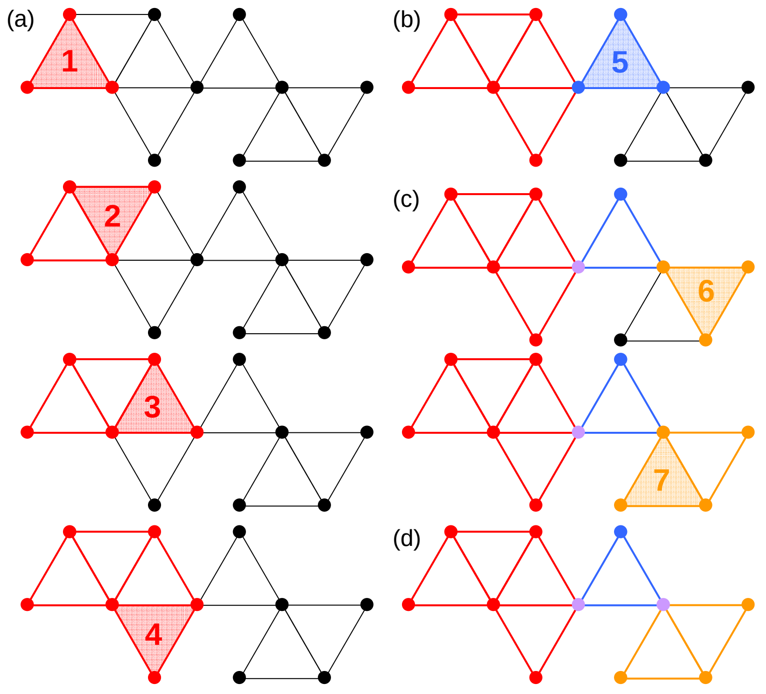


Figure 6.1: Rolling a 3-clique through a graph.

is excessively larger than the other communities. The random network is generated according to the Erdös-Rényi model, that means vertices are connected randomly. An edge between two vertices is included with probability p, independent from the existence of any other edge. The introduced threshold (critical point p_c)

$$p_c(k) = \frac{1}{[(k-1)(|V|-k-1)]^{\frac{1}{k-1}}} \quad \forall k \in \mathbb{N}\backslash\{0,1\} \tag{6.1}$$

depends on the clique-size k. The equation is derived by requiring that the expectation value of the number of adjacent k-cliques, which are candidates to roll *further*, be equal to 1 at $p_c(k)$. An intuitive argument for this requirement is the fact that choosing an expectation value smaller than 1 would result in communities, which are "unfinished". Whereas an expectation value greater than 1 would result in an infinite number of bifurcations for the rolling and so end in a giant component. The expectation value can be written as $(k-1)(|V|-k-1)p_c^{k-1} = 1$, where $k-1$ is the number of candidates for the next relocation, $|V|-k-1$ is the number of possible destinations for the next relocation (the source vertex and the destination vertex of the last step cannot be chosen, since it is desired to roll further). The fraction of acceptable destinations is given by p^{k-1}, since the required $k-1$ edges after the relocation must exist. The equation (6.1) can be simplified for large $|V|$

$$p_c(k) = \frac{1}{[(k-1)|V|]^{\frac{1}{(k-1)}}} \quad \forall k \in \mathbb{N}\backslash\{0,1\} \text{ and } V \text{ large }. \tag{6.2}$$

The introduced threshold helps to avoid the appearance of giant components in random generated networks. This critical point $p_c(k)$ can act as referee for non-random networks as well. Applying the CPM algorithm to a non-random network could also lead to a giant component. The comparison of the critical points in random and non-random networks could show the difference of the graph structures.

6.1.2 Real-World Examples

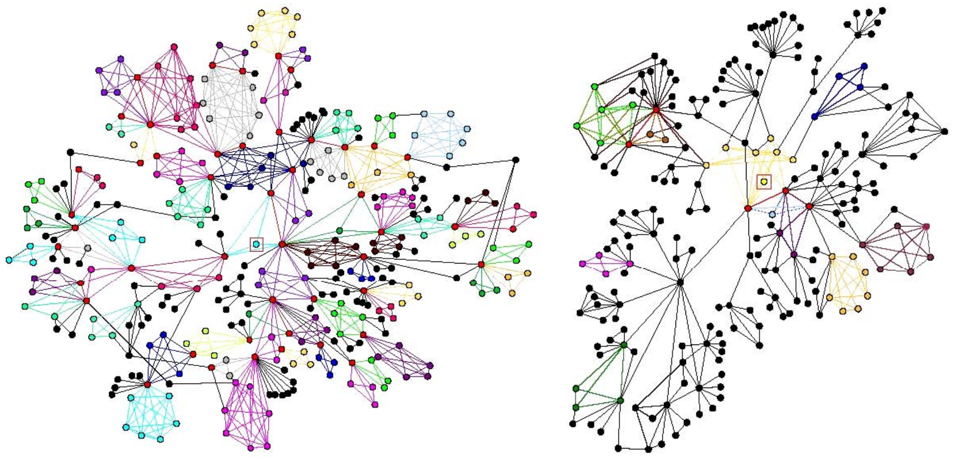


Figure 6.2: Local community structure detected by the CPM in the neighbourhood of a randomly chosen vertex, marked with a red frame (taken from [29]).

The CPM has been applied to two different networks in [29]. The "co-authorship" network is based on the articles in the Los Alamos cond-mat archive and consists of 30 000 authors. The "phone-call" network evolved due to the phone-calls of over 4 million customers of a mobile phone company. The vertices of this network represent the customers.

In Figure 6.2(a) and 6.2(b) the local structure of the neighbourhood of a randomly chosen vertex (marked by a red frame) is given for the two networks. The communities are colour coded. Black vertices and edges are not assigned to a community, red ones belong to more than one community. The co-authorship network is dense and overlaps between communities are significant. In the phone-call network the communities are often separated by inter-community vertices and edges (black).

6.1.3 Algorithmic Implementation

All maximal cliques in a graph can be enumerated with the Bron Kerbosch algorithm (cf. [19] based on [12]) in linear time (relative to the number of cliques). The implementation is efficient because of the *back-tracking method*. This method eliminates multiple solutions without being explicitly examined, by using specific properties of the problem. In case of the Bron Kerbosch algorithm this specific property is the maximal connectedness of a clique. Since the number of cliques could be exponential, the asymptotic running time of this algorithm is exponential as well. In practice however this algorithm is referred to as one of the fastest algorithms, which can enumerate all maximal cliques, and thus it is possible to apply it to relative large graphs.

In Algorithm 5 the clique-clique adjacency matrix B is implemented (line 2 - line 9). $B[i][j]$ represents the number of vertices, which are in the maximal clique i and in the maximal clique j. It is symmetric. The diagonal elements are the sizes of the maximal cliques and are set to 0 , if they are less than k (line 7). All elements, which are not on the diagonal and are less than $k-1$, are set to 0 as well (line 9). If an element is less than $k-1$ this is equivalent to the fact that i and j are not k-clique adjacent. A depth-first-search (DFS) is applied in order to find all connected components (line 12).

Algorithm 5: Finding all k-clique percolation clusters(cf. [19])

```
Input: A set of all maximal cliques $C$ and $k$
Output: All $k$-clique percolation clusters into $P$
1  begin Find-k-CPC($P$, $C$, $k$)
2      $B := 0$;
       // initialise $B$'s elements as 0
       // $M$ is the number of maximal cliques
3      for ($i$ from 1 to $M$) do
4          for ($j$ from 1 to $M$) do
5              $B[i][j] := |C_i \cap C_j|$;
               // # of common vertices of two maximal cliques
6              if $((i = j) \wedge (B[i][j] < k))$ then
7                  $B[i][j] := 0$;
                   // diagonal element < $k$ is replaced by 0
8              if $((i \neq j) \wedge (B[i][j] < k - 1))$ then
9                  $B[i][j] := 0$;
                   // off-diagonal element < $k - 1$ replaced by 0
10     $P := \emptyset$;
11     $i := 1$;
       // initialise output container $P$ and recursion counter $i$
12     DFS($P$, $B$, $i$);
       // DFS to output connected components in $B$ into $P$
13     output $P$;
14 end
15 Find-k-CPC($P$, $C$, $k$);
```

6.1.4 Existence of k-cliques

The existence of a k-clique ($k > 2$) is assured, if the condition of Turán's theorem is fulfilled (cf. [11]). This theorem takes the size of the whole network into account.

> Let $G = (V, E)$ be an undirected graph. If $|E| > \frac{|V|^2}{2} \cdot \frac{k-2}{k-1}$, then there exists a clique of size k within G.

Even for $k = 3$, this condition is not fulfilled for G_{buy} ($|E| = 89590$, $|V| = 5850$, $89590 < \frac{5850^2}{2} \cdot \frac{3-2}{3-1} = 8555625$). Thus there is no guarantee that a k-clique ($k > 2$) exists in this graph.

6.1.5 The Realisation

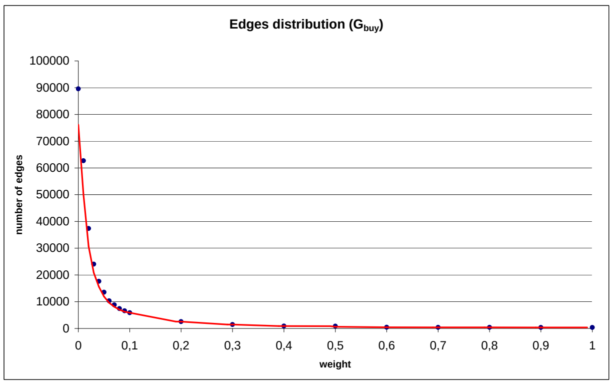
CPM is implemented in a software which is called **CFinder**[1] (cf. [30]).
The input file format of **CFinder** looks like this:

```
vertex1 vertex2 10.3
vertex2 vertex3 3.55
vertex1 vertex3 4.23
```

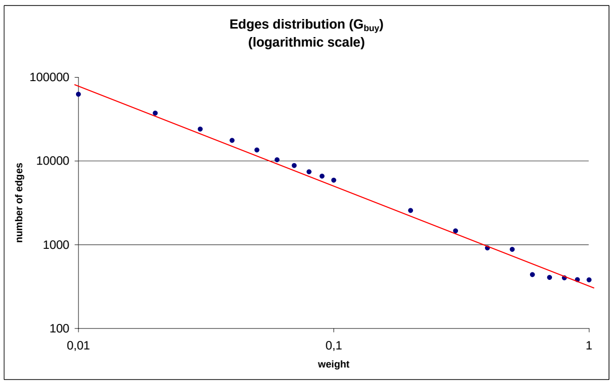
The output files are plain text files.

6.1.6 The Result

The CPM is applied to G_{buy}. For this purpose **CFinder1.21** is used. First of all the distribution of the edge weights is regarded (cf. Figure 6.3(a) and 6.3(b)). The number of edges in the graph is dependent on the applied *weight threshold* w_{min}, which excludes edges with a weight below this threshold. The number of *unfiltered* edges increases with decreasing weight threshold. The unfiltered graph G_{buy}, that means the graph with all edges, is very dense ($|V| = 5850$, $|E| = 89590$). This consideration of the graph is necessary because of the running times given in Table 6.1. The CPM is not applicable to the whole graph with no edge weight threshold, since the algorithm needs more than 30 hours.



(a) normal scale



(b) logarithmic scale

Figure 6.3: Edge weight distribution of G_{buy}.

The CPM is applied to G_{buy} with several edge weight thresholds (cf. Figure 6.4). For each of the thresholds $w_{\text{min}} = 1.0, 0.9, 0.8, 0.7, 0.6$ the clique sizes $k = 3, 4, 5, 6, 7$ appear. For $w_{\text{min}} = 0.5, 0.4, 0.3$

[1]Copyright ©Gergely Palla, Imre Derényi, Illés Farkas, Tamás Vicsek. 2005-2006.

Comparison of the running times				
weight threshold	vertices	edges	detected cliques	running time (hh:mm:ss.ms)
1.0	543	380	35	00:00:00.5
0.9	547	382	35	00:00:00.5
0.8	563	402	41	00:00:00.5
0.7	570	407	42	00:00:00.5
0.6	608	439	47	00:00:00.5
0.5	1 140	880	107	00:00:00.5
0.4	1 175	915	110	00:00:00.5
0.3	1 749	1 463	177	00:00:00.7
0.2	2 674	2 563	315	00:00:01.0
0.1	4 086	5 904	879	00:00:02.1
0.09	4 267	6 593	980	00:00:02.8
0.08	4 461	7 419	1 127	00:00:02.9
0.07	4 711	8 820	1 370	00:00:03.4
0.06	4 964	10 336	1 607	00:00:04.2
0.05	5 256	13 550	2 206	00:00:06.1
0.04	5 476	17 633	3 043	00:00:11.0
0.03	5 614	24 033	4 770	00:00:25.6
0.02	5 708	37 355	11 470	00:02:45.4
0.01[a]	5 788	62 693	106 092	> 16:00:00.0
0.00[a]	5 850	89 590	not applicable	> 30:00:00.0

[a]The command line version of **CFinder** for Linux is used in order to improve the running time.

Table 6.1: Running times of the CPM regarding different weight thresholds.

the clique sizes k range from 3 to 8, for $w_{\text{min}} = 0.2$ the values of k range from 3 to 9 and for $w_{\text{min}} = 0.1$ even from 3 to 10. In Figure 6.4 the *range* and the *average* of the community sizes is visualised for these weight thresholds and occuring values of k. The corresponding table is in the Appendix B (Figure B.1).

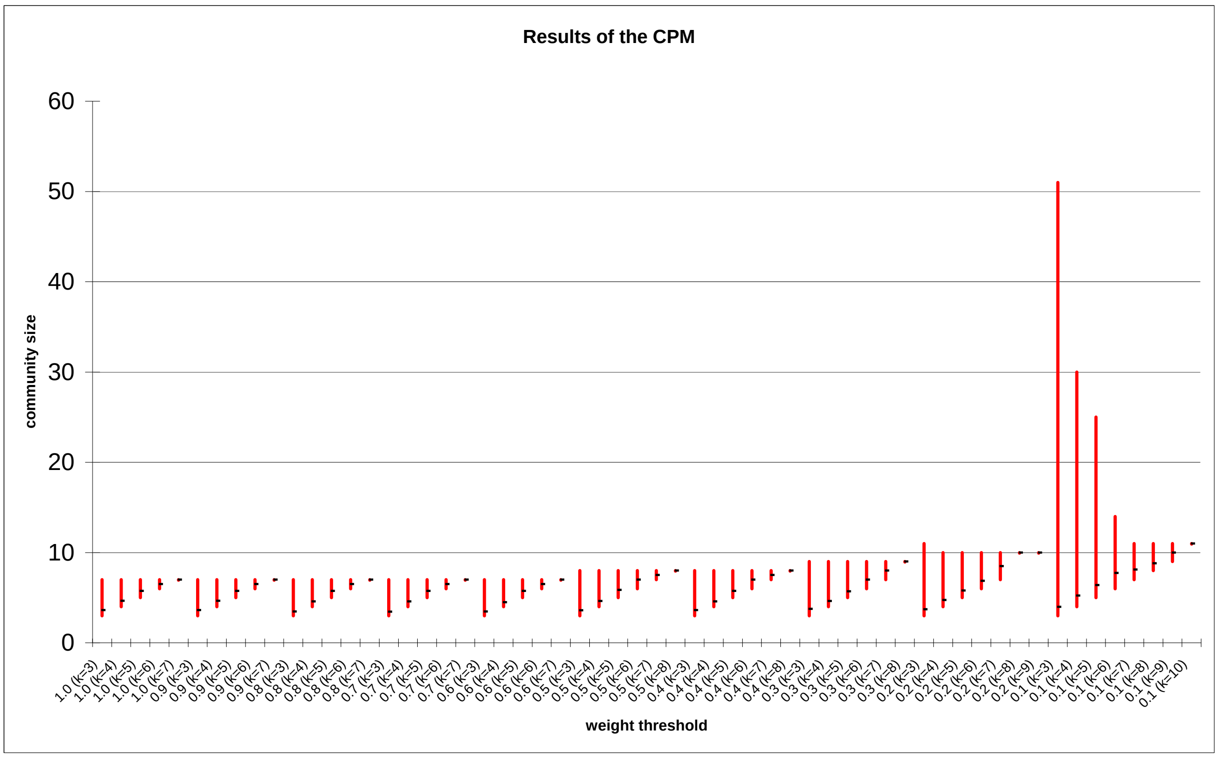


Figure 6.4: Resulting community sizes of the CPM against the given weight threshold and k.

Two of the clusterings are considered more presicely. As shown in Figure 6.4 the range of the community sizes grows for a given k and decreasing weight threshold (increasing number of vertices and edges). The weight threshold $w_{\text{min}} = 0.1$ is the largest threshold which leads to community sizes, which are acceptably large in the context of G_{buy}, at least for $k = 3$. The trend of increasing community size ranges, which is given in Figure 6.4, can be considered for the weight thresholds from 0.09 to 0.02 as well. A smaller weight threshold than $w_{\text{min}} = 0.02$ should not be applied because of the running time (cf. Table 6.1). That is why the thresholds 0.1 and 0.02 are chosen for the consideration in detail. The choice of $k = 3$ is made because of the increasing community size ranges, when k decreases and w_{min} is fixed. Additionally the number of vertices, which are assigned to a community is the largest, if k is the smallest, that means $k = 3$. In Table 6.2 the range and the average community sizes for the chosen weight thresholds and $k = 3$ are given. In the Appendices C - E some tables and diagrams describe the distribution of the community sizes for $w_{\text{min}} = 0.1$ and $w_{\text{min}} = 0.02$ more precisely.

CPM results in detail					
weight threshold	k	cliques	min comm. size	max comm. size	comm. size (average)
0.1	3	579	3	51	3.98
0.02	3	1 131	3	3 342	6.56

Table 6.2: CPM results in detail.

6.1.7 Advantages and Disadvantages of the CPM

Advantages

- Several generalisations are possible (cf. [14]):
  For example it is thinkable that **k-clique connectedness** be defined in a weaker form as follows: Two k-cliques are connected if it's possible to roll from the first k-clique to the second k-clique through $(k - l)$-cliques.
  Another possiblity is the consideration of k-cliques with weighted edges.
- Even unmodified this method is useable for weighted networks to some degree. Therefore the egdes whose weight is smaller than a certain threshold are removed. Afterwards the remaining graph is regarded as unweighted and the CPM is applied.

Disadvantages

- The first impression with real-world networks is that you have to reduce your graph in a crucial way to get an acceptable running time.
- The optimal value of k is not known in advance.
- Without any modification this method only supports binary graphs (that means an edge either exists or not) (cf. [16]).
- If the input graph does not consist of many k-cliques, the CPM is not applicable. The existence of k-cliques can be analysed with Turán's theorem.
- It is possible that one of the detected communities is a giant component (the maximal community size for $w_{\text{min}} = 0.02$ and $k = 3$ is 3342 cf. Appendix C).

6.2 CPMw

6.2.1 The Idea

The CPM which is introduced above is only practicable in the case of unweighted networks. In an unweighted network two vertices are either connected by an edge (if they are neighbours) or vertices are not connected and so there is not an edge which links them. This modelling does not satisfy special demands of real world networks. Therefore an extension of CPM was developed. This method is called *the clique percolation method with weights (CPMw)* (cf. [16]). It allows detecting overlapping communities in weighted graphs. In a weighted graph the vertices are linked with weighted edges. In contrast to the alternative of filtering edges with small weights in a preprocessing step of the CPM the clique percolation method with weights (CPMw) is better adapted to weighted graphs. For this purpose the *intensity* $I(g)$ of a subgraph g is introduced (cf. [28])

$$I(g) = \left(\prod_{(ij) \in l_g} w_{ij} \right)^{\frac{1}{|l_g|}} .$$

The weight between vertex i and vertex j is w_{ij}. Only nonnegative weights are allowed. The variable l_g stands for the edges of g and the absolute value $|l_g|$ for the number of edges in l_g. The intensity $I(g)$ is nothing else than the geometric mean of the edge weights.

A k-clique is only accepted by CPMw if its intensity I is larger than a given threshold. Within a k-clique, $\mathcal{C}$, $\frac{k(k-1)}{2}$ edges between the k vertices exist and so the intensity of a k-clique can be expressed in the following way

$$I(\mathcal{C}) = \left(\prod_{i<j} w_{ij} \right)^{\frac{2}{k(k-1)}}_{i,j \in \mathcal{C}} .$$

This definition leads to a considerably different treatment of weighted edges than the CPM with a preprocessing weight threshold. By setting a threshold for the intensity I in the CPMw, k-cliques are permitted to include edges with a weight below this threshold. Another edge in the same k-clique has to balance this small weight so that the geometric mean of all edge weights of this k-clique is above the threshold for the intensity. The k-clique adjacency in the CPMw is defined like in the CPM. And a weighted network module is the union of all k-cliques that are k-clique connected to a certain k-clique. In this context k-clique connectedness means that it is possible to roll from one k-clique to the other while all the k-cliques of the k-clique chain (it's defined as in the CPM) have intensities greater than I.

6.2.2 Clique Size and Intensity Threshold

A proposal on choosing the parameters k (clique size) and I (intensity threshold) of the CPMw is made in [16].

For each possible k the optimal value of I is determined. This is realised by lowering I until the *critical point* is reached. If I is below this critical point a giant component appears. The optimal value of I is just above this critical point, since this value of I is small enough to involve many k-cliques in the CPMw and high enough to avoid a giant component. Afterwards the parameter k for which the size distribution $p(n_\alpha)$ of the communities is the broadest (at its optimal I) is chosen.

The procedure to derive the optimal value of I (for a given k) in practice is described in the following. The largest community becomes a dominant peak of $p(n_\alpha)$, if the intensity threshold I is below the critical point and consequently a giant component appears. At the beginning the intensity is set to the maximal value of the edge weights $I = \max_{\{i,j\}\in E} w_{ij}$. Then I is lowered until the ratio of the two biggest community sizes $\frac{n_1}{n_2}$ is (for example) 2. This constant makes sure that the two largest communities have *similar* sizes, that means that there does not exist one giant component while all other communities are small-sized.

6.2.3 The Result

The proposal for selecting the parameter I given above, is now used in order to improve the quality of the CPM results for G_{buy} with $w_{\min} = 0.02$ and $k = 3$ (cf. Chapter 6.1.6). The giant component (cf. Table 6.2) with 3342 vertices has to be avoided. In Table 6.3 the procedure of lowering I until the critical point is reached, is shown. The size of the largest community is denoted by n_1 and the size of the second largest by n_2. Hence $I = 0.056975$ is chosen. This approach does not completely correspond to the proposal above: $k = 3$ is set first, since the number of assigned vertices should be large. Then $I = 0.056975$ is set. In the proposal above it is vice versa: first I and then k is chosen.

Selecting I for G_{buy} with $w_{\min} = 0.02$, $k = 3$			
intensity threshold I	n_1	n_2	*ratio* $= \frac{n_1}{n_2}$
0.9	7	6	1.17
0.8	7	6	1.17
0.7	7	6	1.17
0.6	8	7	1.14
0.5	8	7	1.14
0.4	9	9	1.0
0.3	11	10	1.1
0.2	19	19	1.0
0.1	86	85	1.01
0.09	94	89	1.06
0.08	101	99	1.02
0.07	109	104	1.05
0.06	246	195	1.26
0.056975	286	228	1.25
0.056970	504	122	4.13
0.05	606	135	4.49
0.04	1650	57	28.95
0.03	2673	49	54.55
0.02	3307	25	132.28
0.019	3342	25	133.68

Table 6.3: Selection of I for G_{buy} ($w_{\min} = 0.02$, $k = 3$).

The analysed graph G_{buy} with the weight threshold $w_{\min} = 0.02$ and $k = 3$ consists of 5 708 vertices and 37 355 edges (cf. Table 6.1). According to formula (6.2) a giant component (in a random network) appears at the critical point of $p_c(3) = \frac{1}{(2\cdot 5\,708)^{0.5}} \approx 0.0094$. The number of possible edges in the analysed graph is $m_{\text{possible}} = \frac{5\,708\cdot(5\,708-1)}{2} = 16\,287\,778$. In the corresponding random network the number of edges (at the critical point) would be $m_{\text{random}} = p_c(3)\cdot m_{\text{possible}} = 0.0094\cdot 16\,287\,778 \approx 153\,000$. This is much more than the actual number of edges (37 355) in G_{buy}. The intensity threshold

I for G_{buy} with $w_{\text{min}} = 0.02$, $k = 3$ is considerably larger than $p_c(3)$ for the random network ($I = 0.056975 >> 0.0094 = p_c(3)$), although the number of edges in G_{buy} is considerably less than the number of edges in the random network (37 355 << 153 000). The non-random nature of G_{buy}'s graph structure becomes evident.

6.2.4 Advantages and Disadvantages of the CPMw

Advantages

- The consideration of the intensity $I(\mathcal{C})$ of a k-clique $\mathcal{C}$ leads to a more differentiated treatment of weighted edges than the CPM with a preprocessing weight threshold.

Disadvantages

- As in the CPM the optimal value of k is not known in advance.
- If the input graph does not consist of many k-cliques, the CPMw is not applicable either.

6.2.5 Comparison of CPM and CPMw

Communities, which are derived from the CPM, include only edges with a weight higher than the given weight threshold. By contrast communities derived from the CPMw also include edges with a weight smaller than the given intensity.

The results of the methods depend strongly on the input graph (cf. [16]). If edges with a high weight tend to have adjacent edges with a high weight the two methods (CPM and CPMw) derive similar results. The intensity threshold of the CPMw is less rigorous in filtering edges than the weight threshold of the CPM. Thus communities of the CPM are enlarged by the CPMw by adding k-cliques with intensities higher than the given intensity threshold to the communities. These additional k-cliques include edges with a weight smaller than the weight threshold of the CPM.

The results of the two methods differ strongly if edges with a high weight tend to have neighbours with a small weight. This is shown in Figure 6.5 (cf. [16]).

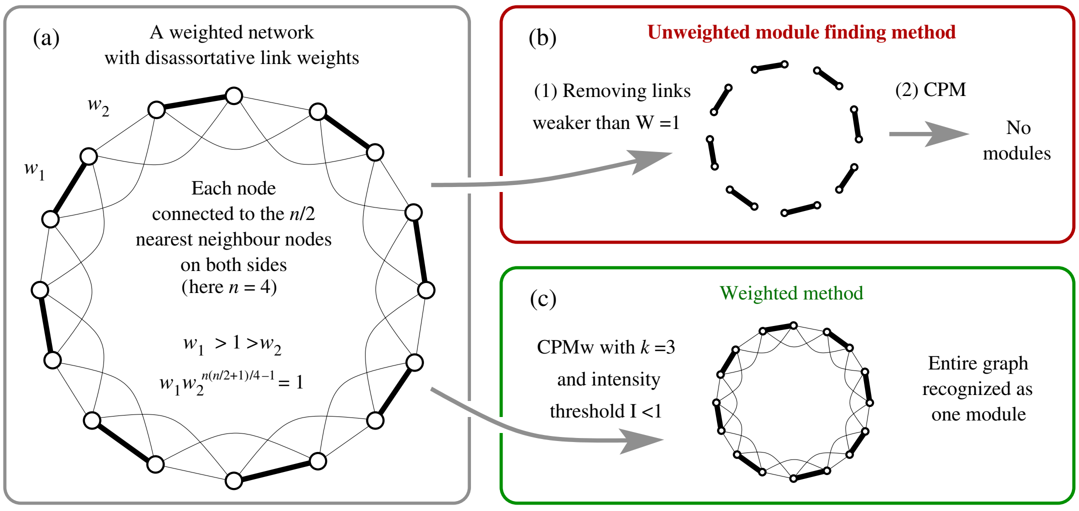

(a)
A weighted network with disassortative link weights
w_2
w_1
Each node connected to the $n/2$ nearest neighbour nodes on both sides (here $n = 4$)
$w_1 > 1 > w_2$
$w_1 w_2^{n(n/2+1)/4-1} = 1$
(b)
Unweighted module finding method
(1) Removing links weaker than W =1
(2) CPM
No modules
(c)
Weighted method
CPMw with k =3 and intensity threshold I <1
Entire graph recognized as one module

Figure 6.5: Comparison of the CPM and the CPMw (taken from [16]).

In this example every edge with a high weight is only connected to edges with a small weight. While the CPM does not find communities, the CPMw considers the whole graph as one community.

6.3 CNM

6.3.1 The Idea

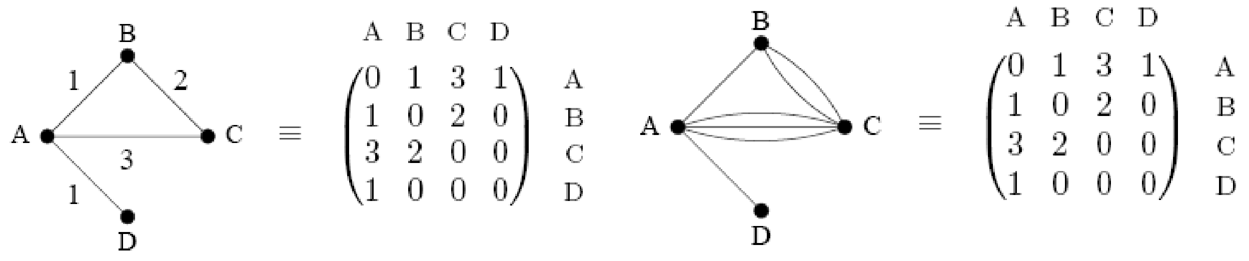
The optimisation of the modularity mod($\mathcal{C}$) (cf. Chapter 4.3) is computationally hard [8]. That is the reason why approximative or at least heuristic optimisation techniques have to be found. One of these is introduced in [13] and is called CNM (Clauset, Newman, Moore). The pseudocode is given in Algorithm 6 (cf. [33]).

Algorithm 6: CNM

```
Input: Graph $G = (V, E)$
Output: Clustering $\mathcal{C}$ of graph $G$
1 $\mathcal{C} := \{v \in V | \{v\}\}$ ;
2 while (true) do
3     $\forall C_i, C_j \in \mathcal{C}$.
4     $\Delta Q^{\mathcal{C}}_{C_i,C_j} := Q(G, \mathcal{C} - C_i - C_j + (C_i \cup C_j)) - Q(G, \mathcal{C})$;
5     Find $(C_i, C_j) \in \mathcal{C}^2$ that has maximum $\Delta Q^{\mathcal{C}}_{C_i,C_j}$.
6     if ( $\max(\Delta Q^{\mathcal{C}}_{C_i,C_j}) < 0$) then
7         break ;
8     $\mathcal{C} := \mathcal{C} - C_i - C_j + (C_i \cup C_j)$ ;
```

This algorithm starts with *singletons*, that means every vertex is the only member of a community. Thus there are as many communities as vertices and every community has size one at the beginning. Subsequently a greedy optimisation of mod($\mathcal{C}$) is applied. The two communities whose amalgamation yields the highest increase in mod($\mathcal{C}$) are merged repeatedly until no further increase in mod($\mathcal{C}$) is possible.

In [13] only unweighted networks are considered, but a proposal for weighted networks is given which refers to [25]. In Figure 6.6 an example is given, in which all weights are integers. According to the weights multiple edges are set, that means in case of $w(e) = 3$ three edges are set. The adjacency matrix remains the same.



$$\equiv \begin{array}{cc} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{pmatrix} 0 & 1 & 3 & 1 \\ 1 & 0 & 2 & 0 \\ 3 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

(a) weighted network

$$\equiv \begin{array}{cc} \begin{array}{cccc} A & B & C & D \end{array} & \\ \begin{pmatrix} 0 & 1 & 3 & 1 \\ 1 & 0 & 2 & 0 \\ 3 & 2 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} & \begin{array}{c} A \\ B \\ C \\ D \end{array} \end{array}$$

(b) multigraph

Figure 6.6: Translation of a weighted network into a multigraph (taken from [25]).

In case of non-integer weights each weight can be divided by the greatest common divisor of all weights. Thus integers are obtained and the modified weights can be treated as above.

However using the weighted case of the modularity formula (cf. equation (4.8)) is the much more simple alternative. The increase of mod($\mathcal{C}$) which can be derived by merging community k and l together is given by ΔQ_{kl} and is initially set to (in case of an undirected graph)

$$\Delta Q_{kl} = \begin{cases} 2\left(\frac{w(e_{kl})}{2m} - \frac{s_k s_l}{(2m)^2}\right) & \text{, if k and l are connected} \\ -2\frac{s_k s_l}{(2m)^2} & \text{, otherwise.} \end{cases} \tag{6.3}$$

The weight of the edge which connects vertex k and l is denoted by $w(e_{kl})$.

After merging community k and l, labeling the combined community l, only the l-th row and column have to be updated and the k-th row and column are removed. The update of ΔQ after merging two communities is made in accordance with the following rule

$$\Delta Q'_{lm} = \Delta Q_{km} + \Delta Q_{lm} \quad m \neq k, l \ . \tag{6.4}$$

This implies that mod($\mathcal{C}$) has only one peak during the algorithm. If the largest value of ΔQ becomes negative, ΔQ can only decrease.

The complexity of the CNM algorithm is given by $O((m+n)n)$ or in case of a sparse graph by $O(n^2)$ (cf. [26]). The number of edges is m and the number of vertices is n.

6.3.2 Real-World Examples

The CNM algorithm analysed the network of Zachary's "karate club" and the network of "American college football teams" in [26]. Zachary's network represents friendships between 34 members of a karate club at a university in the USA. Due to a dispute of one of the instructors and the administrator of the club the group split into two groups. One of these groups started their own club. The CNM algorithm results in two communities with 17 members and achieves a modularity of mod($\mathcal{C}$) $= 0.381$. The shapes of the vertices in Figure 6.7 represent the two groups in real life. The dendrogram shows that almost all vertices (except number 10) are correctly classified by CNM.

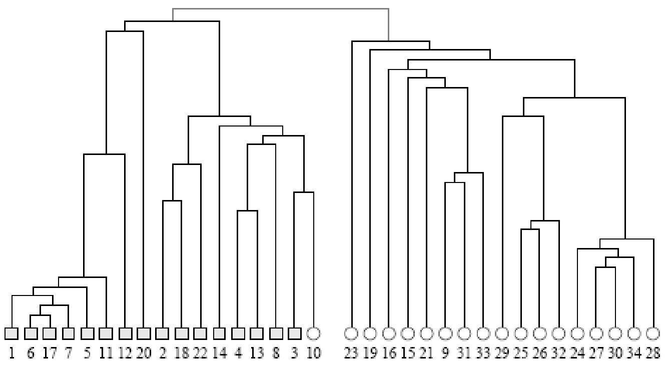


Figure 6.7: Dendrogram of the communities detected by CNM in the "karate club" network of Zachary (taken from [26]).

The network of "American college football teams" represents the schedule of games between the football teams. These 115 teams are devided into 12 "conferences". Intra-conference games occur more often than inter-conference games. The dendrogram is given in Figure 6.8 and has an optimal modularity of mod($\mathcal{C}$) $= 0.546$. The algorithm finds 6 communities. Some of them correspond to real conferences, but most of the communities include vertices of two or more conferences. CNM has missed the structure of this network.

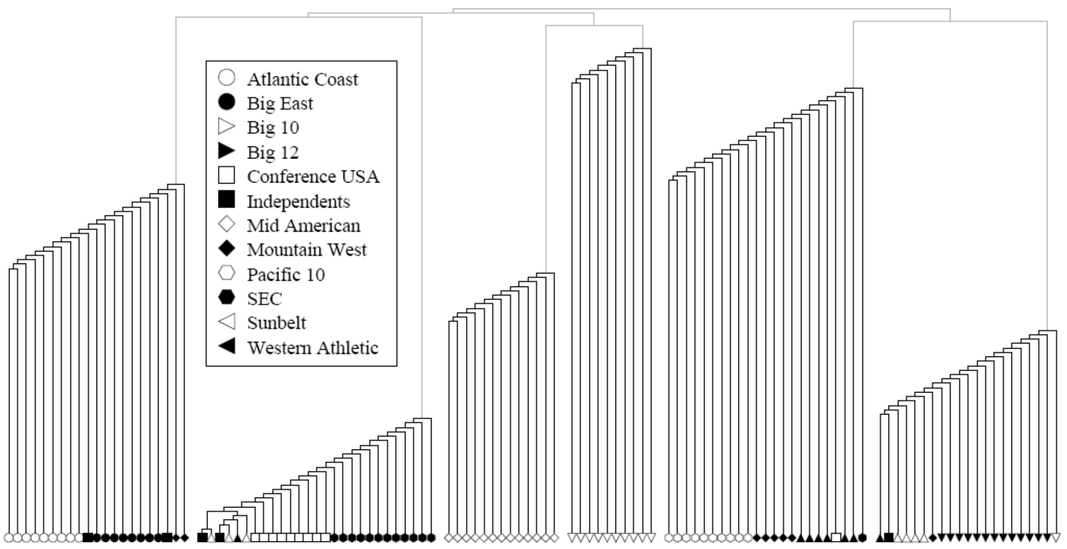


Figure 6.8: Dendrogram of the communities detected by CNM in the network of "American college football teams" (taken from [26]).

6.3.3 The Realisation

The CNM algorithm is implemented in the free software package **igraph**[2] (written in ANSI C) and can be installed as an **R**[3] package. The Pajek format is one of the possible input formats in R. It has the following form

```
*Vertices 5
1 "FirstProduct"
2 "SecondProduct"
3 "ThirdProduct"
4 "FourthProduct"
5 "FifthProduct"
*Edges
1 3 0.2
2 3 0.267
2 4 0.462
2 5 0.287
3 4 0.21
4 5 0.53
```

Vertices are listed with the corresponding labels and edges are represented by the endpoints and the corresponding weight of the edge. In the example given above there exist five vertices and six edges. The weight of the edge between the vertices 1("FirstProduct") and 3("ThirdProduct") is

[2]Copyright ©2007, Gabor Csardi, csardi@rmki.kfki.hu, MTA RMKI, Konkoly-Thege Miklos St. 29-33, Budapest 1121, Hungary, GNU General Public License

[3]Copyright ©1999-2003, R Foundation for Statistical Computing, GNU General Public License, www.r-project.org

0.2 for example. Several additional attributes can be provided by this format, such as colours and coordinates of the vertices. Such attributes are not necessary in this case.
The Pajek format can be derived with the **JUNG**[4] framework and the R code can be described in the following way

```
(01) > library(igraph)

(02) > graph<-read.graph('File.txt',
(03) + format=c('pajek'))

(04) > x <- fastgreedy.community(graph)

(05) > xfastgreedy <- community.to.membership(graph,
(06) + x$merges, steps=which.max(x$modularity))

(07) > for (i in 1:length(xfastgreedy$membership))
(08) + {
(09) + cat (xfastgreedy$membership[i], V(graph)[i-1], "\n",
(10) + file = "OutputFile.txt", +  + append=TRUE)
(11) + }
```

Afterwards the results can be interpreted.

6.3.4 Tuning the Efficiency

The performance can be improved by using efficient data structures (cf. [13]). In the matrix of value ΔQ_{kl} only the elements of pairs k, l are stored which are connected to each other, since the merge of unconnected communities never increases the modularity mod($\mathcal{C}$). Consequently the matrix ΔQ_{kl} is sparse. Each row of the matrix is stored as a balanced binary tree and as a max-heap. A binary tree is a tree data structure in which each node has two children at most. It is balanced if the depth of all leaves differs by at most 1. A max-heap is a specialised tree-based data structure that satisfies the *heap property*: if B is a child of A, then value(A) $\geq$ value(B). The update rule has to be modified.

If community m is connected to k and l:

$$\Delta Q'_{lm} = \Delta Q_{km} + \Delta Q_{lm} \quad . \tag{6.5}$$

If m is connected to k but not to l:

$$\Delta Q'_{lm} = \Delta Q_{km} - 2a_l a_m \quad . \tag{6.6}$$

If m is connected to l but not to k:

$$\Delta Q'_{lm} = \Delta Q_{lm} - 2a_k a_m \quad . \tag{6.7}$$

Additionally an efficient data structure is needed to identify the largest value of ΔQ_{kl}. Therefore the largest value of each row is stored in a max-heap as well. The values $a_l = \frac{s_l}{2m}$ are stored in a vector array. Replacing the balanced binary tree and the max-heap by a doubly-linked list ordered by the community id is even more efficient (cf. [33]).

[4]This software library is freely provided under the BSD open-source license.

6.3.5 The Result

In Table 6.4 the derived cluster sizes are listed. The development of the modularity is shown in Figure 6.9 and more detailed view is in the Appendix F. The achieved modularity is $\operatorname{mod}(\mathcal{C}) = 0.2628$, this is a little less than 0.3, which is according to [13] a reference to significant community structure. The weighted modularity is $\operatorname{mod}_{\mathrm{w}}(\mathcal{C}) = 0.5008$. 5652 merges are executed, that means $5850 - 5652 = 198$ communities are derived. There exist 90 *isolated* communities, thus they are not connected to the main part of the graph. The implementation of igraph stops as soon as all possible merges within the isolated components are executed. Isolated communities are not merged with each other. The information contained in the results of this algorithm are quite useful for the corresponding online shop operator.

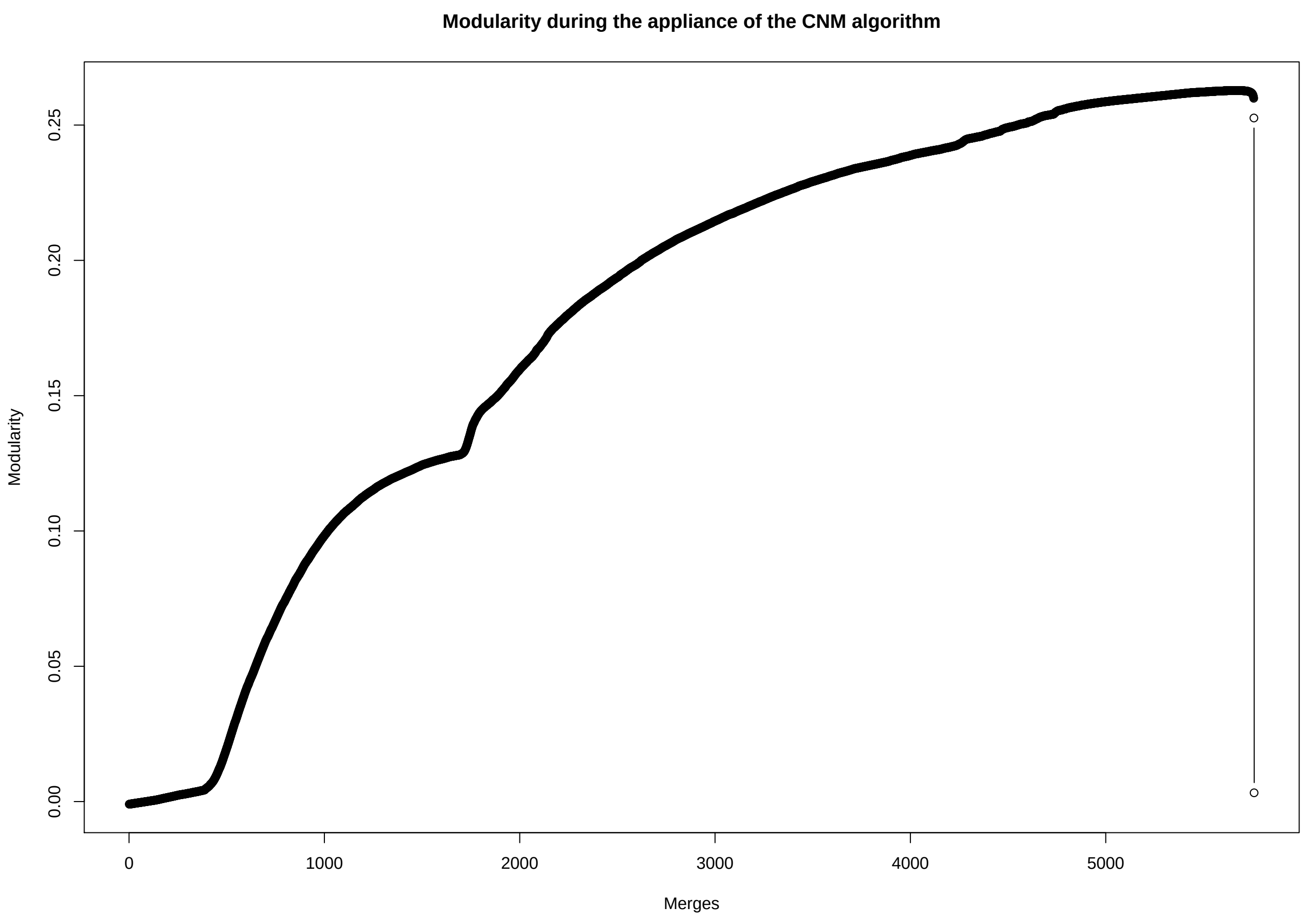


Figure 6.9: Modularity against number of merges in case of G_{buy}.

6.3.6 Advantages and Disadvantages of the CNM Algorithm

Advantages

- The results are useful for an online shop operator.
- With efficient data structures an acceptable running time can be achieved.
- The derived modularity $\operatorname{mod}(\mathcal{C}) = 0.2628$ is near the desired minimum value of 0.3. The number of isolated communities is an explanation for this "relative" small value.

id	cluster sizes														
(1)	3090	1596	173	212	38	55	28	19	23	18	16	16	6	15	6
(16)	11	12	6	6	2	3	7	4	3	11	2	6	3	6	9
(31)	4	6	3	2	7	5	11	5	3	4	2	3	5	5	7
(46)	2	5	4	4	3	8	4	5	2	6	3	2	2	4	4
(61)	2	5	2	7	4	2	3	3	3	4	2	3	3	3	4
(76)	3	4	2	3	2	3	4	3	3	2	3	3	3	3	3
(91)	3	3	3	3	3	3	3	3	3	3	2	2	2	2	2
(106)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(121)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(136)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(151)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(166)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(181)	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
(196)	2	2													

Table 6.4: Cluster sizes derived by the CNM algorithm.

Disadvantages

- As mentioned in [7] the CNM algorithm tends to produce *giant components*, that means the majority of the vertices are in the same cluster. The appliance of the algorithm to G_{buy} confirmed this statement. The maximum cluster size is 3090 (more than 50% of the vertices).
- If the largest value of ΔQ is ambiguous, a tie-breaking solution is needed. A "bad" strategy could deliver an infinite large approximation factor (cf. [10]). This factor is defined as the ratio of the value obtained by this strategy and the real optimal value. Consequently a worst-case tie-breaking strategy could provide arbitrarily bad results.
- Modularity is blindly trusted. But modularity suffers non-locality, scaling behaviour, sensitivity to satellites and a resolution limit (cf. [3], [8]).

6.4 Guillaume

6.4.1 The Idea

In order to get a hierarchical structure another algorithm is introduced in [7]. As the CNM algorithm this algorithm starts with singletons. The optimisation of the modularity $\mathrm{mod}(\mathcal{C})$ is derived in a more "local" way. For each vertex v all neighbour communities C_i are considered and the possible gain of $\mathrm{mod}(\mathcal{C})$ by putting v into C_i, ΔQ, is calculated. For an isolated vertex v the formula is given by

$$\begin{aligned}\Delta Q &= \left[\frac{\sum_{\text{in}}^{C_i} + \deg_{C_i}^{\text{in}}(v)}{|E|} - \left(\frac{\sum_{\text{tot}}^{C_i} + \deg(v)}{2|E|}\right)^2\right] - \left[\frac{\sum_{\text{in}}^{C_i}}{|E|} - \left(\frac{\sum_{\text{tot}}^{C_i}}{2|E|}\right)^2 - \left(\frac{\deg(v)}{2|E|}\right)^2\right] \\ &= \frac{\deg_{C_i}^{\text{in}}(v)}{|E|} - \frac{2 \cdot \sum_{\text{tot}}^{C_i} \cdot \deg(v)}{4|E|^2} \; . \qquad (6.8)\end{aligned}$$

The number of edges whose endpoints are both in C_i is given by $\sum_{\text{in}}^{C_i}$, the number of edges from v to C_i is denoted by $\deg_{C_i}^{\text{in}}(v)$ (all edges are considered as undirected, that means every edge is counted once). The total number of edges is $|E|$, the number of edges incident to vertices in C_i is $\sum_{\text{tot}}^{C_i}$ and the number of edges incident to v is $\deg(v)$.

Vertex v remains in its current community if no increase in $\mathrm{mod}(\mathcal{C})$ is possible. Otherwise it is put in one of the communities C_i with the maximum value of ΔQ. This is done repeatedly over all vertices until no further increase in $\mathrm{mod}(\mathcal{C})$ is possible. After this first phase a new graph is built. The communities of the first phase become the vertices of the second phase. The weights of the edges between the vertices of phase two are built as the sum of the weights of the edges which connected the corresponding communities of phase one. Phase one is reapplied to this newly generated graph. This procedure can be repeated as often as desired or until there are no more changes.

In Algorithm 7 this approach is given as a pseudocode. The clustering depends on the current step ($\mathcal{C}(step)$), this indicates the hierarchical nature of this algorithm. In line 2 the initialising step with singletons is defined as step zero. After step one in line 4 the graph G is transformed for the first time (line 7 - line 12). The communities of step one become the vertices of step two (line 9). The weight of the edge which connects vertex v_{C_i} and vertex v_{C_j} is given by $w_{(v_{C_i}, v_{C_j})}$ (line 12). After this the next step is applied to the transformed graph (G_t) in line 14. In this example 20 steps are desired as can be seen in line 5, that means the resulting hierarchy contains 21 levels (including the initialising level). In Algorithm 8 the procedure `cluster(`$G = (V, E)$`)` is given as a pseudocode as well. The community which contains v is written as $\mathcal{C}(v)$ (line 4). The increase in $\mathrm{mod}(\mathcal{C})$, ΔQ, is calculated for every neighbour community ($C_i \in \mathcal{C}_{neighbour}(v)$) of v (line 6). If all $\Delta Q < 0$, the next vertex is considered (line 9 - line 10). Otherwise vertex v is merged with the neighbour community, which delivers the most increase in $\mathrm{mod}(\mathcal{C})$. This whole procedure is repeated as often as a further increase is possible.

6.4.2 Real-World Examples

Guillaume et al. analysed the "Belgian mobile phone" network in [7]. This network consists of 2.6 million customers. The weights of the edges represent the total number of phone calls during six months. The especialness of this network is the existence of two linguistic communities (French and Dutch). The homogeneity of the communities detected by an algorithm provides a measure for the validity of this algorithm.

Algorithm 7: Algorithm proposed by Guillaume et al [7]

```
Input: Graph $G = (V, E)$
Output: Hierarchical clustering $\mathcal{C}$ of graph $G$
1  $step = 0$ ;
2  $\mathcal{C}(step) := \{v \in V | \{v\}\}$ ;
   // start with singletons
3  $step + +$ ;
4  $\mathcal{C}(step) = cluster(G)$ ;
5  desiredSteps = 20 ;
6  while $((step < desiredSteps)$ and $(\mathcal{C}(step) \neq \mathcal{C}(step - 1)))$ do
       // as often as desired and new communities evolve
7      $G_t = (V_t, E_t) = \emptyset$ ;
8      for every $C_i \in \mathcal{C}(step)$ do
9          $V_t = V_t \cup \{v_{C_i}\}$ ;
           /* communities of the last step become the vertices of the next step */
10     for every $v_{C_i} \in V_t$ do
11         for every $v_{C_j} \in V_t \backslash \{v_{C_i}\}$ do
12             $w_{(v_{C_i}, v_{C_j})} = \sum_{\substack{\{v,w\} \in C_i \times C_j \\ (C_i, C_j) \in \mathcal{C}(step-1)^2}} w_{(v,w)}$;
               // the new weights are built
13     $step + +$ ;
14     $\mathcal{C}(step) = cluster(G_t)$ ;
15     $V = V_t$ ;
16     $E = E_t$ ;
17     $G = G_t$ ;
```

Algorithm 8: cluster($G = (V, E)$) (Guillaume et al [7])

```
   Input: Graph $G = (V, E)$
   Output: Clustering $\mathcal{C}$ of graph $G$
1  $\mathcal{C} := \{v \in V | \{v\}\}$;
   // start with singletons
2  repeat
3      for every vertex $v \in V$ do
4          $\mathcal{C}(v) := \{C_v | v \in C_v\}$ ;
           // $\mathcal{C}(v)$ is the community, which contains $v$
5          for every $C_i \in \mathcal{C}_{neighbour}(v) := \{\bigcup_{C_i \in \mathcal{C}} C_i | (u, v) \in E, u \in C_i\}$ do
               /* $\mathcal{C}_{neighbour}(v)$ is the set of neighbour communities of vertex $v$ */
6              $\Delta Q^{\mathcal{C}}_{C_i,\mathcal{C}(v)} := Q(G, \mathcal{C} - C_i - \mathcal{C}(v) + (C_i \cup \{v\}) + (\mathcal{C}(v) \backslash \{v\})) - Q(G, \mathcal{C})$;
7          Find $C_i \in \mathcal{C}_{neighbour}(v)$ that has maximum $\Delta Q^{\mathcal{C}}_{C_i,\mathcal{C}(v)}$.
8          if ( $\max(\Delta Q^{\mathcal{C}}_{C_i,\mathcal{C}(v)}) < 0$ ) then
9              IncreaseIsPossible=FALSE ;
10             break ;
11         $\mathcal{C} := \mathcal{C} - C_i - \mathcal{C}(v) + (C_i \cup \{v\}) + (\mathcal{C}(v) \backslash \{v\})$ ;
12 until no further increase in Q is possible ;
```

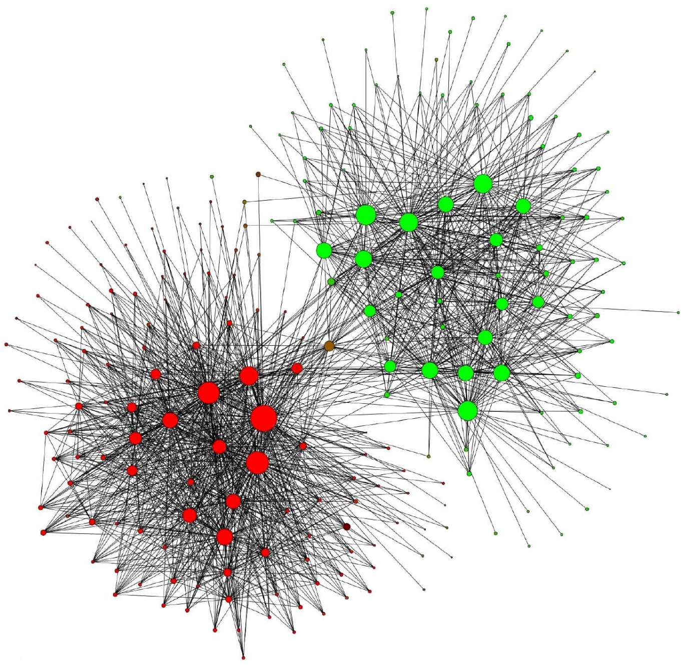


Figure 6.10: Communities detected by the algorithm of Guillaume in the "Belgian mobile phone" network (taken from [7]). The size of a vertex is proportional to the number of customers in the community. The main language spoken in the community is represented by the colour of the vertex (red: French, green: Dutch).

The algorithm of Guillaume detected six hierarchy levels. In the most abstract level exist 261 communities with more than 100 customers (this corresponds to 75% of the customers). Most of these 261 communities are almost monolingual. One community has a balanced distribution of languages and builds the interface between the two language clusters in Figure 6.10 and is therefore mixed-coloured. Only the 261 communities with more than 100 customers are drawn in this figure.

6.4.3 The Realisation

This algorithm is applied to G_{buy} with a program called **community detection**[5]. The latest version can be downloaded at `http://findcommunities.googlepages.com/`. More details about the required system and compiling software can be found in Appendix G. The graph has to be stored in a file, which contains a list of the edges:

```
vertex1 vertex2
vertex2 vertex3
vertex1 vertex3
```

In the first step this text format is converted (command 1). The vertices are renumbered from 0 to $|V|-1$, if they are numbered differently in the input file. Afterwards the communities are computed and the hierarchy is shown (command 2). The number of hierarchical levels and vertices per level is shown with command 3. The clusters of the vertices of a given level can be displayed with command 4.

```
(1) ./convert -i graph.txt -o graph.bin
(2) ./community graph.bin -l -1 > graph.tree
(3) ./hierarchy graph.tree
(4) ./hierarchy graph.tree -l 2 > graph_node2comm_level2
```

6.4.4 The Result

In level 0 every vertex of the graph G_{buy} represents a cluster. The 5850 singletons of level 0 build 446 clusters in the first level, 137 clusters in the second level and 125 clusters in the third level. A local maximum of the modularity of 0.2952 is derived at level 3. This value is almost 0.3, which is according to [13] a reference to significant community structure.

6.4.5 Advantages and Disadvantages of Guillaume

Advantages

- The algorithm is extremely fast because of the rigorous decrease of communities from one step to the next.
- It delivers a hierarchical tree of communities and allows the user to "zoom" into the network.
- The complexity is suggested to be linear on typical and sparse graphs (cf. [6]). This assumption can be made due to the easy computation of the increase in modularity ΔQ. Additionally the number of communities decreases enormously within the first few steps, so the first iterations take the most time.

[5]Based on the article "Fast unfolding of community hierarchies in large networks". Copyright (C) 2008 V. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre. This program or any part of it must not be distributed without prior agreement of the above mentionned authors.

Disadvantages

- Only local properties of the graph are considered during the maximisation process.
- In the current version (date: 07/04/08) weighted networks are not considered.
- Such as in Chapter 6.3 modularity is blindly trusted.

6.5 Fortunato

6.5.1 The Idea

The following algorithm detects both overlapping communities and a hierarchical structure (it is introduced in [24]). Instead of optimising modularity, within this algorithm a *fitness function* is optimised.

$$f_{C_i} = \frac{deg_{in}^{C_i}}{(deg_{in}^{C_i} + deg_{out}^{C_i})^{\alpha}} \tag{6.9}$$

The fitness of community $C_i \in \mathcal{C}$ can be expressed in terms of $deg_{in}^{C_i}$, the total internal degree of the vertices in C_i, and $deg_{out}^{C_i}$, the total external degree of the vertices in C_i, and α, a positive real number, which is a parameter to control the size of the communities. The internal degree is the double of the number of intra-cluster edges (each intra-cluster edge is counted twice, once for each direction), the external degree is the number of edges, which connects each vertex $v \in C_i$ with the rest of the graph (each inter-cluster edge is counted once). The *natural community* of vertex v is the maximal subgraph starting from vertex v, so that neither including another vertex nor excluding a vertex from the subgraph would further increase the fitness of this community. This is a local optimisation, because of the fact, that the global maximum for each vertex is the whole graph. In this case the total external degree $deg_{out}^{C_i} = 0$ and f_{C_i} reaches the largest value, which can be derived for a given α. It is also possible to define a vertex fitness $f_{C_i}^{v}$ of a vertex v.

$$f_{C_i}^{v} = f_{C_i \cup \{v\}} - f_{C_i \setminus \{v\}} \tag{6.10}$$

The fitness $f_{C_i \cup \{v\}}$ is the fitness of the subgraph C_i, which is merged with the vertex v, whereas the fitness $f_{C_i \setminus \{v\}}$ describes the fitness of the subgraph C_i without the vertex v.

The natural community of vertex v is detected by the following procedure. At the beginning C_i contains only vertex v (that means $deg_{in}^{C_i} = 0$). Each iteration is performed in the same manner. For every neighbour vertex w of C_i, which is not part of C_i, the fitness $f_{C_i}^{w}$ is calculated (step 1). The neighbour vertex, which delivers the largest fitness, is added to C_i (step 2). The fitness of each vertex in $C_i' = C_i \cup \{w\}$ is recalculated (step 3). If the fitness of a vertex is negative, it is removed from C_i' and a new subgraph C_i'' is obtained (step 4). If step 4 occurs the procedure is repeated from step 3, otherwise it is repeated from step 1 for C_i''. The whole procedure stops when all neighbour vertices in step 1 have negative fitness. In order to avoid trivial maxima the fitness of each vertex is recalculated each time when a vertex is included or excluded. The algorithm permits the detection of overlapping communities.

6.5.2 Real-World Examples

Some results for real-world networks derived by the algorithm, which has been introduced by Fortunato, are given in [24]. The algorithm introduced by Fortunato detects two hierarchical levels in the "karate club" network of Zachary (cf. Chapter 6.3.2). The most stable partition consists of four communities. Pairwise merging of these four communities results in two overlapping clusters, which are shown in Figure 6.11. The green vertices belong to both communities. The non-overlapping vertices (yellow and blue coloured) are correctly classified compared to the observations of Zachary (represented by the shapes of the vertices).

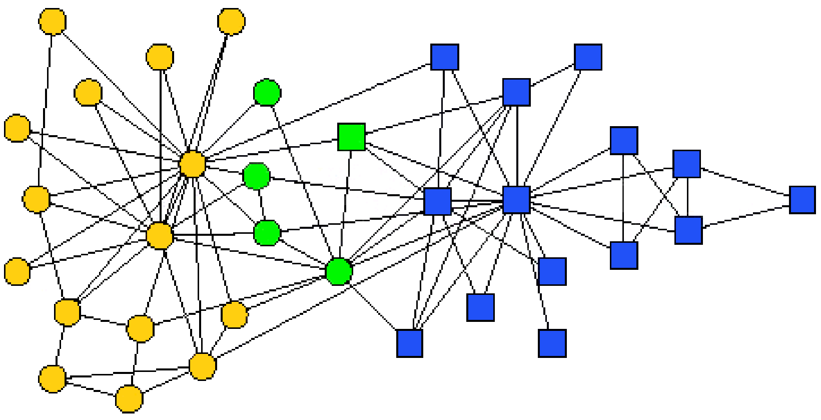


Figure 6.11: Overlapping communities detected by Fortunato in the "karate club" network of Zachary (taken from [24]).

The most stable partition of the network of "American college football teams" (cf. Chapter 6.3.2) consists of 12 groups, which exactly agrees with the conferences. Furthermore no appreciable hierarchy has been detected. Lusseau's network of "bottlenose dolphins" represents the social interactions of bottlenose dolphins in New Zealand. The biologist David Lusseau studied these animals and devided them into two groups according to their age. The result of Fortunato's algorithm in Figure 6.12 agrees with the observations of Lusseau (represented by the shapes of the vertices). Two overlapping communities are detected.

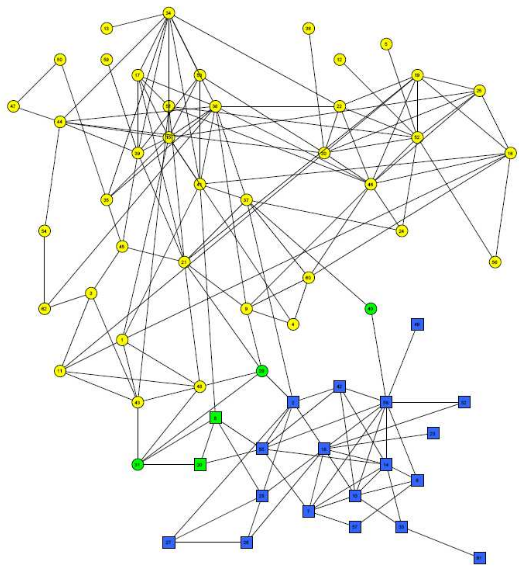


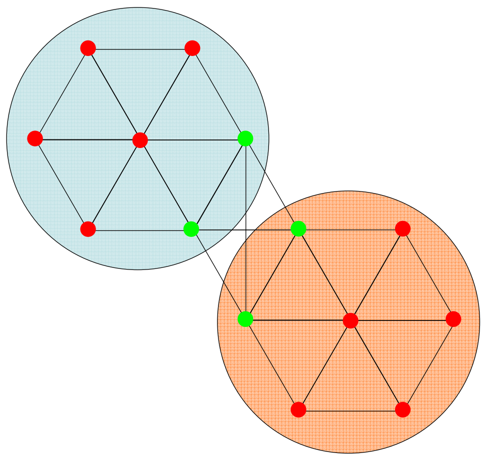
Figure 6.12: Overlapping communities detected by Fortunato in the "bottlenose dolphins" network of Lusseau ([24]).

6.5.3 Improving the Effectiveness

The procedure described in the section above is computationally expensive if it is applied to every vertex. A more economic approach is the following. A vertex v is selected randomly. The natural community of vertex v is explored. Afterwards a vertex w, which is not part of a natural community yet, is chosen randomly. While exploring the natural community of w, all other vertices (no matter if they are part of another community or not) are taken into consideration. After detecting the natural community of w, another vertex, which is not part of a community yet, is selected randomly as before. This is repeated until every vertex is assigned to at least one community.

This improved procedure can be reasoned by the following argument. A vertex v which is part of a community C_i, can either be part of this community C_i and other overlapping communities $C_j, C_k, \ldots$ or it only belongs to C_i. If community C_i is explored by starting from vertex v_1, starting from any other vertex in C_i would either result in the same community or in one of the overlapping communities $C_j, C_k, \ldots$. These overlapping communities can also be explored by starting from a vertex $v_2 \notin C_i$. Thus *almost all* communities can be detected without starting from every vertex. Overlapping communities are derived due to the fact that during the exploration of a new community vertices, which are already assigned, are considered as well.

According to [24] the loss in accuracy by applying this more effective procedure is minimal, considering extensive numerical results. The adumbration of an example for a loss of accuracy is given in Figure 6.13. If one of the red vertices is the starting vertex two communities could be detected, whereas starting from one of the green vertices could result in overlapping communities (this depends on the exact graph structure and the given α). These different results are derived due to the different sequences of the random chosen vertices, which are not assigned yet. Thus the randomness can produce artefacts.


(a) starting from one of the red vertices could result in two detected communities

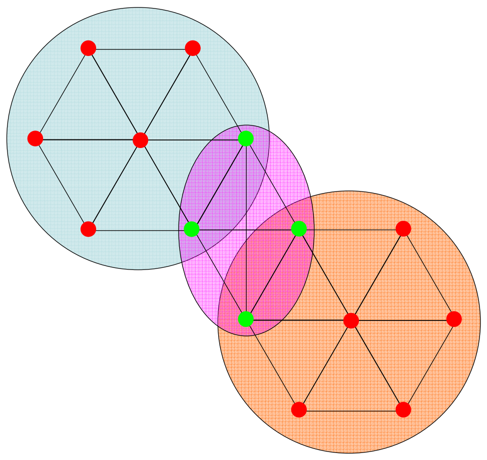

(b) starting from one of the green vertices could result in three detected communities

Figure 6.13: Example of the indeterminate behaviour of Fortunato's algorithm.

This procedure has some degree of stochasticity, that means it is not completely deterministic because of the random choice of the vertices, which are not assigned yet. Thus resulting clusterings can differ from each other for a given α. But nevertheless these clusterings are quite close to each other. In order to choose α in a good manner, several runs should be done to produce a more reliable result (by averaging over the plateau lengths). Discovering an appropriate α is discussed in the following more precisely.

6.5.4 The *Right* Choice of α

The parameter α in equation (6.9) permits to *zoom* through the hierarchy levels. Regarding the clustering for a given α corresponds to a given scale of the graph. Small values of α deliver large communities, large values of α instead deliver small communities. In [24] it is claimed that the choice of $\alpha = 1$ is equivalent to the definition of a community in a weak sense according to [31] (see also chapter 3.2). A counter-example is given in Figure 6.14.

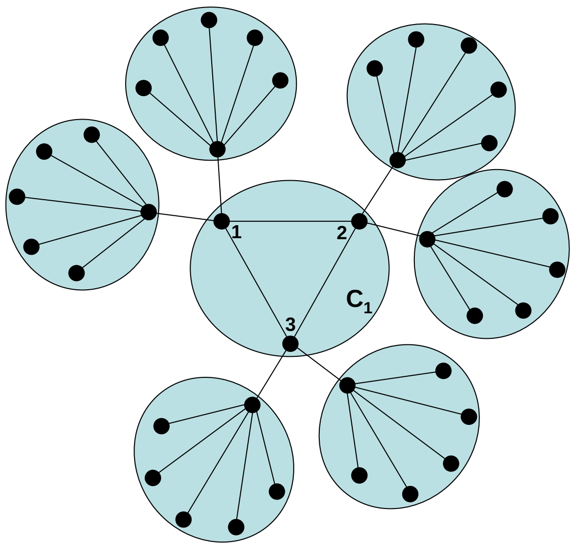


Figure 6.14: C_1 is derived by Fortunato's algorithm ($\alpha = 1$), but it is not a weak community.

If the algorithm is applied to this graph and $\alpha = 1$ is chosen, the result has the following form

```
---------------------------------------------
alpha: 1
number of modules: 7
overlap: 0
homeless nodes: 0
average weak fitness: 0.850649
---------------------------------------------

weak = 0.5 number = 1 overlap = 0
1 2 3

weak = 0.909091 number = 2 overlap = 0
9 4 5 6 7 8

weak = 0.909091 number = 3 overlap = 0
15 10 11 12 13 14

weak = 0.909091 number = 4 overlap = 0
21 16 17 18 19 20

weak = 0.909091 number = 5 overlap = 0
22 23 24 25 26 27

weak = 0.909091 number = 6 overlap = 0
```

```
28 29 30 31 32 33

weak = 0.909091 number = 7 overlap = 0
34 35 36 37 38 39

---------------------------------------------
```

The derived communities are non-overlapping and all vertices are assigned (that means there do not exist homeless vertices). The term "weak" is defined as weak $= \frac{\sum_{i \in G'} deg_{G'}^{in}(i)}{\sum_{i \in G'} deg_{G'}^{in}(i) + \sum_{i \in G'} deg_{G'}^{out}(i)}$. In cluster 1 this ratio is 0.5, that means $\sum_{i \in C_1} deg_{C_1}^{in}(i) = \sum_{i \in C_1} deg_{C_1}^{out}(i)$. Thus the condition of a weak community is not fulfilled.

$$\begin{aligned}
\sum_{i \in C_1} deg_{C_1}^{in}(i) &= deg_{C_1}^{in}(1) + deg_{C_1}^{in}(2) + deg_{C_1}^{in}(3) && (6.11)\\
&= 2 + 2 + 2 && (6.12)\\
&\not> \sum_{i \in C_1} deg_{C_1}^{out}(i) && (6.13)\\
&= deg_{C_1}^{out}(1) + deg_{C_1}^{out}(2) + deg_{C_1}^{out}(3) && (6.14)\\
&= 2 + 2 + 2 = 6 && (6.15)
\end{aligned}$$

Changing the parameter α permits to explore the whole hierarchy of the clusterings. The communities, which are derived for α_1, can be recovered by the communities for $\alpha_2 > \alpha_1$ by merging communities of the α_2-hierarchy-level.

A clustering is said to be *stable*, if it can only be destroyed by changing the value of α considerably. In other words a clustering is derived for an α, which is in some range and if this range is large, the clustering is stable. The average fitness of the communities of a clustering $\mathcal{C}$ can be expressed in the following way

$$\bar{f_\mathcal{C}} = \frac{1}{n_c} \sum_{i=1}^{n_c} f_{C_i}(\alpha = \alpha_1) \ . \qquad (6.16)$$

The number of communities is given by n_c and $f_{C_i}(\alpha = \alpha_1)$ is the fitness of the community C_i for a given value of α. The plot of $\bar{f_\mathcal{C}}$ in relation to α displays several plateaus. The length of these plateaus indicates the ranks of the clusterings. The larger the number of α values used to run the algorithm, the better are the detected hierarchies.

The complete analysis can be carried out quickly, because of the independence of the calculations for different values of α. Thus the calculations can be parallelised on different computers.

6.5.5 The Computational Complexity

The complexity depends on the size of the detected communities and the extent of the overlaps. These characteristics are strongly influenced by the structure of the graph and the value of α. An estimation of the complexity is given in [24] as $O(n_c < s^2 >)$, where n_c is the number of detected communities and $< s^2 >= E(s^2)$ is the second moment of the community size s. In general the k-th moment of a random variable X is the expectation value of the k-th power of X, that means $E(X^k)$. The expectation of s^2 is considered because of the fact, that after every move of a vertex the fitness of the vertices in the community is recalculated. The worst-case complexity amounts to $O(n^2)$ for any single α, where n is the number of vertices, this means the size of the communities is about n.

6.5.6 The Realisation

The algorithm is applied to G_{buy} with a program named **LFM**[6], which is only compilable under Linux. The program performs five loops on α from $\alpha = 0.6$ to $\alpha = 1.6$ with a step of 0.01. That means 505 different clusterings are possible (due to the degree of stochasticity) at most. If the algorithm finds the same clustering for several values of α it means this clustering is more stable. Equivalently the relative occurrence is higher. For $\alpha = 1.6$ (the largest α) the clusters are the smallest, this corresponds to the view of a single vertex. For $\alpha = 0.6$ (the smallest α) the clusters are the largest and this corresponds to the view of the entire graph.

6.5.7 The Result

Applying LFM to G_{buy} leads to 505 clusterings, which have all the same relative occurrence ($\frac{1}{505} \approx 0.00198$). That means no clustering is more stable than the others. This is the result, which would be obtained for a random graph, or a graph whose community structure is not strong enough to let the algorithm find the same clustering more times. Especially for big graphs (more than 1000 vertices) fluctuations can destroy these peaks (cf. an email from Andrea Lancichinetti [arg.lanci@gmail.com] on 07/09/08).

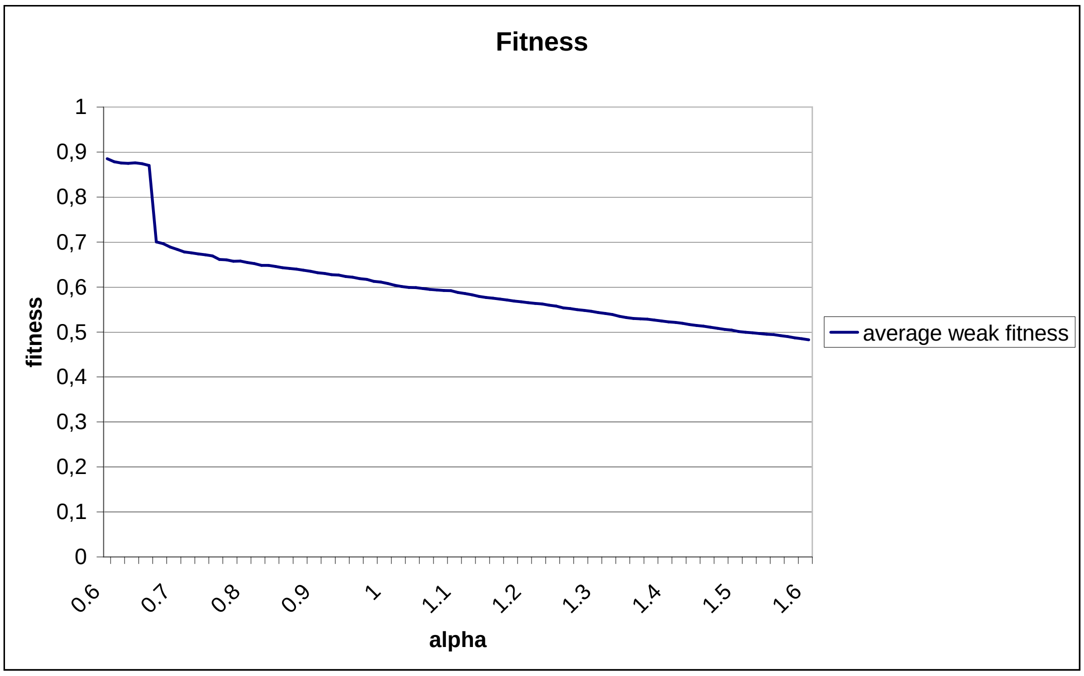


Figure 6.15: The trend of the fitness with respect to α.

In the Appendix H a table with the results is given (for every α the clustering of the run with the largest fitness is listed, that means only 101 clusterings are listed), which are visualised in Figure 6.15 and Figure 6.16. The number of communities is strictly monotonic increasing (when α increases), the number of vertices, which belong to more than one community (overlap) is more or

[6]The program is soon available at `http://santo.fortunato.googlepages.com/inthepress2` (cf. an email from Andrea Lancichinetti [arg.lanci@gmail.com] on 07/07/08) and distributed under the terms of the GNU General Public License

less monotonic decreasing (with one extreme exception between $\alpha = 0.66$ and $\alpha = 0.67$) and the number of homeless vertices (in other words the vertices, which do not belong to any community) is more or less monotonic increasing. The development of all four statistics (number of communities, overlap, homeless vertices, average weak fitness) shows that the clusterings derived for $\alpha = 0.66$ and $\alpha = 0.67$ differ enormously. It is desirable to assign every vertex to a community, since the vertices represent products in the context of this diploma thesis. Assigning a product to a community delivers additional information with respect to this product. Each of the non-assigned vertices should lead to a kind of penalty cost when the derived clustering is judged. The result is not satisfying, since an exploration of the hierarchy structure through stable clusterings is not possible.

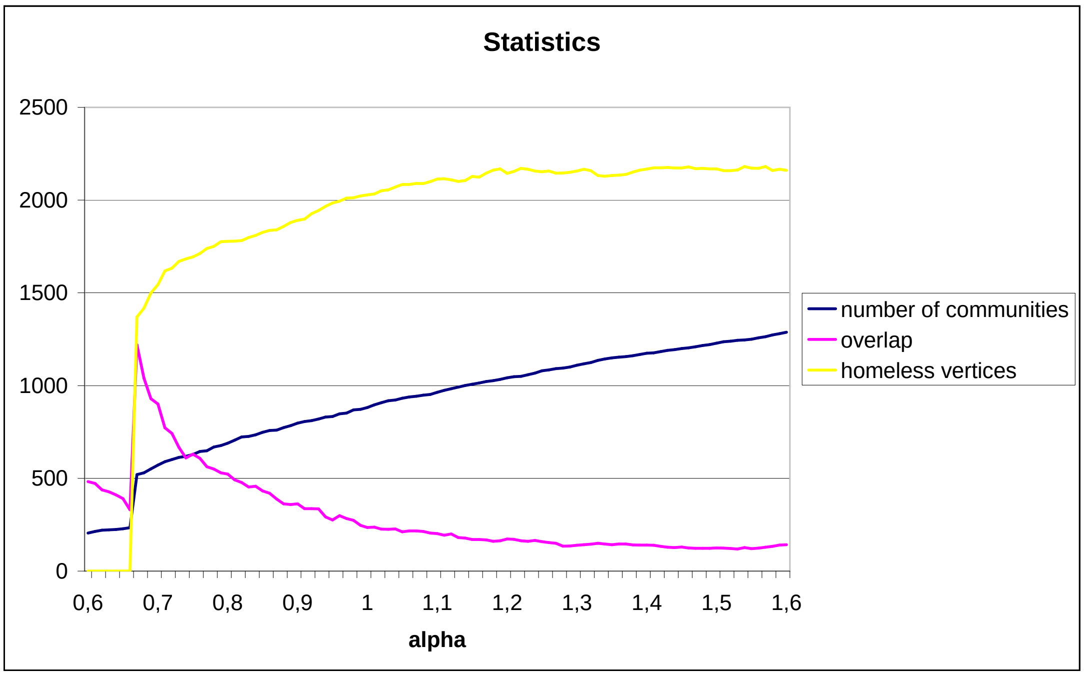


Figure 6.16: Some statistics with respect to α.

6.5.8 Advantages and Disadvantages of Fortunato

Advantages

- The effectiveness of the *original* algorithm in Chapter 6.5.1 can be improved in an easy manner (cf. Chapter 6.5.3).
- The resolution parameter α permits to explore the whole hierarchy of the clusterings.
- Overlapping communities are permitted.

Disadvantages

- For large graphs (more than 1000 vertices) the derivation of stable clusterings is tricky.
- Some degree of stochasticity leads to the fact, that the algorithm is not determinate, and thus indeterministic.

- The number of runs in the program LFM should depend on the number of vertices in the graph to analyse.

Chapter 7

Comparison of the Results

The introduced clustering algorithms deal in different ways with the properties of the graph G_{buy}. In Table 7.1 an overview of the algorithms' features is given.

algorithm	weighted graph	overlaps are permitted	hierarchy is detected
CPM	false	true	false
CPMw	(true)	true	false
CNM	true	false	(true)
Guillaume	false	false	true
Fortunato	true	true	true

Table 7.1: Features of the applied clustering algorithms.

The resulting partitions $\mathcal{C}$ of the vertex set V are judged with regard to the following requirements:

- The optimised quality indices have to be considered, since they are a measure of the partition quality regarding the graph structure.
- Almost every vertex v should be assigned to a cluster C_i (detailed results in Table 7.2). Products, which are represented by a non-assigned vertex, cannot be taken into consideration, when association rules are generated in the next process step (cf. Chapter 2.4).
- Giant components have to be avoided (detailed results in Table 7.2), since the immediate environment of a vertex is of special interest.
- The running time of the algorithm is acceptable. This is especially important, if large graphs are analysed.
- The edge weights are taken into account, that means the graph is considered as weighted (detailed results in Table 7.1). This point of view leads to a different treatment of a high-weighted edge in comparison to a low-weighted edge. By contrast in an unweighted graph every edge is treated in the same way.
- A vertex v can be part of more than one cluster, thus overlaps of clusters are appreciated (detailed results in Table 7.1).
- Detecting hierarchical structures is desirable (detailed results in Table 7.1), since the granularity can play an important role.

The requirements of avoiding giant components and assigning almost every vertex to a cluster are considered in Table 7.2. The running times of the algorithms are given by:

- **CPM(w)** The running time is exponential. According to [1] the software **cfinder** is very efficient for locating cliques of large (at least) sparse graphs and networks with millions of vertices. Nevertheless G_{buy} is too dense for achieving an acceptable running time.
- **CNM** $O(md \log n)$.
- **Guillaume** On typical and sparse graphs the running time is suggested to be linear (cf. [6]).
- **Fortunato** $O(n^2)$.

algorithm	number of clusters	minimal cluster size	maximal cluster size	average cluster size	assigned vertices	overlaps
CPM (min weight 0.02)	1131	3	3342	6.56	4570	2852
CPM (min weight 0.1)	579	3	51	3.98	1957	349
CPMw	1475	3	286	4.96	4008	3301
CNM	198	2	3090	29.55	5850	0
Guillaume						
(level 0)	5850	1	1	1.00	5850	0
(level 1)	446	2	1690	13.12	5850	0
(level 2)	137	2	2383	42.70	5850	0
(level 3)	125	2	2466	46.80	5850	0
Fortunato	234	2	4850	25.54	5849	331

Table 7.2: Statistics of the results for G_{buy}.

7.1 Quality Indices Discussion

The derived values for the results of CNM and Guillaume document that the overlapping formulas generalise the non-overlapping ones (cf. Table 7.3).

7.1.1 Modularity

The calculation of $\text{mod}^{\text{ov}}(\mathcal{C})$ and $\text{mod}^{\text{ov}}_{\text{w}}(\mathcal{C})$ is only applicable for clusterings without a giant component. In the non-overlapping case the second summand of modularity can *easily* be computed (cf. equation (4.7) and (4.8)). Every vertex has to be visited once. But it is expensive to derive the second summand in the overlapping case (cf. equation (4.28) and (4.29): $[-\frac{1}{4|E|^2}\sum_{\{v,w\}\in V\times V} c_{v,w}\cdot \deg(v)\cdot\deg(w)]$ or $[-\frac{1}{4m^2}\sum_{\{v,w\}\in V\times V} c_{v,w}\cdot s_v\cdot s_w]$) due to the large number of vertex combinations $|V|\cdot|V|$, which is $5850\cdot 5850 \approx 3.4\cdot 10^7$ for G_{buy}. It is desirable to reduce these combinations. An efficient implementation is possible, where only the combinations of vertices in a cluster have to be considered. For every cluster C_i there exist $|C_i|\cdot|C_i|$ vertex combinations. If v, w is one of the vertex combinations in C_i, $|\mathcal{C}(v\cdot w)|$ (cf. commonness $c_{v,w}$ in equation (4.27)) is incremented by one and integrated into the second sum. Thus this implementation has to deal with

Comparison of the Quality Indices								
quality index	CPM	CPM	CPMw	CNM	Guillaume			Fortunato
	$w_{\text{min}} = 0.02$	$w_{\text{min}} = 0.1$			level 1	level 2	level 3	
$\text{mod}(\mathcal{C})$				0.2629	0.2848	0.2951	0.2955	
$\text{cov}(\mathcal{C})$				0.7498	0.5603	0.5914	0.5930	
$\text{perf}(\mathcal{C})$				0.6469	0.8442	0.7478	0.7363	
$\text{mod}_{\text{w}}(\mathcal{C})$				0.5008	0.5662	0.5305	0.5253	
$\text{cov}_{\text{w}}(\mathcal{C})$				0.8323	0.7016	0.7557	0.7589	
$\text{perf}_{\text{w}}(\mathcal{C})$				0.6473	0.8449	0.7486	0.7372	
$\text{mod}^{\text{ov}}(\mathcal{C})$		0.0289	0.0075	0.2629	0.2848	0.2951	0.2955	
$\text{cov}^{\text{ov}}(\mathcal{C})$	0.9314	0.0421	0.2023	0.7498	0.5603	0.5914	0.5930	0.9805
$\text{perf}^{\text{ov}}(\mathcal{C})$	0.8319	0.1161	1.5268	0.6469	0.8442	0.7478	0.7363	0.2672
$\text{mod}^{\text{ov}}_{\text{w}}(\mathcal{C})$		0.2591	0.2973	0.5008	0.5662	0.5305	0.5253	
$\text{cov}^{\text{ov}}_{\text{w}}(\mathcal{C})$	0.8955	0.3068	0.5688	0.8323	0.7016	0.7557	0.7589	0.9407
$\text{perf}^{\text{ov}}_{\text{w}}(\mathcal{C})$	0.8317	0.1175	1.5288	0.6473	0.8449	0.7486	0.7372	0.2670

Table 7.3: Quality indices for G_{buy}.

fewer vertex combinations. If a clustering contains a giant component, the described implementation becomes inefficient as well. Therefore $\text{mod}^{\text{ov}}(\mathcal{C})$ and $\text{mod}^{\text{ov}}_{\text{w}}(\mathcal{C})$ for the CPM with $w_{\text{min}} = 0.02$ and Fortunato have not been calculated. But the modularity of a clustering with a giant component is bad (close to 0) anyway.

7.1.2 Coverage

The difference of the alternative coverage formulas (cf. equation (4.30) and (4.31)) and the coverage formulas in equation (4.9) and (4.10) can be observed in case of G_{buy}. Table 7.4 shows that the alternative formulas derive smaller values.

Comparison of the coverage values				
quality index	CPM	CPM	CPMw	Fortunato
	$w_{\text{min}} = 0.02$	$w_{\text{min}} = 0.1$		
$\text{cov}^{\text{ov}}(\mathcal{C})$	0.9314	0.0421	0.2023	0.9805
$\text{cov}^{\text{ov}}_{\text{alternative}}(\mathcal{C})$	0.5209	0.0326	0.0642	0.9641
$\text{cov}^{\text{ov}}_{\text{w}}(\mathcal{C})$	0.8955	0.3068	0.5688	0.9407
$\text{cov}^{\text{ov}}_{\text{w,alternative}}(\mathcal{C})$	0.5356	0.2603	0.3020	0.9099

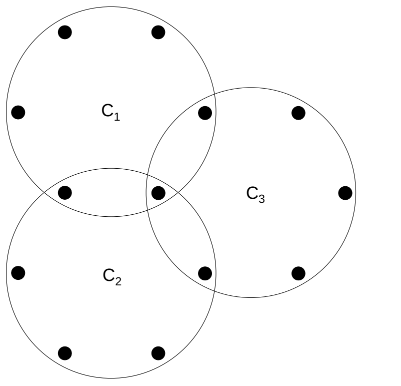
Table 7.4: The alternative coverage formulas derive smaller values.

7.1.3 Performance

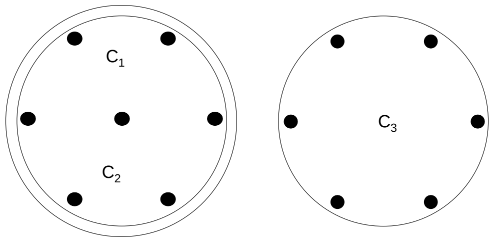
The performance values differ scarcely regarding $\text{perf}^{\text{ov}}(\mathcal{C})$ and $\text{perf}^{\text{ov}}_{\text{w}}(\mathcal{C})$ for the unweighted and the weighted case.

$$
\begin{aligned}
\mathrm{perf}^{\mathrm{ov}}(\mathcal{C}) &= \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{missing inter}}}{|V||V-1|\frac{1}{2}} \\
&= \frac{(\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{missing inter}}) \cdot \bar{w}}{(|V||V-1|\frac{1}{2}) \cdot \bar{w}} \\
&= \frac{(\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{missing inter}}) \cdot \bar{w}}{|E| \cdot \bar{w} + (|V||V-1|\frac{1}{2} - |E|) \cdot \bar{w}} \\
&= \frac{(\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) + m_{\text{missing inter}}) \cdot \bar{w}}{m + (|V||V-1|\frac{1}{2} - |E|) \cdot \bar{w}} \\
&= \frac{\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) \cdot \bar{w} + m_{\text{missing inter}} \cdot \bar{w}}{m + (|V||V-1|\frac{1}{2} - |E|) \cdot \bar{w}}
\end{aligned}
\tag{7.1}
$$

Equation 7.1 shows that $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C})$ differs only in terms of $\sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) \cdot \bar{w}$ from $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C})$. If the average weight of the intra-cluster edges is above the average weight of all edges, that means $\sum_{\{v,w\}\in E} (1_{\mathrm{intra}}(\{v,w\}) \cdot w(\{v,w\})) > \sum_{\{v,w\}\in E} 1_{\mathrm{intra}}(\{v,w\}) \cdot \bar{w}$, $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{w}}(\mathcal{C})$ is larger than $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C})$.



(a) $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C}) = \frac{36}{78}$, $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{approx}}(\mathcal{C}) = \frac{48}{78}$



(b) $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C}) = \frac{42}{78}$, $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{approx}}(\mathcal{C}) = \frac{84}{78} > 1$

Figure 7.1: Comparison of $\mathrm{perf}^{\mathrm{ov}}(\mathcal{C})$ and $\mathrm{perf}^{\mathrm{ov}}_{\mathrm{approx}}(\mathcal{C})$.

The performance values of the CPMw result need to be explained. It is curious that these values are larger than 1. This is due to an approximation, which is used in the implementation. In order to avoid the analysis of the $|V| \times |V|$ matrix (cf. Chapter 7.1.1), the total number of possible inter-cluster edges $m_{\text{inter max}}$ is computed with the help of the possible inter-cluster edges for every combination of two clusters C_i and C_j

$$m^{C_i,C_j}_{\text{possible inter}} = (|C_i| - |C_i \cap C_j|) \cdot (|C_j| - |C_i \cap C_j|) \ . \tag{7.2}$$

The intersection $C_i \cap C_j$ describes vertices, which are assigned to both clusters. The approximation

$$m_{\text{inter max}} \approx \frac{1}{2} \sum_{C_i,C_j \in \mathcal{C}, i \neq j} m^{C_i,C_j}_{\text{possible inter}}$$

is adequate as long as the overlaps are moderate. Since the number of overlaps is large for CPMw (3301), the approximation leads to an oversized number of missing inter-cluster edges, which results in values greater than 1 (cf. Figure 7.1). The performance values for CPM with $w_{\min} = 0.02$ and CPMw are large in comparison to the performance values of CPM with $w_{\min} = 0.1$ and Fortunato (cf. Table 7.3). That is caused by the large number of overlaps in case of CPM with $w_{\min} = 0.02$ and CPMw (cf. Table 7.2).

7.2 CPM and CPMw

In Table 7.2 the role of the weight threshold of the CPM algorithm gets clear. A high threshold results in fewer clusters (579), which are smaller (maximal cluster size is 51, the average size is 3.98), but the number of unassigned vertices ($5850 - 1957 = 3893$) is large. A low threshold leads to more clusters (1131), which are larger (maximal cluster size is 3342, the average size is 6.56). The number of unassigned vertices is considerably less ($5850 - 4570 = 1280$), but a giant component appears.

The CPMw algorithm has been applied to G_{buy} with a weight threshold of 0.02. The intensity threshold has been set to 0.056975 and leads to a less restrictive condition than the weight threshold of 0.02 for the CPM. That is the reason why less than 4570 (as in the CPM with the threshold 0.02) vertices are assigned (in case of the CPMw 4008 vertices are assigned). This results in more unassigned vertices ($5850 - 4008 = 1842$ unassigned vertices in contrast to 1280 unassigned vertices(CPM threshold 0.02)). In comparison to the CPM with the weight threshold 0.1, the number of unassigned vertices in case of the CPMw is less ($1842 << 3893$). The giant component has been avoided (maximal cluster size is 286, the average is 4.96), which is right between the two CPM results. The number of clusters is 1475, which is above both CPM cluster numbers (1131 and 579).

Running time The network analysis of G_{buy} in Chapter 5 showed that the majority of the vertices are part of the same component (cf. Table 5.1). The average degree over all vertices is 30.6284, thus the graph is relatively dense. The betweenness centrality is consistent with the observation retrieved by the degree distribution. The vertices with a high betweenness centrality are probably the ones, which also have high degrees. Another interesting measure in order to understand the running time of the CPM and the CPMw algorithm is the clustering coefficient. The probability that two neighbours of a vertex are also neighbours of each other, corresponds to the probability that these three vertices build a 3-clique. The clustering coefficient of G_{buy} is $0,5773$ and this is a further indicator of the dense structure of the graph. In summary all measures indicate the density of the graph. This property is a hard challenge for the CPM and the CPMw algorithm and explains the immense running time if no constraints have been made (this could be for example a threshold for the edge weights).

Overlaps Both algorithms, CPM and CPMw, allow overlaps between the derived clusters. Although the number of assigned vertices in case of CPMw (4008) is less than the number of assigned vertices of CPM with $w_{\min} = 0.02$ (4570), the number of overlaps is larger ($3301 > 2852$).

Consideration of weighted graphs Applying CPM to a weighted graph means using a threshold for the edge weights. The CPMw takes the edge weights with the help of the intensity of the cliques into account, this is more differentiated. Consequently in both cases the weights are taken into consideration.

Hierarchy As can be seen in Table 7.1 the CPM and the CPMw do not detect hierarchical structures. An approach to get hierarchies out of the cfinder results is introduced in the following. On the one hand k-cliques with a large value of k indicate a strong connection of the components, on the other hand many of the vertices are not assigned to a community (detected by the CPM and CPMw), if the clique size k is large.

Exploring the communities from small k to large k shows that there does not exist a "real" hierarchy. Some of the k-cliques are part of a $k+1$-clique, but not all. If all vertices, which are part of a k-clique, are also part of a $k+1$-clique, a real refinement would take place. Otherwise nothing can be said, because vertices could get lost from one step to the other.

But exploring the communities from large k to small k delivers a solution. A $k+1$-clique always consists of k-cliques. Vertices, which are not part of the $k+1$-clique, but of the k-cliques, could be added from one step to the other. None of the already assigned vertices could get lost. And thus the communities for the largest k ($k_{\max}$), which contain v, are searched in a first step. These communities build the lowest hierarchy level of v. Afterwards for every k below $k_{\max}$ the communities of v are detected and provide the communities of the corresponding hierarchy level of v. The community sizes increase with decreasing k.

In conclusion the results of the CPMw are more appropriate than the results of the CPM. The edge weights are considered more precisely in the CPMw than in the CPM, a giant component can be avoided with the help of the CPMw and the number of assigned vertices is comparable to the number of assigned vertices in the CPM (without an intensity threshold but the same weight threshold).

7.3 CNM

As expected the 91 disconnected components of the graph G_{buy} (cf. Chapter 5.1 (Table 5.1) and Chapter 6.3.5) are detected by this algorithm. The requirement of assigning every vertex to a community is satisfied completely. Unfortunately a giant component appears (more than one half of the vertices (3090 vertices (cf. Table 7.2) build one cluster)).

Running time The running time is applicable.

Consideration of weighted graphs The augmentation of the modularity formula for weighted graphs (cf. equation (4.8)) allows the analysis of weighted graphs.

Overlaps This hierarchical agglomeration algorithm does not allow overlapping communities due to the agglomerative nature of the approach.

Hierarchy The overview in Table 7.1 shows that the CNM algorithm can detect hierarchical structures to some degree. In Figure 6.9 the increase in modularity with each merge of clusters is shown, a more detailed view is given in Appendix F. Each merge, which increases modularity, leads to a new hierarchical level. If the increase in modularity is relatively large, this hierarchical level can be regarded as a significant one. With this artificial approach a hierarchical structure can be obtained, which consists of the significant levels. The most abstract hierarchy level has the maximal modularity. With this approach it is also possible to supervise for example the number or the size of the clusters. All this can only be obtained at the expense of modularity.

Quality Indices The modularity ($\text{mod}(\mathcal{C}) = 0.2629$, $\text{mod}_{\text{w}}(\mathcal{C}) = 0.5008$), which has been optimised by the CNM algorithm, indicates the proper detection of the community structure.

7.4 Guillaume

As the CNM algorithm, this algorithm assures the assignment of every vertex. The only drawback is the relative large maximal cluster size in level 1 (1690, cf. Table 7.2). But this component does not grow considerably in the other levels, in level 2 it contains 2383 vertices and in level 3 it contains 2466 vertices. This is still less than the giant component of the CNM algorithm (3090 vertices). The number of communities is in every level moderate (from 446 in level 1 to 125 in level 3, cf. Table 7.2).

Quality Indices The modularity is calculated for the unweighted graph G_{buy} and is appropriate large ($\text{mod}(\mathcal{C}) = 0.2955$ in level 3). It is notable that $\text{mod}(\mathcal{C})$ increase from level 1 to level 3, whereas $\text{mod}_{\text{w}}(\mathcal{C})$ decreases. But in both cases all modularity values are above those of CNM (cf. Table 7.3).

***k*-core decomposition** The k-core decomposition delivers wide shells (cf. Figure 5.5). The hierarchical levels are hard to detect. The result of the applied algorithm of Guillaume confirms this. Four hierarchical levels are obtained, these correspond to the four most significant ones.

Overlaps The algorithm of Guillaume does not allow overlaps (cf. Table 7.1), thus assigning a product to several communities is not possible.

Consideration of weighted graphs The algorithm, which was introduced by Guillaume et al. in [7] and applied to G_{buy}, has been augmented. The algorithm has been named MAM, a multi-level aggregation method for optimising modularity, and is described in [6]. The advancement, which would be interesting in the context of this diploma thesis, is the possibility to analyse weighted graphs. But only integer weights are allowed and thus MAM is only applicable for the unweighted graph G_{buy}. The latest version can be downloaded at `http://findcommunities.googlepages.com/`[1].

Running time Referring to [6] detecting communities in a graph with 118 million vertices and 1 billion edges took only 152 minutes and the running time is suggested to be linear on sparse graphs. Hence this algorithm is applicable even for very large graphs.

7.5 Fortunato

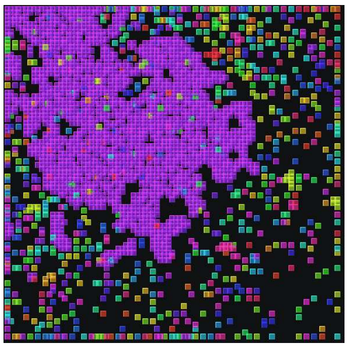
The algorithm, which has been introduced by Fortunato et al., is the only one which allows the consideration of weighted graphs, overlaps and the detection of hierarchical structures. The running time is acceptable. If the number of runs does not depend on the number of vertices as in the LFM program, which is applied, the derived hierarchy levels are not stable and thus not meaningful. Due to the request for assigning almost every vertex of G_{buy}, the clustering, which fulfills this condition and also delivers a large fitness should be chosen. As can be seen in Table 7.2 a giant component (4850 vertices) is the consequence of this choice. The fitness of the chosen clustering is 0.8699, which is good compared to the occurring fitnesses of the other clusterings (cf. Figure 6.15: $\bar{f}_{\mathcal{C}} \in [0.4827, 0.8854]$).

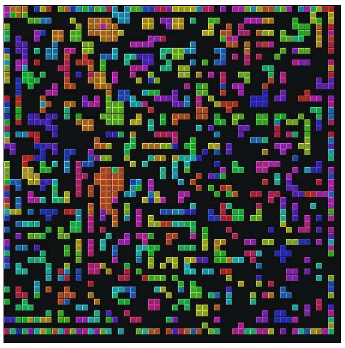
[1]Based on the article "Fast unfolding of community hierarchies in large networks"
Copyright (C) 2008 V. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre
This program or any part of it must not be distributed without prior agreement of the above mentionned authors.

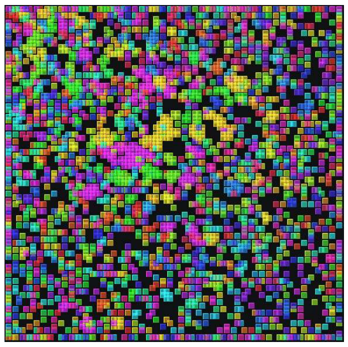
***k*-core decomposition** The algorithm introduced by Fortunato et al. in [24] considers the weighted graph. Due to the extreme connectedness of the vertices (they are arranged very close to each other, cf. Figure 5.5) and the huge number of hierarchical levels (73), these overlapping hierarchical levels are hard to detect. Without further parameter adjustment, the results of the algorithm are not meaningful.

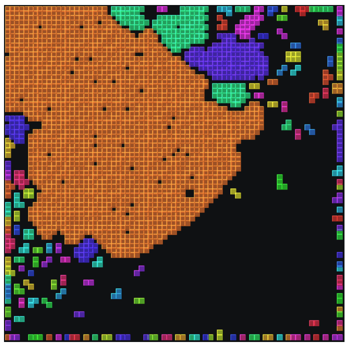
7.6 Visualisations for $G_{\mathbf{buy}}$

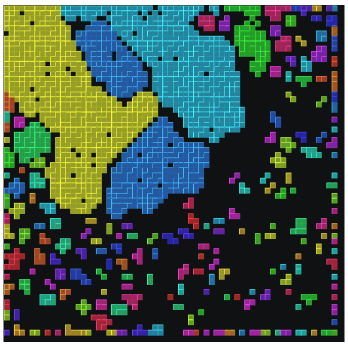
Both results, CPM and CPMw, are fine granular (cf. Figure 7.2(b) - Figure 7.2(c)). The result of CNM corresponds to a more abstract point of view, there are only 198 communities, but the giant component dominates (cf. Figure 7.2(d)). The hierarchy levels of Guillaume (cf. Figure 7.2(e) (level 1) - Figure 7.2(g) (level 3)) are observable. In Figure 7.2(h) the giant component of Fortunato is even larger than the giant component of CNM (cf. Figure 7.2(d)).

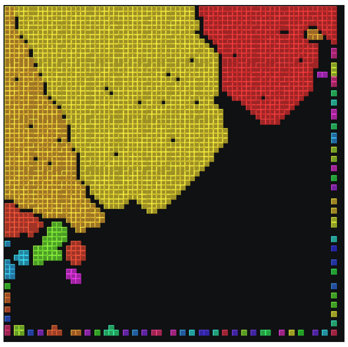

(a) CPM $w_{\text{min}} = 0.02$

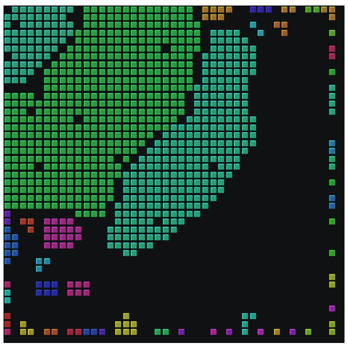

(b) CPM $w_{\text{min}} = 0.1$


(c) CPMw


(d) CNM


(e) Guillaume level 1


(f) Guillaume level 2


(g) Guillaume level 3

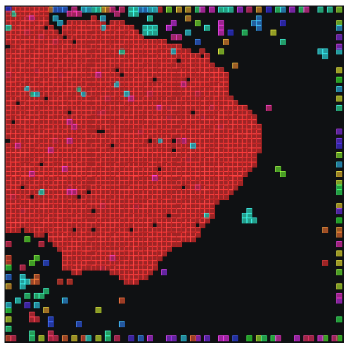

(h) Fortunato

Figure 7.2: Visualisations of the results for G_{buy}.

Chapter 8

Transfer to Another Graph

In order to confirm the validity of the results, which have been derived concerning G_{buy}, another graph is analysed in this chapter. The basis of the new graph is a product portfolio of another online shop, it is denoted by $G_{\text{buy}}^{\text{anotherShop}} = (V_{\text{buy}}^{\text{anotherShop}}, E_{\text{buy}}^{\text{anotherShop}})$. The graph modelling is done in the same manner as in Chapter 3.1 described. The vertices represent the products of the online shop and the edge weights are built as conditional probabilities.

8.1 Network Analysis

8.1.1 Visualisation

The visualisation of $G_{\text{buy}}^{\text{anotherShop}}$ in Appendix I is similar to the illustration of G_{buy} in Appendix A. The graph $G_{\text{buy}}^{\text{anotherShop}}$ is also disconnected. The number of vertices $|V| = 329$ and edges $|E| = 2615$ is much smaller. The chosen layout is organic classic as before, but the preferred edge length is longer and so the vertices are arranged more sparse.

8.1.2 Degree Distribution

The average degree of $G_{\text{buy}}^{\text{anotherShop}}$ is $\frac{2\cdot|E|}{|V|} = 15.8967$, which is much lower than the average degree of G_{buy}. The small number of vertices and the low average degree implicate for example that the running time of the **cfinder** is better for $G_{\text{buy}}^{\text{anotherShop}}$ than for G_{buy}. The degree distribution in Figure 8.1 does not show a logarithmic trend: there are too little vertices with low degree. In general vertices with low degree are *good to handle*, since they build sparse regions of the graph. Clustering algorithms could explore these regions easily. Only a few sparse regions exist in $G_{\text{buy}}^{\text{anotherShop}}$ and consequently the algorithms have to dispense with these regions.

8.1.3 Betweenness Centrality

Considering the visualisation in Appendix I and the betweenness centrality in Figure 8.2 of $G_{\text{buy}}^{\text{anotherShop}}$ shows that the vertices with high betweenness centrality are universal products (cf. Chapter 5.3).

8.1.4 Clustering Coefficients

The clustering coefficients (cf. equation 5.4) of $G_{\text{buy}}^{\text{anotherShop}}$ are shown in Figure 8.3. For 118 vertices it is $c(v) = 1$. These vertices are much more than the 22 vertices, which have degree 1 and consequently $c(v) = 1$. 186 of the 329 vertices possess a value greater than 0.5. These values indicate the

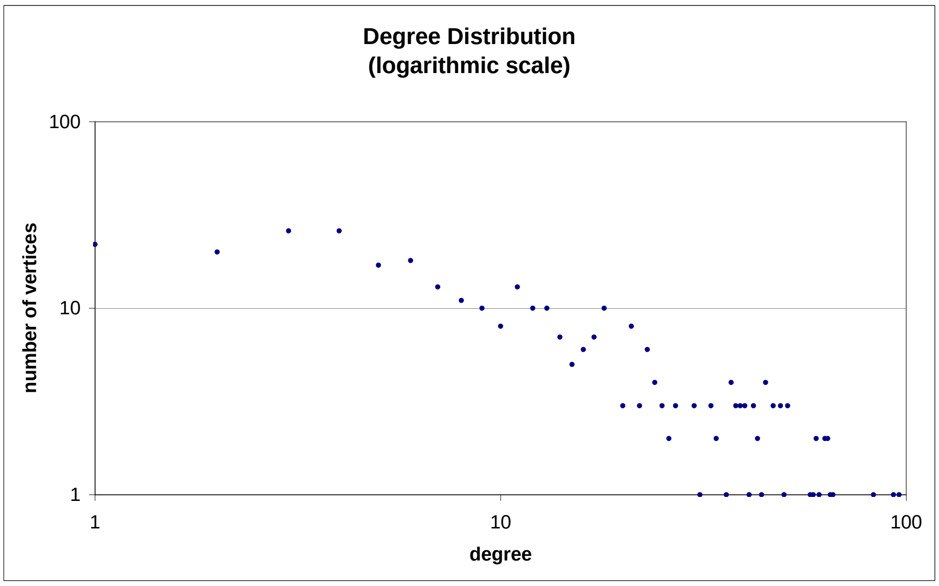


Figure 8.1: Degree distribution of the vertices of $G_{\text{buy}}^{\text{anotherShop}}$.

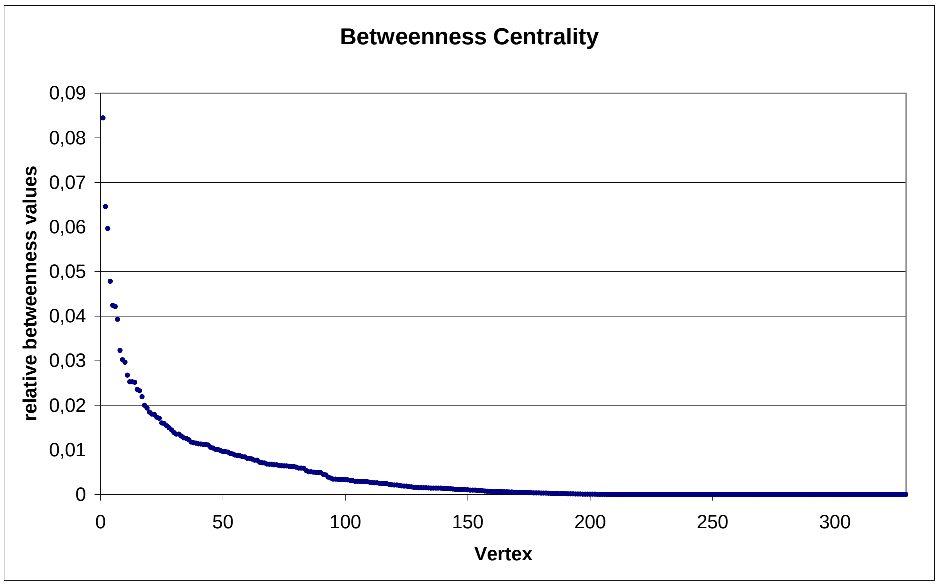


Figure 8.2: Betweenness centrality of the vertices of $G_{\text{buy}}^{\text{anotherShop}}$.

high probability that two neighbours of a vertex are also neighbours of each other. The clustering coefficient of $G_{\text{buy}}^{\text{anotherShop}}$ is 0.6369, this is even larger than in case of G_{buy}. This coefficient is of special interest with regard to the existence of 3-cliques (which is the same as a triangle) in the context of the CPM algorithm. The graph G_{buy} is too dense for the CPM, a weight threshold has to be applied. Although the average degree of $G_{\text{buy}}^{\text{anotherShop}}$ is less than the average degree of G_{buy}, the clustering coefficient is higher. That means $G_{\text{buy}}^{\text{anotherShop}}$ is sparser, but the probability of triangles is higher. Hence there exist *many* 3-cliques in $G_{\text{buy}}^{\text{anotherShop}}$ and this is also an indicator for *many* k-cliques in general. This is a good basis for the CPM(w).

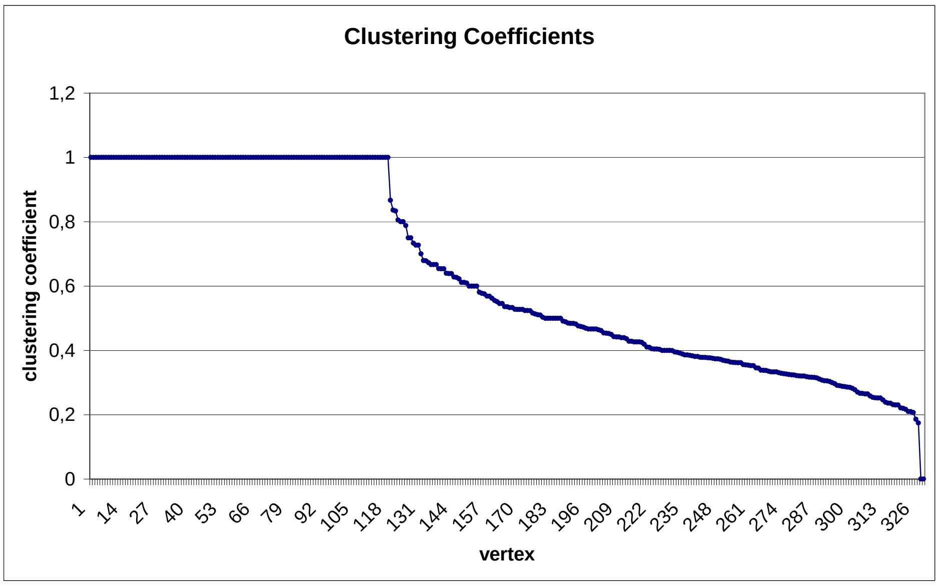


Figure 8.3: Clustering coefficients of the vertices of $G_{\text{buy}}^{\text{anotherShop}}$.

8.1.5 Edge Weight Distribution

Another interesting distribution is the edge weight distribution of $G_{\text{buy}}^{\text{anotherShop}}$ in Figure 8.4. It is far away from a uniform distribution. Hence the steep run of the curve possibly supports the correct classification of a high fraction of vertices by the CNM algorithm, since the edges with a large edge weight are the first considered ones.

8.1.6 k-Core Analysis

The k-core analysis is realised with the tool **LunarVis** (cf. [20]) and the result is shown in Figure 8.5. The 19 shells are arranged in an annulus. Edges with a large (small) weight are coloured red (turquoise). The area of the vertices is proportional to the degree of the vertices. Vertices with a high (low) betweenness are coloured red (blue). The lowest shell is the shell on the left, which only consists of vertices with low degree and low betweenness, while the maximum shell is right below this and consists only of vertices with high degree and betweenness. The shells are connected to each

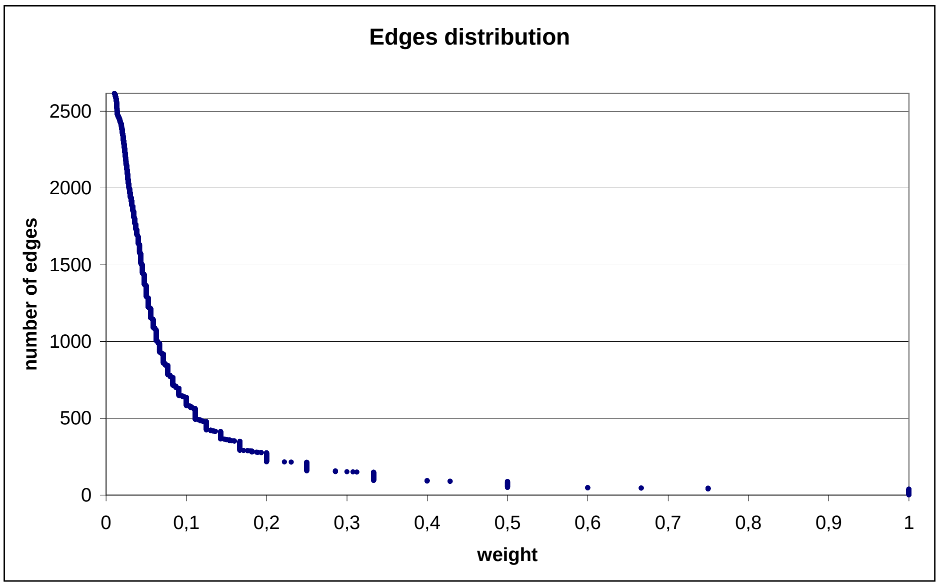


Figure 8.4: Edge weight distribution of $G_{\text{buy}}^{\text{anotherShop}}$.

other by edges with low weights. Note that the maximum shell (the core) is connected to the other shells via edges with larger weights in comparison to the rest of the inter-shell edges. High edge weights appear only within the shells. They are not well interconnected. In general the degree and betweenness of the vertices increases with increasing shell number. Thus the universal products (cf. Chapter 5.3) are in the higher shells. There is also a tendency that vertices with high betweenness are more connected to lower shells, which is shown in the visualisation by the arrangement of these vertices towards the lower shells. Furthermore the shells are arranged well separated from each other. The sizes of the shells are inhomogeneous. In summary this means that the detection of hierarchy levels should be realisable.

8.2 Results

The quality indices of the results for $G_{\text{buy}}^{\text{anotherShop}}$ are evaluated according to the formulas in Table 4.2. The red highlighted values in Table 8.1 are the values of interest for the respective algorithm. In case of Guillaume the more abstract hierarchy level is level 2. Collection 0 of Fortunato is derived for $\alpha = 1.02$ and achieves a fitness of $\bar{f}_{\text{Coll. }0} = 0.6946$ (cf. equation (6.16)). Collection 1 appears at $\alpha = 0.68$ and the fitness is $\bar{f}_{\text{Coll. }1} = 0.8709$. Thus the more abstract level of Fortunato is Collection 1. The hierarchies of Guillaume and Fortunato become clear in Figure 8.6(c), 8.6(d) and Figure 8.6(e), 8.6(f). The visualisation of the CPM result in Figure 8.6(a) shows that most of the vertices are in one cluster. No weight threshold has to be introduced, since the running time is acceptable. In case of G_{buy} the giant component of the CPM with $w_{\text{min}} = 0.02$ has been decomposed with the help of the CPMw with intensity threshold $I = 0.056975$ (cf. Chapter 6.2.3). In case of $G_{\text{buy}}^{\text{anotherShop}}$ a giant component has been tried to be avoided with another procedure. Larger values of k correspond to a higher strictness during the detection of communities and provide smaller communities with a

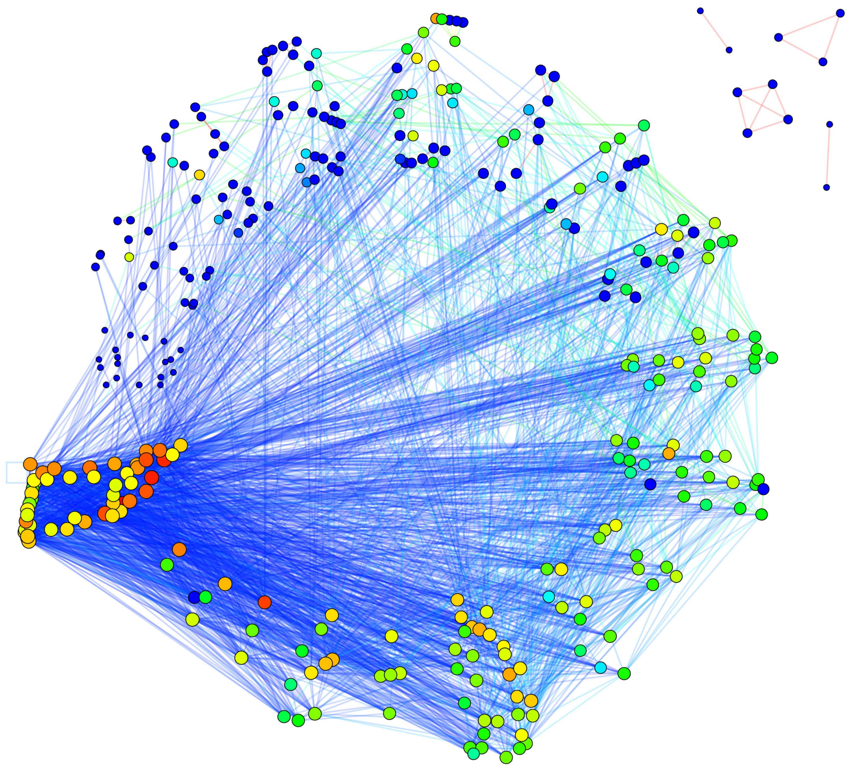


Figure 8.5: k-cores of $G_{\text{buy}}^{\text{anotherShop}}$.

Comparison of the Quality Indices						
quality index	CPM	CNM	Guillaume		Fortunato	
	$k = 6$		level 1	level 2	collection 0	collection 1
$\text{mod}(\mathcal{C})$		0.2676	0.2791	0.3115		
$\text{cov}(\mathcal{C})$		0.5769	0.3784	0.5386		
$\text{perf}(\mathcal{C})$		0.7822	0.9391	0.8511		
$\text{mod}_{\text{w}}(\mathcal{C})$		0.5405	0.5693	0.5823		
$\text{cov}_{\text{w}}(\mathcal{C})$		0.7448	0.6210	0.7258		
$\text{perf}_{\text{w}}(\mathcal{C})$		0.7903	0.9508	0.8601		
$\text{mod}^{\text{ov}}(\mathcal{C})$	0.0963	0.2676	0.2791	0.3115	0.1791	0.0473
$\text{cov}^{\text{ov}}(\mathcal{C})$	0.8027	0.5769	0.3784	0.5386	0.6612	0.9679
$\text{perf}^{\text{ov}}(\mathcal{C})$	0.4693	0.7822	0.9391	0.8511	0.8115	0.2805
$\text{mod}_{\text{w}}^{\text{ov}}(\mathcal{C})$	0.2503	0.5405	0.5693	0.5823	0.4989	0.2929
$\text{cov}_{\text{w}}^{\text{ov}}(\mathcal{C})$	0.6592	0.7448	0.6210	0.7258	0.7108	0.9385
$\text{perf}_{\text{w}}^{\text{ov}}(\mathcal{C})$	0.4624	0.7903	0.9508	0.8601	0.8139	0.2791

Table 8.1: Quality Indices for $G_{\text{buy}}^{\text{anotherShop}}$.

algorithm	number of clusters	minimal cluster size	maximal cluster size	average cluster size	assigned vertices	overlaps
CPM ($k = 6$)	26	6	145	12.08	206	108
CNM	12	2	111	27.42	329	0
Guillaume						
(level 0)	329	1	1	1.00	329	0
(level 1)	38	2	39	8.66	329	0
(level 2)	14	2	75	23.50	329	0
Fortunato						
(Coll. 0)	38	2	141	9.00	300	42
(Coll. 1)	15	2	276	21.73	318	8

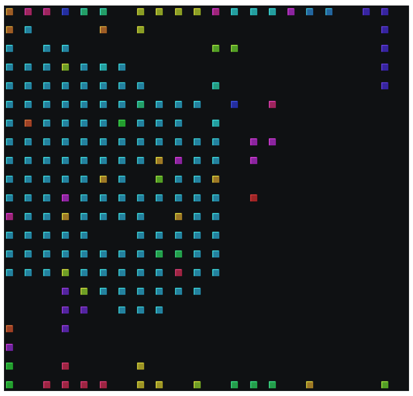
Table 8.2: Statistics of the results for $G_{\text{buy}}^{\text{anotherShop}}$.

higher density inside (cf. [1]). Since the clustering coefficient of $G_{\text{buy}}^{\text{anotherShop}}$ is large, a high value of k is justified. Therefore the largest suggested clique size in [1] $k = 6$ is chosen. Nevertheless a giant component appears. On the contrary the CNM result in Figure 8.6(b) shows the good partition of $G_{\text{buy}}^{\text{anotherShop}}$. Table 8.2 emphasises these observations. The average community size of CPM is only 12.08, but the largest community has size 145. Thus many of the 26 communities are relative small sized. Furthermore only 206 of 329 vertices are assigned (that is one reason why $\text{mod}^{\text{ov}}(\mathcal{C}) = 0.0963$, which is small). Hence the result of the CPM is not appropriate. The good results of CNM can be seen in Table 8.2 as well. There are only 12 communities, the largest has *only* size 111 and the average community size is nevertheless 27.42, which is very good. The advantage of CNM of assigning all vertices is one more argument for CNM. The hierarchy levels of Guillaume and Fortunato consist of almost the same number of communities (Guillaume level1 (level 2): 38 (14), Fortunato Collection 0 (Collection 1): 38 (15) communities). But the maximum community sizes of Guillaume are much lower (level 1: 39, level 2: 75) than in case of Fortunato (Collection 0: 141, Collection 1: 276). This large difference is not only because of the overlaps of Fortunato. Assigning every vertex is another argument for Guillaume.

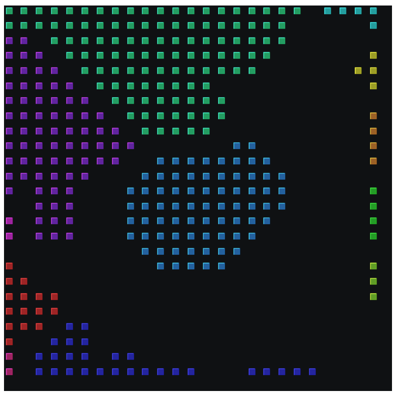
The modularity of Guillaume increases from level 1 ($\text{mod}(\mathcal{C}) = \text{mod}^{\text{ov}}(\mathcal{C}) = 0.2791$) to level 2 ($\text{mod}(\mathcal{C}) = \text{mod}^{\text{ov}}(\mathcal{C}) = 0.3115$). That means the more abstract level achieves a higher modularity. The opposite can be observed for Fortunato's results. The modularity decreases from collection 0 ($\text{mod}^{\text{ov}}(\mathcal{C}) = 0.1791$) to collection 1 ($\text{mod}^{\text{ov}}(\mathcal{C}) = 0.0473$). This is also observable regarding the values of the modularity for the weighted case ($\text{mod}_{\text{w}}^{\text{ov}}(\mathcal{C})$).

Conclusion In summary the result of Guillaume is the most appropriate one for the graph $G_{\text{buy}}^{\text{anotherShop}}$. Both level 1 and level 2 achieve the highest modularity in comparison to the other algorithms (cf. Table 8.1) for the weighted and the unweighted case (level 1: $\text{mod}(\mathcal{C}) = \text{mod}^{\text{ov}}(\mathcal{C}) = 0.2791$ and $\text{mod}_{\text{w}}(\mathcal{C}) = \text{mod}_{\text{w}}^{\text{ov}}(\mathcal{C}) = 0.5693$, level 2: $\text{mod}(\mathcal{C}) = \text{mod}^{\text{ov}}(\mathcal{C}) = 0.3115$ and $\text{mod}_{\text{w}}(\mathcal{C}) = \text{mod}_{\text{w}}^{\text{ov}}(\mathcal{C}) = 0.5823$). Unfortunately Guillaume does not allow overlaps, but on the other hand all vertices are assigned. The number of clusters is in both levels moderate (level 1: 38, level 2: 14, cf. Table 8.2). Furthermore the detected hierarchy is a desired feature.

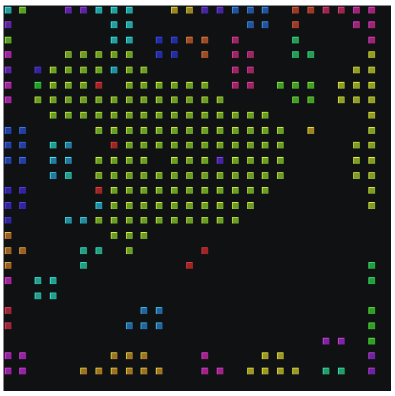
CNM delivers a good result as well (which has been supposed due to the edge weight distribution). The achieved modularity ($\text{mod}(\mathcal{C}) = \text{mod}^{\text{ov}}(\mathcal{C}) = 0.2676$ and $\text{mod}_{\text{w}}(\mathcal{C}) = \text{mod}_{\text{w}}^{\text{ov}}(\mathcal{C}) = 0.5405$, cf. Table 8.1) is comparable to the modularity of Guillaume's level 1. The number of clusters (cf. Table



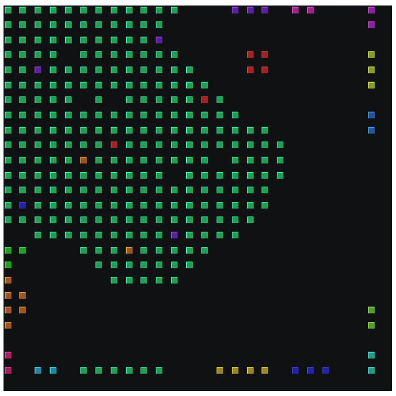
(a) CPM $k = 6$



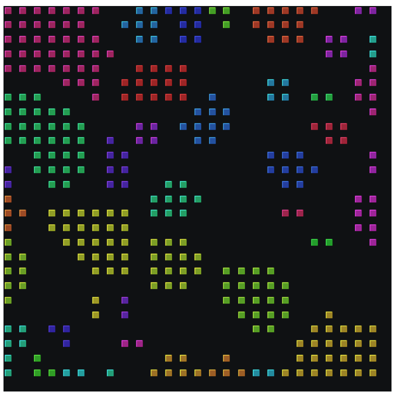
(b) CNM



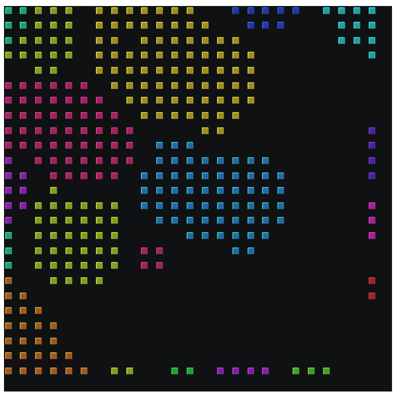
(c) Fortunato coll. 0



(d) Fortunato coll. 1



(e) Guillaume level 1



(f) Guillaume level 2

Figure 8.6: Visualisations of the results for $G_{\text{buy}}^{\text{anotherShop}}$.

8.2) is moderate as well. CNM also assigns all vertices but does not allow overlaps. Since CNM does not support detecting hierarchies, the result of Guillaume is more appropriate.

8.3 Conclusion

Each of the clustering algorithms, which are analysed in this diploma thesis, has advantages and disadvantages. None of them is *universal*. But all of them are of current interest for sciences. The introduction of quality indices, which are valid for the weighted and the overlapping case, makes it possible to evaluate the derived clusterings of the selected algorithms. The product portfolio of every online shop can be represented by a graph and the *most appropriate* algorithm for this online shop can be detected with the help of the quality indices.

8.3.1 Using the Results

The most appropriate clustering of $G_{\text{buy}}^{\text{anotherShop}}$ is taken as the initial point of the preprocessing step for the FP-Growth approach (cf. Chapter 2.4). This is level 2 of Guillaume. One special example product is regarded in order to judge the proposed procedure. Therefore some of the correlation rules, which are derived with FP-Growth, are analysed with respect to the clusters of the products, which are included in these correlation rules. The correlation rules, which have the special example product as antecedent, have almost exclusively products of the same cluster as consequent. The correlation rules with products of another cluster as consequent have small confidence values. Thus the background knowledge derived by this procedure is temperate. But the large number of products of the same cluster in the correlation rules emphasises the correct classification (according to FP-Growth) of the products derived by the clustering algorithm. The products of a cluster *fit* to one another.

8.3.2 Semantics in the Detected Clusters

During the preparation of this diploma thesis, the non-overlapping clusters turned out to be more comprehensible. The evident structure of the non-overlapping clusterings is more suitable for example for visualising the product groups of an online shop. Guillaume and CNM could detect clusters like *gift accessories*, *European championship packages*, *product sets* and others in a very good manner. Besides, flavoures of a special product were also assigned to one cluster for example. Apart from that Guillaume achieves a good compromise of diversification and unification (cf. Figure 8.6(e) for instance).

One future perspective could be the combination of a cluster with an action. That means if a customer buys a product of the *children's toys* cluster, it could be appropriate to advert to the new test report of "Stiftung Warentest" concerning the security of children's toys.

8.3.3 Next Steps

In a next step the results of this diploma thesis are integrated into the process of generating correlation rules. So the preprocessing step of FP-Growth can be used in the live operation. If a customers of an online shop is interested in a special product, FP-Growth is not applied to the set of all products, but to the set, which is restricted to the products, which are assigned to the cluster of the product, which is interesting for the customer. Consequently only products of the same cluster are recommended to the customer. The live operation makes a conclusive judgement possible.

Chapter 9

Implementation Chapter

In this chapter the implementations, which have been required in the course of this diploma thesis, are described. The software library **JUNG**[1] is used.

Graph Creator The vertices and edges are read out of the database. The *buycount* of a vertex describes how often the product, which is represented by this vertex, has been bought. Every vertex is furnished with this buycount. An edge is furnished with the number of customers, which have bought the two products represented by the endpoints of the edge. Afterwards these numbers are used to calculate the edge weights. The vertices are labelled with the corresponding product id. The possibility to *slim* the graph is realised because of the required weight threshold for the CPM(w). Only edges with a weight greater than this threshold are allowed.

Network Analysis The analysed properties of the generated graph (for example: degree distribution, betweenness centrality, clustering coefficient) are realised with the already implemented classes of the JUNG library. Subsequently the given results have to be interpreted with a spreadsheet analysis tool.

Graph Formats The generated graph has to be exported in several formats. The Pajek format is already implemented in JUNG, but other formats like a list of weighted or unweighted edges had to be realised as well.

After this the clustering algorithms are applied with the help of several softwares.

- CPM(w): **cfinder**[2]
- CNM: R package **igraph**[3]
- Guillaume: **community detection**[4]
- Fortunato: **LFM**[5]

[1]This software library is freely provided under the BSD open-source license.

[2]Copyright ©Gergely Palla, Imre Derényi, Illés Farkas, Tamás Vicsek. 2005-2006.

[3]Copyright ©2007, Gabor Csardi, csardi@rmki.kfki.hu, MTA RMKI, Konkoly-Thege Miklos St. 29-33, Budapest 1121, Hungary, GNU General Public License

[4]Based on the article "Fast unfolding of community hierarchies in large networks"
Copyright (C) 2008 V. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre
This program or any part of it must not be distributed without prior agreement of the above mentionned authors.

[5]The program is soon available at `http://santo.fortunato.googlepages.com/inthepress2` and distributed under the terms of the GNU General Public License

Transformer The result files of the applied clustering algorithms differ from each (due to the different softwares). They must be transformed into a standardised format:

```
cluster id
0 2
0 4
0 6
1 1
1 5
2 3
2 7
```

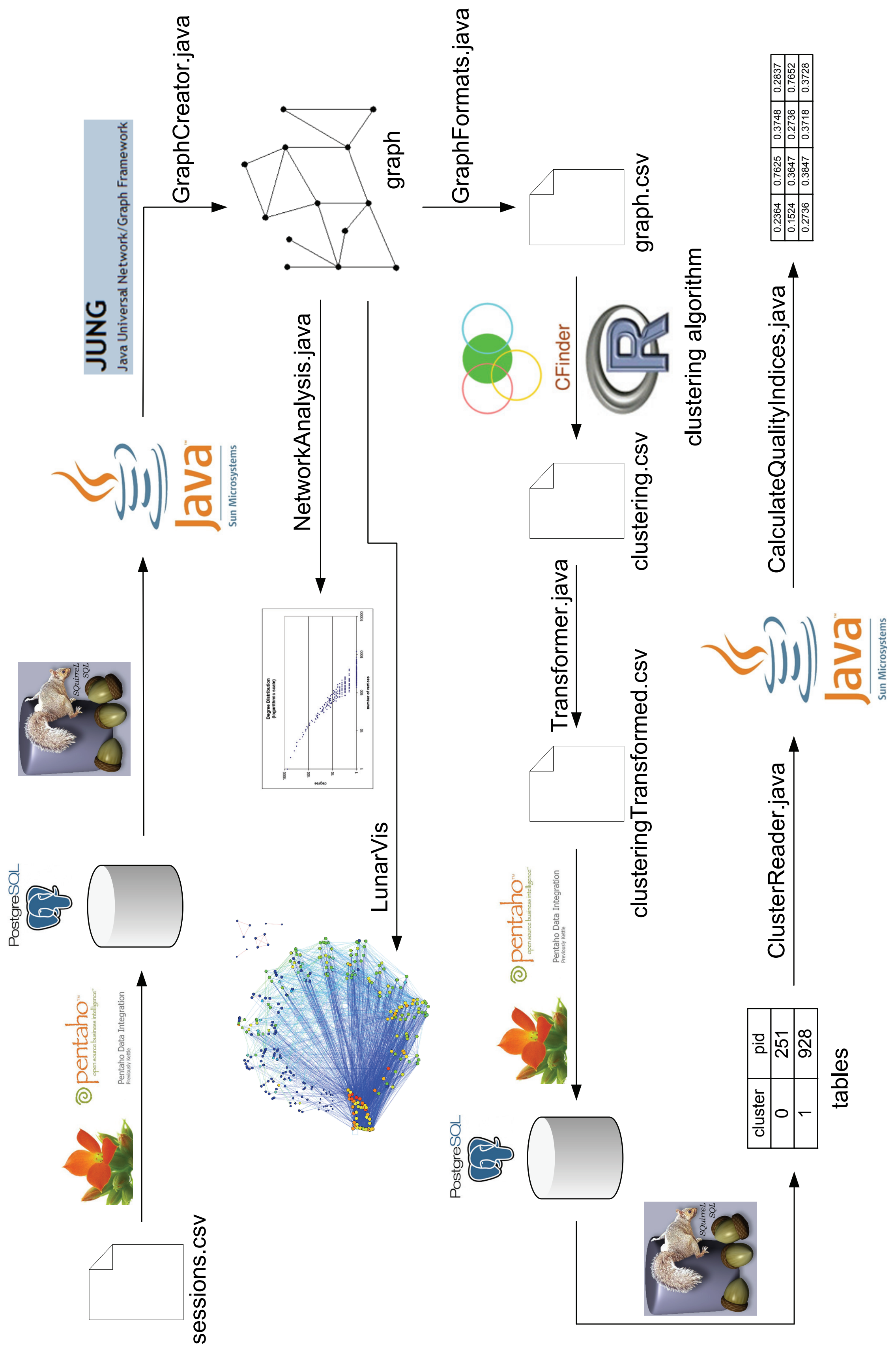
The output formats of **cfinder** and **LFM** must be transformed. The result of the CNM algorithm is reconciled with the standardised format by means of an **R** script.

Cluster Reader The result files in the standardised format are imported to JUNG.

Quality Functions The formulas given in Table 4.2 are implemented and the imported clusterings (cf. Cluster Reader) could be regarded with respect to these quality indices.

In Figure 9.1 an overview of the analysis procedure is given. As a first step the shopping cart data, given in the `sessions.csv` file, is written into a **PostgreSQL**[6] database. This is done with the **Pentaho Data Integration**[7] tool, which delivers many extraction, transformation and loading (ETL) capabilities. Afterwards the data is prepared for the next step with **SQuirreL SQL**[8], an open-source Java SQL Client program. With the help of the software library **JUNG**[9] a graph is created in **Java**. This graph is the initial point for further steps. The network analysis is realised in **Java**, the k-core analysis is made with **LunarVis**. The graph is stored in several formats (`graph.csv`): the Pajek format, a list of the weighted edges and a list of the unweighted edges. These files are the input for the clustering algorithms. Applying the different clustering algorithms to the graph required several software tools: **cfinder**, **R**, **community detection** and **LFM**. The output of the algorithms (`clustering.csv`) is transformed in **Java** into `clusteringTransformed.csv`, if it is necessary (cf. **cfinder**, **LFM**). The results are written into the **PostgreSQL** database with the help of the **Pentaho Data Integration** tool. With SQL the results are processed in such a way that they become interpretable and visualisable. In **Java** the visualisation and the evaluation of the quality indices is realised as a last step.

[6]PostgreSQL is released under the BSD license.
[7]The Pentaho BI Platform is distributed under the terms of the GNU General Public License Version 2.
[8]SQuirrel is released under the GNU Lesser General Public License.
[9]This software library is freely provided under the BSD open-source license.



cluster	pid
0	251
1	928

0.2364	0.7625	0.3748	0.2837
0.1524	0.3647	0.2736	0.7652
0.2736	0.3847	0.3718	0.3728

Figure 9.1: Analysis Procedure.

Appendix A

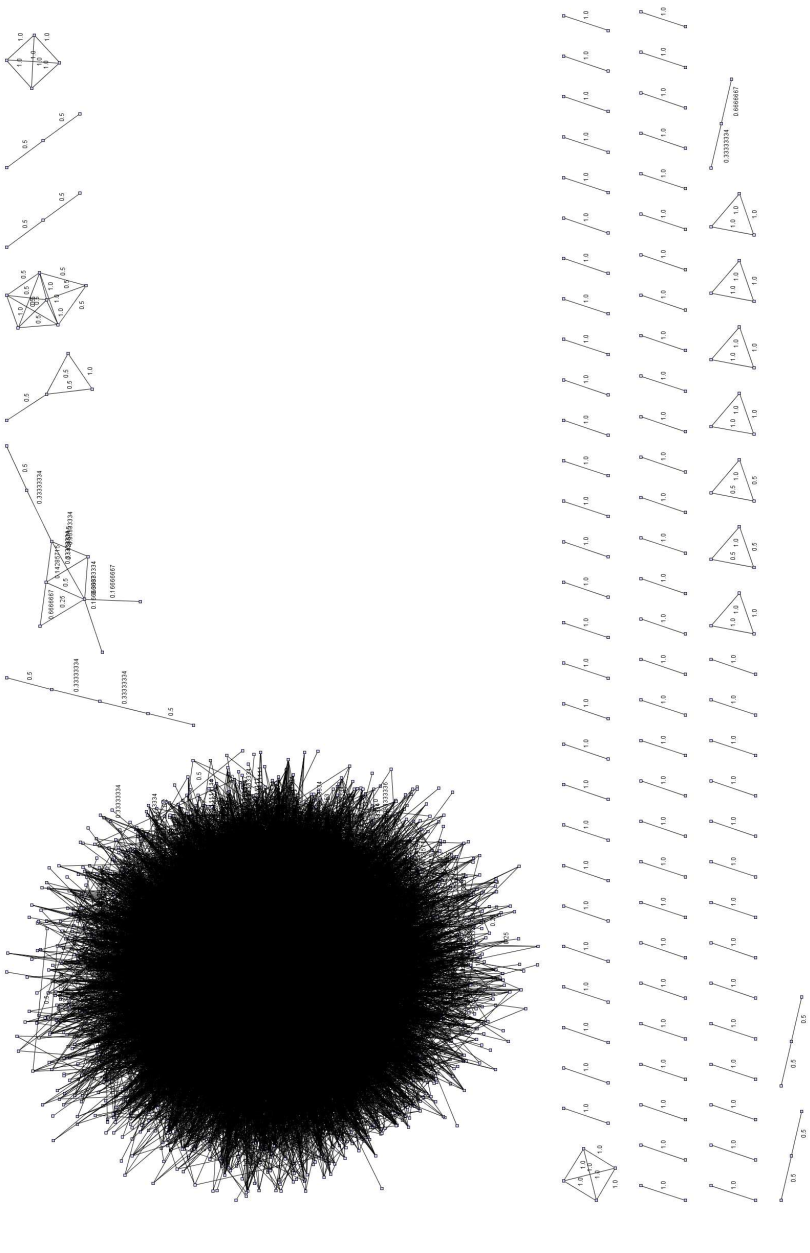


Figure A.1: Visualisation of G_{buy}.

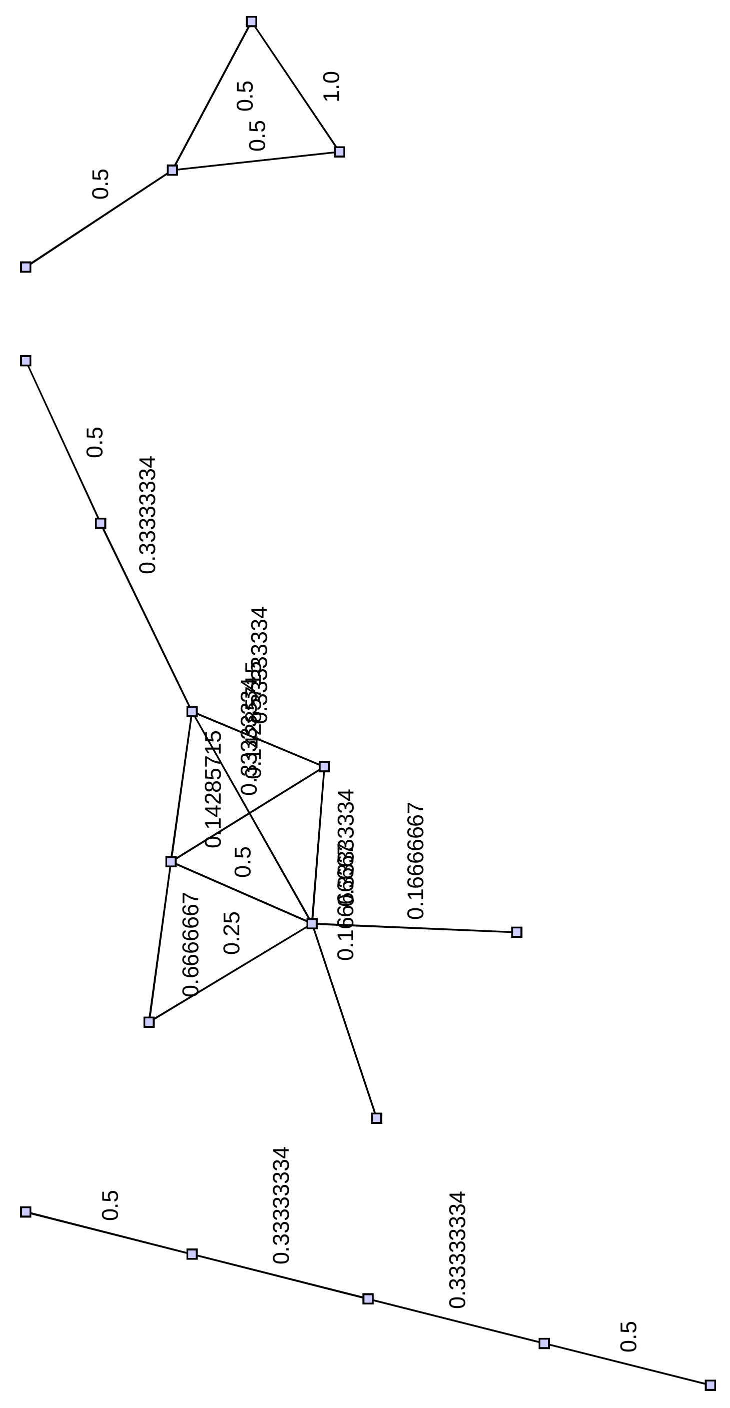


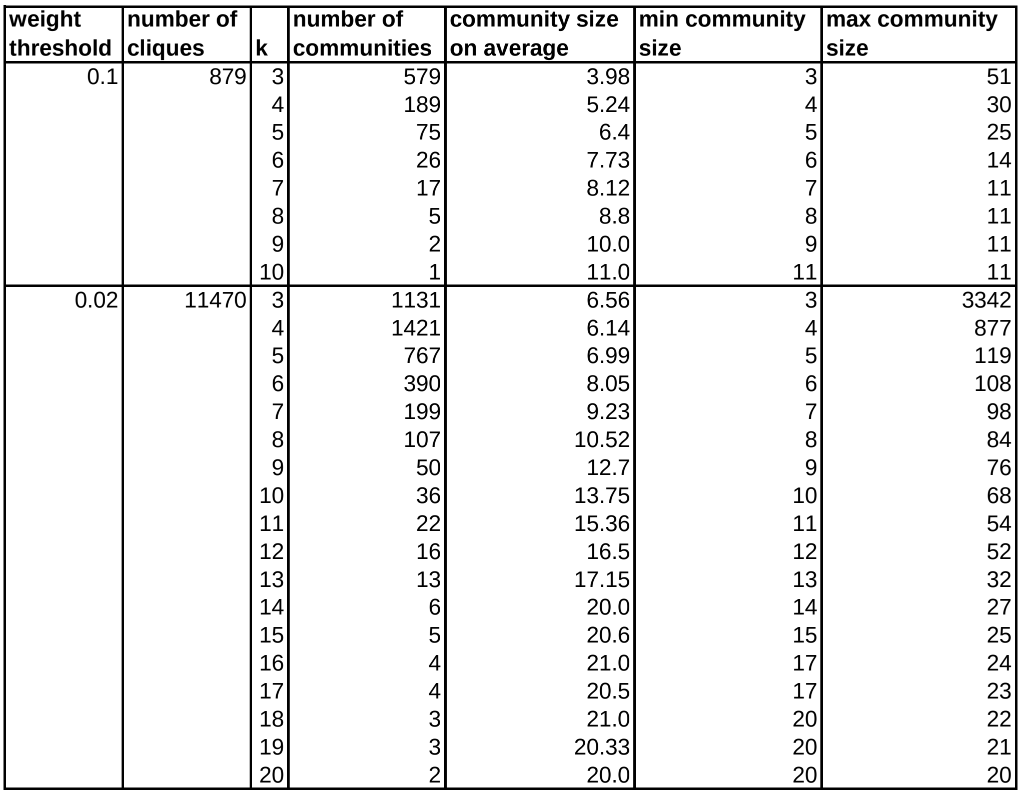
Figure A.2: Enlargement of the visualisation given in Figure A.1.

Appendix B

weight threshold	number of cliques	k	number of communities	community size on average	min community size	max community size
1.0	35	3	35	3.63	3	7
		4	11	4.64	4	7
		5	4	5.75	5	7
		6	2	6.5	6	7
		7	1	7.0	7	7
0.9	35	3	35	3.63	3	7
		4	11	4.64	4	7
		5	4	5.75	5	7
		6	2	6.5	6	7
		7	1	7.0	7	7
0.8	41	3	41	3.46	3	7
		4	12	4.58	4	7
		5	4	5.75	5	7
		6	2	6.5	6	7
		7	1	7.0	7	7
0.7	42	3	42	3.45	3	7
		4	12	4.58	4	7
		5	4	5.75	5	7
		6	2	6.5	6	7
		7	1	7.0	7	7
0.6	47	3	45	3.47	3	7
		4	14	4.5	4	7
		5	4	5.75	5	7
		6	2	6.5	6	7
		7	1	7.0	7	7
0.5	107	3	90	3.6	3	8
		4	24	4.63	4	8
		5	7	5.86	5	8
		6	3	7.0	6	8
		7	2	7.5	7	8
		8	1	8.0	8	8
0.4	110	3	93	3.61	3	8
		4	26	4.58	4	8
		5	8	5.75	5	8
		6	3	7.0	6	8
		7	2	7.5	7	8
		8	1	8.0	8	8
0.3	177	3	140	3.75	3	9
		4	48	4.63	4	9
		5	13	5.69	5	9
		6	4	7.0	6	9
		7	2	8.0	7	9
		8	1	9.0	9	9
0.2	315	3	252	3.71	3	11
		4	80	4.73	4	10
		5	27	5.78	5	10
		6	7	6.86	6	10
		7	2	8.5	7	10
		8	1	10	10	10
		9	1	10	10	10
0.1	879	3	579	3.98	3	51
		4	189	5.24	4	30
		5	75	6.4	5	25
		6	26	7.73	6	14
		7	17	8.12	7	11
		8	5	8.8	8	11
		9	2	10	9	11
		10	1	11	11	11

Figure B.1: Weight threshold and community size of the CPM for G_{buy} (table).

Appendix C



weight threshold	number of cliques	k	number of communities	community size on average	min community size	max community size
0.1	879	3	579	3.98	3	51
		4	189	5.24	4	30
		5	75	6.4	5	25
		6	26	7.73	6	14
		7	17	8.12	7	11
		8	5	8.8	8	11
		9	2	10.0	9	11
		10	1	11.0	11	11
0.02	11470	3	1131	6.56	3	3342
		4	1421	6.14	4	877
		5	767	6.99	5	119
		6	390	8.05	6	108
		7	199	9.23	7	98
		8	107	10.52	8	84
		9	50	12.7	9	76
		10	36	13.75	10	68
		11	22	15.36	11	54
		12	16	16.5	12	52
		13	13	17.15	13	32
		14	6	20.0	14	27
		15	5	20.6	15	25
		16	4	21.0	17	24
		17	4	20.5	17	23
		18	3	21.0	20	22
		19	3	20.33	20	21
		20	2	20.0	20	20

Figure C.1: Distribution of the community sizes for G_{buy} (table).

Appendix D

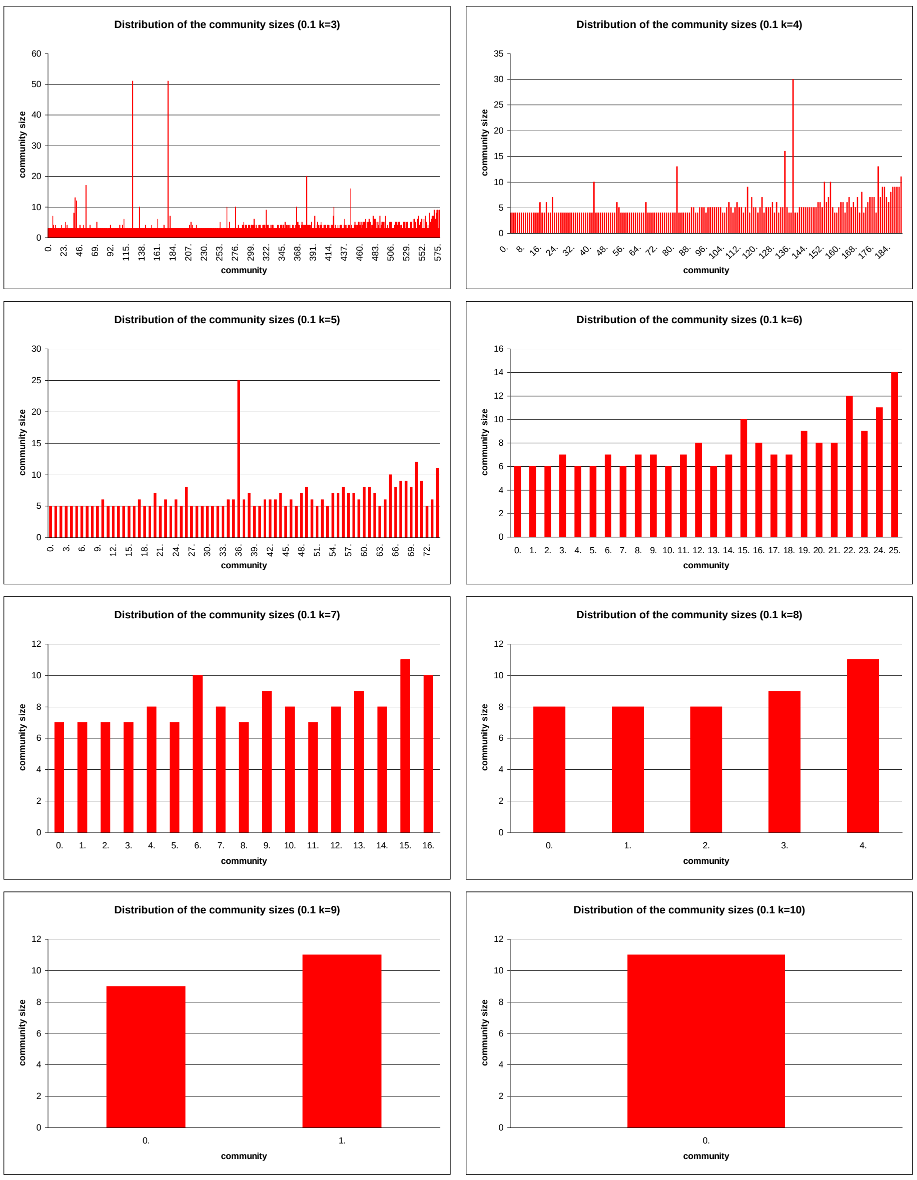


Figure D.1: Distribution of the community sizes for G_{buy} in detail (threshold $= 0.1$).

Appendix E

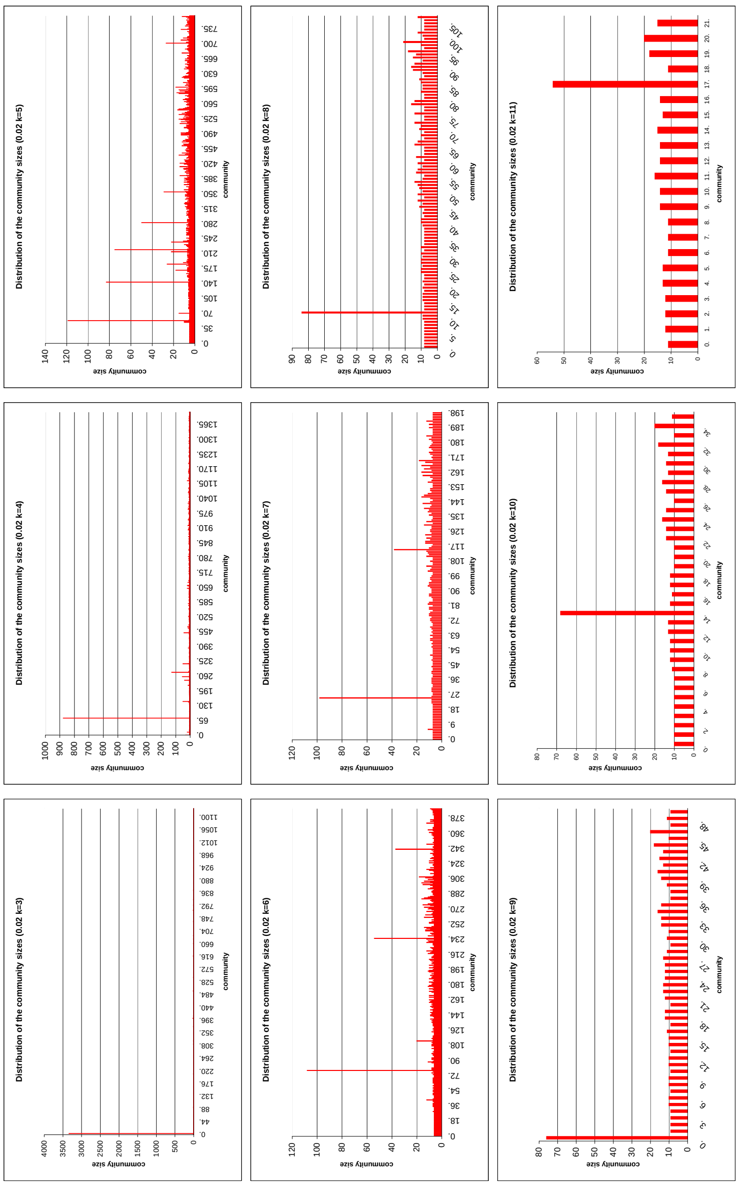


Figure E.1: Distribution of the community sizes for G_{buy} in detail (threshold = 0.02).

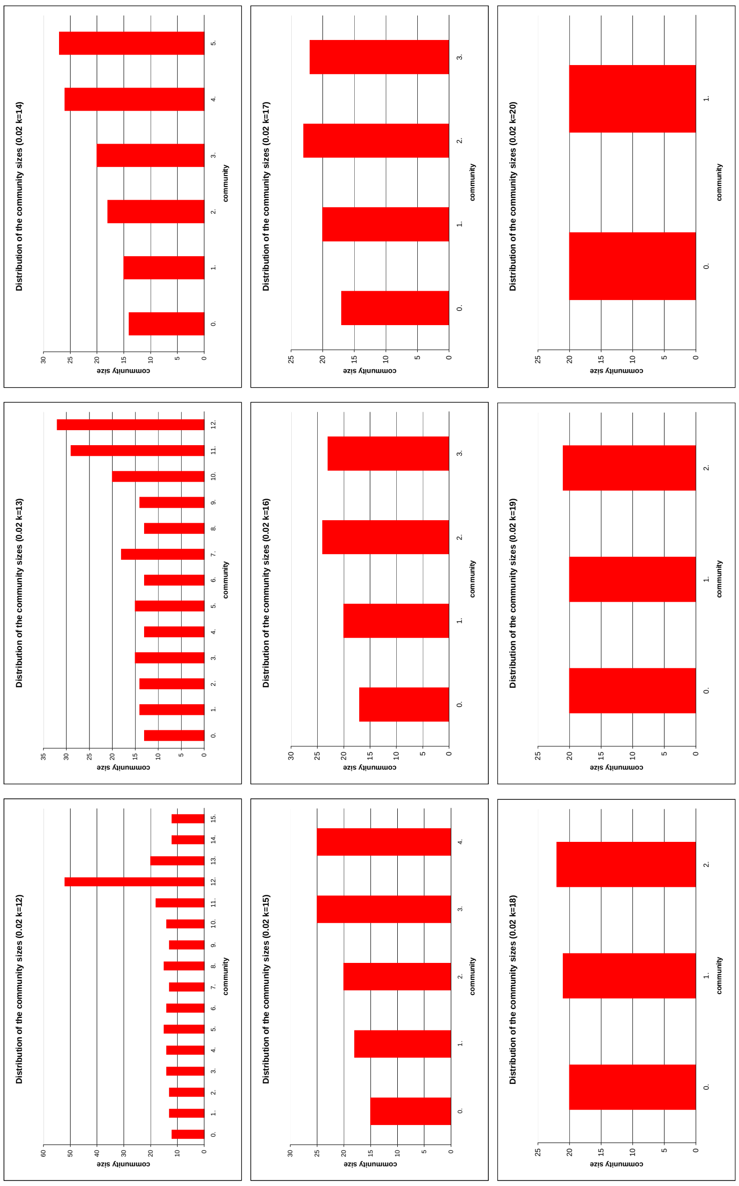


Figure E.2: Distribution of the community sizes for G_{buy} in detail (threshold = 0.02).

Appendix F

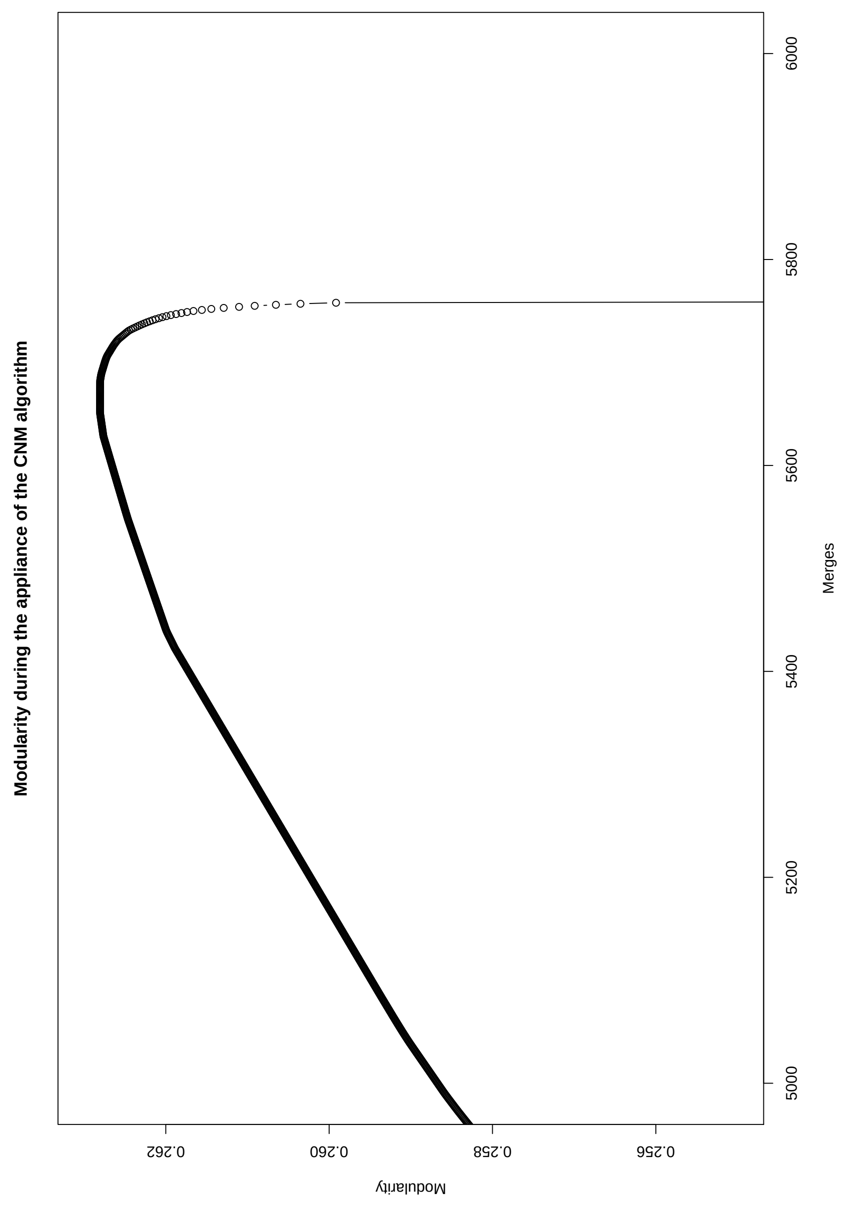


Figure F.1: Modularity against number of merges in case of G_{buy} in detail.

Appendix G

The program **community detection** does only work under linux and with a particular compiling software. Other configurations are problematical.

```
guillaum@delhi:Community_BGLL$ g++ --version
g++ (GCC) 4.1.2 20061115 (prerelease) (Debian 4.1.1-21)
Copyright (C) 2006 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.
There is NO warranty; not even for MERCHANTABILITY or FITNESS
FOR A PARTICULAR PURPOSE.
```

Appendix H

Result of the Fortunato algorithm				
alpha	number of communities	overlap	homeless vertices	average weak fitness
1.6	1288	142	2161	0.482677
1.59	1280	140	2166	0.485083
1.58	1273	133	2160	0.487217
1.57	1264	129	2181	0.489728
1.56	1258	124	2171	0.491952
1.55	1250	121	2172	0.494394
1.54	1246	127	2181	0.495056
1.53	1244	119	2163	0.496413
1.52	1239	122	2159	0.498091
1.51	1236	124	2159	0.499495
1.5	1229	125	2168	0.501149
1.49	1221	123	2168	0.504052
1.48	1216	123	2171	0.505588
1.47	1209	123	2169	0.508239
1.46	1204	125	2179	0.510623
1.45	1200	130	2173	0.5128
1.44	1194	127	2173	0.5146
1.43	1190	129	2176	0.516716
1.42	1183	133	2174	0.51942
1.41	1177	139	2174	0.521328
1.4	1175	140	2167	0.52246
1.39	1168	140	2162	0.524416
1.38	1161	141	2151	0.526593
1.37	1156	146	2138	0.528418
1.36	1153	146	2135	0.529484
1.35	1149	142	2133	0.530217
1.34	1144	146	2129	0.532022
1.33	1136	150	2133	0.534942
1.32	1124	145	2159	0.538775
1.31	1118	142	2166	0.541254
1.3	1110	139	2157	0.543421
1.29	1100	135	2150	0.545865
1.28	1094	134	2146	0.547974

1.27	1091	150	2145	0.549781
1.26	1085	154	2157	0.552092
1.25	1080	159	2153	0.55373
1.24	1067	165	2157	0.557411
1.23	1059	160	2166	0.559652
1.22	1050	163	2171	0.562549
1.21	1048	171	2155	0.563547
1.2	1042	173	2144	0.565169
1.19	1033	163	2168	0.567234
1.18	1027	160	2162	0.568897
1.17	1022	168	2145	0.571141
1.16	1014	170	2124	0.573148
1.15	1007	170	2128	0.575262
1.14	1001	178	2106	0.576806
1.13	992	181	2101	0.579051
1.12	983	200	2109	0.58259
1.11	974	193	2115	0.585327
1.1	964	202	2113	0.587823
1.09	952	205	2100	0.591812
1.08	948	214	2089	0.592362
1.07	943	217	2090	0.593583
1.06	939	217	2084	0.594712
1.05	932	212	2084	0.596785
1.04	922	227	2071	0.598492
1.03	918	225	2055	0.599054
1.02	908	226	2050	0.601099
1.01	896	237	2033	0.603875
1	882	235	2028	0.607706
0.99	872	247	2022	0.611108
0.98	869	274	2013	0.612492
0.97	852	283	2011	0.617002
0.96	848	299	1994	0.618433
0.95	833	276	1985	0.621939
0.94	830	292	1966	0.62321
0.93	820	335	1944	0.626502
0.92	811	336	1927	0.627344
0.91	806	336	1898	0.629999
0.9	798	363	1891	0.631748
0.89	784	359	1879	0.634836
0.88	773	363	1858	0.637501
0.87	760	389	1840	0.639691
0.86	758	420	1837	0.641274
0.85	748	432	1826	0.642952
0.84	735	457	1810	0.645721
0.83	726	453	1798	0.647949
0.82	723	478	1782	0.64817
0.81	706	492	1779	0.652068
0.8	689	523	1778	0.654559

0.79	677	530	1776	0.657685
0.78	669	550	1751	0.657311
0.77	649	563	1739	0.66053
0.76	645	608	1712	0.661358
0.75	628	630	1694	0.669064
0.74	619	610	1683	0.671738
0.73	613	667	1670	0.673601
0.72	601	742	1633	0.675912
0.71	590	772	1618	0.67796
0.7	571	901	1544	0.683651
0.69	551	929	1499	0.688873
0.68	530	1040	1417	0.696126
0.67	520	1220	1370	0.700395
0.66	234	331	1	0.869895
0.65	228	391	0	0.87407
0.64	224	411	0	0.876174
0.63	222	427	0	0.87487
0.62	220	438	0	0.875615
0.61	214	473	0	0.878284
0.6	205	482	0	0.8854

Table H.1: Statistics of Fortunato's algorithm (applied to G_{buy})

Appendix I

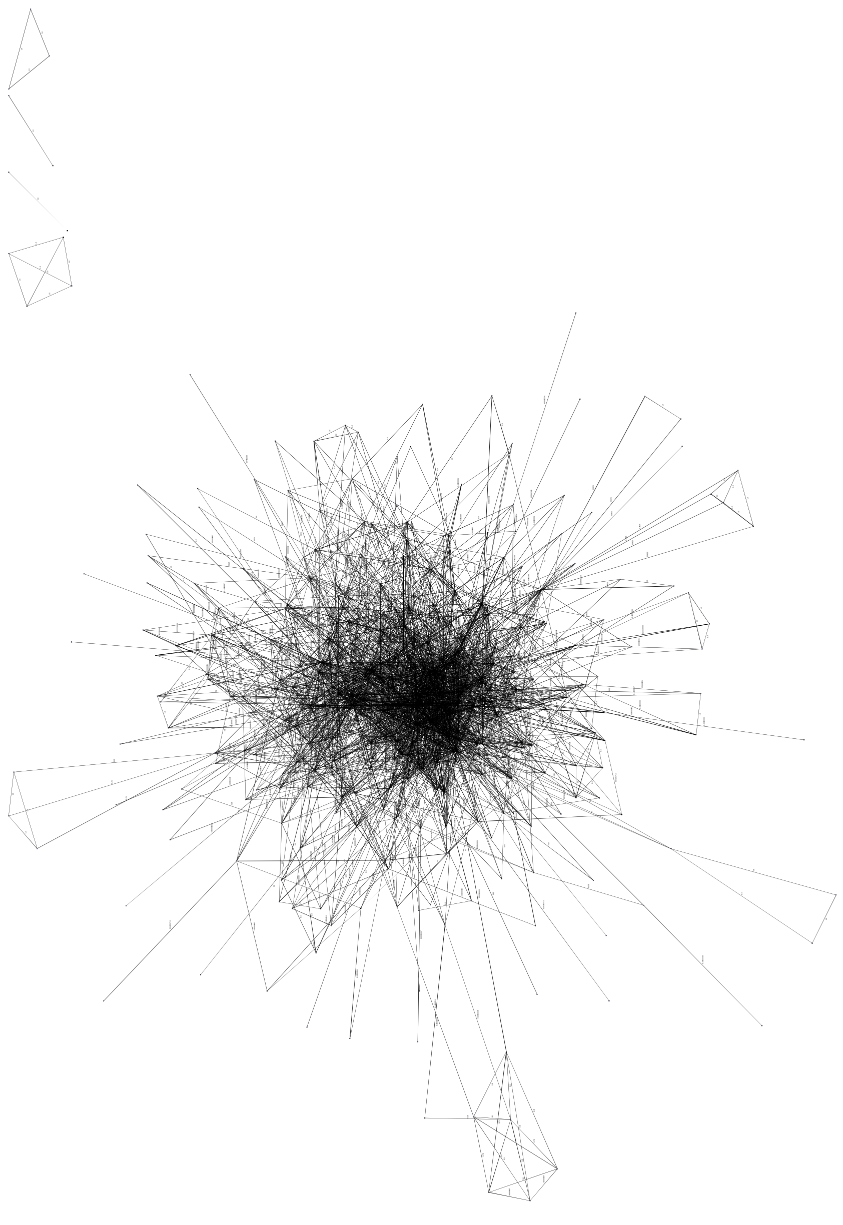


Figure I.1: Visualisation of $G_{\text{buy}}^{\text{anotherShop}}$.

Bibliography

[1] Balázs Adamcsek, Gergely Palla, Illés J. Farkas, and Imre Derényi. Cfinder: locating cliques and overlapping modules in biological networks. *Oxford Jourmals: Bioinformatics Applications Note*, 22(8):1021–1023, 2006.

[2] Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Lanet-vi in a nutshell. *http://xavier.informatics.indiana.edu/lanet-vi/lanetvi-2_2.pdf*, 2006.

[3] Marc Barthélémy and Santo Fortunato. Resolution limit in community detection. *Proceedings of the National Academy of Science of the United States of America*, 104(1):36–41, 2007.

[4] Vladimir Batagelj and Matjaz Zaversnik. An o(m) algorithm for cores decomposition of networks. *Detailed version of the part of the talk presented at Recent Trends in Graph Theory, Algebraic Combinatorics and Graph Algorithms, September 2001, Bled, Slovenia*, 2002.

[5] Michael Baur, Marco Gaertler, Robert Görke, Marcus Krug, and Dorothea Wagner. Augmenting k-core generation with preferential attachment. *Networks and Heterogeneous Media, 3(2):277-294*, 2008.

[6] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *arXiv:0803.0476v2 [physics.soc-ph] 25 Jul 2008*, 2008.

[7] Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of community hierarchies in large networks. *arXiv:0803.0476v1 [physics.soc-ph] 4 Mar 2008*, 2008.

[8] U. Brandes, D. Delling, M. Gaertler, R. Goerke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity - np-completeness and beyond. *Technical Report 2006-19, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH)*, 2006.

[9] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology 25 (2)*, pages 163–177, 2001.

[10] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity - np-completeness and beyond. *Technical Report 2006-19, ITI Wagner, Faculty of Informatics, Universität Karlsruhe (TH)*, 2006.

[11] Ulrik Brandes and Thomas Erlebach, editors. *Network Analysis.* Springer-Verlag Berlin Heidelberg, 2005.

[12] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16:575–577, 1971.

[13] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, 70(066111), 2004. cond-mat/0408187.

[14] Imre Derényi, Gergely Palla, and Tamás Vicsek. Clique percolation in random networks. *Physical Review Letters*, 94(160202), 2005. DOI: 10.1103/PhysRevLett.94.160202.

[15] Birgit Eimeren and Beate Frees. Internetverbreitung: Größter zuwachs bei silver-surfern. *Media Perspektiven 7/2008*, pages 330 – 344, 2008.

[16] Illés J Farkas, Dániel Ábel, Gergely Palla, and Tamás Vicsek. Weighted network modules. *New Journal of Physics*, 9(180), 2007. DOI: 10.1088/1367-2630/9/6/180.

[17] G. W. Flake, S. Lawrence, C. L. Giles, and F. M. Coetzee. Self-organization and identification of web communities. *IEEE Computer*, 35(3, 66 - 71), 2002. http://www.ghostscript.com/ raph/flake02selforganization.pdf.

[18] Marco Gaertler. *Network Analysis*, chapter Clustering, pages 178 – 215. ©Springer-Verlag Berlin Heidelberg, 2005.

[19] Wei Gao, Kam-Fai Wong, Yunqing Xia, and Ruifeng Xu. *Clique Percolation Method for Finding Naturally Cohesive and Overlapping Document Clusters.* Springer-Verlag Berlin Heidelberg 2006, 2006. Y. Matsumoto et al. (Eds.): ICCPOL 2006, LNAI 4285, pp. 97-108, 2006.

[20] Robert Görke, Marco Gaertler, and Dorothea Wagner. Lunarvis - analytic visualizations of large graphs. *Proceedings of the 15th International Symposium on Graph Drawing (GD'07)*, 2007. Lecture Notes in Computer Science. Springer, 2008.

[21] Jiawei Han and Micheline Kamber. *Data Mining.* Morgan Kaufmann Publishers, 500 Sansome Street, Suite 400, San Francisco, CA 94111, second edition, 2006.

[22] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. *Data Mining and Knowledge Discovery.* Kluwer Academic Publishers, 2004. Mining Frequent Patterns Without Candidate Generation: A Frequent-Pattern Tree Approach.

[23] Yanqing Hu, Hongbin Chen, Peng Zhang, Menghui Li, Zengru Di, and Ying Fan. A new comparative definition of community and corresponding identifying algorithm. *arXiv:0802.0242v1 [physics.soc-ph] 2 Feb 2008*, 2008. PACS: 89.75.Hc, 89.75.Fb.

[24] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure of complex networks. *arXiv:0802.1218v1 [physics.soc-ph] 8 Feb 2008*, 2008.

[25] M. E. J. Newman. Analysis of weighted networks. *preprint cond-mat/0407503*, 2004.

[26] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Phys. Rev. E*, 69(066133), 2004. arXiv:cond-mat/0309508v1 [cond-mat.stat-mech] 22 Sep 2003.

[27] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2003. arXiv:cond-mat/0308217v1 [cond-mat.stat-mech] 11 Aug 2003.

[28] Jukka-Pekka Onnela, Jari Saramäki, János Kertész, and Kimmo Kaski. Intensity and coherence of motifs in weighted complex networks. *Physical Review E*, 71(065103), 2005. DOI: 10.1103/PhysRevE.71.065103.

[29] Gergely Palla, Albert-László Barabási, and Tamás Vicsek. Quantifying social group evolution. *Nature 446, 664 (2007)*, 2007.

[30] Gergely Palla et al. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435:814 – 818, 2005.

[31] Filippo Radicchi, Claudio Castellano, Federico Cecconi, Vittorio Loreto, and Domenico Parisi. Defining and identifying communities in networks. *Proc. Natl. Acad. Sci. USA*, 101:2658 – 2663, 2004. arXiv:cond-mat/0309488 v2 27 Feb 2004.

[32] v. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending modularity definition for directed graphs with overlapping communities. *arXiv:0801.1647v3 [physics.data-an] 29 Jan 2008*, 2008.

[33] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. *arXiv:cs/0702048v1 [cs.CY] 8 Feb 2007*, 2007.