Diplomarbeit

Analysis and Visualisation of Social Networks

Keyan Mahmoud Ghazi-Zahedi

Eberhard Karls
Universität
Tübingen

Diplomarbeit im Fach Informatik

Analysis and Visualisation of Social Networks

vorgelegt von

Keyan Mahmoud Ghazi-Zahedi

April 2001

Betreuer: Prof. Dr. Michael Kaufmann

Wilhelm-Schickard-Institut
Arbeitsbereich Paralleles Rechnen
Eberhard-Karls-Universität Tübingen
Sand 13 · 72076 Tübingen

Erklärung

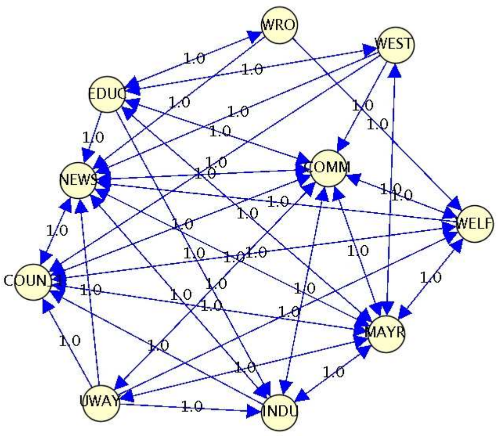
Hiermit versichere ich, diese Arbeit selbständig verfaßt und nur die angegebenen Quellen benutzt zu haben.

(Keyan Zahedi)

Contents

Chapter 1

Introduction



Firms exchanging information forming a social network (Knoke Information Exchange Data)

A social network is a formal representation of actors and the relations between them. Actors, in this context, can be persons, like students in a classroom, or companies, having economic relations (see Figure above). Representing a social structure by a (social) network allows the analysis by graph theoretical methods. In a social network, or more precise the graph of a social network, actors are represented by nodes, and the relations between actors are represented by edges between the according nodes. In the Chapter *Social Networks* we briefly review the history of social network analysis, discussing the data we have, and the advantages of formal methods.

In the Figure above, we see companies that have an information exchange network. Having a brief look at the Figure, one can see, that there are actors having a lot of direct connections to others, some receiving more or less information as they offer, and some only having very few direct connections to others. Social scientists are interested in determining the structure of a social network, as well as determining the distribution of power among the actors. The power of an actor arises from the structure of the relations in the social network.

The structure of network is best shown using clustering methods. A cluster is a group of actors in the network, that have more relations among each other, then to actors that are not a member of the group. The network is divided into distinguishable sub-networks. We assume, that an actor can be part of more then one group. The clustering method must therefor be able to extract overlapping clusters. In the Chapter *Clustering* we discuss different approaches of clustering, finally choosing the k-plex approach as it allows overlapping of the extracted clusters.

The next step in our analysis of the social network is to calculate the distribution of power. An actors is said to have power, if he has an advantageous position compared to others. The power of an actor is modelled by centrality measures. The degree centrality, as an example, implies, that an actor with more direct connections to other actors has more power, or is more important, as he has access to more resources. In the picture above the actor labelled with *WRO* would be said to be less powerful than the actor labelled with *COMM*, as *WRO* has less direct connections, and therefor less access to information. In the Chapter *Centrality* we discuss different approaches to model power, discussing in detail the algorithms to calculate two basic (degree and betweenness) and one extension of a basic centrality (flow-betweenness).

After analysing the data, the next step is a good visualisation of the extracted information. We say that a visualisation is good, if the information, extracted clusters and centrality of the actors, can bee seen by the viewer without much interpretation needed. The result of the visualisation should be able to support the analysis of a social network done by a social scientist, as well as the presentation of the results. In the chapter *Visualisation* we first discuss the problems that occur visualising overlapping clusters. We then develop a method, starting with the visualisation of only one cluster, extending it to handle multiple overlapping or non-overlapping clusters.

In the chapter *Mathematical Foundations* we introduce some definitions and the terminology that is needed to describe and discuss the algorithms used here.

In the Appendix A we discuss the architecture of the application $\mathcal{Y}$SocNet, and related work is discussed in the Appendix B.

Acknowledgement

I would like to thank Prof. Dr. M. Kaufmann for giving me the chance to work on this subject under his direction and for his help. My thanks also go to Monika Lanik, from the Ethnological Institute of Tübingen, for helping in any aspect concerning the usage of the application as well as helping in any social scientific question, and to Markus Eiglsperger, Roland Wiese, Boris Diebold and Frank Eppinger for their help, as well as to Olaf Peisker for his help.

Chapter 2

Social Networks

In this chapter we give a brief overview of the history of social network analysis, the modern analysis, the data that social network scientist use, and the advantages of formal methods in social network analysis.

2.1 History

Scott dates the development of social network analysis as an algebraic analysis with the breakthrough of the Harvard-structuralists around Harrison C. White (ca. 1970) [Jan99]. Before this breakthrough was possible, a lot of work was done by other researchers. The starting point probably was the British scientist Radcliffe-Brown (1881-1955), who, together with Malinowski is deemed to be the father of the functionalistic approach in sociology. The structuralism stadardisises the behaviours of groups in fixed roles. They were interested in how larger units like communities, villages or organisation function. Radcliffe-Brown was the first to use the term of *network* as a metaphor:

> "I use the term ,social structure' to denote this network of actually existing relations."[Radcliffe-Brown, 1940]

The "*Manchester-Group*" (Gluckman, Barnes, Bott, Nadel, Mitchell) of the British social-anthropology augmented the network approach in the fifties and sixties as an alternative to the structure-formalism. They were interested in specific behaviours and specific relationships instead of using attributes for the analysis. Conflicts and power were main aspects of their research.

In their works Barnes (1972) and J. Clyde Mitchell (1969) discussed the term of *network* at the end of the sixties, beginning of the seventies. They developed an instrument to describe network-structure under the aspect of the network terminology. Mitchell understands a network as

> "a specific set of linkages among a defined set of persons, with the additional property that the characteristics of these linkages as a whole may be used to interpret the social behaviour of the persons involved."

2.2 Modern social network analysis

Moreno (1934, 1954) was the first to introduce sociograms as the mathematical representation of social network data. These sociograms were drawn graphs, representing actors and relations as we discuss them here. Drawing even a small network of actors can be a difficult task, if the result should still provide the viewer with information. Intuitively every viewer interpreted something into the distance between the drawn actors, but early sociograms did not code any information in the distance between actors. Social scientists could also use any desired number of different visualisation criteria in their sociograms [Jan99]. Early sociograms therefor did not provide any information. Consequently social science, especially in Germany, did not use social network analysis much, as the results did not promise to support the analysis of social groups.

Today the analysis of social networks is said to be the most promising approach in the sociology [Jan99]. The following ideas mark the analyse of networks, as it is practised today, especially in the United States [Sch89].

1. The network analysis is understood as a general framework for the structural analysis of volatile and tight relationships in a social system.

2. Formal analysis provides an important part of the ethnological network analysis. Todays algorithms were developed in cooperation of mathematic and ethnological scientist. They address the issue of ethnological problems much better and are very powerful in the analysis of social structures.

3. Social network analysis focus two main aspects:

- Gathering of complex and diffuse social structures
- Micro–macro linkage, explaining how local units are bound into higher-level regional and national units.

2.3 Data and resources

A social network is described by actors and by the relations among the actors.

2.3.1 Actors

In the graph of the social network, actors are represented by nodes. Network analysts do not focus on one individual and his attributes, but on the relations among actors. The nodes or actors in non-network studies tend to be the result of independent probability sampling, whereas network studies are much more likely to include all of the actors who occur within some, usually naturally occurring boundary. Such a natural occurring boundary could be a classroom. If we would like to examine the relationships between classmates, we would include every actor in the class.

Levels of analysis

One can think of individuals as being embedded into a network, which is embedded into a network, etc. In the classroom-example, we have students within classes, and classes within the school, and schools within a school-district, etc. Such a structure is described as "multi-modal".

2.3.2 Relations

In the graph of the social network, relations are represented by edges. When it comes to the relations between the actors, some sampling is very likely to occur. Actors might be connected by many relations. In our example this could be, who likes whom, who does not like whom, who plays with whom, etc. Full network data allows very powerful description and analysis of social structures, but requires that all possible information about each actor's ties with all other actors is obtained. Collecting full network data can be very expensive and difficult in any other case than very small groups, as one has to ask every actor about every connection he has to every other person. In many cases the problems are not as severe as it might seems. Most persons, groups and organisations tend to have only a limited number of ties, or at least a limited number of strong ties. The reason might be, that actors often have access to only a limited number of resources. The methods introduced here are designed to work with full network data.

Scales of measurement

Different kind of scales have different mathematical properties. We can distinguish between different measurements, which we will describe here briefly. These are:

- nominal
  - binary
  - multi-category
- ordinal
- interval

Binary measures: The most common measurement, where *true* denotes an existing relation and *false* that the actors have no relation.

Multi-category: Each actor has a category, which describes the type or strength of a relation to another actor. These categories are mainly described by numbers, where 1,2,3,... could be *weak, average, strong,...*.

Grouped ordinal: One of the earliest traditions in social networks. Actors are asked to rate other actors as *liked, neutral, disliked*, represented by $1, 0, -1$.

Full-rank ordinal: Actors are asked to give related actors numbers from $1, \ldots, n$ starting with the lowest favoured actor or the actor with the weakest tie to, to the most liked actor or the actor with to strongest tie to. Such scales reflect differences in the degree of intensity, but not necessarily equal differences.

Interval measures: A more precise level of measurement as the *full-rank ordinal measure*. The difference between a "1" and "2" denotes the same amount of real difference as between "24" and "25".

2.4 Why formal methods?

There are three main reasons for using formal methods to analyse social networks [Han98].

Systematic representation: For a small population of actors, the patterns of relationships can be described completely and effectively using words. For a large number of actors and/or relations between the actors, this can get very tedious. Formal representation ensures that all the necessary information is systematically represented.

Computational Analysis: To analyse a large social network large amounts of data must be manipulated. By hand this would take too long. The usage of computers reduces the time needed significantly.

Patterns: The technique of graphing and the rules of mathematics themselves suggest things that we might look for in our data, that might not have been seen if the data was presented using a description in words. Describing the relations of actors by a matrix can show patterns of the relations that otherwise may not have been seen. This can lead us to ask other questions, which then may lead us to new insights.

Chapter 3

Mathematical Foundations

In this chapter we introduce some graph theoretical definitions, needed to describe and discuss the algorithms of this work.

3.1 Definitions

The definitions are taken from [dBETT99],[CLR99] and [Har69]. The formal representation of a social network is called a *graph*.

Definition 3.1 (graph)
A graph $G = (V, E)$ consists of a finite set V of *vertices* and a finite multiset E of *edges*, that is, unordered pairs (u, v) or vertices. The vertices of a graph are sometimes called *nodes*, edges are sometimes called *links*, *arcs* or *connections*.

Here a node represents an actor, and an edge indicates a relationship between two actors.

This definition is too general, as relations are directed. If an actor A calls actor B , this does not automatically imply, that actor B will also call actor A. So we have to regard directed relations between actors.

Definition 3.2 (digraph)
A directed graph, or digraph, is a graph, which elements of E are ordered pairs of vertices, called *directed edges*. The directed edge (u, v) is an *outgoing edge* of u and an *incoming edge* of v. Vertices without outgoing (resp. incoming) edges are called *sinks* or *targets* (resp. *sources*).

If we talk about graphs in the following chapters, we always mean digraphs. Given an undirected graph $G = (V, E)$, the directed version of G is the directed graph $G' = (V, E')$, where $(u, v) \in E'$ if and only if $(u, v) \in E$. That is, each undirected edge $(u, v) \in E$ is replaced in the directed version by the two directed edges $(u, v) \in E'$ and $(v, u) \in E'$. Given a directed graph $G = (V, E)$, the undirected version of G is the undirected graph $G' = (V, E')$, where $(u, v) \in E'$ if and only if $u \neq v$ and $(u, v) \in E$ or $(v, u) \in E$.

Now we need a notation for the direct neighbourhood of a node, concerning the relations and the neighbours a node can have.

Definition 3.3 (incident)
If (u, v) is and edge in a directed graph $G = (V, E)$, we say that (u, v) is *incident from* or *leaves* vertex u and is *incident to* or *enter* vertex u. If (u, v) is an edge in an undirected graph $G = (V, E)$, we say that (u, v) is *incident on* vertices u and v.

Definition 3.4 (adjacent)
If (u, v) is an edge in a graph $G = (V, E)$, we say that vertex v is *adjacent* to vertex u. When the graph is undirected, the adjacency relation is symmetric.

Definition 3.5 (neighbour)
In a directed graph $G = (V, E)$, a *neighbour* of a vertex u is any vertex that is adjacent to u in the undirected version of G. That is, v is a neighbour of u if either $(u, v) \in E$ or $(v, u) \in E$. In an undirected graph, u and v are neighbours if they are adjacent.

Definition 3.6 (degree)
The *degree* of a vertex in an undirected graph is the number of edges incident on it. In a directed graph, the *out-degree* of a vertex is the number of edges leaving it, and the *in-degree* of a vertex is the number of edges entering it. The *degree* of a vertex in a directed graph is its in-degree plus its out-degree.

Definition 3.7 (path)
A *path* of *length* k from a vertex u to a vertex u' in a graph $G = (V, E)$ is a sequence $\langle v_0, v_1, \ldots, v_k \rangle$ of vertices such that $u = v_0, u' = v_k$, and $(v_{i-1}, v_i) \in E$ for $i = 1, 2, \ldots, k$. The length of the path is the number of edges in the path. The path *contains* the vertices $v_0, v_1, \ldots, v_k$ and the corresponding edges $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$. If there is a path p from u to u', we say u' is *reachable* from u via p, which we sometimes write as $u \overset{p}{\leadsto} v$ if G is directed. A path is *simple* if all vertices in the path are distinct. A shortest

path u, u' is often called a *geodesic*. The *diameter* $d(G)$ of a connected graph G is the length of any longest geodesic. A undirected graph is *connected* if every pair of vertices is connected by a path.

Definition 3.8 (complete graph)
A *complete graph* is an undirected graph in which every pair of vertices is adjacent.

Definition 3.9 (flow network)
A *flow network* $G = (V, E)$ is a directed graph in which each edge $(u, v) \in E$ has a nonnegative *capacity* $c(u, v) \geq 0$. If $(u, v) \notin E$, we assume that $c(u, v) = 0$. We distinguish two vertices in a flow network: a *source* $s \in V$ and a *sink* $t \in V$.

Definition 3.10 (flow)
A *flow* in G is a real-valued function $f : V \times V \mapsto \mathbb{R}$ that satisfies the following three properties:

Capacity constraint: For all $u, v \in V$, we require $f(u, v) \leq c(u, v)$.

Skew symmetry: For all $u, v \in V$, we require $f(u, v) = -f(v, u)$.

Flow conservation: For all $u \in V \backslash s, t$, we require $\sum_{v \in V} f(u, v) = 0$.

Chapter 4

Clustering

Social scientists are interested in the structure of relations between actors. By grouping the actors, that have more relations among each other, the structure of the social network can be presented well, as we divide the graph into distinguishable sub-graphs. Actors are understood as strategists, and their position in the network as well as their power allows a social scientist to draw conclusions about the tactics and possibilities of acting. Therefor the first step in our social network analysis is the clustering of nodes. But before we discuss the clustering, we should first define what a cluster is.

Definition 4.1 (cluster)
Let $G = (V, E)$ be a graph, where V is the set of nodes, and E is the set of edges. A *cluster* C_i is a list of nodes $v_i \in V$, where the elements of C_i are sorted by their degree in an increasing order. The set of clusters $\mathcal{C} = \{C_0, C_1, \ldots, C_k\}$ is called a *clustering* of G, if $\bigcup_{C_i \in \mathcal{C}} C_i = V$. A cluster $C_i \in \mathcal{C}$ includes all sub-cluster, so that $C_j \notin \mathcal{C}$, if $C_j \subset C_i$.

We have 2 main goals in clustering:

1. Finding the maximum number of clusters
2. Maximise each cluster in size

To be sure not to find just a single cluster, we would like to have the maximal number of clusters. This would be fulfilled, if every node declares a cluster on its own. In order to prevent this, we also require that each group is maximised in size.

Overlapping of clusters is wanted, as we assume that a single actor can be part of more than just one group.

4.1 Clustering Algorithms

There are two main categories of clustering algorithms, *bottom-up* and *top-down*. The bottom-up methods start with a single node, trying to find *dyads*, which is a cluster formed by only two nodes. The dyads are then extended to triads and higher-numbered clusters, if possible.

The top-down methods divide the graph into clusters, by analysing the structure of the graph, identifying sub-structures as parts that are locally denser than the graph as a whole.

4.1.1 Top-Down Approaches

Components

A *component* of a graph G is a completely connected sub-graph. It does not matter how closely connected the nodes are. Each component determines a cluster.

Blocks and Cut-points

A *cut-point* of a graph G is a node, which, if removed, divides the graph into an unconnected system. The sets of nodes into which the graph is divided are called *blocks*.

Lambda Sets and Bridges

Each relationship in the network is ranked in terms of importance by evaluating how much of the flow among the actors in the network is going through each link. The network is divided by the set of actors who, if disconnected, would significantly disrupt the flow among all actors. These actors are called *bridges* and the resulting sets are called *Lambda sets*. One property of Lambda sets is that each node in the set has more edge independent paths to every other node in the set than to nodes outside the set.

Conclusion

The top-down algorithms divide the complete graph into non-overlapping sub-graphs. As we would like to allow overlapping, none of the algorithms introduced above seem appropriate.

4.1.2 Bottom-Up Approaches

Cliques

A clique in a graph G is a maximal complete sub-graph of G. That is, a set of nodes $C \subseteq V$ is a clique, if

$$\forall v \in C : \deg_C(v) = |C| - 1,$$

where $\deg_C(v)$ is the degree of the node $v \in V$ in the cluster C:

$$\deg_C(v) = |\{u | u \in C, (u, v) \in E \vee (v, u) \in E\}|.$$

N-Cliques

The N-clique loosens the very hard constraint of a maximal fully connected sub-graph to form a cluster. We could say, an actor is part of a group, if he knows everyone in the group through someone else, which would correspond to being a "friend of a friend". This approach of defining sub-structures is called N-clique, where N stands for the length of the path allowed to make a connection to all other members. A set of nodes $C \subseteq V$ is a N-clique, if

$$\forall v \in C : \forall u \in C : \mathrm{dist}_C(u, v) \leq N.$$

N-Clan

The N-clique approach tends to find long and stringy groupings rather than the tight and discrete ones of the maximal sub-graph approach. In some cases, N-cliques can be found, that have a property that is probably undesirable for many purposes: it is possible for a member of N-cliques to be connected by actors who are not members of the clique themselves.

Therefor some analysts have suggested to insert a restriction of the total span or diameter of a N-clique. This forces all ties among members of a N-clique to pass only over actors, who are themselves members of the group. N-Clans are also known as *distance k-cliques* [ESB99]. A set of nodes $C \subseteq V$ is a N-clan, if

$$\forall v \in C : \forall u \in C : \mathrm{dist}_C(u, v) \leq N$$

and

$$\mathrm{diameter}_C(C) \leq N.$$

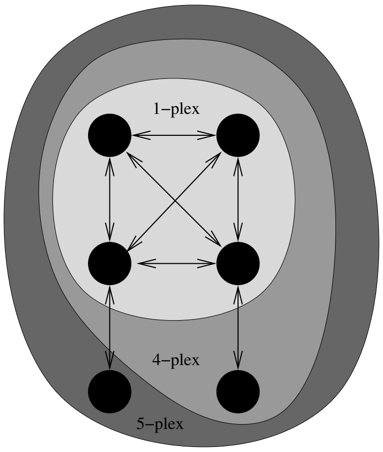


Figure 4.1: k-plex

k-plex

An alternative way of relaxing the strong constraints of the maximal complete sub-graph is to loosen the number of needed neighbours in the cluster. Using the k-plex approach, each node $v \in C$ in a cluster has a direct link to $n - k$ other members of the cluster $C \in \mathcal{C}$, where $n = |C|$ is the number of elements in the clusters C, and k is the k-plex parameter (see Figure 4.1). A 1-plex is a clique. A set of nodes $C \subseteq V$ is a k-plex, if

$$\forall v \in C : \deg_C(v) \geq |C| - k.$$

k-core

A subgraph $H = (W, E|W)$ induced by the set W is a *k-core* or a *core of order k* if $\forall v \in W : \deg_H(V) \geq k$ and H is a maximum subgraph with this property [BMZ99].

The degree $\deg_H(v)$ can be: in-degree, out-degree, in-degree + out-degree, min(in-degree,out-degree),... determining different types of cores. The cores have the following properties:

- The cores are nested: $i < j \Rightarrow H_i \subseteq H_j$, where H_i is a core of order i.
- There exists an efficient algorithm to determine the core hierarchy ($\mathcal{O}(m)$, see [BMZ99])

- Cores are not necessarily connected sub-graphs

Conclusion

From all bottom-up approaches only the N-clan, N-clique and the k-plex algorithm would allow overlapping of the resulting clusters. Using the N-clan/-clique, we would demand, that every node has a limited path to every other node in the cluster, but the path may have a length greater than one. The k-plex on the other hand emphasises the direct connection a node has in a cluster. Which algorithm to choose is a personal decision, as they just highlight different aspects. In consultation with the Ethnological Institute of Tübingen, we decided, that the number of direct neighbours an actor must have would better match our understanding of a group.

4.2 The Clustering Algorithm

4.2.1 Overview

The clustering algorithm used here is based on the k-plex algorithm. That is, each cluster fulfils the k-plex constraint

$$\forall v \in C_i : \deg_{C_i}(v) \geq n_i - k \tag{4.1}$$

during the clustering, where $n_i = |C_i|$ is the size of the cluster C_i, and k is given externally. With $\deg_{C_i}(v)$ we denote the degree of the node $v \in V$ in the cluster $C_i \in \mathcal{C}$. If $k = 1$, then each node has a link to every other node in the cluster. We have two goals for the clustering:

1. maximise the number of clusters

2. maximise each cluster in size

In order to achieve these goals, we declare every node as a cluster on its own. In every clustering-step, the cluster will be extended by all neighbours, that obey the Constraint 4.1. So at the beginning we receive every possible dyad as a initial cluster-core. As we will see later, the dyads are very important for the overlapping between clusters, consequently we will keep them, and only copies will be extended. If no nodes which can be added are found, k is increased by one, until the increasing of k does not change the set of clusters anymore. At the beginning we are looking for highly connected clusters. At every clustering-step, k is increased, which means, every new member needs less direct neighbours in order to be added to the cluster. While maximising the number of nodes for each cluster, we check if a cluster is equal to or completely part of another cluster. If we find such a cluster, we delete it from the list of clusters, as the size of the set of clusters is a very critical factor in the running time of the clustering. At the end, we give up the k-plex constraint to perform a post processing, which merges clusters, that have a significant percentage of shared nodes in respect of the smaller cluster. Why the merging of the clusters is needed is explained in the Section 4.2.4 at the end of this chapter.

4.2.2 Clustering

The first step of the clustering algorithm is to create a cluster from every node $v \in V$. After that, we only need an algorithm capable of extending a cluster in respect of a given parameter k.

Finding k-plex

The algorithm introduced here (see Algorithm 1) is inspired by the clustering algorithm introduced in "*Improved Graph Drawing Via Clustering*" by Six and Tolles in [ST] regarding the Constraint 4.1.

The input of the algorithm is a set of clusters $\mathcal{C}^{(k-1)}$ in respect of $k-1$. The output is the set of clusters $\mathcal{C}^{(k)}$ in respect of k.

For every cluster $C_i \in \mathcal{C}$ we need to find new members in the set of neighbours of the nodes $v \in C_i$. The goal is to perform this without checking every neighbour of every node in each cluster as a potential new member. The good thing is, that we do not have to. As a new member needs at least $n-k-1$ direct neighbours in the cluster, we only have to check for neighbours of $k+1$ members of each cluster. If none of the $k+1$ members has a neighbour, that has enough direct connections into the cluster, we can stop, as for any potentially new member the maximal number of possible neighbours in the clusters is $n-k-2$. If we choose the $k+1$ members with the lowest degree, we have reduced the cost for expanding a cluster to a minimum. Let $L_D^{(k+1)} = C_i|_{k+1}$ be the set of the $k+1$ lowest degree members of the cluster C_i, which is equal to the first $k+1$ elements of C_i. For every node $v \in L_D^{(k+1)}$, we have to create the set of neighbours not in

C_i of v; $N(v, V \backslash C_i) = \{u | u \in V \backslash C_i, (v,u) \in E \wedge (u,v) \in E\}$. For every $u \in N(v, V \backslash C_i)$, we have to calculate the size of the set of neighbours of u in C_i; $N(u, C_i)$. If $|N(u, C_i)|$ exceeds $n-k-1$ then u will be added to the cluster C_i.

A cluster C_i with size one will not be extended itself, but a copy will be added and extended, as we need the single node to receive every possible dyad. Deleting the single nodes, if they and all their neighbours are part of one cluster did not show any significant speed-up. The algorithm is listed in Algorithm 1.

To avoid too much overlapping, a residual graph is calculated for every cluster (see Section 4.2.3), only including the nodes of the clusters, that the current cluster does not share any nodes with. New members can only be found within the residual graph.

Worst-case runtime analysis

The first loop is called $\mathcal{O}(|\mathcal{C}|)$ times, creating the set of lowest degree members L_D in $\mathcal{O}(k)$ time. The time needed to create the rest-graph is discussed later in Section 4.2.3, and is here denoted by $\mathcal{O}(R)$. For each member $v \in L_D$ we create the set of member $N(v, V \backslash C)$ in $\mathcal{O}(n)$ time. For each node $u \in N(v, V \backslash C)$ we create the set of neighbours of u in the cluster C in $\mathcal{O}(n)$ time. The rest of the inner loop can be approximated by $\mathcal{O}(n)$. At the end we clean up after each cluster, which takes $\mathcal{O}(|\mathcal{C}|) \cdot \mathcal{O}(n^2)$ time. This sums to an overall worst case running-time of

$$\mathcal{O}(|\mathcal{C}|) \cdot (\mathcal{O}(R) + \mathcal{O}(k) \cdot \mathcal{O}(n^2) + \mathcal{O}(|\mathcal{C}|) \cdot \mathcal{O}(n^2))$$

Maximising the clusters

We assume, that it is not possible to find the best clustering by finding all k-plexes for a given k. Choosing a too small value for $k \in \mathbb{N}$ could yield in receiving many small clusters (especially dyads), as each cluster has to be highly connected. On the other hand, choosing a too high value for k is very likely to produce only a few or just one very large cluster (in respect of the size of the graph) as each cluster is only very slightly connected. Instead it seems promising to start with the smallest possible value $k = 1$ and increase k after every clustering-step. A clustering-step is over, if no node can be added to any cluster in respect to k.

Algorithm 1: clustering

```
Data: 𝒞, k, G = (V, E)
didCluster = false;
foreach C ∈ 𝒞 do
    m = |C| − k − 1;
    L_D = |C|_{k+1};
    R = calcRGraph(C);
    foreach v ∈ L_D do
        if directed then
            N(v, V\C) = {u|u ∈ R ∩ V\C, (u, v) ∈ E ∧ (v, u) ∈ E};
        else
            N(v, V\C) = {u|u ∈ R ∩ V\C, (u, v) ∈ E ∨ (v, u) ∈ E};
        end
        foreach u ∈ N(v, C) do
            if directed then
                N(u, C) = {w|w ∈ C, (u, w) ∈ E ∧ (w, u) ∈ E};
            else
                N(u, C) = {w|w ∈ C, (u, w) ∈ E ∨ (w, u) ∈ E};
            end
            if |N(u, C)| ≥ m then
                L_D = L_D ∪ {u};
                m = m + 1;
                didCluster = true;
                if |C| = 1 then
                    C̃ = C ∪ {u};
                    𝒞 = 𝒞 ∪ {C̃};
                    C = C̃;
                else
                    C = C ∪ {u};
                end
            end
        end
    end
    foreach Ĉ ∈ 𝒞\C do
        if Ĉ = C then
            𝒞 = 𝒞\Ĉ;
        end
    end
end
return didCluster;
```

The maximal possible value for k would be $n-1$, $n = |V|$ being the number of vertices in the graph, as each node would only need one incident node in any cluster, no matter how large.

A more efficient way to limit the number of iterations is to stop as soon as no node was being added to any cluster in one clustering step. The resulting clustering is not optimal, as clusters can not include others, even if the structure of the graph would indicate that. Thus we need a post processing, which merges clusters, that have enough overlapping, so that they can be considered to be one cluster (see Section 4.2.4).

One can imagine different ways of letting the clusters grow. They were implemented, but did not show any differences in their results. A possibly cause is the merging that is performed at the end. For the reason of completeness they are listed here. The Algorithms 2,3, and 4 only differ in how the sets of neighbours of $N(v, V\backslash C)$ and $N(u, C)$ are calculated in the algorithm, which extracts k-plexes (see Algorithm 1).

Algorithm 2: G1: undirected

Data: $\mathcal{C}, k, G = (V, E)$
directed = false;
while(clustering(k++,$\mathcal{C}$,G));

Algorithm 3: G2: directed, then undirected

Data: $\mathcal{C}, k, G = (V, E)$
directed = true;
while(clustering(k++,$\mathcal{C}$,G));
directed = false;
while(clustering(k++,$\mathcal{C}$,G));

4.2.3 Calculating the residual graph

The calculation of the residual graph is needed in order to avoid too much overlapping between the clusters. The more overlapping we have, the more the quality of the visualisation is reduced (see Chapter 6). We therefor require

Algorithm 4: G3: directed, reset k, then undirected

Data: $\mathcal{C}, k_i, G = (V, E)$
directed = true;
$k = k_i$;
while(clustering(k++,$\mathcal{C}$,G));
directed = false;
$k = k_i$;
while(clustering(k++,$\mathcal{C}$,G));

that two clusters, which share nodes at a certain clustering-step, denoted by the corresponding k-plex parameter $\hat{k}$, will not be allowed to have any further overlapping for any $k > \hat{k}$. To ensure this, the remaining graph $R \subset G$ must be calculated once for every cluster at every clustering-step. The residual graph R includes only nodes of clusters that have no node in common with the current one.

To calculate the residual graph R, we need a binary array that indicates, which clusters overlap, and which do not. This array is denoted by A_O and must be re-initialised every time R is calculated, because the size of the set of clusters $\mathcal{C}$ may vary. New dyads are added to the set of clusters and dispensable clusters are removed. After the overlapping-array A_O is created and filled with values indicating the overlapping, the residual graph R can be created by merging all clusters having a *false* entry in the array. The algorithm is shown in Algorithm 5.

Worst-case running time

With $|C_m|$ we denote the size of the largest cluster in the set of clusters $\mathcal{C}$. Initialising and filling of the overlap-array A_O takes $\mathcal{O}(|\mathcal{C}|) \cdot \mathcal{O}(|C_m|^2)$ time, as for every cluster $C_j \in \mathcal{C}$ in the set of clusters, the intersection with the current cluster C_i is calculated. The residual graph R is created in $\mathcal{O}(|\mathcal{C}|) \cdot \mathcal{O}(n) \cdot \mathcal{O}(|C_m|)$, as for every cluster $C_j \in \mathcal{C}$ in the set of clusters, the cluster C_j is merged with the residual graph R if it does not share any nodes with the current cluster C_i. This gives us an overall worst case running-time of

$$\mathcal{O}(R) = \mathcal{O}(|\mathcal{C}|) \cdot \mathcal{O}(n) \cdot (|C_m|).$$

Algorithm 5: calculate residual graph

```
Data: |𝒞|, C_i
Result: R
m = |𝒞|;
i = index(C_i);
A_O = BinArray(m);
foreach C_j ∈ 𝒞 do
    if i = j then
        A_O(j) = 1;
    else
        if |C_i ∩ C_j| > 1 then
            A_O(j) = 1;
        else
            A_O(j) = 0;
        end
    end
end
R = ∅;
foreach C_j ∈ 𝒞 do
    if A_O(j) = 0 then
        R = R ∪ C_i;
    end
end
```

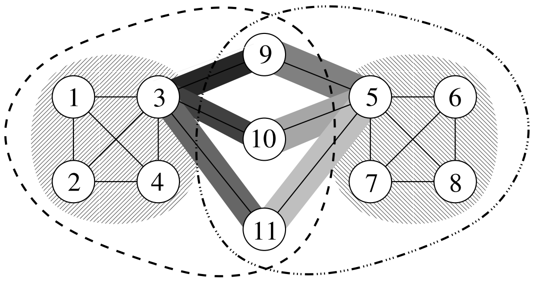


Figure 4.2: Clustering if no merging is performed. The resulting clusters are: $\mathcal{C}_{\overline{M}} = \{\{1,2,3,4\},\{5,6,7,8\},\{3,9\},\{3,10\},\{3,11\},\{5,9\},\{5,10\},\{5,11\}\}$, indicated by the filled areas. With merging the resulting cluster are: $\mathcal{C}_M = \{\{1,2,3,4,9,10,11\},\{5,6,7,8,9,10,11\}\}$, indicated by the surrounding lines.

4.2.4 Merging clusters

There is one main reason why the merging is needed. The restricted overlapping of groups prevents clusters of including other clusters while they are extended, even if the structure of the graph would indicate it. This could lead to an unwanted clustering, that is, we may get good large clusters, but also plenty of small cluster, mainly dyads. This happens when we have bridges or strings in the graph. One example is shown in the Figure 4.2, listing the result of the clustering with and without merging.

Let $C_i, C_j, C_k \in \mathcal{C}$ be clusters out of the set of clusters, such that $|C_i| < |C_j|$ and $|C_i| < |C_k|$. We then say C_i is the smaller cluster, and we mean that in respect of C_j, C_k. We also say that C_j or C_k is the larger cluster, and we mean that in respect of C_i.

To merge two clusters, the smaller cluster must share a certain percentage of its elements with the larger cluster. This percentage is called *merge percentage* and is denoted by $p_{m_{ij}} \in [0,1]$, where $i \in \mathbb{N}$ is the index of the smaller cluster, and $j \in \mathbb{N}$ is the index of the larger clusters. Only if the merge percentage $p_{m_{ij}}$ is higher or equal to a merge constant $\varepsilon_m \in [0,1]$ the cluster will be merged. Obviously a value of 1 for ε_m suppresses any merging. The cluster C_i is then added to the cluster with the highest merge percentage exceeding the merge constant. In some cases the merge percentages are equal $p_{m_{ij}} = p_{m_{ik}}$ for the three clusters C_i, C_j and C_k . This happens especially when many dyads are in the set of clusters. The cluster C_i could be added to the larger cluster of the clusters C_j, C_k. But that would lead to an unpleasant clustering, as the shared nodes in Figure 4.2 would be added completely to one of the two clusters $\{1, 2, 3, 4\}$ or $\{5, 6, 7, 8\}$, depending which cluster would first include one of the dyads. In this case node 3 or node 5 would be the shared node of the resulting two clusters.

Better results are achieved if the diameters of the candidate clusters are compared. The diameter $D(C)$ of the candidate clusters merged with the smaller cluster are compared, and the candidate with the smaller resulting diameter is chosen. This is done, until no clusters can be merged anymore (see Algorithm 6).

Algorithm 6: Merging clusters

Data: $\mathcal{C}$
Result: $\mathcal{C}$
repeat
  $n = |\mathcal{C}|$;
  foreach $C_i \in \mathcal{C}$ **do**
    $O = \{C_j | C_j \in \mathcal{C}, p_{m_{ij}} = \max\limits_{C_k \in \mathcal{C}}(p_{m_{ik}}) \vee |C_j| > |C_i|, p_{m_{ij}} = \frac{|C_i \cap C_j|}{|C_i|}\}$;
    $D = \{C_j | C_j \in O, D(C_j \cup C_i) = \min_{C_k \in \mathcal{C}} (D(C_k \cup C_i))\}$;
    if $D \neq \emptyset$ **then**
      $C_m \in D$;
      $\mathcal{C} = \mathcal{C} \backslash C_i$;
      $C_m = C_m \cup C_i$;
    end
  end
until $n \neq |\mathcal{C}|$;

Chapter 5

Centrality

Once we have the clusters extracted, it is very interesting to know, how the power is distributed within the clusters, and how the power is distributed among the groups. The power is an index of the importance of the actor in the network and allows to draw conclusions about the activity of the actors, showing if they are passive or active. Concerning the distribution of power in the complete network, it shows if the network is homogeneous or heterogeneous.

5.1 Overview

Power is the consequence of patterns of relations [Han98]. We will discuss the degree-, betweenness-, closeness- and flow-betweenness-centrality, as the basic centrality measurements. The problems of the closeness-centrality are discussed in the Section 5.1.4. The Bonacich power index is introduced as an alternative measurement, but is not included in the analysis.

To understand the principles of a power centrality measurement, we take a look at three basic types of networks, the *star, line,* and *circle* (see Figure 5.1). Having a short look at the basic networks, a viewer would probably say, that actor A has the most power in the star-network. In the following we should find enough arguments that validate our assumption.

We then discuss how the centrality measurement can be adopted to calculate the group-centrality, ending the chapter with the algorithms to calculate the basic centrality measurements.

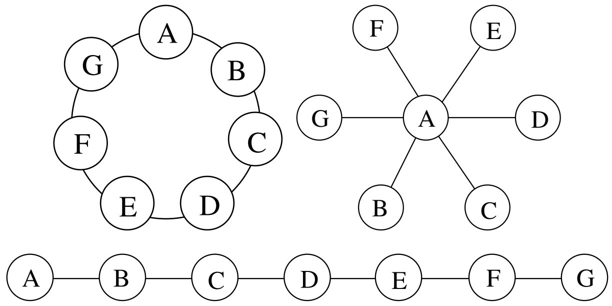


Figure 5.1: The upper left picture shows the circle, next to it the star, and the diagram at the bottom shows the line-network

5.1.1 Degree Centrality

The idea of the degree centrality is, that the more ties an actor has, the more opportunities he has. In our example, actor A can access resources from every other actor in the star network. But the other actors muss exchange with actor A in order to be able to exchange at all. Actors with a higher degree have more choices. This autonomy makes them less dependent on any specific other actor, and hence more powerful.

Considering the circle-network, every actor has exactly the same number of neighbours, so all actors are equally advantaged or disadvantaged. Considering the line-network and the degree centrality, only the actors at the end (A and G) have disadvantaged positions compared to the rest of the nodes, as they have only one direct neighbour.

Let $n = |V|$ be the number of actors in the graph $G = (V, E)$, then the value of the degree centrality of a node v is defined as

$$C_D(v) = \frac{\deg(v)}{n-1} \tag{5.1}$$

where the function $deg(v)$ may be in-degree, out-degree, in-degree + out-degree ..., defining different degree centrality functions.

The algorithm to calculate the degree-centrality is discussed in the Section 5.4.1.

5.1.2 Betweenness Centrality

The basic idea of the betweenness is if an actor is part of many shortest paths, then he is capable of controlling the communication among others. This makes him more powerful.

In the star-network, actor A can block any communication between two actors. So every other actor is highly dependent on A, making A powerful.

Considering the circle-network, no one has a more advantaged position, compared to the others, as everyone lies on of the same number of shortest paths.

Actors in the line-network have more power the closer they are to the centre, as they control more shortest paths than actors closer to the end of the line.

Let $\sigma_{st}(v)$ be the number of shortest paths between node $s \in V$ and node $t \in V$ that pass node $v \in V$, and let σ_{st} be the total number of geodesics between s and t. Then can we define the value of the betweenness centrality of a node v as

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

The algorithm to calculate the betweenness-centrality is discussed in the Section 5.4.2.

5.1.3 Flow-betweenness Centrality

The betweenness centrality only takes care of the shortest path between two actors. But if an actor blocks the shortest path between two actors, and another path exists, it is very likely to happen that an alternative path will be taken. In general, actors may use all of the pathways connecting them, rather than just geodesics. The flow approach to centrality expands the notion of betweenness centrality. It assumes that actors will use all pathways that connect them (proportionally to the length of the pathways).

If $\mathcal{F}_{st}$ denotes the maximal flow between the node $s \in V$ and the node $t \in V$, then $f_{st}(v)$ is the flow over $v \in V$, in respect of $\mathcal{F}_{st}$. Then we can define the flow betweenness of a node v as

$$C_F(v) = \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v,s\}} f_{st}(v). \tag{5.3}$$

In this definition, we do not take care of the length of the pathway. One method would be dividing C_F by the sum of the length of the geodesics between s, v and v, t divided by the length of the geodesic between s and t:

$$C_F(v) = \frac{\sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v,s\}} f_{st}(v)}{\frac{\text{dist}(s,v)+\text{dist}(v,t)}{\text{dist}(s,t)}}. \tag{5.4}$$

The algorithm to calculate the flow-betweenness is discussed in the Section 5.4.3.

5.1.4 Closeness Centrality

Actors who are able to reach other actors at shorter path length, or who are more reachable by other actors at shorter path lengths, have a favoured position. In the star-network again actor A has the best position, as he is closest to all other actors. Taking a look at the circle-network, we see that all the actors have an identical distribution of closeness, as for each actor the sum of distances to the other actors is identical. Actor D is the most powerful in the line network, followed by the couple of C,E, then B,F and finally A,G. Let $dist(v, t)$ be the length of the geodesic between the actors v and the actor t, then we can define the closeness centrality of a node v as

$$C_C(v) = \frac{1}{\sum_{t \in V} \text{dist}(v, t)} \tag{5.5}$$

or

$$C_C(v) = \sum_{t \in V} \text{dist}(v, t). \tag{5.6}$$

The problem with this centrality is, it does not really handle isolated nodes, or completely separated sub-graphs. If unreachable nodes have a distance of infinity, then it is hard to distinguish between nodes that are completely isolated and nodes that have a few neighbours and some nodes they can not reach. Assigning the value 0 for unreachable actors would make a disconnected group of two actors most powerful, as it would have a closeness centrality value of 1.

5.1.5 The Bonacich's power index

Phillip Bonacich completely questions the ideas of centrality as discussed here. His idea is, that if an actor A is connected to central others, he is more influential, as he can reach more actors with less expense. But if the others are themselves well connected, they are not highly dependent on the actor A. If, on the other hand, the others are not well connected, then they depend more on the actor A, making him more powerful. Bonacich argued that being connected to connected others makes an actor central, but not powerful, but being connected to others that are not well connected makes an actor powerful. Bonacich proposed that both centrality and power are a function of the connections of the actors in one's neighbourhood. The more connections the actors in the neighbourhood have, the more central an actors is. The fewer connections the actors in the neighbourhood have, the more powerful the actor is.

5.2 Group centrality

After we have done the clustering and calculated the centrality value for every node in its cluster, we are interested in the distribution of power among the groups. This should be possible using existing individual measures. Everett and Borgatti introduce a method for calculating group centralities, using existing measures [EB99].

5.2.1 General Principles

Any group measure is a proper generalisation of the corresponding individual measure, such as, when applied to a group consisting of a single individual, the measure yields the same answer as the individual version. An immediate consequence of this requirement is that group centrality is not measured by computing centrality on a network of relationships among groups. Instead, the centrality of a group is computed directly from the network of relationships among individuals. A side benefit of this approach is that there are no problems working with overlapping groups, where one individual can belong to many groups.

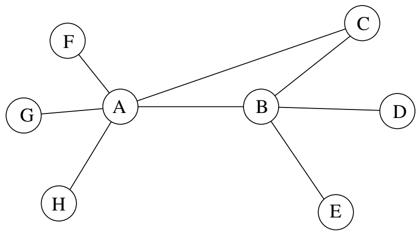


Figure 5.2: Group Degree Centrality: Number of non-group nodes that are connected to group members. Let $\{A, B\}$ be a cluster, then the group degree centrality value of the group is 6. Normalised by the number of non-group nodes, the centrality value is 1.

5.3 Method

We will use the following graph as an example to define *group degree centrality*, and then define the *group betweenness centrality*.

Degree betweenness centrality

We define the *group degree centrality* as the number of non-group nodes that are connected to group members. Multiple ties to the same node are counted only once. Hence, the centrality of the group consisting of the node A and B is 6. Group degree centrality can be normalised by dividing the value by the number of non-group nodes, we then receive 1 as centrality value of the group $\{A, B\}$.

Group betweenness centrality

Let C be a subset of a graph $G = (V, E)$. Let σ_{uv} be the number of geodesics connecting u and v and $\sigma_{uv}(C)$ be the number of geodesics connecting u and v passing through C. Then the *group betweenness centrality* of C denoted by $C_B(C)$ is given by

$$C_B(C) = \sum_{u<v,\, u,v \notin C} \frac{\sigma_{uv}(C)}{\sigma_{uv}} \tag{5.7}$$

One way to compute this measure is:

1. count the number of geodesics between every pair of non-group members, yielding a node-by-node matrix of counts

2. delete all ties involving group members and redo the calculation, creating a new node-by-node matrix of counts

3. divide each cell in the new matrix by the corresponding cell in the first matrix

4. take the sum of all these ratios

5.4 Algorithms

We implemented the basic centrality measures of *degree* and *betweenness* centrality, and one alternative to betweenness, the *flow betweenness* centrality.

5.4.1 Degree centrality

Let $G = (V, E)$ be a graph, and $N(v) = \{u | u \in V, (u, v) \in E \vee (v, u) \in E\}$ be the set of neighbours $u \in V$ of $v \in V$. We define the set of edges leaving a cluster $E_{\overline{C}} = \{(u, v) | (u, v) \in E, (u \in C \wedge v \in V \backslash C) \vee (u \in V \backslash C \wedge v \in C)\}$ and the set of neighbours u of v that are in the same cluster $N_C(v) = C \cap N(v)$.

The values for $C_D(v)$ and $C_D(C)$ may be normalised as suggested here (see Algorithm 7). One alternative method would be to normalise the values by the theoretical maximum given by a star-network (see Figure 5.1) of the same size. Normalising by the theoretical maximums allows to compare the centrality values of different networks. As we combine the three centrality measures by a weighted sum, we decided to normalise each value by its maximum.

5.4.2 Betweenness centrality

A fast algorithm to calculate betweenness centrality was introduced by Ulrik Brandes. The description in this section follows the publication "*Faster Evaluation of Shortest-Path Based Centrality Indices*" of Ulrik Brandes. Proofs are omitted here, as they give no additional information. For any proof or a more detailed description see [Bra00].

Algorithm 7: Degree Centrality including Group Degree Centrality

---

Data: $\mathcal{C}$
Result: $\forall C_i \in \mathcal{C} : C_D(C_i)$, $\forall v \in C_i : C_D(v)$
foreach $C_i \in \mathcal{C}$ **do**
  foreach $v \in C_i$ **do**
    $E_{\overline{C}} = \emptyset$;
    $N(v) = \{u | u \in V, (u, v) \in E \wedge (v, u) \in E\}$;
    foreach $u \in N(V)$ **do**
      if $u \in C_i$ **then**
        $N_C(v) = N_C(v) \cup \{u\}$;
      else
        if $(u, v) \in E$ **then**
          $E_{\overline{C}} = E_{\overline{C}} \cup \{(u, v)\}$;
        end
        if $(v, u) \in E$ **then**
          $E_{\overline{C}} = E_{\overline{C}} \cup \{(v, u)\}$;
        end
      end
    end
    $C_D(v) = \frac{|N_C|}{|C_i|}$;
  end
  $C_D(C) = \frac{|E_{\overline{C}}|}{|V| - |C_i|}$;
end

The betweenness centrality is an essential measurement for the analysis of social networks. The fastest known algorithms require $\Theta(n^3)$ time and $\Theta(n^2)$ space, where $n = |V|$ is the number of vertices. The algorithm introduced by Ulrik Brandes requires $\mathcal{O}(n+m)$ space and runs in $\mathcal{O}(n(m+n))$ or $\mathcal{O}(n(m+n\log n))$ time for unweighted or weighted graphs, where $m = |E|$ is the number of edges.

To obtain the betweenness centrality index of a vertex v, we simply have to sum the pair-dependencies of all pairs on that vertex

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{5.8}$$

where σ_{st} is the number of shortest paths between $s \in V$ and $t \in V$, and $\sigma_{st}(v)$ is the number of shortest paths between s and t passing the node $v \in V$.

Therefor betweenness centrality is traditionally determined in two steps.

1. compute the length and number of shortest paths between all pairs
2. sum all pair-dependencies

Corollary 5.1 *Given a source $s \in V$, both the length and number of all shortest paths to other vertices can be determined in time $\mathcal{O}(m + n\log n)$ for weighted, and in time $\mathcal{O}(m+n)$ for unweighted graphs.*

The Corollary 5.1 tells us, that the complexity of determining the betweenness centrality is dominated by the second step, the $\theta(n^3)$ time summation and $\theta(n^2)$ storage of pair-dependencies. This situation is remedied by the algorithm introduced by Ulrik Brandes.

Accumulation of Pair-Dependencies

Let $G = (V, E)$ be a graph, and let $\sigma_{st} = \sigma_{ts}$ denote the number of shortest paths between $s \in V$ and $t \in V$, where $\sigma_{ss} = 1$ by convention. Then $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass the vertex $v \in V$. The distance between s and t is described by $d_G(s,t)$, where $d_G(s,s) = 0$ for every $s \in V$, and $d_G(s,t) = d_G(t,s)$ for $s, t \in V$. Given pairwise distances and shortest paths counts, the *pair-dependency* $\delta_{st}(v) = \frac{\sigma_{st(v)}}{\sigma_{st}}$ of a pair $s, t \in V$ on an intermediary $v \in V$. The lemma and theorem are taken from [Bra00].

We define the *dependency* of a vertex $s \in V$ on $v \in V$ as

$$\delta_{s\bullet} = \sum_{t \in V} \delta_{st}(v), \tag{5.9}$$

Lemma 5.1 *If there is exactly one shortest path from $s \in V$ to each $t \in V$, the dependency of s on any $v \in V$ obeys*

$$\delta_{s\bullet}(v) = \sum_{w: v \in P_s(w)} (1 + \delta_{s\bullet}(w)) \tag{5.10}$$

where

$$P_s(v) = \{u \in V | (u, v) \in E, d_G(s, v) = d_G(s, u) + \omega(u, v)\}$$

is the set of predecessors *of a vertex v.*

Theorem 5.1 *The dependency of $s \in V$ on any $v \in V$ obeys*

$$\delta_{s\bullet} = \sum_{w: v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{st}(w)).$$

With this theorem, we can determine the betweenness centrality index by solving the single-source shortest-paths problem for each vertex. At the end of each iteration, the dependencies of the source on each other vertex are added to the centrality score of that vertex.

5.4.3 Flow-betweenness centrality

Let $G = (V, E)$ be the graph, where V is the set of vertices $v \in V$ and E is the set of edges. The $maxFlow$-function was implemented using the lift-to-front algorithm, see [CLR99] for a description of the algorithm. With c_f we describe the array of values returned by $maxFlow$, and with $c_f(v)$ we can access the max-flow-value of the node $v \in V$, and $C_{F,C_i}(v)$ is the flow-betweenness of the node v in the cluster C_i. In this algorithm the length of the path the node v is part of is not taken into account. A rule to normalise the value of $C_{F,C_i}(v)$ is not given here, but we normalise any centrality index by the maximum. The flow index of each group is described by $C_F(C_i)$.

To calculate the flow index of a node in a group, the maximum flow within the cluster is calculated for every combination of $s, t \in C_i, s \neq t$. The flow values for every node in the cluster are added together, receiving the flow centrality value.

Algorithm 8: Flow-Betweenness Centrality including Group Flow-Betweenness Centrality

```
Data: 𝒞
Result: ∀C_i ∈ 𝒞 : C_F(C_i), ∀v ∈ C_i : C_F(v)
foreach C_i ∈ 𝒞 do
    foreach s ∈ V\C_i do
        foreach s ≠ t ∈ V\C_i do
            c_f = maxFlow(G, s, t);
            foreach v ∈ C_i do
                C_F(C_I) = C_F(C_I) + c_f(v);
            end
        end
    end
    foreach s ∈ C_i do
        foreach s ≠ t ∈ C_i do
            V' = C_i;
            E' = {(u, v)|u ∈ V', v ∈ V', (u, v) ∈ E};
            G' = (V', E');
            c_f = maxFlow(G', s, t);
            foreach v ∈ C do
                C_{F,C_i}(v) = C_{F,C_i}(v) + c_f(v);
            end
        end
    end
end
```

Chapter 6

Visualisation

After clustering the graph, and determining the distribution of power or centrality, the extracted information should be visualised.

The visualisation should support the social scientist in two ways. It should help the scientist with the analysis concerning the social network and also support the presentation of the results of the analysis, as a good visualisation can provide additional comprehension aids.

The information should therefor be clearly visible, without much interpretation or searching needed.

6.1 Data and Properties

Before we discuss the visualisation, we should recapitulate what kind of data we have, and define the properties of the visualisation.

6.1.1 Data

Let $G = (V, E)$ be a graph representing the social network. We then have nodes, representing actors, and edges representing the relations between actors. The relations between actors are directed. Two actors A, B have a relation, if the corresponding vertices $v_A, v_B \in V$ are incident, that is $(v_A, v_B) \in E$ or $(v_B, v_A) \in E$. The nodes are combined in clusters, which denote separated groups within the graph. A node can be part of more than one group or cluster. Each node has a centrality value in its cluster. If a node is part of more than one cluster, it may have a different centrality value in

Uniformity	all nodes should be positioned equally spaced around the centre
Distance	the distance of a node must be in direct relationship of the difference of its centrality value and the value of the centre node(s)
Convexity	the nodes should be positioned in such a way, that the crossings of a node with the edges are minimised
Centre	the most central node of a group must be seen immediately without any interpretations needed

Table 6.1: Visualisation properties

each cluster. Each cluster has a centrality value representing the centrality of the group according to the rest of the graph. There is one or more central actors in each group. A central actors is an actor with the highest centrality value in its group. The number of central nodes depends on the distribution of power in the cluster.

6.1.2 Properties

There are two levels of information, which have to be visualised. The lower level is each group, which nodes must be layouted. The higher level are the groups, which must be positioned.

To receive a good visualisation of the nodes within a cluster, we define the following four properties, which must be fulfilled for each group-layout (see Table 6.1). The *uniformity* is needed, so that a cluster can be easily distinguished from others, as each group is regionally limited by its members, and no node in a cluster overlaps with any other node in the cluster. We require the property of *distance*, as the centrality of a node in each group, should be seen easily in comparison to other members of the group. The layout should be *convex*, so the overlapping of edges through nodes is minimised, as the edges run mainly within the layout of the group. To see the most central node(s) at once, only the nodes with the highest centrality value are positioned in the *centre* of the group. The central nodes are also scaled in size, so they have a second property that distinguishes them from the non-central nodes.

We say, a node is positioned *nicely*, if it fulfils the visualisation properties. A shared node is positioned *nicely*, if it fulfils the visualisation properties for each cluster it is in.

6.2 Problems

We first define a *clan* and the different types of nodes and clusters, before discussing the the problems that must be handled if more than one cluster with overlapping is to be visualised.

Definition 6.1 (clan)
Let $G = (V, E)$ be a graph. Let $\mathcal{C} = \{C_i\}_{i\in\mathbb{I}}$ be a valid clustering of G, such that $\bigcup_{i\in\mathbb{I}} C_i = V$ but the $C_i \in \mathcal{C}$ may not be disjunct. A *clan* is a subset $\Phi \in \mathcal{C}$ such that the elements $C_i \in \Phi$ *overlap*, that is $\bigcap_{C_i\in\Phi} C_i \neq \emptyset$, and $\forall_{C_i\in\mathcal{C}} C_j \notin \Phi : C_j \cap \left(\bigcap_{C_i\in\Gamma} C_i\right) = \emptyset$. The set of clans is denoted by Φ.

6.2.1 Types of Nodes

A cluster $C_i \in \mathcal{C}$ can be divided into central $\mathcal{Z}_{C_i}$ and non-central nodes $\overline{\mathcal{Z}}_{C_i}$. So we can distinguish between four different classes of nodes.

Type A Node $v \in V$ is part of only one group:

$$v \in \mathcal{T}_A^A : [\,\exists_1 C_i \in \mathcal{C} : v \in C_i, v \notin C_j \forall j \neq i\,].$$

Type B Node $v \in V$ is part of any number of clans, having a non-central position in each cluster included in the clan:

$$v \in \mathcal{T}_B^A : [\,\exists\Gamma \in \Phi : \forall C_i \in \Gamma : v \in \overline{\mathcal{Z}}\,].$$

Type C Node $v \in V$ is part of any number of clans, having a central position in each cluster included in the clan:

$$v \in \mathcal{T}_C^A : [\,\exists\Gamma \in \Phi : \forall C_i \in \Gamma : v \in Z_i\,].$$

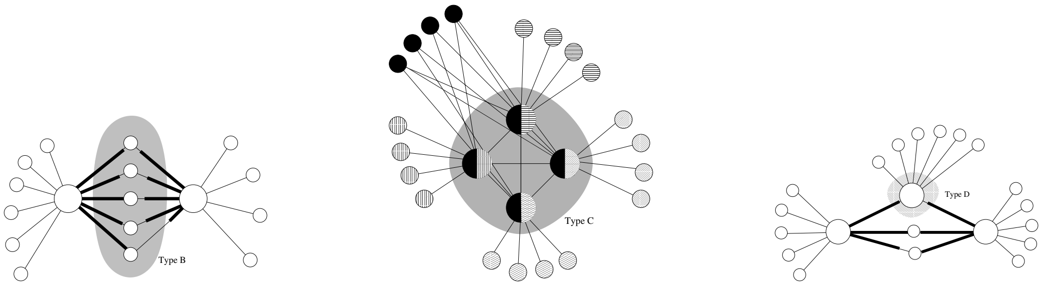


Figure 6.1: From left to right we see actor type $\mathcal{T}_B^A, \mathcal{T}_C^A$ and $\mathcal{T}_D^A$. Shared nodes are marked by the filled areas.

Type D Node $v \in V$ is part of any number of clans, and may or may not be in a central position:

$$v \in \mathcal{T}_D^A : [\, \exists \Gamma \in \Phi : \forall C_i \in \Gamma : v \in C_i \,].$$

Types B to D are displayed in Figure 6.1.

6.2.2 Types of Clusters

The basic types of clusters directly follow from the definition of the types of nodes.

Type A Cluster $C_i \in \mathcal{C}$ is a cluster that only includes type A actors:

$$C_i \in \mathcal{T}_A^G : [\, \forall v \in C_i : v \in \mathcal{T}_A^A \,].$$

Type B Cluster $C_i \in \mathcal{C}$ is a cluster with type A or type B actors:

$$C_i \in \mathcal{T}_B^G : [\exists v_i \in C_i : v_i \in \mathcal{T}_B^A, \forall_{v_j \notin \{v_i\}} v_j \in C_i : v_j \in \mathcal{T}_A^A].$$

Type C Cluster $C_i \in \mathcal{C}$ is a cluster with type A or type C actors:

$$C_i \in \mathcal{T}_C^G : [\exists v_i \in C_i : v_i \in \mathcal{T}_C^A, \forall_{v_j \notin \{v_i\}} v_j \in C_i : v_j \in \mathcal{T}_A^A].$$

Type D Cluster $C_i \in \mathcal{C}$ is a cluster with type A or type D actors:

$$C_i \in \mathcal{T}_D^G : [\, \forall v \in C_i : v \in \mathcal{T}_A^A \text{ or } v \in \mathcal{T}_D^A \,].$$

A cluster can have any combination of the basic types, so we must take care of them too.

Type E Cluster $C_i \in \mathcal{C}$ is a cluster of type B and C¿

$$C_i \in \mathcal{T}_E^G : [\exists v_i, v_j \in C_i : v_i \in \mathcal{T}_B^A, v_j \in \mathcal{T}_C^A, \forall_{\substack{v_k \notin \{v_i\} \\ v_k \notin \{v_j\}}} v_k \in C_i : v_k \in \mathcal{T}_A^A].$$

Type F Cluster $C_i \in \mathcal{C}$ is a cluster of type B and D:

$$C_i \in \mathcal{T}_D^G : [\exists v_i, v_j \in C_i : v_i \in \mathcal{T}_B^A, v_j \in \mathcal{T}_D^A, \forall_{\substack{v_k \notin \{v_i\} \\ v_k \notin \{v_j\}}} v_k \in C_i : v_k \in \mathcal{T}_A^A].$$

Type G Cluster $C_i \in \mathcal{C}$ is a cluster of type C and D:

$$C_i \in \mathcal{T}_D^G : [\exists v_i, v_j \in C_i : v_i \in \mathcal{T}_C^A, v_j \in \mathcal{T}_D^A, \forall_{\substack{v_k \notin \{v_i\} \\ v_k \notin \{v_j\}}} v_k \in C_i : v_k \in \mathcal{T}_A^A].$$

Type H Cluster $C_i \in \mathcal{C}$ is a cluster of type B, C and D:

$$\begin{aligned} C_i \in \mathcal{T}_D^G : \quad & \exists v_i, v_j, v_k \in C_i : v_i \in \mathcal{T}_B^A, v_j \in \mathcal{T}_C^A, v_k \in \mathcal{T}_D^A, \\ & \forall_{\substack{v_r \notin \{v_i\} \\ v_r \notin \{v_j\} \\ v_r \notin \{v_k\}}} v_r \in C_i : v_r \in \mathcal{T}_A^A. \end{aligned}$$

Figure 6.2 gives an overview over the non-basic types of clusters.

6.2.3 Type B Analysis

In this section we discuss how nodes of type B $v \in \mathcal{T}_B^A$ are to be placed within two or three clusters $C_i, C_j, C_k \in \mathcal{T}_B^C$ of type B. The visualisation of overlapping clusters must still obey the visualisation properties, in particular the property of distance (see Table 6.1), so we must place the shared nodes accordingly.

Overlapping within 2 clusters

Let $\Phi = \{C_0, C_1\}$ be a clan of two clusters. And further, let $\mathfrak{D} = \{v \in V | v \in C_0, v \in C_1\}$ be the set of shared actors. We can then define the set of the most central common actors $\mathcal{V} = \{v \in \mathfrak{D} | d_0(v) + d_1(v) = \min_{u \in \mathfrak{D}}(d_0(u) + d_1(u))\}$,

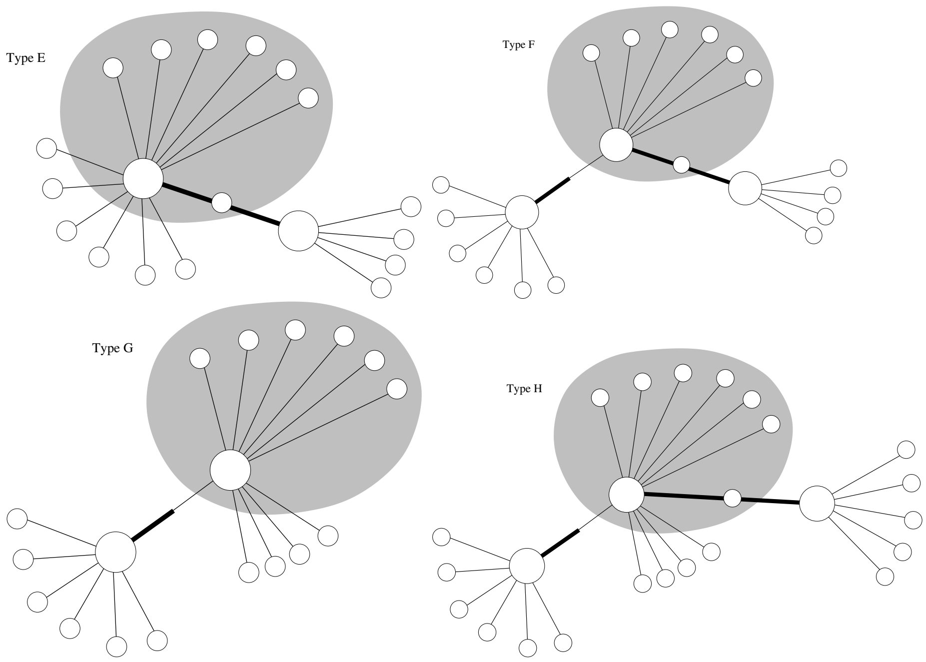


Figure 6.2: From upper left to lower right we see the non-basic cluster types E,F,G and H. The highlighted actors belong to the cluster we assigned the type to.

where $d_i(v)$ is the distance-value of the actor v in the cluster C_i, as defined in the Algorithm 9.

The distance between the centre of the groups is then defined by

$$D = \mathrm{dist}(C_0, C_1) = d_0(v) + d_1(v), \qquad v \in \mathcal{V}.$$

The following constraint must be valid for all nodes $v \in \mathcal{V}$ if we want the distance of v to have the same proposition as all the other nodes, that are part of the group.

$$\forall v \in \mathfrak{D}: \quad |d_0(v) - d_1(v)| \leq D$$

If this constraint is valid for every $v \in \mathfrak{D}$, we can place them well according to their distance-values, but we can not guarantee a nice placement, as Figure 6.3 demonstrates.

The following problems can occur

1. $|\mathcal{V}| > 1$
   If there is more than one actor with the smallest sum of distance-values, we will have a problem in placing the second one, as it would need the same position.

2. $\exists v \in \mathfrak{D}: \quad |d_0(v) - d_1(v)| > D$
   If there is an actor, who has distance-values that have a bigger difference than D, it can not be placed correctly, because we cannot keep the distances to both clusters exact.

3. $\exists v \in \mathfrak{D}: \quad d_0(v) > D \vee d_1(v) > D$
   If there is an actor, who has one distance-value bigger than D, it can not be placed nicely anymore, that means it will not be be clearly separated from the actors, that do not belong to both groups (see Figure 6.3). If both distance-values are bigger than D it will be placed somewhere, but not close to the other actors or their centre.

For all the other shared nodes, we need a super-edge. The super-edge overlays the original edge, with a length that indicates, where the node should be placed, if it was possible to position the node correctly (see Figure 6.1). We could solve the problems discussed above by scaling D, but that would make the distances between the centres larger, so a lot more space would be needed. This does not increase the readability of the resulting visualisation.

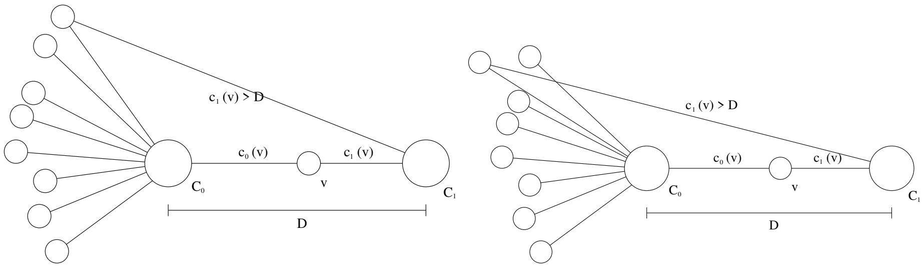


Figure 6.3: Possible overlap between 2 groups. Here we see an actor, which fulfils the constraint, but overlapping and non-overlapping actors are not separated clearly anymore

Overlapping in 3 groups

Let $\Gamma = \{C_0, C_1, C_2\}$ be a clan of three cluster, and $\mathfrak{O} = \{v | \forall C \in \Phi : v \in C\}$ be the set of shared actors. The set of actors with the smallest sum of distances is described by

$$\mathcal{V} = \{v \in \mathfrak{O} | \sum_{i=0}^{|\mathcal{C}|-1} d_i(v) = \min_{u \in \mathfrak{O}} (\sum_{i=0}^{|\mathcal{C}|-1} d_i(u))\}.$$

We also need the sets $\mathcal{V}$ and $\mathfrak{O}$ restricted to only two clusters,

$$\mathfrak{O}_{ij} = \{v \in V | v \in C_i \vee v \in C_j, \}$$

$$\mathcal{V}_{ij} = \{v \in \mathfrak{O}_{ij} | d_i(v) + d_j(v) = \min_{u \in \mathfrak{O}_{ij}} (d_i(u) + d_j(u))\}.$$

We define

$$D_{ij} = \text{dist}(C_i, C_j) = d_i(v) + d_j(v), \qquad v \in \mathcal{V}_{ij}$$

and as before we need

$$\forall v \in \mathfrak{O}_{ij} : \quad |d_i(v) - d_j(v)| \leq D_{ij}.$$

The centres of the clusters limit the area in which the shared nodes of the clusters have to be placed. In the centre of this area we would like to see the nodes of the set of the most central shared nodes $v \in \mathcal{V}$. All the other shared nodes have to be placed around them, not leaving the area defined by the centre of the groups. Nodes that are shared by only two clusters have to be placed as discussed before. If we layout three clusters, we still have to solve

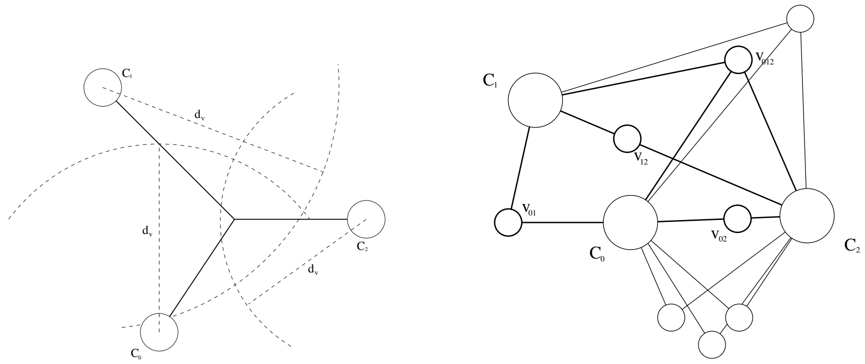


Figure 6.4: The centre of a cluster is denoted by C_i, and the distance value of a node $v_{ij} \in \mathcal{V}_{ij}$ or $v_{ijk} \in \mathcal{V}$ is described by d_{ij} or d_{ijk}. On the left we see, that for a given node $v \in \mathfrak{D} \backslash \mathcal{V}$ there must not always be a clearly defined position within the area limited by the centre of the groups. The right figure shows, that the pairwise overlapping can move the centres together, so that no node can be placed nicely anymore. If all nodes are to be placed according to their centrality values, then C_1 and C_2, must move towards C_0.

the problems with the overlapping of two cluster, as the three clusters can pairwise share nodes. Additionally we have to solve the problems of three overlapping clusters, described in the following.

First of all, it is not guaranteed that the the shared node $v \in \mathfrak{D}$ can be placed in the area limited by the centres of the three clusters, as their position is also determined by the pairwise overlapping (see Figure 6.4, right side). It can happen that the pairwise overlapping restricts the possible distance between the centres of the clusters, in such a way, that they have to move together too much. The central nodes $v \in \mathcal{V}$ are then not placeable within the area limited by the three centres. And even if this does not occur, the distance values for a node $v \in \mathfrak{D}$, $d_0(v), d_1(v), d_2(v)$ can be too large, so that a position within the area, limited by the group-centres can not be found without violating the property of *distance*. (see Figure 6.4, left side).

Even if we have only 2 clusters, and one of the shared nodes is a node of type D, we will have a problem in placing the other nodes, without loosening the visualisation properties. But if we loosen the visualisation constraints in some cases, the resulting picture will show misleading information. It therefor seems reasonable to look for another solution.

6.3 Conclusion

We have discussed the problems that occur, when having to layout a type B group, overlapping with one or two other type B groups that have no further overlapping. The complexity of visualising overlapping clusters arise from the range of possible combinations of clusters, that yield from the following properties:

1. a node can be part of any number of clusters
2. in each cluster a node can be central or non-central
3. a cluster can overlap with any number of other clusters
4. an overlapping cluster can overlap with any type of a cluster

It is therefor non-trivial to find a set of rules, that can handle all possible cases of overlapping groups. We tried to create a graph for each clan, representing the groups and their overlapping. Groups were represented by nodes, and the weighted edges between nodes represented different overlapping types. This graph was then layouted by a spring embedder. After that, the shared nodes were placed between the involved clusters, and the unshared nodes were placed, so that they did not intersect with the area of the shared nodes. The result was very un-pleasing, and it was very difficult to read the desired information from the resulting picture, as central nodes were moved away too far from the centre of their groups, and overlapping of three groups could not be handled. We therefor had to develop an alternative layout, which is introduced in the next section.

6.4 Single Cluster Layout

The algorithm to visualise a social network is first explained using a graph containing only one cluster. The data for the analysis is the Knoke information exchange data, taken from [Han98]. Figure 6.5 shows the graph before any analysis or visualisation is performed. We first have to define, what a *layout* in this context is.

Definition 6.2 (Layout)
Let $G = (V, E)$ be a graph, and let $G' \subseteq G$ be a subset of G such that $G' = (V' \subseteq V, E' \subseteq E)$ with $V' \neq \emptyset$. A *layout* of G' is defined by $\mathfrak{L}(G') =$

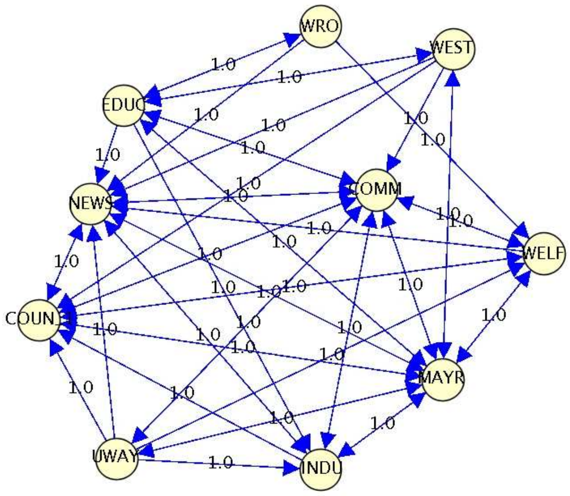


Figure 6.5: Knoke information exchange data

$\{\mathbb{C}, \mathcal{Z}, \overline{\mathcal{Z}}, c(v), p, \omega, \Delta_{min}, \Delta_{max}\}$ where $\mathbb{C} = \{(x_v, y_v) \in \mathbb{R}^2 | v \in V'\}$ is the set of coordinates for every node $v \in V'$, $\mathcal{Z} = \{v \in G' | c(v) = max_{u \in G'}(c(u))\}$ is the list of the central nodes with $c(v) \in \mathbb{R}$ as a valid centrality measure, and $\overline{\mathcal{Z}} = G' \backslash \mathcal{Z}$ is the list of non-central nodes. The elements in the list $u \in \overline{\mathcal{Z}}$ are put in descending order in respect to their centrality value $c(u)$. The centre and the orientation of the layout are given by $p = (x_l, y_l) \in \mathbb{R}^2$ and $\omega \in \mathbb{R}$, while the *distance parameters* $\Delta_{min}, \Delta_{max} \in \mathbb{R}$ determine the dimension of the layout, in respect of the centre nodes.

The values $\Delta_{min}, \Delta_{,max}$ limit the distance of a non-central node from a central node. In some cases, if for example the cluster has a large number of members, the nodes must be placed further away from the centre, as $\Delta_{min}, \Delta_{max}$ would indicate. In this case the distance parameters must be adopted.

The sets of central and non-central nodes $\mathcal{Z}, \overline{\mathcal{Z}}$ can be received using the centrality function $c(v) \in \mathbb{R}$. To layout a cluster, we therefor must determine the set of coordinates $\mathbb{C}$, in respect of the layout parameters $\mathcal{Z}, \overline{\mathcal{Z}}, c(v), p, \omega$.

Algorithm

The first step in the algorithm for the layout of a group (see Algorithm 9) is to position the central nodes $v \in \mathcal{Z}$ in a circular layout. This is not discussed here, as it would not provide any additional information. A circular layout algorithm is discussed in [dBETT99]. The radius of the resulting circle of central nodes is denoted by $r_c \in \mathbb{R}$. After that, the radius for the non-central nodes is calculated, respecting the radius of the inner circle r_c and the number of vertices in the set of non-central nodes $\overline{\mathcal{Z}}$. The resulting value is denoted by $r_{\overline{c}} \in \mathbb{R}$, and is the minimum distance for each non-central node from the centre of the layout $p \in \mathbb{R}^2$. The first non-central node is placed in the direction given by the orientation of the layout $\omega \in \mathbb{R}$. The next two nodes are placed in the same angle from the orientation, but on different sides. This is done for every following node, so that the cluster forms a kind of oval or heart (see Figure 6.6). The distance of the non-central node $v_{\overline{\mathcal{Z}}} \in \overline{\mathcal{Z}}$ from the nearest central node $v_{\mathcal{Z}} \in \mathcal{Z}$ is determined by the difference of the centrality values $c(c_{\mathcal{Z}}) - c(v_{\overline{\mathcal{Z}}})$ mapped to the difference of the distance values $\Delta_{max} - \Delta_{min}$. The result of the visualisation of the Knoke information exchange data is shown in Figure 6.7. One can see, that if the orientation is 0, thus the first node is exactly above the centre, then there is an ordering from top to bottom, in which the node is placed further down, as its centrality value decreases. In the following we will assume that the orientation ω is 0 for every layout.

Running time

The visualisation of the non-central nodes of one cluster in done in $\mathcal{O}(n)$ time (see Algorithm 9).

6.5 Multi cluster visualisation

The visualisation of more then just one cluster does differ only in two aspects. We have to take care of the shared actors, and each group must be positioned according to its group centrality value.

Algorithm 9: Layout

Data: $C_i \in \mathcal{C}$
Result: $\mathbb{C}$
$s = max_{v \in \overline{\mathcal{Z}}}(\mathrm{size}(v))$;
$d_{min} = s \cdot (1 + \epsilon) + r_c$;
$\delta_\alpha = \frac{2\Pi}{|\overline{\mathcal{Z}}|}$;
$\mathbb{C} = \emptyset$;
if $|\overline{\mathcal{Z}}| < 2$ **then**
    $r_{\overline{c}} = d_{min}$;
else
    $r_{\overline{c}} = \frac{d_{min}}{2 * \sin(\frac{\gamma}{2 \cdot |\overline{\mathcal{Z}}|})}$;
end
if $r_{\overline{c}} > \Delta_{min}$ **then**
    $\Delta_{max} = \Delta_{max} + \Delta_{min} - r_{\overline{c}}$;
    $\Delta_{min} = r_{\overline{c}}$;
end
foreach $v \in \overline{\mathcal{Z}}$ **do**
    $\alpha = \omega + (-1)^{\mathrm{i}_{\overline{\mathcal{Z}}}(v)} \cdot \mathrm{i}_{\overline{\mathcal{Z}}}(v) \cdot \delta_\alpha$;
    $d = \mathrm{MIN} + c(v) \cdot (\Delta_{max} - \Delta_{min}) + r_c$;
    $x_v = \sin(d)$;
    $y_v = \cos(d)$;
    $\mathbb{C} = \mathbb{C} \cup \{(x_v, y_v)\}$;
end

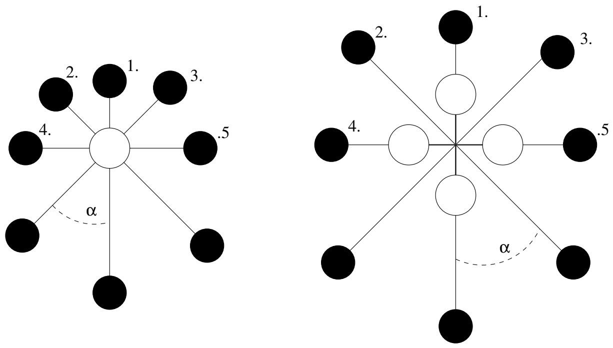


Figure 6.6: Placement order of the nodes in a group. The central node(s) are placed first. The most central non-central node is placed exactly above the central nodes. Following non-central nodes are placed alternatively left and right in equal angles, ordered from top to bottom. Higher central non-central nodes are placed on the top, lower central non-central nodes are placed below.

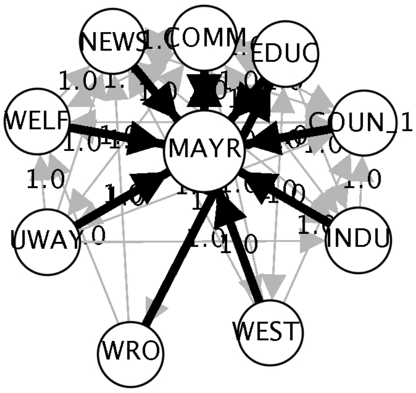


Figure 6.7: Knoke Information Data using Degree- & Betweenness- and Flow-betweenness-centrality.

6.5.1 Shared actors

A desired layout would position the shared actors only once in the complete graph, so that the position of the node indicates to what groups the node belongs and how central the node is in each group it belongs to. This kind of visualisation is non-trivial and the problems were discussed in the Section 6.2. In this section we discuss a simpler form of a visualisation for several overlapping clusters. Each cluster will be visualised with all members. This means, that a shared node will be drawn more then once. It must be visible, without much interpretation needed, to which groups the shared actor belongs. This can be done using the colour of every cluster the node is part of. Every cluster will have a unique colour, in detail, every node of a cluster shares the same colour. An actor, who is part of more than one cluster will have every colour of every cluster it belongs to, drawn in equally spaced arcs. An example is shown in the Figure 6.8, showing the relations between different parties in the city of Tübingen. Actor 8 functions as bridge between the small groups, and was able to unite the small groups in a voting against the large party.

6.5.2 Group layout

For the positioning of each cluster, we create a graph of clusters, in which every group is represented by a node, which we will call a *group-node* here. Every group-node will have a node centrality assigned, taken from the group centrality value. The graph of clusters is then treated like a single cluster, layouted by the same algorithm (see above). The centre of each cluster $p \in \mathbb{R}^2$ is then adopted to the position of the corresponding group-node.

6.6 Conclusion

The objective of the visualisation of a social network was to present a picture of the social network, that enables the viewer to easily read the extracted information. That is, the membership of each actor to each group, the power of each actor in each group, and the power of each group in the social network.

The membership of an actor to a group is shown by the zoned area each group uses. The members of each group share one colour, only the shared actors have the colour of each group they belong to.

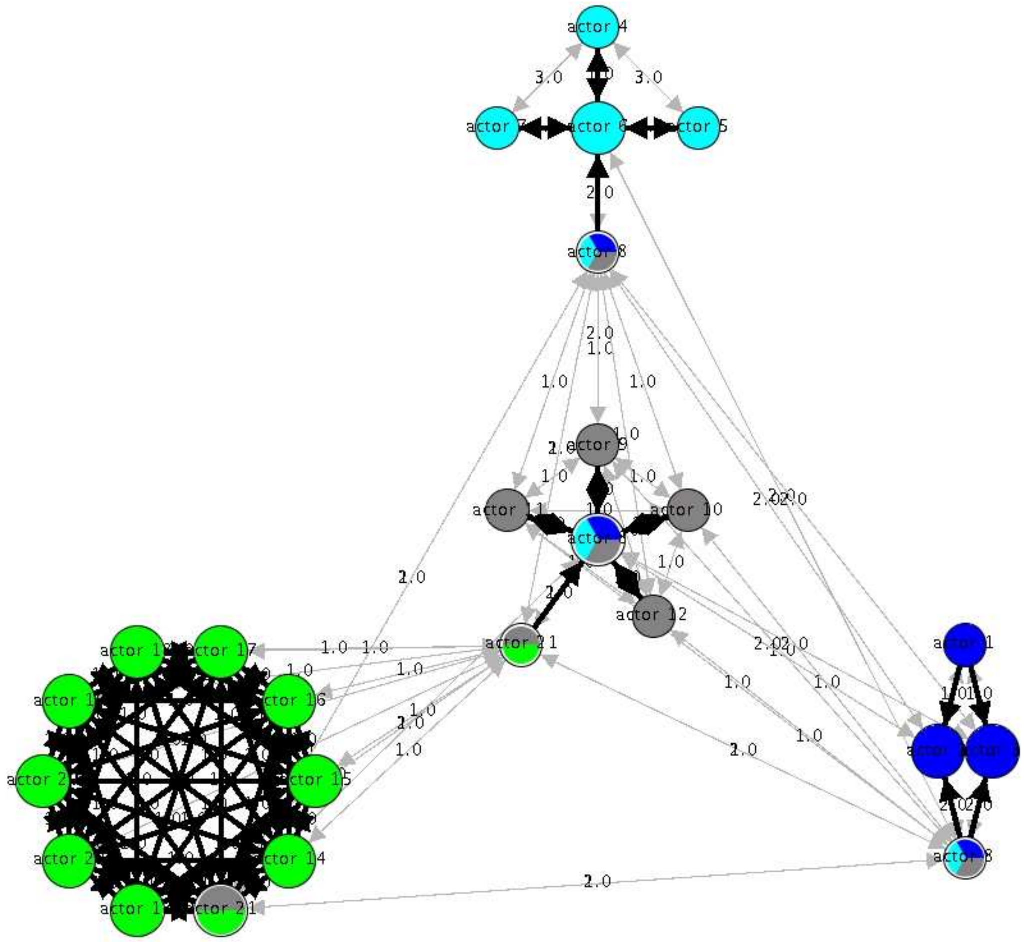


Figure 6.8: Visualisation of several clusters, with overlapping

The centrality of each actor is displayed by the distance of an actor to the centre of its group. A node is closer to the centre of its group, the higher its centrality value is. The centre of its group only includes the central node, that is, the nodes with the highest centrality value. If we have more than one central node, the central nodes form a circle around the centre. Central nodes are also scaled in size. All non-central nodes are placed in equidistant angles around the centre. The most central non-central node is placed exactly above the centre, then, in decreasing order of their centrality, the nodes are placed on the left and right side of the centre. The group forms a kind of oval or heart, depending on the distribution of the power within the group. Therefor we also have an ordering of the non-central actors, where actors with a higher centrality value are placed above actors with a lower centrality value.

The centrality of each group is visualised, using the same algorithm that is used to position the actors within a cluster. Central groups are placed central, and the ordering of the other groups is equivalent to the non-central nodes in a cluster.

We can therefor say, that the algorithm introduced here provides the viewer with easy to read information about the analysed social network.

Chapter 7

Conclusions & Future Work

The analysis of social networks was divided into two separated sections, the clustering and the calculation of centrality.

The first step is the clustering of the actors in the social networks, since clustering is a good method to analyse the structure of a social network. The clustering must correspond to the social-scientific understanding of groups. We did introduce several clustering algorithms, that were developed under this premise. The k-plex algorithm was chosen in consultation with the Ethnological Institute of Tübingen, as it correlates most with our understanding of a group. The results are very pleasing, as the k-plex algorithm described here is capable of clustering the graph according to the requirements of the social scientist, including the correct analysis of bridges.

The calculation of power or centrality is the second step of the social network analysis. We introduced several basic concepts of centrality, finally discussing three (degree-, betweenness-, flow-betweenness centrality) in detail. Ulirk Brandes introduced a method that is able to calculate the betweenness in a minimum of time and space [Bra00]. The flow-betweenness centrality measure was implemented using the lift-to-front algorithm discussed in [CLR99]. No optimisation was found yet, as the max-flow calculation for every combination of sources $s \in C_i$ and sinks $t \in C_i$ in a cluster C_i needed did not seem to be replaceable. Future work could find new algorithms for the calculation of flow-betweenness, reducing the high cost.

The visualisation of the social network analysis must be able to present the extracted information without much interpretation needed. The information, that is to be presented are the clusters, the centrality for each node and cluster, and the overlapping of clusters. We discussed the problems, that occur in the visualisation of overlapping cluster, finally introducing an algorithm capable of visualising the clusters and the centrality of each node in a cluster well. The algorithm can also be used to visualise the centrality of each group in the cluster. Shared nodes are placed more then once. The region of each node is divided into equally spaced arcs, each sharing the colour of a group the node belongs to. This method is good as long as a node is not part of too many clusters, as the arcs might get too small to identify the colours. Practically this did not happen. Future work could include the visualisation of overlapping clusters, placing shared nodes only once. We have briefly discussed a method here, using a graph to represent overlapping clusters. Each cluster was assigned a node in the graph. Each edge between two nodes represented one of four possible overlaps, that is, the node is central — centre, non-central — central, non-central — non-central and no overlap (indicated by an edge with weight 0). The graph was layouted using a spring embedder. Early results seemed promising, but the limited time did not allow any detailed evaluation of the problems that occurred. Future work could achieve good results in the visualisation of overlapping clusters.

Appendix A

$\mathcal{Y}$SocNet

The two standard software packages, *UCINET* as the analysis software, and *KrackPlot* (see Appendix B) mainly to visualise, are very good considering the results, but they are not easy to handle. The analysis is not interactive and the usage is not very intuitive. $\mathcal{Y}$SocNet was created trying to compensate that.

A.1 Requirements

Talking to social scientist, we extracted some requirements for a social network analysis program:

Ethnological Methods: The methods used in the program should correspond to social scientific understanding of groups and power.

Good Visualisation: The visualisation should support the analysis as well as the presentation of results.

Handling: The handling of the software should be intuitive and easy.

Interactive: A social scientist should be able to "play" with the data, that is, to easily be able to select what data to include/exclude from the analysis.

In the next section we will discuss, what we mean by handling and interactivity.

A.2 Application Properties

The goal is to develop a social network analysis tool, that allows a social network scientist to interactively analyse the data. A researcher should be able to load all the data into the analysis software, and then be able to decide what data should be included in the analysis, and to choose any combination of centrality functions wanted. That is what we understand under the term *interactive analysis*. Future work could include different clustering algorithms and other centrality functions to choose from.

Visualisation of overlapping clusters was not found in the standard software packages. The following list summarises the desired properties:

1. easy graphical input of social network data
2. input of independent relations-matrices
3. input of independent actor attributes
4. possibility to choose actors by attribute for the analysis
5. possibility to choose relations for the analysis
6. possibility to choose any combination of centrality functions for the analysis
7. possibility to visualise overlapping groups of actors

In the following we describe the architecture of the $\mathcal{Y}$SocNet application, explaining how the application properties were realised.

A.3 Architecture

There are three main parts in the architecture of $\mathcal{Y}$SocNet, the input, the analysis, and the analysis interface (see Figure A.1).

A.3.1 Data Input

There are three possible ways to input data: input file, input dialog, and graphical interface.

Input File

Let $n = |V|$ the number of actors in the social network. The input file can be best described by the following grammar $G = \{T, N, S\}$, where $T = \{Z, L, ',', ',', '.', '\_'\}$, and $N = \{WORD, NUMBER,$ $ATTRIBUTENAMES, ACTOR, ACTORLIST, REALATION,$ $MATRIX\}$.

$$
\begin{aligned}
L &= \{a, b, c, \ldots, z, A, B, C, \ldots, Z\} \\
Z &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \\
WORD &= L^{+}(L|\_) \\
NUMBER &= (-)^{?} Z^{+} (.Z^{+})^{?} \\
MATRIX &= NUMBER^{\{n^2\}} \\
ATTRIBUTENAMES &= WORD\,(\,,WORD)^{*} \\
ACTOR &= WORD\,(,WORD)^{*} \\
&\quad\ (NUMBER,\ NUMBER)^{?} \\
ACTORLIST &= ACTOR^{\{n\}} \\
RELATION &= (WORD)^{?}\,MATRIX \\
S &= ((ATTRIBUTENAMES)^{?} \\
&\quad\ (ACTORLIST)^{?})\ RELATION^{+}
\end{aligned}
$$

An input file must include at least a definition of a relation, where a relation is either just the matrix $M \in M_{n\times n}(\mathbb{R})$, or a relation-name followed by a relation matrix. If no attribute-names or actors are defined, then default actors and one default attribute-name will be created. The default attribute-name is the name of the actor, and each actor will have a name assigned; {"actor 1",...,"actor n"}. An actor is a list of attributes which can end with the coordinates of the actor an the screen. If actors are given, the number of actors must obey $n = |V|$. The number of attributes must be the same for each actor. If there are attribute-names defined, the number of attributes for each actor must obey the number of attributes-names (not including the coordinates). If no attribute-names are defined, default attribute-names will be created; {"attribute 0",...,"attribute 1"}.

Input Dialog

The input dialogs are an alternative way to input data. There are two dialogs, the attribute-dialog and the relations-dialog.

The attribute-dialog provides an interface, that allows to add and delete actors, to add and delete attributes, and to set the value of each attribute for each actor.

The relations-dialog provides an interface, that allows to add and delete relations (relation name and relation matrix) and to set or delete relations between actors.

The data is then stored in the *SocNetData*-Module.

A.4 Modules

A.4.1 SocNetData

The module *SocNetData* holds all the input data. It provides the graph of the social network with all desired information. The information wanted can be controlled by the elements of the *Data controlling*, which allow to select which attribute, and which relations are to be displayed and included in the analysis.

A.4.2 SocNet-Interface

The *SocNet-Interface* provides an user interface, that enables the user, to control the analysis. That is, the user can select, what relations are to be included in the analysis, what centrality functions are to be combined, and set some parameters for the clustering, including the initial value for k for the k-plex algorithm, and the desired merge percentage.

A.4.3 Clustering

The *Clustering*-module receives the current graph of the social network, as well as the user defined or default parameters for the clustering. It then clusters the graph, returning a list of clusters to the *SocNetAnalysis*-module.

A.4.4 Centrality

The *Centrality*-module receives the list of clusters from the *SocNetAnalysis*-module, the graph of the social network, and the list of centrality functions to use. It then calculates the centrality value for each node in each cluster, and the centrality of each group in the graph of the social network. A list with the centrality values is then returned.

A.4.5 Visualisation

The input to the *Visualisation*-module is the graph $G = (V, E)$ of the social network, one cluster and the centrality values for each node in the cluster. It then calculates the positions and size for each node, as central nodes are scaled in size. The layout structure (positions and sizes) is returned to the *SocNetAnalysis*-Module. For each cluster, the relation to the strongest neighbour of an actors is highlighted, so that more important relations are easily distinguishable from relations, that might not be as advantageous.

A.4.6 SocNetAnalysis

The analysis of the social network data is handled in the module *SocNet-Analysis*. At the beginning of each analysis, the data is taken from the module *SocNetData*, as the interaction can change the data after each analysis. The data is passed on to the clustering, which then returns the clustered graph. The list of clusters is passed on to the *Centrality*-module. The *SocNetAnalysis*-module receives the centrality values for each node in each group, as well as for each group in the graph. For each group, a layout is calculated, using the visualisation-module. After that a graph of groups is created. Each group will have one node in the graph of groups assigned. Each group-node will have a size equivalent to the size of the layout of its group. The centrality value of each group-node is equivalent to the centrality value of the corresponding group. The graph of groups is treated as one cluster, and the positions of each group-node is determined by the *Visualisation*-module. The centre of each group-layout is then moved to the coordinates of the assigned group-node. The sizes of the group-nodes are ignored.

A.5 Framework

$\mathcal{Y}$SocNet uses the $\mathcal{Y}$-framework, called *yFiles*. The *yFiles* is a collection of libraries written in Java and will run on every computer with a Java2 VM properly installed. The libraries provide powerful tools for viewing, editing, layouting and animating graph-like structures. The layouting-tools of *yFiles* are included in $\mathcal{Y}$SocNet, and can be used to layout the graph of the social network.

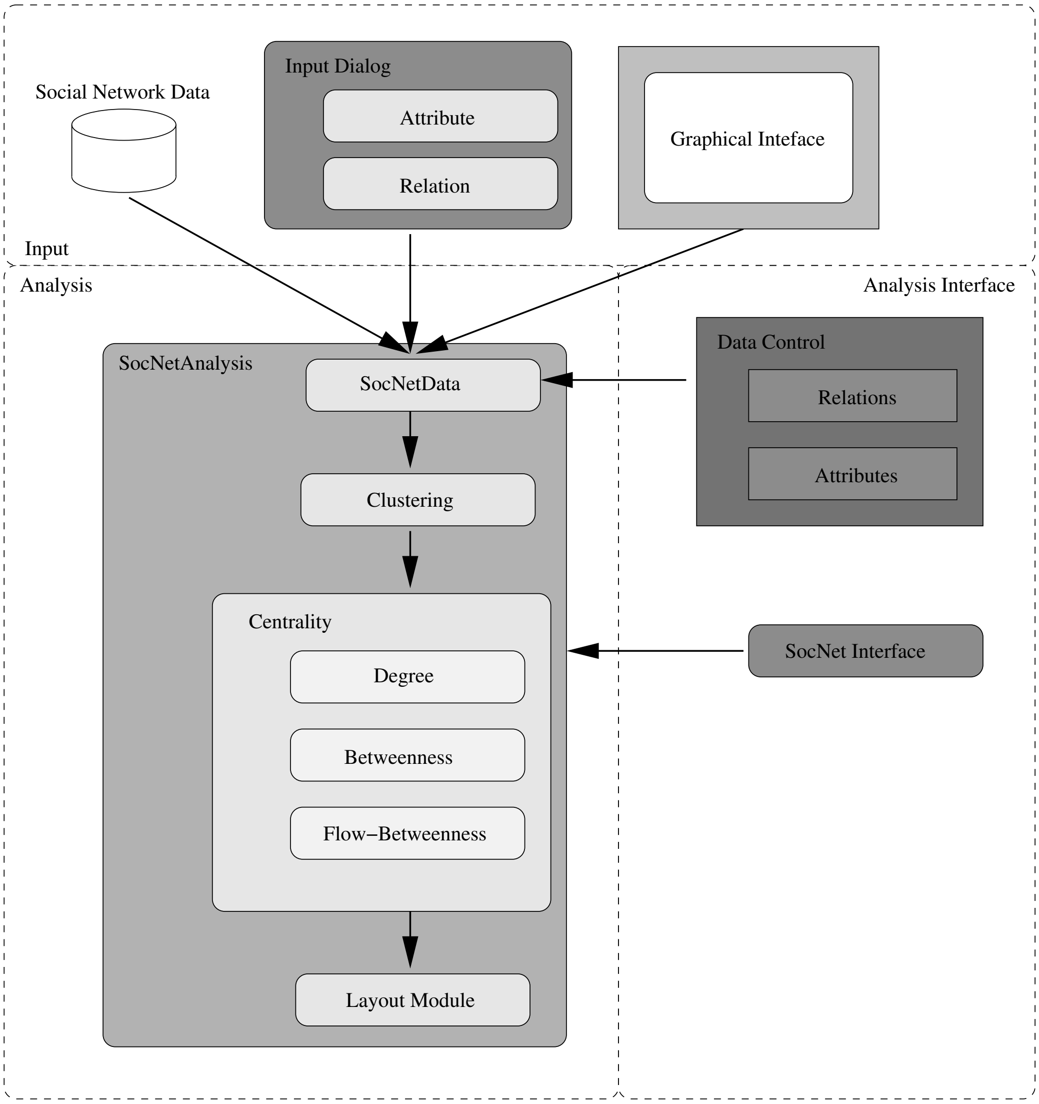


Figure A.1: Architecture of $\mathcal{Y}$SocNet . There are three possibilities to input data: file input, dialog input, and graphical interface input. The analysis is controlled by the *SocNetAnalysis*-Module. The *SocNet-Interface* allows to control the analysis by setting parameters for the clustering and the centrality functions. The *Data Control*-Module allows the selection of the desired relations and to select/deselect nodes depending on their attributes.

Appendix B

Related Work

B.1 Social Network Analysis Programs

B.1.1 UCINET

The first version of the program *UCINET* was developed in 1986 under the administration of Linton C. Freeman [BEF92],[Frea].

UCINET is a DOS program and the input is an ASCII-file. The input file contains matrices, that can be unlabelled or labelled. Concerning the analysis of social networks, it offers the following centrality measures

- Degree
- Betweenness
- Flow betweenness
- Information centrality
- Eigenvector centrality
- Bonacich's power index

and the following clustering algorithms

- Cliques
- N-cliques

- N-clans
- k-plexes
- k-cores
- Lambda sets
- Fractions
- f-groups

of which a few where described here. It can also group actors by structural similarity[Han98], such as

- Structural equivalence
- Automorphic equivalence
- Regular equivalence

The output can be visualised by *KrackPlot*.

B.1.2 KrackPlot

The visualisation program *KrackPlot* was originally designed by David Krackhardt. Jim Blythe and Cathleen McGrath are involved in the version 3 of the program. The program does not support network analysis, but has several layout-functions included [Hom]:

User: the user can specify the coordinates

Circle: the nodes are positioned in a circle, so that every edge is within the circle

Multidimensional Scaling: The main applications of factor analytic techniques are

1. to reduce the number of variables and
2. to detect structures in the relationships between variables, that is to classify variables

KrackPlot uses the geodesic distance between nodes for the factor analysis.

Random: random placement of the nodes

Anneal: is also known as spring embedder [Bra99], organic layouter or force directed layout layout (see Figure B.1).

Jiggle: is the same as anneal, but the diameter of the area that the node will move is smaller. It allows optimising the graph, after it has been layouted with annealing.

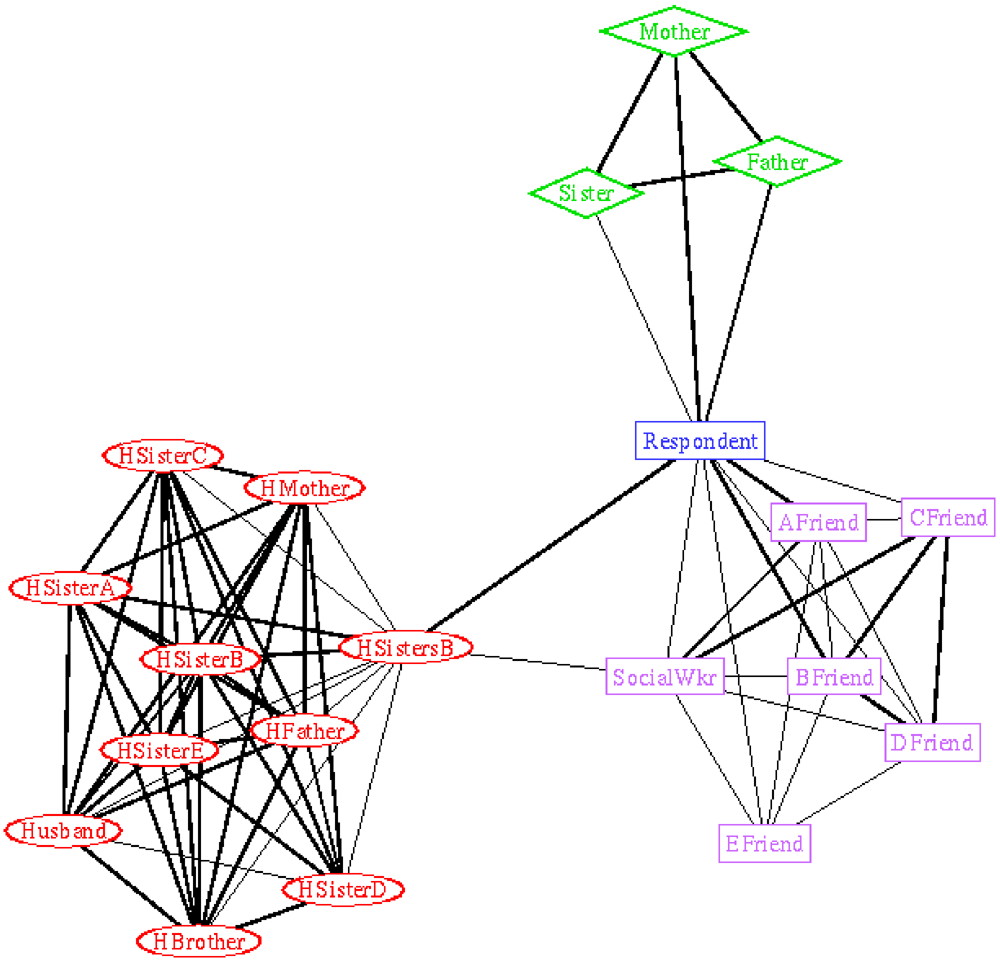


Figure B.1: A graph layouted by *KrackPlot* using simulated annealing

List of Algorithms

List of Figures

Bibliography

[BEF92] S.P. Borgatti, M.G. Everett, and L.C. Freeman. *UCINET IV*. Analytic Technologies, 1992.

[BKW99a] Ulrik Brandes, Patrick Kenis, and Dorothea Wagner. Centrality in Policy Network Drawings. *Proc. 7th Intl. Symp. Graph Drawing (GD'99) LNCS 1731*, pages 250–258, 1999.

[BKW99b] Ulrik Brandes, Patrick Kenis, and Dorothea Wagner. Explorations into the Visualization of Policy Networks. *Journal of Theoretical Politics*, 1(11):75–106, 1999.

[BLW96] Giuseppe Di Battista, Giuseppe Liotta, and Sue H. Whitesides. The Strength of Weak Proximity. Technical report, Universitá Degli Studi di Roma Tre, Dipartimento die Discipline Scientifice, 1996.

[BM99] S.P. Borgatti and M.G.Everett. Models of Core/Periphery Structures. Unpublished manuscript submitted to *Social Networks*, 1999.

[BMZ99] Vladimir Batagelj, Andrej Mrvar, and Matjaž Zaveršnik. Partitioning Approah to Visualization of Large Graphs. *Graph Drawing*, pages 90–97, 1999.

[Bor] Stephen P. Borgatti. A SOCNET Discussion on the Origions of the Term Social Capital. *CONNECTIONS*, 21(2):–46.

[Bra99] Ulrik Brandes. Layout of Graph Visualizations. Ph.D. Thesis. http://www.ub.uni-konstanz.de/kops/volltexte/1999/255, June 1999.

[Bra00] Ulrik Brandes. Faster Evaluation of Shortest-Path Based Centrality Indices. *Konstanzer Schriften in Mathematik und Informatik 120*, May 2000.

[BW00] Ulrik Brandes and Dorothea Wagner. Contextual Visualization of Actor Status in Social Networks. In *Data Visualization 2000*, pages 13–22. 2nd Eurographics/IEEE TVCG Symp. Visualization (VisSym'00), Springer, 2000.

[CLR99] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, 22th edition, 1999.

[dBETT99] Giuseppe di Batitista, Peter Eades, Roberto Tamassia, and Ioanning G. Tollis. *Graph Drawing*. Prentice-Hall, Inc., 1999.

[DFK+] P. Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. Clustering in large graphs and matrices.

[EB] Martin G. Everett and Stephen P. Borgatti. Peripheries of Cohesive Subsets.

[EB99] M.G. Everett and S.P. Borgatti. The Centrality of Groups and Classes. *Joural of Mathematical Sociology*, 23(3):181–201, 1999.

[ESB99] Jubin Edachery, Arunabha Sen, and Fran J. Brandenburg. Graph Clustering Using Distance-k Cliques. *Graph Drawing*, pages 98–106, 1999.

[Frea] Linton C. Freeman. UCINET - Network Analysis Software - Webpage. http://eclectic.ss.uci.edu/˜lin/ucinet.html.

[Freb] Linton C. Freeman. Visualizing Social Networks. *Journal of Social Structure*.

[Fre97] Linton C. Freeman. Uncovering Organizational Hierachies. *Computational & Mathematical Organization Theorie*, 1(3):5–18, 1997.

[FWK98] Linton C. Freeman, Cynthia M. Webster, and Deirdre M. Kirke. Exploring social structure using dynamic three-dimensional color images. *Social Network*, 20:109–118, 1998.

[GWW] Yoram Gdalyahu, Daphna Weinshall, and Michael Werman. Stochastic Clustering and its Application to Image Segmentation.

[Han98] Robert A. Hanneman. Introduction to Social Network Methods. http://wizard.ucr.edu/˜rhannema/networks/text/textindex.html, 1998.

[Har69] Frank Harary. *Graph Theorie*. Addison-Wesley, 1969.

[Hom] KrackPlot Homepage. http://www.heinz.cmu.edu/˜krack/.

[Jan99] Dorothea Jansen. *Einführung in die Netzwerkanalyse*. leske + budrich, 1999.

[JB] Robert O. Johnson and John Boyd. *e*-centrality. http://eclectic.ss.uci.edu/ rjohns/ecent.htm.

[Lea] Sonia Leach. Singular Value Decomposition – A Primer.

[MBK] Cathleen McGrath, Jim Blythe, and David Krackhardt. Seeing Groups in Graph Layouts.

[MC] Peter R. Monge and Noshir S. Contractor. Emergence of Communication Networks. A chapter prepared for Publication in Jablin, F.M., & Putnam, L.L. (Eds.) *Handbook of Organizational Communications* (2nd Ed.). Thousand Oaks, CA. Sage. 1999.

[Sch] Robby Schönfeld. *k*-layer Straightline Crossing Minimization by Speeding upn Shifting.

[Sch89] Thomas Schweizer. *Netzwerkanalyse*. Dietrich Reimer Verlag, 1989.

[ST] Janed M. Six and Ioannis G. Tollis. Improved Graph Drawing Via Clustering.

[Ved98] Balázs Vedres. Locked in Centrality. *Sunbelt XVIII and 5th European International Conference on Social Networks*, 1998.