University of Szeged

Faculty of Science and Informatics

Department of Computational Optimization

Methods for the description and analysis of processes in real-life networks

PhD Dissertation

András Bóta

Supervisors:
Miklós Krész, Dr.
András Pluhár, Dr.

Szeged, 2014.

Abstract

The roots of graph theory lead back to the puzzle of Königsberg's bridges. In 1736 Leonhardt Euler published a paper on this problem, and also proposed a solution for it. Since then much has been learned about the mathematical properties of graphs, and the field has a long history of applications including sociology, biology or operations research and optimization. Things changed when computers became accessible and affordable to most researchers, allowing them to collect, store, share and study large amounts of data on graphs observed in real-life. This gave rise to the interdisciplinary field of network science, which is dedicated to the description and analysis of real-life graphs borrowing methods from mathematics, physics, computer science and sociology. The topics of networks science include the organization principles of real-life networks, for example their degree distribution and the formation of groups, as well as several processes taking place on the networks themselves.

The goal of this dissertation is to provide an overview on the authors' works focusing on three major topics of network science: overlapping community detection, dynamic community detection and the study of infection processes. Based on our experiences with known community detection algorithms and the works of Csizmadia et al. we have developed an overlapping community detection method, the hub percolation method, that is capable of handling various types of networks or discover different layers of the community structure of single network. We have tested our method on several benchmarks including Newman's networks and Lancichinetti's community based graph generator. We have given two real-life case-studies: one of them examines the properties of an ownership network of Hungarian companies. The other one compares the community structures of a Hungarian and an English word association graph. The hub percolation method and its applications are the first main topic of this work, and they can be found in chapter 2.

Motivation for examining another aspect of community detection – dynamic community detection – came from applications on other social and economic networks. Based on the works of Palla et al., we have developed an algorithm to efficiently match the communities of two neighboring networks in a time series of graphs. Our method distinguishes eleven community events, and its mechanism does not depend on the specifics of the used community detection algorithm. We have evaluated our method on two real-life networks. The dynamic community detection method is the second topic of this dissertation, and it can be found in chapter 3.

The last topic of this dissertation is the study of infection processes in networks with a focus on economic applications. A common problem of these applications is, that the edge infection probabilities required to compute these processes are often not available, missing values are either estimated or constants are used. In order to provide a more systematic approach we have proposed the Inverse Infection Problem, its task being the computation of the edge infection probabilities with the help of

observations on the beginning and the end of the process. We have proposed the Generalized Cascade model to help with the computations required for this model, as well as several heuristics to further improve performance. We have given a learning method based on Particle Swarm Optimization as a solution. Finally we have tested the performance of our solution on benchmark networks, and we have also published a case-study dealing with the application of our method in the estimation of credit default in banking. Chapter 4 contains our findings in the topic of inverse infection.

This work begins with a short introduction into the basics of graph theory, including the concepts required to understand further chapters. Then we are going to discuss each of the aforementioned topics, beginning with the hub percolation method and its evaluation. Chapter 3 deals with dynamic community detection, and chapter 4 covers the Inverse Infection Problem. We will draw conclusions regarding each topic at the end of the corresponding chapters, and we will summarize our findings at the end of the dissertation.

Contents

Chapter 1

Introduction

In the beginning of this chapter, to give the reader a proper taste of the topics in this dissertation, we will give a short review on a few selected landmarks in graph theory.

The first paper in the history of graph theory was written by Leonhard Euler on the problem called "The Seven Bridges of Königsberg". The problem itself can be summarized in a simple manner. To find a walk through the city of Königsberg, that crosses each bridge once and only once. Euler quickly realized, that the choice of route inside a landmass was irrelevant, only the sequence of bridges to be crossed matters. This allowed him to introduce the concepts of abstract nodes (or vertices) and the connections between them. This way the city could be represented as a graph, the nodes corresponding to the landmasses, and the bridges are given as the connections (or edges) between the nodes.

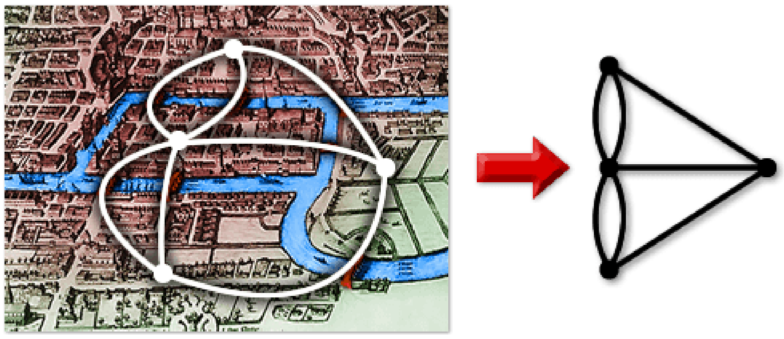

Figure 1.1: The seven bridges of Königsberg and its graph representation[1]

Euler proved that the problem has no solution in the following way. He observed, that in order to walk to a different landmass, one must enter and leave a bridge. As a consequence during any walk, the number of times one enters a landmass is equal to the number of times one leaves it. The only exceptions are the beginning and the end of the route. The original problem allowed the crossing of each bridge only once, therefore for each landmass the number of bridges connecting them must be even, again with the exception of the beginning and the end of the route. In the city of Königsberg, this is untrue: all landmasses are connected by an odd number of bridges. Since a walk only has one beginning an one end, and the graph has four

[1]Figure taken from `http://world.mathigon.org/Graph_Theory`

nodes, this leads to a contradiction, hence the negative result published by Euler. The problem was later generalized to any given graph, and the solution – a walk using each edge once and only once – was named as an Eulerian walk in his honor.

In the following centuries graph theory was expanded by many authors' contributions. Euler's formula lead to the development of topology, and the enumeration of graphs having particular properties. The term graph was introduced by J. J. Sylvester in 1878. The first textbook on graph theory was published by Dénes Kőnig in 1936.

Graph theory was always a popular branch of science and has many topics: the description of planar graphs and Euler's formula, Kuratowski's and Wagner's theorems, the connectivity of graphs and Menger's theorem, or finding matchings and the description of bipartite graphs including the works of Dénes Kőnig and Jenő Egerváry. Pál Erdős and Alfréd Rényi studied the probability of graph connectivity giving rise to random graph theory, while the infamous four color problem motivated the development of factorization problems by Julius Petersen and Dénes König. Pál Turán started extremal graph theory, his work was continued by Pál Erdős. A conjecture of Erdős and Turán lead Endre Szemerédi to develop a combinatorial mechanism that could be destilled to his famous regularity lemma. The theory of perfect graphs originated from a 1958 result of Tibor Gallai.

Graph theory has many other branches apart from those listed above, and the historical introduction given here is far from complete, but the trail this work picks up begins in the final decade of the 20th century, with the emergence of the interdisciplinary field of "network science". Before we resume our tale however, it is necessary to familiarize ourselves with the basic concepts of graph theory.

1.1 Basic concepts and definitions

After the appetizer, let us take a look at some of the basic concepts of graphs. In this section we will introduce all definitions, theorems and problems relevant to the topics of this dissertation.

Graphs, degrees and representations

The simplest way of defining a *graph* would be to consider an ordered pair $G = (V, E)$, where V is a set of *vertices* or *nodes*, and E contains two-element subsets of V called *edges*. A more general description defines an *incidence relation* I, that maps elements of E to two-element sets of vertices. Edges are often denoted as $e = (u, v)$ or simply as $e(u, v)$ or $e_{u,v}$, where $e \in E$ is the edge itself, and $u, v \in V$ are vertices of the graph. Vertices associated with an edge are the *end points* of the edge, are *connected* by the edge, and the edge is *incident* to each vertex. If two vertices are connected by an edge, they are *adjacent* or *neighboring*. We will the denote the set of vertices adjacent to vertex v as n_v^+. It is possible for a vertex not to be incident to any edge, therefore these vertices are not adjacent to any other vertices. In this case $n_v^+ = \emptyset$. The definition above allows two additional things: the pair of vertices mapped to an edge may contain the same vertex twice. These edges are called *loops*, an example of this is $e_{v,v}$. It is also possible for two vertices to be connected by multiple edges. These are called *parallel edges*. If there is a parallel edge in a graph it is called a *multigraph*. If there are neither parallel edges nor loops in a graph, it is a *simple* graph. In a *complete* graph, all vertices are connected by an edge. Complete graphs will be denoted as K_n, n indicating the number of vertices. It is easy to see, that a complete graph has exactly $(n(n-1))/2$ edges. If V and E are finite, then G is a *finite* graph, otherwise G is *infinite*. In this dissertation we will only work with simple, finite graphs.

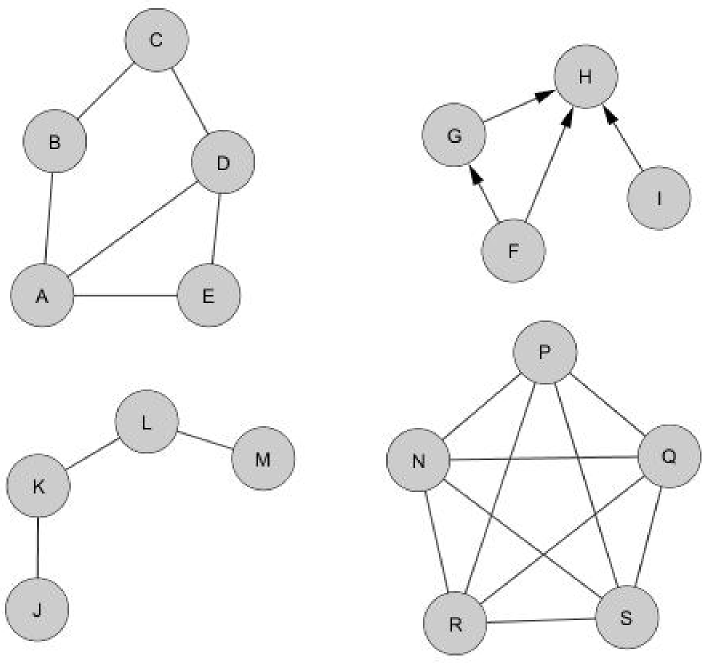

Figure 1.2: Graphs. Upper left: undirected graph with five nodes and six edges. Upper right: Directed graph with four nodes and directed edges. Lower left: a path graph with four nodes. Lower right: a complete graph with five nodes.

If I maps ordered pairs of vertices to edges, we call G as a *directed graph*, and the edges as *directed edges*. Directed edges are also called *arcs* or *arrows*, and we can make a distinction between the first and second element of the ordered pair associated with it. Consider edge $e_{u,v}$, u is the *tail* of the edge, and v is called the *head* of the edge. The edge is said to be *pointing* from u to v, and v is the *successor* of u, u being the *predecessor* of v. According to the direction of its incident edges, the neighbors of a node u can be split into two categories: *in-neighbors* are connected to u by an edge pointing towards u denoted by n_u^-, and *out-neighbors* by edges pointing from u, n_u^+. The *inverted edge* of $e_{u,v}$ is $e'_{v,u}$. A directed graph is *symmetric* if for every edge, its inverted edge is also present in the graph. We can create an undirected graph from a directed one by *omitting* the direction of edges: replacing each directed edge with an undirected one.

It is possible to assign labels to the vertices and edges of the graph, resulting in *labeled* graphs. These may hold additional information, for example the states of certain graph algorithms. If numbers are assigned to the edges of a graph, the graph is a *weighted* graph. The numbers on the edges may represent distances, similarities, costs, etc. We will denote the weight of edge $e_{u,v}$ as w_e or $w_{u,v}$. Sometimes we will access the labels of nodes and the labels or weights of edges with the help of functions, for example: $w_e = w(e)$, or a label a of a vertex v like $a(v)$.

An important property of any node in a graph is its *degree*. In an undirected graph the degree of a node is simply the number of edges incident to it, which is equal to the number of adjacent vertices it has[2], which is equal to the cardinality of n_v^+ for any vertex v. If a graph is directed, we can make a distinction between the *in-degree* and the *out-degree* of the node. We can construct a probability distribution on the graph using the concept of degree; the *degree distribution* $P(k)$ of a graph is the fraction of the nodes in the graph with degree k. Finally if we consider a weighted graph, we have another property. The *strength* of a node is the sum of the weights of the edges incident to it. Like before, we can make a distinction between *in-strength* and *out-strength* in the case of directed and weighted graphs.

The three simplest and most popular graph representations are the following: an *adjacency list* contains for each node all adjacent nodes. Organized in records or rows, each row stores a single node and its neighborhood. This form is very efficient in storing sparse graphs, and the addition of a node or edge only takes constant time. Checking whether two nodes are neighboring scales with the number of vertices in the worst case however. Dense graphs are often stored in *adjacency matrices*. The rows and columns of the matrix represent the nodes of the graph, and the elements of the matrix signal a connection between them. This form is rather space-consuming, but the addition and removal of edges can be done in constant time, and testing the adjacency relation between nodes is also simple. In an *incidence matrix* the rows represent the nodes of a graph, columns represent the edges and and the values of the matrix indicate if an incidence relation exists between the corresponding edge and node. Additional factors must be taken into account when selecting a graph representation: while all of them is able to handle directed graphs, the storage of weighted graphs is more simple in the latter two. The matrix representations also have advantages in specific tasks: the computation of network flows, PageRank and HITS values, etc.

Graph operations and graph classes

Graphs can be modified with the help of *graph operations*. Here we will define the ones required for this dissertation. Operations vary in their scale, starting from the addition and removal of edges and vertices to the union of two graphs. A *subgraph* H of a graph G is graph where $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$ and for all $e_{u,v} \in E(H)$, $u, v \in V(H)$. If $V(G) = V(H)$, then H is a *spanning subgraph* of G. A subgraph H is a *induced subgraph* of G if for any vertex $u, v \in V(H)$, $e_{u,v} \in E(H)$ if and only if $e_{u,v} \in E(G)$. We can partition the vertices of a graph into two subsets with the notion of *cuts*. A *cut* $C = (S, T)$ on a graph G divides the elements of $V(G)$ into two sets S, T.The *cut-set* is a set of edges $e_{u,v}, u \in S, v \in T$. Another important notion is the *complement graph* H of a graph G. The vertex set is the same in both graphs, while for any vertex u, v, $e_{u,v} \in E(H)$ if and only if $e_{u,v} \notin E(G)$. A *line graph* of a graph $L(G)$ is the intersection graph of the edges of G: each vertex of $L(G)$ represents an edge in G and the vertices of the line graph are connected if their corresponding edges share a common endpoint. Finally the two binary graph operations required in this dissertation are union and intersection. The *union* of two graphs G and H is $G \cup H = (V(G) \cup V(H), E(G) \cup E(H))$, the *intersection* of these graphs is $G \cap H = (V(G) \cap V(H), E(G) \cap E(H))$.

According to their properties graphs can be categorizes into a multitude of differ-

[2]We only consider simple graphs in this dissertation.

ent classes. The discussion of these classes is beyond the scope of this dissertation, so again we will only cover classes directly tied to the topics of this work. There are many equivalent definitions for a *tree*. In almost all occurrences a tree is undirected. A tree is a connected graph without cycles. A tree is connected, but leaving any edge from it ruins this property. A tree does not contain cycles, but adding a single edge between any of its vertices creates one. Any two vertices in a tree are connected by exactly one path. A tree is a connected graph with n nodes and $n-1$ edges. A tree is a graph without cycles, n nodes and $n-1$ edges. The nodes of a tree can be divided into two groups: *leaf* nodes have exactly one neighbor, *internal nodes* have at least two neighbors. A *forest* is a disconnected graph with trees as its components. Equivalently, a forest is an undirected cycle-free graph, or a disjunct union of trees. A single vertex can be selected as the *root* of a tree resulting in a *rooted* tree. We can create a *partial ordering* on the nodes of a rooted tree with $u \leq v$ if and only if the unique path from the root to v passes through u. We can also measure the distance from the root for all vertices. We will define a *directed tree* in the following way. Given a rooted undirected tree, we will assign direction to the edges so that for all vertices $v \in V(G)$ with the exception of the root, the edges of the unique path from v to the root point *towards* the root. In this work we will also allow a different definition, that is similar to the above one but with edges pointing *away* from the root.

Paths and connectedness

An alternating sequence of vertices and edges is a *walk*, $v_0, e_1, v_1, e_2, v_2, e_3, \ldots, v_{\ell-1}, e_\ell, v_\ell$. A walk begins and ends with vertices. A walk *connects* vertices v_0 and v_ℓ, $v_i : i > 0, i < \ell$ are the *inner points* of the walk. We will denote the vertices of a path S as $V(S)$, the edges as $E(S)$. If the graph is directed e_i points from v_i to v_{i+1}. If $v_0 = v_\ell$, the walk is *closed*, otherwise it is *open*. The *length* of a walk is the number of edges it contains, in the above formulation, it is ℓ. In a weighted graph, the *weight* of a path is $w_S = \sum_{e:e\in S} w_e$. In an unweighted graph, the length and weight of a path is considered to be equal. In simple graphs $V(S)$ is enough to define an arbitrary walk S. A *line* is a walk, where all edges are distinct. A closed line is often called as a *tour*. A *path* is a walk, where no vertices are repeated (as a consequence no edges are repeated as well). If the a path is closed, it is called a *cycle*. Two paths covering different vertices are *disjoint*. Graphs containing a single walk or cycle of n vertices are called P_n or C_n respectively. A graph without a cycle is *acyclic*. A cycle with an odd length is an *odd* cycle, otherwise it is an *even* cycle. A *Hamiltonian path* covers all of the vertices of a graph exactly once. In *Hamiltonian cycle* the first and last vertices are the same, and these are the only reoccurring vertices. A graph containing a Hamiltonian cycle is *Hamiltonian*. An *Eulerian* trail uses all edges of the graph exactly once. A graph containing an Eulerian cycle is *Eulerian*.

A common task in many applications of graph theory is finding the *shortest path* between two given vertices in a graph. Starting from v_0 and ending in v_k consider the set of paths T between them. As we have mentioned before, we can assign a weight (or length) to each path $w_S, S \in T$. The shortest path among all paths connecting these two vertices, is the path where the weight of the path is minimal. The length or weight of the shortest path between to vertices gives the *distance* between the vertices. It is possible that there are multiple shortest paths between the vertices, but it is easy to see, that the length of all shortest paths is the same, hence the distance between them is unambiguous. The shortest path problem has three variants. Either it computes all shortest paths *stating from* a vertex, *ending in* a vertex, or between every pair of vertices in the graph. Dijkstra's algorithm solves the first problem if the edge weights are non-negative. The algorithm of Ford and Bellman is able to solve the same problem if there are negative edge weights. This algorithm is also able to detect *negative cycles*: in these all edge weights are negative. This is important, because the use of a negative cycle can make all incident paths infinitely cheap. Finally the Floyd-Warshall solves the all-pairs shortest path problem.

As we can recall, two vertices are adjacent if there is an edge incident to both of them. Two vertices u and v are *connected* if there is a path starting from u and ending in v. A graph G is a *connected graph* if there is at least one path between all of pairs of vertices it contains, otherwise the graph is *disconnected*. A *connected component* (or simply component) of a graph, is a set of vertices, where each pair of vertex is connected, and no other vertex in the graph outside the component is adjacent to the vertices of the component. The connected components of a graph give a partitioning of the vertices. If there is only one connected component in a graph, the graph itself is connected. It is easy to see, that the connectedness relation between two vertices is an equivalence relation. With it we can define equivalence groups inside the graph, these are the connected components. If there is only one equivalence group in a graph, it is connected. A directed graph is *weakly connected* if it is connected after omitting the directions of its edges. A directed graph is *connected* if for all vertices u and v it contains a directed path from u to v or v to u. A directed graph is *strongly connected* if it has paths from u to v and v to u for all verices. We can use similar formulations to define weakly connected, connected and strongly connected components.

Independent sets, cliques and degeneracy

An *independent set* in graph is a set of vertices, no two of which is adjacent: for every pair of vertices there is no edge connecting them. If we cannot add a vertex to an independent set without ruining this property, we have a *maximal* independent set. The largest maximal independent set is the *maximum* independent set. The size of this set in a graph G is the independence number of G, $\alpha(G)$. Obviously, any independent set with multiple vertices may be split into two independent sets. Finding maximal and maximum independent sets is difficult: there can be at most $3^{n/3}$ maximal independent sets in an arbitrary graph [90], and finding the maximum set was proven to be NP-hard [63].

Given a graph G with an independent set S, the vertices of S form a complete subgraph or *clique* in the complement of G. Like above we can define a clique as a set of vertices, all of them adjacent. We can also define the concepts of *maximal* and *maximum* clique in the same way: the maximal clique is not a subgraph of any other clique, and the maximum clique is the largest maximal clique. The difficulty of finding all maximal cliques and finding the size of the maximum clique is also the same. A famous problem associated with cliques is the *clique decision problem*, which is among Karp's 21 NP-complete problems, and asks whether a given graph and a number k does the graph contain a clique larger than k?

Listing all maximal cliques will be crucial to most of the community detection methods discussed in this dissertation. We have already stated, that an arbitrary graph may contain at most $3^{n/3}$ maximal independent sets or cliques. In some graph classes, especially sparse ones, this number may be significantly lower. The oldest and most useful algorithm for listing all maximal cliques is the Bron-Kerbosch algorithm [25]. Matching the given bound, it has a worst-case running time of $O(3^{n/3})$. Another approach to finding maximal cliques is the output-sensitive algorithm of Tsukiyama et al. [111], which lists maximal cliques with a polynomial delay. Several other methods were proposed in [27, 31, 109] to further improve the performance of this algorithm. As a consequence it is possible to list all maximal cliques in reasonable time in graph classes, where the number of cliques is polynomially bounded, like planar graphs, triangulated graphs, complete graphs, etc.

Despite this theoretical result, in most applications the Bron-Kerbosch algorithm has better performance than other methods. In this disseration we will use the BK algorithm to find maximal cliques in graphs. The basic form of this algorithm is a simple recursion manipulating three sets of vertices R, P and X.

Initially R and X are empty, and P contains the vertex set of the graph. During the process R is used for selecting the vertices belonging to a possibly maximal clique, P contains the candidate vertices, with which R might be extended, and X contains the vertices, that were previously used to form a maximal clique, or vertices known not to be part of any clique. In the same paper, the authors have shown, that performance can be improved by *pivoting*; selecting the vertex from P (or $P \cup X$), that is most likely a member of a maximal clique. They have proposed a few pivot selecting strategies: in random graphs selecting the vertex with the greatest degree was the best choice, but in some graph classes random selection was better. Since then many other selection strategies were proposed. Here we will only mention the one proposed by Cazals and Karande in [27]. Their idea is based on the one introduced by Bron and Kerbosch: the pivot is selected from $P \cup X$ maximizing the cardinality of the set $P \cap n^+_{v_p}$, where v_p is the pivot vertex. Finally Eppstein and Strash have shown in [41], that performance can be further improved by replacing pivoting at the outermost level of the recursion with the degeneracy ordering of the vertices. Algorithm 1.2 uses both pivoting and degeneracy ordering.

The *coloring number*[3] κ of graph is the least κ for which there exists an ordering of the vertices in which each vertex has fewer than κ neighbors that are earlier in the ordering. The coloring number was introduced by Erdős and Hajnal in 1966 [42]. Sometime later the *degeneracy* of a graph was defined by Lick and White [82] to be the least k such that every subgraph has a vertex with k or fewer neighbors. The authors have also proven, that $k + 1 = \kappa$ in finite graphs, implying that these properties are equivalent. The ordering used to define κ is also known as the *degeneracy ordering* of the graph, and can be computed in linear time with the algorithm of Matula and Beck [88].

A short detour into time complexity

We will propose several algorithms in this dissertation, and we will also measure the performance of them. One of the core performance indicators of any algorithm is its *running time* or *time complexity*. The time complexity of an algorithm is usually

[3]Not to be confused with the chromatic number

1.1. Algorithm. The Bron-Kerbosch algorithm.

Input: Graph G

Output: Set Q containing all maximal cliques

```
procedure BronKerboschUpper(G)
    $R \leftarrow \emptyset$
    $X \leftarrow \emptyset$
    $P \leftarrow V(G)$
    for all $v \in P$, in the degeneracy ordering of $G$ do
        BronKerbosch($R \cup \{v\}$, $P \cap n_v^+$, $X \cap n_v^+$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
    end for
end procedure

procedure BronKerbosch(R, P, X)
    if $P = X = \emptyset$ then
        $Q \leftarrow Q \cup \{R\}$
        return
    end if
    Choose a pivot vertex $u$ from $P \cup X$
    for all $v \in P \setminus n_u^+$ do
        BronKerbosch($R \cup \{v\}$, $P \cap n_v^+$, $X \cap n_v^+$)
        $P \leftarrow P \setminus \{v\}$
        $X \leftarrow X \cup \{v\}$
    end for
end procedure
```

given as a function of its *input size*[4], because if the size of the input is increased, the time it takes for the algorithm to compute its result also increases. The nature of the input or task also matters. Given two different inputs with the same sizes, the same algorithm may have two different running times for each of them. We could say, that one task is more difficult, than the other.

The simplest way to give the running time of an algorithm is to measure it empirically by setting a timer. This way the running is given in nanoseconds for example. The drawback of this is, that an algorithm is a concept in itself without a concrete implementation. The measured empirical running time however depends on several other factors: the language it is implemented in, the platform it runs on, the programmer's skill, the difficulty of the task etc. Even so empirical running time analysis is sometimes the only available choice of the algorithm is too complex for other approaches.

Theoretical time complexity analysis measures running time as a *function* of the input size measured in *steps* or computational units. The most frequent notation – the big O notation – gives an upper bound to the for the run-time of an algorithm. More precisely, for a given input size n greater than some n_0 and a constant c, the running time of that algorithm will never be larger than $c * f(n)$. In theoretical analysis often the *worst-case* time complexity is given, these means the most difficult

[4]At least since the paper of Edmonds.

task of a given size n.

This concludes the basic definitions required in this dissertation. This was of course only a short introduction, since the field of graph theory is too large to summarize in a single work. If the reader is interested various textbooks and reviews can be found in the references including [12, 56, 113].

1.2 Properties of real-life networks

At a certain point in the development of graph theory, it become necessary to introduce the concept of a "typical" graph. Contrary to the previously mentioned graph classes, these graphs were meant to represent graphs without any specific properties, or in different terms, researchers were interested whether these specific properties were present in an arbitrary, random graph one can encounter anywhere in nature. This gave rise to *random graph theory*. In the beginning, the aim of random graph theory was to propose models for the creation of graphs with or without certain properties. Most researchers consider the founders of this branch of graph theory to be Pál Erdős and Alfréd Rényi who proposed the so-called $G(n, M)$ model named after them in 1959 [43]. A similar model $G(n, p)$ was proposed by Edgar Gilbert independently and contemporaneously [46].

These models have a probabilistic rule set to add edges to a certain number of isolated nodes. The $G(n, M)$ model considers all graphs with n vertices and M edges each with a uniform probability of occurrence, and chooses one at random. In the $G(n, p)$ model each vertex is connected with an independent probability $0 < p < 1$. The two models are interchangeable for almost all graphs with $M = \binom{n}{2}p$ if $pn^2 \to \infty$. The main goal behind the creation of these models was to study the connectivity behavior in these graphs with respect to p. Questions like, at which value will the graph contain a large connected component of size greater than some value, or at which value does the graph contain isolated vertices were successfully answered. The study of these models continued in the following decades.

In the same time it became apparent, that the above two models produced graphs quite dissimilar to the ones scientists encountered in other fields of science, like sociology. As a result, researchers either continued to study small graphs with specific properties, or shifted their attention to graphs directly created from real-life phenomena. Things changed when computers became accessible and affordable to most researchers, allowing them to collect, store, share and study large amounts of data on graphs observed in real-life. Focus shifted towards the analysis of the properties of real-life graphs, and these graphs were often greater than the ones previously studied. These graphs were named *complex networks* and they often contained non-trivial topological features unlike the ones that occur in the more simple graph classes: trees or lattices.

The study of these graphs sprouted a community mainly composed of physicists, sociologist, biologists and some computer scientists and mathematicians. The field of *network science* was born, and the term *network* became the synonym of graph for these people. The goal of this field is to study the statistical properties of networks occurring in nature and develop models to help scientists understand these properties. Another goal added more recently is the prediction of the behavior of these networks in time.

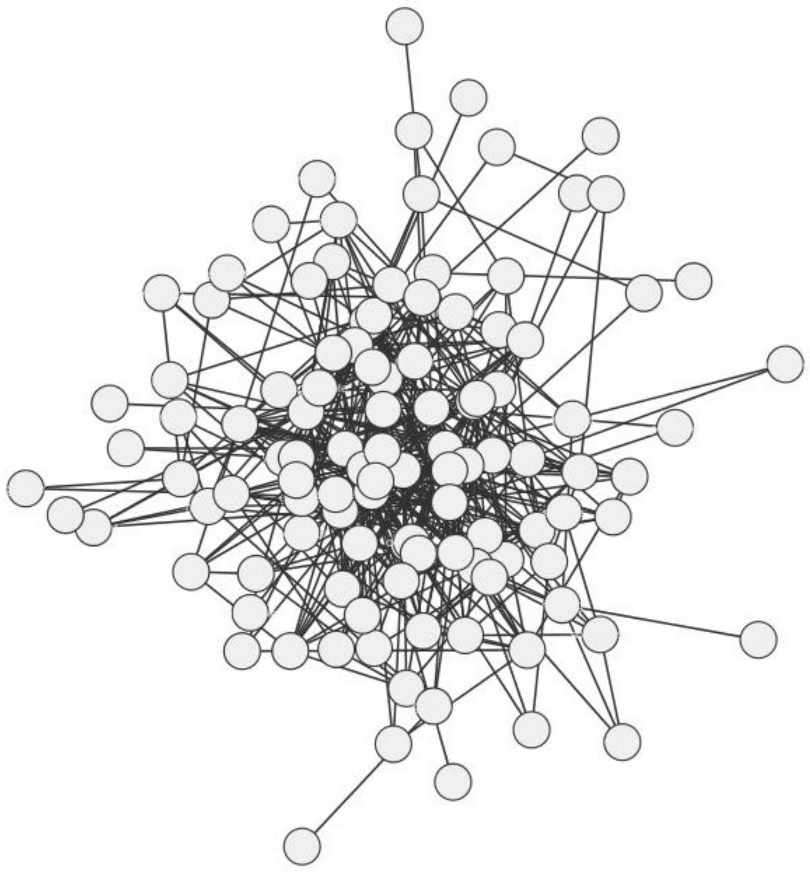

Figure 1.3: The graph of common words in the novel "David Copperfield" by Charles Dickens. Edges connect any pair of words that occur in adjacent position in the text of the book. The graph was created in [94].

Networks in nature

Before we go into further details regarding the properties of these graphs, it is wise to take a look at the various networks one can encounter in nature. Here we will loosely follow the classification Newman used in [95]. The study of *social networks* dates back well before the proposal of random graphs by Edrős. The analysis of these networks was a sub-field of sociology and sociometry. However, traditional approaches to the studies of these networks were plagued with small sample size, inaccuracy and subjectivity. Again, this changed with the computer revolution and the introduction of techniques from graph theory and the newly founded network science. Social networks are often composed of people or groups of people: sports clubs, families, etc. The connections between them are those, that occur naturally between these entities. A famous representation of friendships between people was Zachary's work on the members of a karate club in [114], but friendships between animals were also studied [47]. Recently, with the rise of Internet-based social networking sites, these networks became the focus of many works. Other well-known forms of social networks is the network of marriages between families discussed in [60], the network of tournaments in various sports [47], or the network of telephone calls in [4]. Collaboration networks are also composed of people, and the connections between them came from common works in science or industry. One of the most famous collaboration networks is the network of actors taken from the Internet Movie

Database. The network of scientific collaborations is also well studied, with data coming from various publication indexing services, and arxiv.com is also a popular source of information in this field. Collaboration in a certain scientific fields also created amusing facts. In discrete mathematics for example, a measurement called *Erdős number* was created. Erdős was a very productive scientist, he wrote more than a thousand publications with various co-writers. The aforementioned number was defined as a distance measurement on this network: the number of any author in the network is the length of the shortest path from that author to Erdős, if such a path exists.

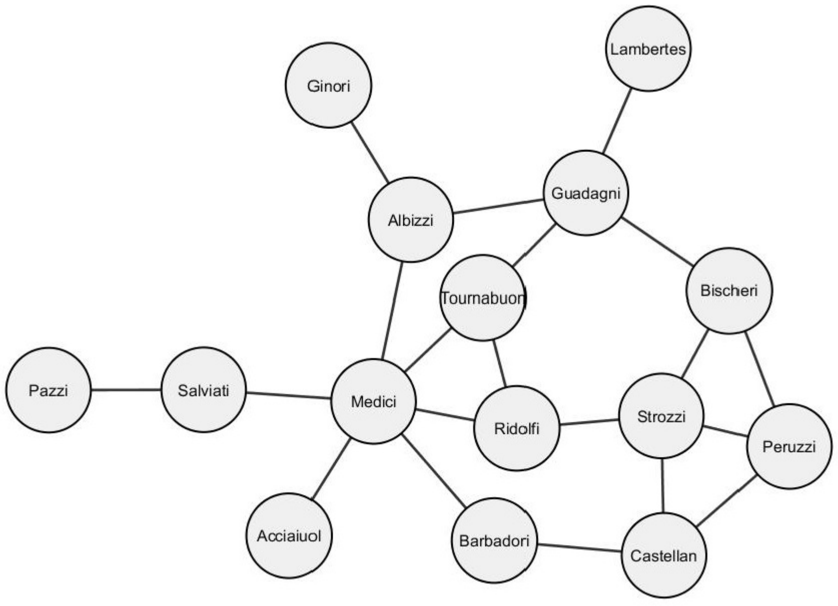

Figure 1.4: The graph of family connections in renaissance Florence [60]. Links indicate marriages between families.

Information networks are very closely related to social networks to the point, that they are sometimes considered to be a specific branch of them. Examples include scientific citation networks [106], where the nodes are publications and directed edges are pointing from one to another if there is a citation present between them. Besides being directed, these networks have another specific property, they do not contain directed cycles, because one can only cite papers that have been already written. Like collaborations networks, these networks can be easily constructed from various journals, indexing services and publication databases. The Internet provides ample means of creating information networks. The links of the World Wide Web from a directed network, nodes serving as web pages and the links between them are the edges [59]. More specialized variations of this network is formed by a medium on the web: twitter accounts "twitting" each other, various blog engines, etc.

Typically representing some sort of infrastructure, *technological networks* are centered around the distribution of resources. The simplest examples are road maps, railways or air traffic maps, but electric power grids or plumbing networks belong to this category as well. The technical parts of certain above mentioned networks are here as well. The network of telephone calls is based on a physical network of wires and cable connecting devices, or the Internet runs upon physical hardware, the connections between routers, servers and various other devices.

Often considered to be between social and information networks, *economic networks* will be very important in this dissertation. Economic networks are composed of economic actors as nodes including sovereign states, partnerships or various types of companies like joint-stock companies, ltd-s or banks. The diversity of economic networks also comes from the connections between the nodes. Edges may represent bank transactions [16], some form of cooperation [38], common shareholders [18], etc. Economic network may be difficult to come by, since access to specific information regarding companies may be prohibited by law, or considered to be a trade secret of the company. In the latter case, it is possible to cooperate with the company, but the publishability of the results is very limited. Even if public databases are available, advanced data mining techniques are often required to construct the respective networks, due to the erroneous, missing or partial data, a characteristic of these databases in many countries.

Networks can be found in other fields of science as well. Examples in biology include the network of protein interactions [62], gene regulation [54] or food chains [33]. Physicists have studied the network of free energy minima and saddle points in glasses [39] and the network of conformations of polymers and the transitions between them.

Properties of real-life networks

As we have mentioned before, the properties of real-life networks are fundamentally different than the properties of the Erdős-Rényi random graph. In the remaining part of this section we will take a look at these specific properties.

Perhaps the most well-known property of these networks is the so-called small-world effect. The idea, that the person to person distances in social networks are surprisingly small, first occurred in the short work of the famous Hungarian author Frigyes Karinthy. In it, he played with the thought of now many people he has to go through if he wanted to contact the Nobel laureate Selma Lagerlöf. He suggested a chain of connections: each link of the chain is a personal acquaintance of the next one. He imagined that the number of possible connections grows exponentially as the length of the chain increases. The first mathematical examination of this phenomenon was done by Pool and Kochen in [104] in a preprint that circulated between various scholars for almost 20 years before being published.

One of these scholars was Stanley Milgram, who devised an experiment [61, 89]: he sent packages to several randomly selected individuals in the United States with instructions to send the package forward to a target person. If they knew the target personally, they would simply sent the package to him. Otherwise, they were required to think of friend who was more likely to know the target and send the package to him. If the individuals in the chain agreed to participate in this experiment, they would add their contact information to the package and forward it according to the rules above. Even though a large fraction of the packages never arrived to their destination the experiment was successful, and generated enormous publicity. The average number of contacts required to reach the contact person was six, hence the phrase "six degrees of separation".

After several decades, the small-world phenomenon was picked up again by researchers Duncan Watts and Steven Strogatz. In their groundbreaking paper [112] they have made several observations on the average distances in real-life networks, and proposed a random graph generation model with this property. Watts and

Strogatz defined the small-world network to be network where the typical distance L between two randomly selected nodes grows proportionally to the logarithm of the number of nodes in the network: $L \sim \log |V(G)|$. They listed several empirical networks with this property including social networks, the Internet or gene networks.

Unlike the Erdős-Rényi random graph, whose degree distribution is binomial, the degree distribution of many real-life networks is fundamentally different. In 1965 Price studied scientific citation networks [105] and found, that the degree distribution of these networks were heavily right-skewed and followed a Pareto distribution or a *power law*. This concept is often simply referred to as the Pareto-rule, and it was used to describe the allocation of wealth: a large portion of the wealth of a society is owned by a small percentage of the individuals. This law was identified in life many times, from the areas burnt by forest fires to the rates of hard-drive errors. In networks terms, it appears as a small fraction nodes having large degrees as opposed to the rest of the nodes with small degrees. Price discovered this law in his networks, and he also identified the exponent of this distribution to be $P(k) \sim k^{-\alpha}, \alpha = 3.04$. This property was observed in real-life many times [7], and this kind of network was dubbed as a *scale-free network* by Albert and Barabási. Finally, several network growth models producing networks with scale-free degree distributions were proposed. The earliest one was the preferential attachment model [7], but others soon followed [72, 81].

A behavior observed in social interactions is, that people tend to form groups according to their lines of interest, occupation, etc. In networks, this behavior presents itself as a tendency for nodes forming into sets, so that the nodes inside the sets are densely connected and the links between the sets are relatively sparse. This feature is called *community structure*. The detection of the community structure of networks, and its properties form a large part of this dissertation. Therefore we will give an introduction into this concept in the next section in greater detail. The authors' own findings in this topic can be found in chapters 2 and 3.

There are many other properties of real-life networks we have not discussed before, like transitivity [112], maximum degree [2], robustness [5] or the temporal behavior of networks [20,99]. Again the field of network science is too large to review in a single dissertation. Further material on this topic can be found in [60, 95].

1.2.1 Community structure

We have pointed out in the previous section, that people group together based on family relationships, hobbies or other common lines of interest. In network terms this means, that the nodes of the network are organized into vertex sets, such that most edges run inside these sets as opposed to be between them. This is the community structure of a network, and the discovery of this structure is called *community detection*. This property is not only present in social networks, but many other complex networks as well. Still the communities of social networks have been studied by sociologists well before the emergence of complex networks.

A famous example, that is still used as a benchmark of community analysis nearly 30 years after its publication, is Zachary's karate club network [114]. The vertices of this network represent the members of a karate club, and edges are drawn between individuals who were observed to interact outside the club. At a certain point during their observation, the club president and the instructor had a serious argument, that resulted in a split: two groups were formed, one centered around the president, the other around the instructor. Is it possible to create a method that predicts who are the members of these groups by looking at the network of connection before the split, researchers asked. A more recent benchmark was presented in [47], and shows the network of interactions of the bottlenose dolphins living in Doubtful Sound. The nodes of the networks are the dolphins, and there are links between them if they were seen together more often than expected by chance. The identification of friendship groups between them is a typical task of community detection.

As we have mentioned before, community structure is present in all kinds of networks, not only social ones. Another famous benchmark we will use in this dissertation is a word association graph built from the University of South Florida free association norms [91, 100]. The nodes of this network are English words, and two words are connected, if enough people associated one word from another. This is typical example of a directed complex network. A person associates one word from another, giving the edges a natural direction. This direction is usually omitted in applications however.

Other examples of communities in complex networks, are the presence of research groups in scientific citation and collaboration networks [96], geographically separated groups of companies in [18], but proteins form communities in their interactions as well [15, 62].

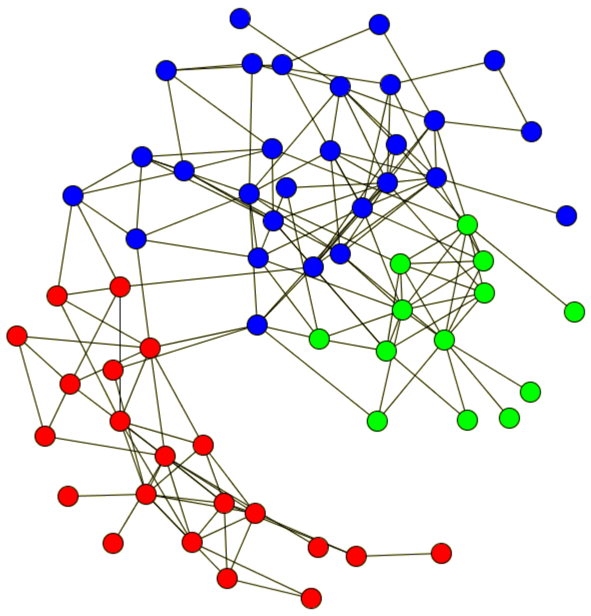

Figure 1.5: The community structure of the dolphins of Doubtful Sound [47]. Communities werer created with k-means clustering [86].

After taking a look at the communities of real-life networks, let us consider the task of detecting the communities themselves. Before going into detail we will restrict the object of our observations to specific types of networks:

- Meaningful communities are only found in certain complex networks. It is useless to look for them in other kinds of networks: lattices, trees, etc. do not contain communities.

- The identification of community structure is only possible if the network is sparse, otherwise the distribution of edges is too homogeneous for communities to make sense.
- While there are existing methods for the detection of communities in directed networks [100], almost all existing methods deal with undirected ones. We will also follow this approach.

While discussing community detection will only consider networks adhering to these specifications. Another thing to note is, that most methods only consider unweighted networks, although there are examples of weighted detection algorithms as well. Most of these rely on transforming the weighted network into an unweighted one by pruning: omitting edges with weights below (or above) a predefined criterion. In this dissertation we will show, that the method proposed by the authors can be extended to use the edge weights without the use of pruning.

Traditional community detection

The concept of community structure described above is very intuitive and easy to understand in layman's terms. Finding a proper definition for communities however is quite the task in itself. Initially, most scholars settled on the following loose definition. Find a partitioning of the vertices of the network that maximizes the number of edges running inside the sets and minimizes those between the sets. This definition has its weaknesses: a single set containing all of the vertices of a graph satisfies this criterion for example. We will keep this definition for some time, and will ignore the trivial solution above.

Approaches to community detection have roots in graph clustering. The first methods to find the community structure of networks were borrowed from this field: the Kernighan-Lin algorithm [69], k-means clustering [86], or various spectral algorithms [8, 85]. One of the most useful tools turned out to be hierarchical clustering [57]. It starts with the introduction of a similarity measurement between the vertices of the graph. After this, there are two approaches: agglomerative methods start with each vertex in a different initial cluster, and join similar clusters iteratively. The second, divisive approach starts with a cluster containing all of the vertices, and iteratively splitting this cluster by removing edges connecting dissimilar vertices. Further reference can be found in [45].

One of the first and most popular algorithms specifically created for community detection is the method of Girvan and Newman [47]. It is a divisive method similar to the ones above, and uses the concept of edge betweenness: the number of shortest paths among all paths running across a single edge. The algorithm itself is iterative: in each iteration the edge betweenness value is calculated for all edges, then the edge with the largest value is removed until a certain stopping criterion. The betweenness value itself can be calculated in $O(nm)$, the number of nodes times the number of edges, and the maximum number of iterations is m, resulting in the total running time of $O(m^2n)$, a rather slow method for large graphs.

A stopping criterion called *modularity* for the Girvan-Newman algorithm proposed by the same authors two years later [97], caused an even greater impact to the scientific community then the method itself. Modularity is based on the idea,

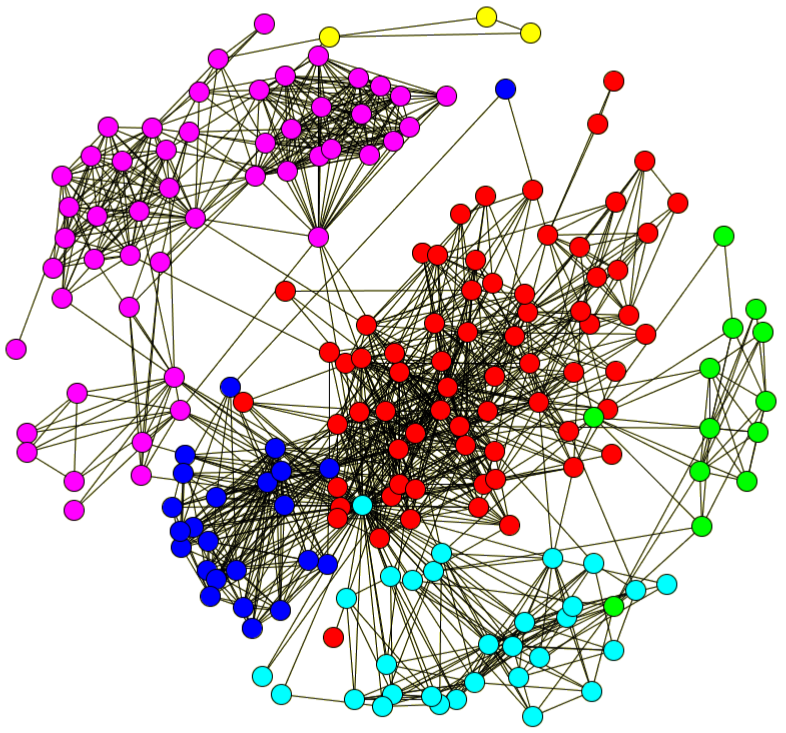

Figure 1.6: The community structure of a fraction of a social networking site. The communities were created with the method of Girvan and Newman [93].

that a random graph is not expected to have a community structure, therefore comparing the communities of the observed network with the ones of a similar network with no community structure may yield a good measurement. The latter network is created with the configuration model in [83]. It has the same number of nodes as the original one, has the same expected degrees for each node, but the edges are drawn randomly between the vertices. The exact formula for computing the modularity of the result of a detection algorithm is:

$$Q = \frac{1}{2m}\sum_{i,j}(A_{ij} - \frac{k_i k_j}{2m})\delta(C_i, C_j),$$

where the sum runs through all vertex pairs, m is the number of edges in the network, A denotes the adjacency matrix of the network, k_i or k_j the degrees of i and j, C_i, C_j the communities i and j belongs to, and $\delta(C_i, C_j)$ yields one if $C_i = C_j$, otherwise it is zero.

Starting with its creator, Newman, many authors proposed methods for creating community structures with optimal modularity. Various optimization techniques were used: greedy [97], spectral [94], extremal [40] optimization methods as well as simulated annealing [55]. Modularity has a resolution limit however; if the used network is large enough so the expected number of edges between two groups of nodes in the configuration model may be smaller than one. In this case even a single edge between these groups is interpreted as a strong correlation between the groups, so an optimization method would join these groups. As a result, modularity has a tendency to create large communities in large networks: it is unable to discover smaller ones.

Overlapping community detection

In contrast to the approaches listed above, researchers soon realized, that in real-life networks an individual can belong to multiple communities at once. This means, that our currently used definition is too strict: since the vertex sets representing communities are no longer disjoint, we cannot look for a partitioning. As a consequence we can neither use modularity. In reality the definition of *overlapping communities* often varies depending on the used benchmark or the application. There have been attempts to establish a measurement or benchmark as a standard, but neither of them became very popular. For example, several ways have been proposed to extend modularity to overlapping communities [53, 98], but neither of them are widely accepted. The graph generator of Lancichinetti et al. [76] also looks promising but it has its limitations as well. As we have noted before, overlapping community detection methods are often measured by their applicability to real-life examples. These applications have clearly stated goals to achieve, and the method is judged according to its ability to satisfy these goals.

The concept of completely connected subgraphs or cliques is central to many algorithms [18, 79, 100]. Cliques have very high densities, they are the ideal representation of closely-knit communities. A possible weakness of detecting cliques is their computational complexity mentioned in the beginning of this chapter. On the other hand, real-life networks are often very sparse, so that they contain significantly less cliques than the theoretical maximum. Sparsity may cause problems as well. A common flaw of clique based methods surfaces in networks too sparse to contain enough cliques: a large fraction of nodes in the graph may not be assigned to any communities.

A popular technique, the clique percolation method (CPM) was published by Palla et al. in [100]. It is based on the idea of "sliding" k-cliques in the target network. They thought, that these sliding k-cliques would become trapped in the communities of the network. The sliding also allows nodes on the border of communities to be members of multiple groups. More formally, they defined the concept of adjacent cliques: two k-cliques are adjacent if and only if they share $k-1$ nodes. By taking the union of adjacent cliques they formed k-clique chains, and introduced the concept of clique connectivity. Two cliques are connected if they is a chain of cliques from one to another. Finally they define the k-clique as a maximal connected subgraph built form the union of its connected k-cliques. The method was very well received because its clean theoretical foundations, and it has been extended to weighted, directed and bipartite graphs [80, 100]. A fast implementations was also published in [75]. The method has serious drawbacks however, for example the size of the cliques k is a parameter of an algorithm, but its choice is not trivial. It also has definition problems, since it detects chains of cliques and not communities, so the resulting objects often do not have a high internal edge densities. The opposite can also happen in networks containing a high number of cliques: the method might return trivial community structure, like a single group containing the whole network as a single cluster.

An example of non-clique based overlapping community detection algorithm is the method of Gregory [52]. Gregory's method points out the difference between traditional and overlapping community detection: a node may belong to multiple communities at the same time. If these nodes are copied for each group they belong

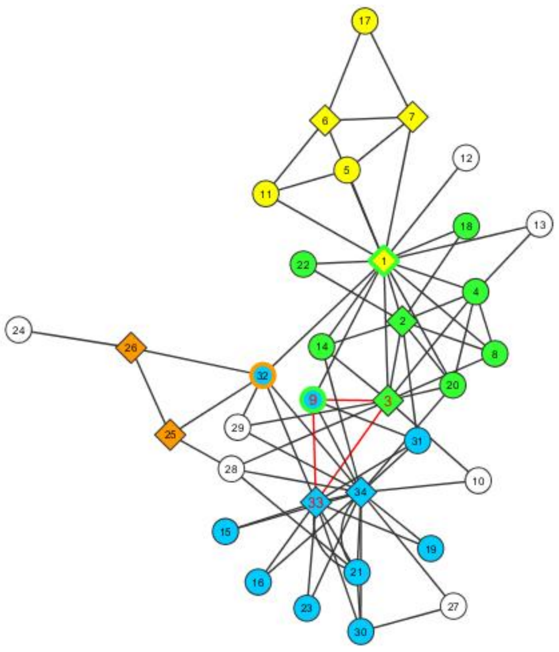

Figure 1.7: The community structure of Zachary's karate club network [114]. The communities were created with the hub percolation method discussed in the next chapter citeinfocom.

to, a traditional community detection method could be used. Gregory introduces split betweenness to measure the likelihood of a node belonging to multiple communities. The method itself has three phases: the graph is transformed into another graph without the overlapping nodes, a non-overlapping detection method is applied to it, then the resulting partitions are mapped back into the original graph by replacing the copied nodes with their original ones. This algorithm is popular because its speed: an approximation can be computed in $O(n \log n)$. It also has a drawback however, the number of overlaps between the resulting communities is usually small.

There are many other approaches to overlapping community detection. Examples include fuzzy clustering [92, 115], hierarchical clustering [3, 11], line graph partitioning [44], block models [35, 48], label propagation [65] or optimization according to some fitness function [78]. An excellent review on community detection can be found in [45].

Dynamic community detection

The final aspect of community detection we will discuss in this dissertation is *dynamic community detection*, the changing of communities in time. This task is even more difficult, than static detection, because the inaccuracies of traditional or overlapping community detection are multiplied in a temporal setting.

Dynamic community detection deals with a time series of graphs $\{G_i\}_{i \in \mathcal{T}}$, where $\mathcal{T} = 1, ..., \ell$ represents a discrete time set. Time elapses as i increases, and the task is discovering how the community structure of one graph relates to the community structures of the others in the series. Even before examining the community structure we have to deal with several difficulties of the graphs themselves. Ideally the time series has two factors, that have to be taken into account. Each graph in the series represents a snapshot of the changing community structure. The number of available graphs is governed by the length of the observation period and the resolution: in a given time period, how many times do we observe the network. Selecting the right resolution is important: if the communities change too much between the time instances, it may be impossible to track them. If they change too little, time is spent matching graph with nearly the same structure. To make matters worse, in applications the availability of the data limits our choices, usually both the number of graphs, and the elapsed time between them is given.

Algorithms for dynamic community detection usually consider two neighboring time instances at the same time, G_i and G_{i+1} for example. Most authors begin with the definition of *events* that may occur to communities between the instances. Palla et al. [99] defines the basic events to be: birth, death, growth, contraction, join and split. Other authors have expanded this list according to their observations [6, 20]. We will discuss these in detail in chapter three. After the definition, the communities of the two graphs are matched with some algorithm according to the defined community events. This matching may also be computationally difficult in large networks. Some authors circumvent this by using advanced computation techniques [6] or by using a special property of the static detection algorithm [99].

The results of dynamic community detection are also influenced by the used static detection algorithm. One of the first studying this field, Hopcroft used hierarchical clustering in [58], Palla et al. used their overlapping community detection algorithm, the previously discussed Clique Percolation Method [100].

The authors' works regarding community detection can be found in chapters two and three. An overlapping community detection called the hub percolation method is proposed in chapter two. It is a clique-based method with adjustable parameters capable of influencing the resulting community structure in multiple ways to match the requirements of diverse applications. We have used several benchmark methods to illustrate its usability including Newman's graphs and the graph generator of Lancichinetti. We have also given two case studies, one dealing with economic networks, the other with word association graphs. The authors findings in the field of dynamic community detection are in chapter three. We will present a method that uses an overlapping community detection method to efficiently match the communities of the neighboring networks, that is more general than the method of Palla et al. We will test this method on two real-life network series: a bank transaction graph and a social network.

1.2.2 Diffusion models

Graphs serve as the basis of many processes. A significant part of this dissertation deals with the spread of economic events, which falls into the category of *diffusion processes*. These models deal with the diffusion or spread of a wide array of things: behavior, information, influence, diseases, etc. These methods were studied multiple times by many different branches of science.

Perhaps the oldest approach comes from the medical sciences, where the prediction of the spread of infectious diseases is important. Even the earliest methods defined states for the individuals corresponding to various stages of infection. Individuals were categorized into these states, and the transitions between them were given as difference equations. Initially, these methods did not use graphs, rather they dealt with fractions of a fixed population. One of the fundamental models, the

SIR model was proposed in 1927 by Kermack and McKendrick [68]. This model used three states: *susceptible* individuals have not yet been infected with the disease, *infected* individual have caught the disease and are actively spreading it, and *removed* individuals have been infected and then removed from the disease, either due to immunization or death. In order to customize the SIR model to different kinds of diseases, additional models were defined with different states and updated the equations defining the transitions between them. For example, the SIS model allows individuals recovered from the disease to be infected again, or the SIRS models, in which the immunity is limited to a time period. A good review on epidemic modeling can be found in [14, 37].

One of the landmarks in this field was Granovetter's work in [51]. He studied the collective behavior of populations: whether individuals would participate in a riot. Based on the work of Schelling [108], he pointed out that an individual's behavior may depend on the number of other individuals already following that behavior. As such he could describe the behavior of a single individual separated from the rest of the mass. He proposed a threshold model to represent this phenomenon. A person adopts a certain behavior (for example, participation in a riot), if enough people already follows this behavior. In his definition, the threshold is "the number or proportion of others who must make one decision before a given actor does so". He also suggested that some individuals may have more influence on the population than other. He described the spreading process in discrete time steps, using difference equations. In the end of his paper however, he also suggested the use of several other structures that can represent the effect people have on each other more accurately, one of these was the use of graphs.

In 2001 Domingos and Richardson published a paper on virus marketing [38]. They wanted to improve the effect of the process by considering the effect buyers have on each other, the so called network effect. They proposed several revolutionary ideas. To represent the interaction between customers they have used a probabilistic framework, that closely resembles graphs. For each customer, they defined the set of customers able to directly influence it. This translates to a graph with buyers as nodes, and connections between nodes if they could directly influence each other. Their models also took the size of the influence into account, represented by weights between the customers. They used to same probabilistic model to describe the network effect itself: the way the opinion of one customer may spread in the network. The goal of their work was to identify the few key individuals that have the greatest influence on the network of buyers. By targeting these individuals with good offers, they proposed, a cascade effect would start with friends recommending the product to other friends and so on. In their paper they proposed the problem of selecting these individuals and several heuristics for solving it.

In graph terms

Somewhat later in 2003 Kempe, Kleinberg and Tardos reformulated the methods above [64]. The Linear Threshold and the Independent Cascade models were proposed. Both models were defined on specific graphs: directed ones with edges showing the orientation of the influence between the nodes, and a value $0 \leq w_{u,v} \leq 1$ was assigned to the edges representing the size of the influence. We will refer to these values as the *edge infection probabilities*. The nodes in the networks have at least two states: *active* and *inactive*. We will also use *infected* and *not infected* as a synonym for the corresponding states. In both models, a non-empty set of *initially active* nodes A_0 are selected. Apart from them, the other nodes start in an inactive state, may become activated during the process and once activated, they remain active until the end. Both processes unfold in discrete time steps or *iterations*, and each iteration can be characterized with the number of active nodes.

Let us consider the Linear Threshold (LT) model first. It has a further requirement for the edge weights: for each node v, $\sum_{w \in N_v^+} w_{u,v} \leq 1$: the weights on the in-edges of a node may sum up to 1 at maximum. This models assigns thresholds to each node at the beginning of the process either randomly or with a user specified strategy. These thresholds are denoted as θ_v for all $v \in V(G)$. The model starts from the set of initially infected nodes at iteration 0. At iteration i all nodes remain active that were active in the previous iteration, and any node v becomes active if the total weight of its active neighbors exceed θ_v: $\sum_{w \in N_v^+} w_{u,v} \geq \theta_v$.

The Independent Cascade (IC) model is somewhat different. It uses edge weights or edge infection values to infect neighboring nodes and its starts from the set of initially active nodes, progressing in discrete time steps. As opposed to the previous models, at any iteration i, only the newly infected nodes in iteration $i-1$ are infectious. This effectively creates another state for the nodes: activated, newly activated and inactive. These newly activated nodes have one chance to active each of their inactive neighbors according to the edge infection probability assigned to the edge connecting them. If the attempt is successful, the target node will become infected in iteration i. For example v becomes infected in iteration $i-1$ it is connected to an inactive node u with an edge weight of $w_{v,u}$. The attempt is made, and it is successful, then u becomes active in iteration i. Since only newly activated nodes are infectious, v cannot infect anyone in iteration i or above. If u has multiple newly activated neighbors, the activation attempts are made in an arbitrary order independently of each other.

It is easy to see that both models are finite, they both terminate if there are no new infections in an iteration. In the LT model this means that for an inactive node, the sum of the weights of the neighboring nodes will remain the same, less than the threshold, so the node will remain inactive from there on. In the IC model if there are no newly infected nodes, no activation attempts will be made. Another important remark of Kempe et al. is that the generalization of the LT and IC models are equivalent.

Tasks for infection models

In the same paper, Kempe et al. have formalized the problem of Domingos and Richardson, the influence maximization problem. For a set of nodes A, the expected number of active nodes at the end of the process is $\sigma(A)$, given that $A = A_0$. The problem is, that for a given graph with edge infection probabilities, which set of initially infected nodes A gives the largest expected infection $\sigma(A)$. They have used the notion of submodularity to prove, that for both the LT and IC models, the influence maximization problem is NP-hard. The proofs can be found in [64]. Fortunately they have also stated that the problem is easy to approximate, and they have given a greedy hill-climbing heuristic that efficiently estimates the results with an approximation guarantee of $(1 - 1/e)$.

In a later paper [65] the same authors have given a different proof of the complexity of their algorithm. It is based on the following idea. In the IC model each activation attempt is made independently of each other. Since each node has only one chance to infect its neighbors, for each edge $w_{u,v}$ will be evaluated at most once. Also, a node $v \notin A_0$ will be infected if it is connected to a node $u \in A_0$ by a chain of successful activation attempts. This way we can rephrase the infection process of the IC model. Consider a graph G' with $V(G') = V(G)$, then add an edge $e_{u.v}$ to G' if v and u are connected in G and the attempt $w_{u,v}$ is successful. In G' a node becomes infected if it is reachable from an initially infected node. This alternative formulation will be important for this dissertation in the construction of the heuristics of our Generalized Cascade model.

A problem closely tied to influence maximization is the computation of $\sigma(A)$ for any given A. More precisely, given any initial vertex set, we are looking for the probability of infection for all vertices in the graph. Again, this proves to be a difficult problem: it is #P-hard for the LT and IC models according to [26]. There are existing heuristics for the efficient computation of these values, like the ones proposed in [28, 29, 30], and we will propose additional heuristics for it in chapter four of this dissertation.

Recently several authors have pointed out a problem in most approaches to infection processes. In order to compute the above models an appropriate graph is required with edge weights between 0 and 1 on all edges: the edge infection probabilities. In many real-life applications the real values of these probabilities are unavailable, they are considered to be constants, or some intuition guided trial-and-error estimation is used. Based on this experience a few models were proposed to give a systematic approach to the estimation or learning of the edge infection probabilities. Goyal et al. [49] gave a learning method on social graphs. Their method is based on "actions", which contain information on the node performing this action, the type of the action and a time-stamp. The method takes a data table with these actions as an input, and uses propagation to estimate the edge infection probabilities. They have successfully tested their method on a network built from Flickr groups. A somewhat more theoretical approach following a similar idea was given by Saito et al. [70]. Their method follows the notion of the IC model closely. They assume, that in each discrete timestep, the set of newly activated nodes is observable, and a probabilistic model is used to predict the edge infection values to estimate the transition between each successful step. They have used a network of blogs as a real-life example to underline their results.

The largest part of the authors' work is devoted to this topic. Chapter four will give a recollection on the Inverse Infection Problem (IIP), its preliminaries, a global optimization method to predict the edge infection probabilities, and an application. Developing the Inverse Infection Problem required a series of additional works. We will start with the introduction of the Generalized Cascade Model, the infection framework IIP uses, and we will give four different heuristics for the efficient computation of it. Then we will describe the Inverse Infection Problem itself, and we will give an optimization method to give a good approximation of its solution. The approach is different from the ones reviewed in the previous paragraph. It does not require the steps of the infection model to be known, rather it uses one or more observations on the beginning and the end of the process. We will end the chapter with a detailed case-study on a very important part of the financial industry: the prediction of default probabilities on bank transaction networks.

Chapter 2

Community detection

We have seen in the previous chapter, that community structure in complex networks is a popular and well-studied field of research. There are many traditional approaches, but recently a large number of overlapping community detection methods have been proposed. The field suffers from a defect though: the concept of overlapping communities lacks a strict definition. There have been attempts to create goodness measurements [53, 77, 98] or benchmarks [76] but these attempts did not become widely accepted. This is clear if we take a look at the results of various detection algorithms, for example the clique percolation method [100], COPRA [52] or line graph approaches [44]. Authors of these papers have either chosen an available benchmark, or have provided a real-life application to prove the worth of their method.

The root of this problem lies in diverse real-life examples. If we take a look at the applications of overlapping community detection methods, we can see, that they have different requirements. These requirements may concern the total number of communities, there average sizes of them, the distribution of their sizes, the number and scale of the overlaps between them. If the task of the scientist is to examine a given network, especially in applied research, he seeks a way to discover meaningful results loosely defined by such requirements. These requirement can be tied to the specific application. For example if we take a look at the word association graph in [74, 91, 100], each of the overlapping community detection methods gives a meaningful, although quite different result. Neither of these results are wrong, each capture a different aspect of the community structure in this network.

We do not aim to solve this problem in this dissertation by proposing another goodness measurement or point out a benchmark to be used. We will also not limit ourselves to a single application. Rather we aim to propose a method that is flexible enough to handle multiple applications with different requirements. As a continuation of our discussion, in this chapter we will present the authors' findings in the field of overlapping community detection.

In the beginning of this chapter we will propose a high-resolution clique-based overlapping community detection algorithm for small world networks called the *hub percolation method*. This method has flexible parameters that govern the various aspects of the resulting communities, like the number and size of the overlaps between them. This allows us to discover different kinds of communities from large, loosely overlapping groups to ones with a dense, highly overlapping structure. We will use well-known benchmark networks [47, 96, 100] to demonstrate the difference between different parameter settings in terms of community sizes, the size of overlaps and the number of singletons: nodes without communities. The method also has a natural extension to weighted networks, that does not require the pruning of edges before the process. The development of the *hub percolation method* was heavily influenced by our works with the clique percolation method [100] and the N^{++} method in [17] as well as several well known benchmark networks like Zachary's karate club [114] and Newman's works [47, 96]. Much of the details of this algorithm came from experiences gained during test runs on these networks.

In subsequent sections, we will evaluate the performance of our method in different ways. We will use the community based graph generator of Lancichinetti and Fortunato [76] to compare the results of our method to the clique percolation method of Palla et al. [100]. We have chosen this benchmark because its relative acceptance and its versatile parametrization to create networks with different kinds of community structure.

We will also present two case-studies. We will examine the communities of an economic network constructed from the Hungarian company register. We will focus our attention on three aspects of the companies: the geographical location of them, the industrial sector they belong to and the age of the companies. In a detailed case study, we will take a look at two word association graphs in different languages and compare the community structure of them. Our main goal is to study the organizing laws behind the formation of communities. Are communities formed around specific words and if so, what characteristics do these words have?

In the end of this section, we will summarize the works of authors on overlapping community detection and present a unified framework, that is capable of representing different kinds of community detection algorithms, both traditional and overlapping ones.

2.1 The hub percolation method

The hub percolation method has two simple ideas at its core. A property of most approaches for overlapping community detection[1] is that cliques (fully connected subgraphs) are considered to be the purest communities. Therefore our method builds communities from cliques. An important observation on real-life networks is, that inside a community some members are more important than others with respect to the role of the nodes in connecting different communities. We will denote these nodes as hubs, and we will use them to extend and join the cliques of a given graph. Cliques containing hubs become extended cliques by absorbing other cliques sharing a certain number of nodes. This is called percolation after [100]. Finally, the joining phase of our method merges these extended cliques if they share the same hubs. Considering these ideas, an outline of the hub percolation algorithm can be seen in algorithm 2.1.

As we have stated in the first chapter, finding the set of all maximal cliques in a graph is difficult, because an n-vertex arbitrary graph may contain $3^{n/3}$ maximal cliques in the worst case [90]. Because of their unique structure, this number is significantly lower in complex networks allowing algorithms like in [27,41,109] to list

[1]In the following chapters, we will refer to overlapping community detection simply as community detection.

2.1. Algorithm. The hub percolation method - outline

Input: Graph G

Output: Set K containing all communities

```
1: Find the set $C$ of all maximal cliques of size greater or equal than 3 on graph $G$.
2: Select the set of hubs $H$.
3: Create the set of extended cliques $C'$.
4: Compute the set of communities $K$: Take the union of extended cliques if one of them contains all the hubs in the other one.
```

the set of maximal cliques in reasonable time even for large networks. In this work we used the modified Bron-Kerbosch algorithm described in the first chaper [41].

The hub selection strategy is an important part of the algorithm. Hubs represent the locally important nodes in the network. As a consequence, whether a node is a hub should depend on the t-neighborhood[2] of the given node, where t is a small number. In our interpretation hubs connect communities, therefore the deciding factor in hub selection should be the number of cliques the vertex belongs to. Each node v is assigned a hub value h_v according to the above rule, then some of them are selected if their value is higher than the average or median hub values in their t-neighborhood. It is also possible to extend the selection strategy to weighted networks. We will discuss hub selection in the next subsection.

In our method, cliques of the network are extended with a a one-step percolation rule, then merged if they share the same hubs. Introducing the filtering parameter $k \geq 2$, let us consider all cliques of size equal to k on the subgraph induced by the set of hubs H. We will denote the set of k-cliques on $G[H]$ as C_H. Then, we expand the elements of C_H according to a one-step percolation rule. Let C_e denote the set of merged cliques $c_e = c_H \cup c_0 \cup \cdots \cup c_\ell, c_e \in C_e$ with $c_H \in C_H$, $c_0, \ldots, c_\ell \in C$ and $|c_0 \cap c_H| \geq 2, \ldots, |c_\ell \cap c_H| \geq 2$ [3].

The last step of our method corresponds to the joining phase of the community detection framework. We merge elementary communities if they contain the same hubs, more precisely, we take the union of two elementary communities c_{e_0} and c_{e_1} if $c_{e_0} \cap H \subseteq c_{e_1} \cap H$. We iterate this process by adding the new merged clique to C_e and removing the original ones. At the end of the process C_e contains the communities of graph G. Note, that depending on the hub selection strategy C_e may contain duplicate members, the merging process eliminates this problem as well. Each element of C_e is a union of the cliques of G and contains at least k hubs. We will refer to the members of C_e as elementary communities. A full description of this method can be seen on algorithm 2.2

The community structure of Zachary's karate club network [114] can be seen on Figure 2.1. This network is a well-known social network, that represents friendships between the members of the club. Our method identifies five communities[4], the most interesting ones being the green and blue ones, as well as the one represented by the red triangle. During Zachary's observation the club split into two parts, because

[2]A t-neighborhood of a vertex v is the set of vertices N_v^t, where $u \in N_v^t$ only if the length of the shortest path from u to v is less or equal then t.

[3]Our experiances on various benchmark networks indicate that this value gives the best performance.

[4]The median hub selection strategy was used with $k = 2$, see subsection 3.1.

2.2. Algorithm. The hub percolation method

```
Input: Graph $G$, parameter $k$
Output: Set $K$ containing all communities
1: Find all maximal cliques of graph $G$ using any exact algorithm or heuristic. Let $C$ denote the set of cliques.
2: For all $v \in V(G)$, let $h_v = |H_v|$, $H_v = \{h|v \in h, h \in C\}$.
3: Select the set of hubs $H$ according to the hub selection strategy.
4: Let $C_H$ denote the set of $k$-cliques on the subgraph induced by the hubs $G[H]$.
5: Create the set of extended cliques $C_e$ according to the following rule: for all $c_H \in C_H$ find all cliques $c \in C$ where $|c \cap c_H| \geq 2$. Let $c_0, \dots, c_\ell$ denote the cliques satisfying this criterion. Create the union of cliques $c_e = c_H \cup c_0 \cup \dots \cup c_\ell$, and add $c_e$ to $C_e$.
6: For all $c_{e_0}, c_{e_1} \in C_e$ add the union of them to $C_e$ if $c_{e_0} \cap H \subseteq c_{e_1} \cap H$, and remove $c_{e_0}$ and $c_{e_1}$ from $C_e$. Iterate until there are no more merges.
7: The set $K = C_e$ contains the communities of graph $G$.
```

some friendships were broken. Most community detection algorithms are able to identify these subgroups even before the actual split. In our case the borders of the green and blue communities represent the borders between the two subgroups, and the red group contains the edge that was broken when the split occurred.

2.1.1 Hub selection strategies

Hub selection is a crucial part of the algorithm. As we have mentioned before, each node in the graph is assigned a hub value based on the number of cliques it belongs to. Based on this value the selection strategy chooses the set of hubs H. Hubs represent "locally important" nodes so the criterion of the hub property of nodes or "hubness" should depend only on the tight neighborhood of the node. In our interpretation, this criterion depends on some simple statistical property of the first or second neighborhood of the given node, namely the average or median of neighboring hub values.

At the beginning of our work on several famous benchmark networks [96, 114] we have quickly found out, that the 2-neighborhood strategies are often not robust enough to select the appropriate hubs: hubs were relatively rare, which resulted in small overlaps and a larger than acceptable number of nodes without community memberships. A general experience was, that hubs should be "common enough", so that most of the nodes have one or more in their direct neighborhood.

Considering this, the 1-neighborhood median selection strategy provided the best community structure on these benchmark networks. This may not be the case, however, with other real-life networks. In order to extend our algorithm to handle different kinds of networks, we can suggest another hub selection rule. Still considering only the direct neighborhood of nodes, we calculate the average hub value and multiply it with a parameter $q > 0$. If the hub value of the node is higher than the mean, we select it as a hub. This approach makes hub selection more flexible, allowing the algorithm to adapt to different requirements. A small value of q selects higher number of hubs resulting in larger communities with greater overlaps, while increasing q has the opposite effect. This also allows the algorithm

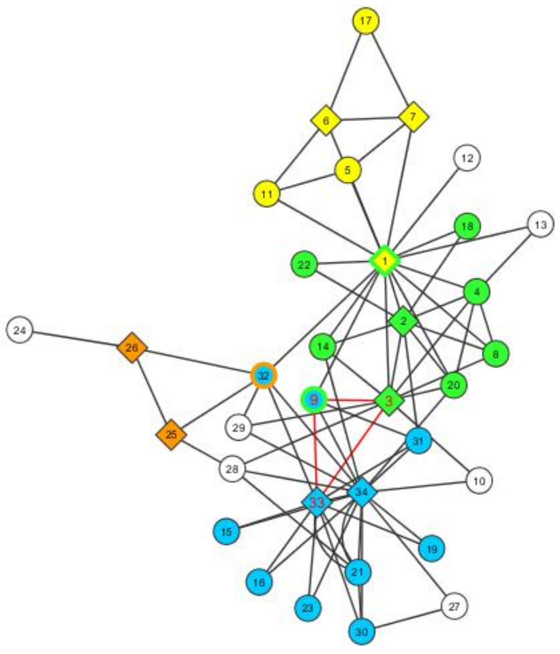

Figure 2.1: The communities of Zachary's karate club network [114]. Hubs are marked as diamond shapes. Nodes with multiple colors indicate overlapping nodes. The median hub selection strategy was used with $k = 2$. Nodes $9, 3, 33$ form an additional community and node 9 belongs to three communities.

to discover several layers of community structure on the network.

Finally, hub selection can be extended to weighted graphs in a natural way. As before, the hub value of a node is the number of cliques it belongs to. Then the values are multiplied with the strength of the node. After this, the process is the same as in the previous strategies.

As a summary, we propose the following hub selection strategies:

- 1-neighborhood median: A node is selected as a hub if its hub value is greater than the median of hub values in its one-neighborhood.
- 1-neighborhood mean with multiplier: A node is selected as a hub if its hub value is greater than the mean of hub values in its one-neighborhood multiplied with a parameter $q > 0$.
- 1-neighborhood weighted mean with multiplier: The hub values are multiplied with the strength of the nodes. Beside this, the strategy is the same as above.

As a recommendation, the median strategy should be tried first, and if it does not give satisfactory results the average strategy should be used with $q = 1$ initially, decreasing or increasing its value in small steps depending on the requirements. In practice $0 < q < 2$ seems to hold.

2.1.2 Implementation

The bottleneck of the algorithm is finding all maximal cliques in graph G. A general graph with n vertices may contain up to $3^{n/3}$ maximal cliques. In correspondence, the original algorithm of Bron and Kerbosch has a worst-case running time of

$O(3^{n/3})$. In small-world networks however, the number of maximal cliques is smaller by magnitudes, decreasing the running time of the algorithm. Furthermore, refinements of the Bron-Kerbosch algorithm have been published in recent years, enabling the use of this method on large sparse networks [41, 109]. In cases when even faster computation is required, there are existing heuristics for clique search [31].

The hub value of each node can be calculated in a single pass on the set C of cliques. All of the hub selection strategies suggested in the previous section have a local fashion: they can be computed in a single pass on the vertices and their one-neighborhoods.

The computation of C_H does not require a repeated run of the Bron-Kerbosch algorithm on $G[H]$, since the cliques of G contain the cliques of $G[H]$ as subsets. Therefore it is enough, that for each $c \in C$, if $|c \cap H| \geq k$, simply add all k-combinations of c to C_H. Depending on the size of the network and the hub selection strategy, H may be quite large, but the use of flags on the nodes of the graph G to signal the hub property can reduce the computation of this step to a single pass on C. The percolation step can be executed by computing the 1-neighborhood of each $c_H \in C_H$. Let c_H^+ contain all of the direct neighbors of vertex set c_H, and initially let $c_e \leftarrow c_H$. For all nodes $v \in c_H^+$, if $|\{v\}^+ \cap c_H| \geq 2$ add v to c_e. Again this step can be computed by making a single pass on C_H.

In order to make the joining step, the computation of the hubs of each elementary community is required: for each $c_e \in C_e$ let $c_{H_e} = c_e \cap H$. Let C_{H_e} denote the sets of hubs of the elementary communities. An important remark is, that $C_{H_e} \neq C_H$ since in the previous step additional hubs may have been added to the elements of C_H. Removing the "sub-hubs" (hubs being contained in other elements of C_{H_e}) can be executed in quadratic time in worst case. In general, performance can be improved by sorting C_{H_e} in descending order according to the sizes of $c_{H_e} \in C_{H_e}$. After this, starting from the first element, remove all the sets of vertices from C_{H_e} which are subsets of the first one, then repeat for the second, third, ... until no more vertex sets can be removed from C_{H_e}. Finally, the elements of C_e and C_{H_e} must be compared: for all $c_{H_e} \in C_{H_e}$ find all $c_e \in C_e$ where $c_e \cap H \subseteq c_{H_e}$ and take the union of these vertex sets.

We can conclude, that the two most time-consuming steps of the method is the computation of C and C_H, all other operations take at most quadratic time[5]. The algorithm of Eppstein and Strash [41] is able to list all maximal cliques in large sparse networks in reasonable time. For faster computation heuristics [31] or the use of quasi-cliques [87] can be applied. The size of C_H depends on two factors: the size of H and k. The former is governed by the hub selection strategy, the latter is a parameter of the algorithm. Choosing a different hub selection strategy, that produces a smaller number of hubs, or decreasing k may speed up computations.

2.1.3 Sensitivity to parameters

We have created the hub percolation method with the intent to provide a versatile tool for community detection. Therefore, an important question arises: how does the hub selection strategy and the filtering parameter influence the community structure found by the algorithm? In order to examine their effect, we will use several well-

[5]We have also conducted experiments, and found that clique detection may take from 60% up to 95% of the running time.

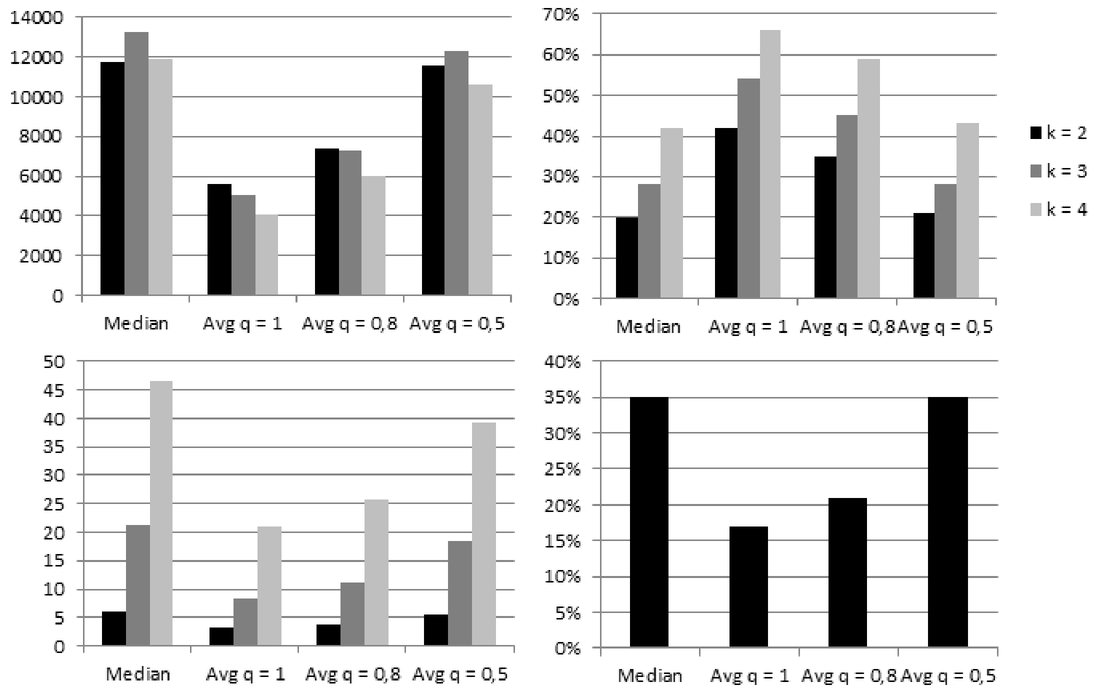

Figure 2.2: Upper left: Number of communities for different hub selection strategies and values for k; Upper right: The percentage of nodes without communities; Lower left: The average overlap; Lower right: The percentage of hub nodes in the network.

known benchmark networks including the word association graph of Palla et al. [100], a scientific collaboration network [96] and a graph of American football games [47].

The first network we will examine was created by Newman [96] on the condensed matter archive at www.arxiv.org based on preprints posted to the archive between January 1, 1995 and March 31, 2005. The graph is undirected, unweighted and contains 39540 nodes and 175683 edges. We will evaluate the median and average hub selection strategies and we will also experiment with different values for k. We will measure the number of communities, the average overlap[6], the number of singletons[7] and hubs in the network. We will also present the community size distribution for each selection strategy.

On Figure 2.2 we have compared four different hub selection properties: the median strategy and the average strategy with values $q = 1, 0.8$ and 0.5. The number of communities is the largest and the number of singletons is the smallest with the median strategy and the average strategy with $q = 0.5$; these strategies provide the largest cover on the network. The number of hubs is also the largest with these strategies: roughly one in three nodes, this confirms our expectations, that hubs should be "common". We can see, that the average overlap and the number of singletons increases with k, while the number of communities does not change. The reason for the above fact is that by increasing k, the nodes are concentrated in highly overlapping communities keeping the number of communities constant, while many nodes are left out of the community building process.

We will further examine the average hub selection strategy with $k = 2$ on Figure 2.3. The main observations remain the same with higher values for k. As before,

[6]The sum of the cardinalities of all community divided by the number of nodes.
[7]Nodes without communities.

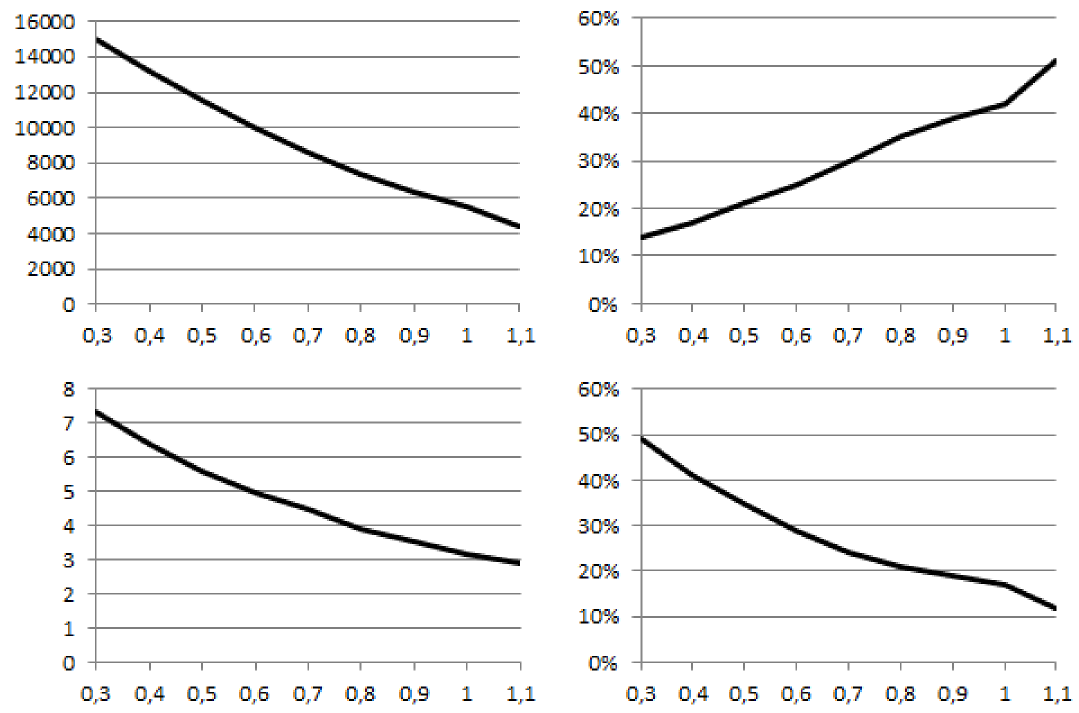

Figure 2.3: Upper left: Number of communities for the average hub selection strategy with $q = 0.3, \ldots, 1.1$ and $k = 2$; Upper right: The percentage of nodes without communities; Lower left: The average overlap; Lower right: The percentage of hubs in the network.

the number of hubs and communities grows inversely proportional with q, while the number of singletons grows proportionally with it. The average overlap slowly decreases when q is increased, indicating that decreasing the number of hubs causes communities to become smaller and scarcer.

We can see, that the community size distributions follow a power-law on Figures 2.4 and 2.5. The median hub selection strategy is depicted with different values for k. Increasing k results in much larger communities: With $k = 2$, the largest community had 255 members, with $k = 4$ the maximum was 869. This confirms our previous observation, that increasing k creates a highly overlapping community structure. Similar observations can be made with the average hub selection strategy. Increasing q decreases the number of communities evenly among the community sizes, even the size of the largest communities does not change much.

A strict requirement for all community detection algorithms should be, that the number of nodes left without community memberships should be minimized. Therefore we can conclude, that the filtering parameter should be kept as low as possible, and the ratio of hubs should be above 30%.

We have measured the running time of our method as well[8]. The results for the average hub selection strategy with different values of q and k can be seen on Figure 2.6. We have seen before, that decreasing q increases the number of hubs – the size of H and C_H. This directly increases the computational time of the joining phase. The filtering parameter k also has an impact on the running time of the method,

[8]We have implemented our method in JAVA, and we have used a computer with an Intel i7-2630QM processor, and 8 gigabytes of memory.

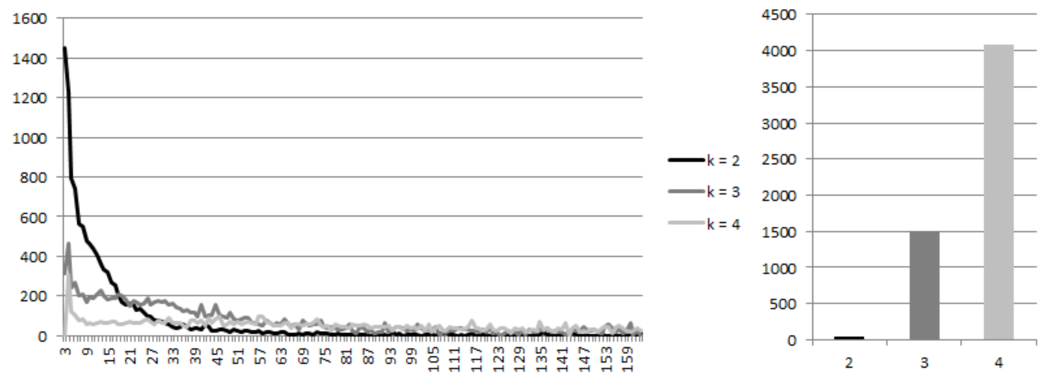

Figure 2.4: The community size distribution of the median hub selection strategy with different values of k. Left: The number of communities with size below 150. Right: The number of communities with size greater than 150.

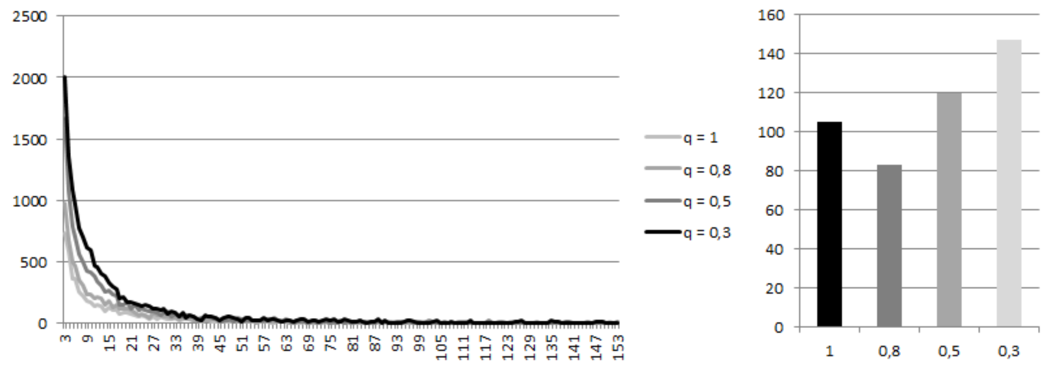

Figure 2.5: The community size distribution of the average hub selection strategy with different values of q, $k = 2$. Left: The number of communities with size below 150. Right: The number of communities with size greater than 150.

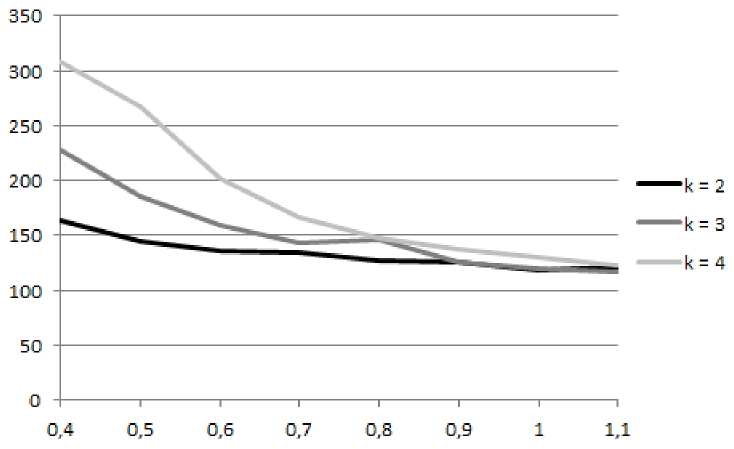

Figure 2.6: The running time measured in seconds with the average hub selection strategy and different values of q and k.

since it influences the size of C_H. As a conclusion we can say, that the filtering parameter should be kept as low as possible, and the average hub selection strategy should be used to further refine the results of the algorithm.

We can draw similar conclusions on the other two networks, with a few exceptions. The relationship between the ratio of hubs, the community size and the average overlap is the same in all networks. The ratio of singletons shows a similar behavior as it grows inversely proportional to the ratio of hubs. There is a difference however. The graph of football games contains no singletons for the majority of the parameter configurations, while the ratio of singletons never goes below 30% in the word association network. This can be explained by the difference in the structure of the networks. The graph of football games is an union of cliques by definition, while word associations do not have this property. Since our method is clique-based, it is able to cover all nodes of the former test set, while in the latter case nodes not part of any triangles are left out of the building process.

The relationship between the ration of hubs and the hub selection strategies is also similar, that is for the average selection strategy increasing q decreases the number of hubs. However, the exact pairs of these values change together with the networks. For example setting $q = 0.5$ results in 35% of nodes being selected as hubs on the collaboration network, 21% on the word association network and 90% on the graph of football games. Therefore in any application, it is important to find the hub selection strategy that produces the ratio of hubs so that the number of communities, the size of the overlaps and the number of singletons move according to the specifications of the application.

We have previously concluded that the filtering parameter should be kept as low as possible to reduce both the ratio of singletons and the computation speed. As we will see below, there are some situations where a higher value is desirable. In the next chapter we are going to examine networks with a large number of highly overlapping communities.

2.2 Performance on benchmark networks

In order to evaluate our method, we have used benchmark networks created with the graph generator of Lancichinetti and Fortunato [76]. We have generated both weighted and unweighted networks, with the following parameters:

- We have created undirected graphs with $|V(G)| = 1000$
- The average degree was 15
- The maximum degree was 50
- The exponent of the degree distribution was -2
- The minimum community size was 3
- The maximum community size was 25
- The exponent of the community size distribution was -1
- The mixing parameter μ_t was between 0.1 and 0.2

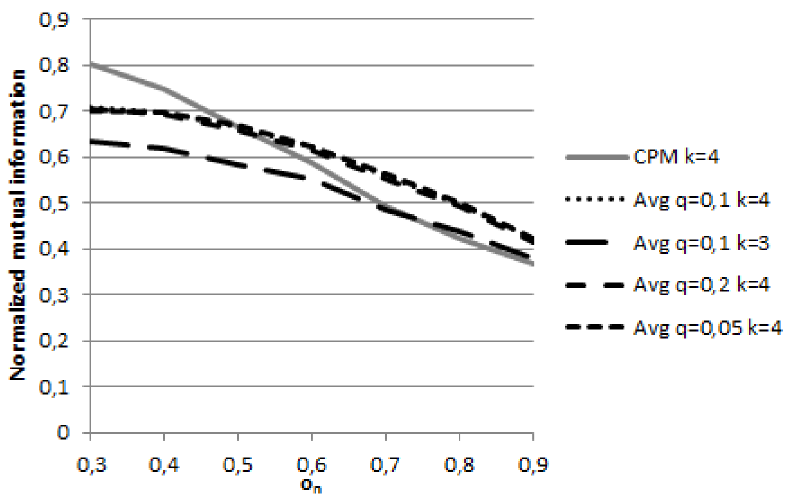

Figure 2.7: The performance of hub percolation compared to CPM with $\mu_t = 0.1$.

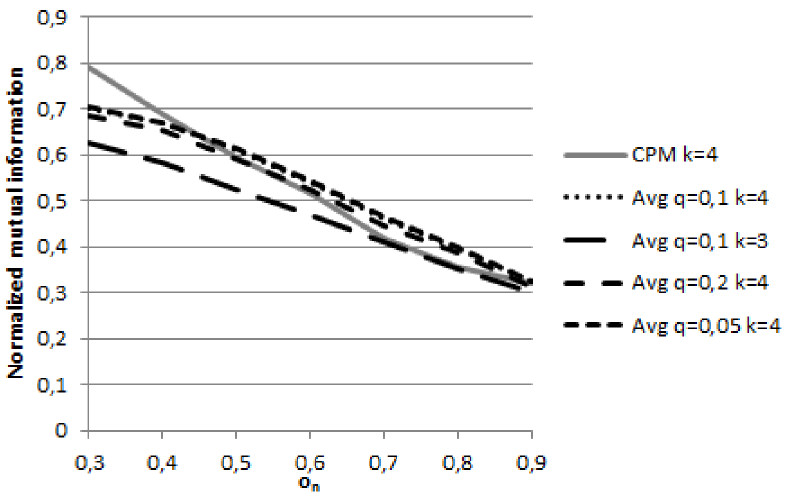

Figure 2.8: The performance of hub percolation compared to CPM with $\mu_t = 0.2$.

- The fraction of overlapping nodes o_n was between 0.3 and 0.9

A detailed description of the used model and its parameters can be found in [76]. We have selected the parameters above, because they are close to the recommendations of Lancichinetti and Fortunato, yet they provide a challenge to our method. Again following the recommendations of the above authors, we have used mutual information [77] to measure the similarity between the communities given by our method and those of the benchmark. Because of the probabilistic nature of the benchmark we have generated 10 different networks for each parameter configuration and averaged the similarity measurements. We have also compared the performance of our method to that of the clique percolation method. We have tried several values for k-clique percolation, and have found that $k = 4$ clearly provides the best results, therefore we have used this parameter setting for comparison.

On Figures 2.7 and 2.8 we can see that the best results were provided by the 1-neighborhood average hub selection strategy with low q values and $k = 4$. We can also see, that our method reaches peak performance at $q = 0.1$, but the selection of q has little influence on the results. The median selection strategy performs poorly on

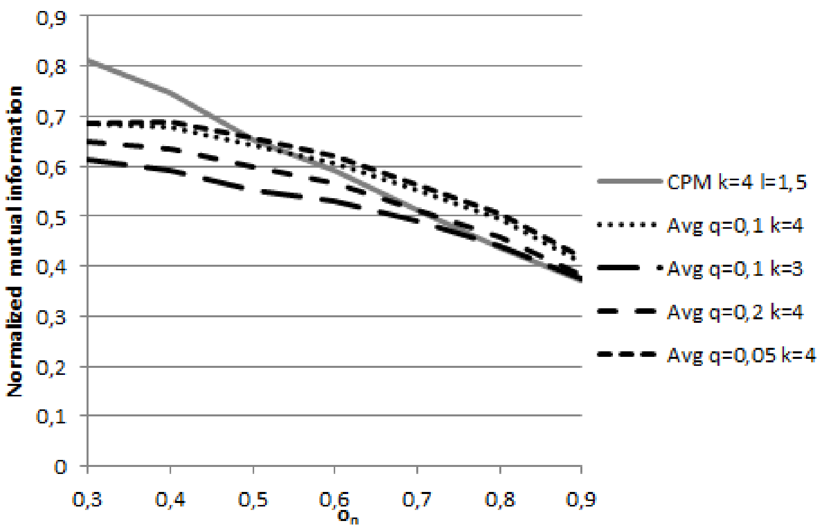

Figure 2.9: The performance of hub percolation compared to CPM with $\mu_t = \mu_w = 0.1$.

these networks, and increasing or decreasing k worsens performance. If we compare our method to CPM, we can conclude, that hub percolation gives better results on networks with a high number of overlapping nodes.

Our observations remain the same when using the weighted benchmarks of the same authors with the recommended parameters $\mu_t = \mu_w$ and $\beta = 1.5$. Low values of q and $k = 4$ gives the best results for hub percolation, and 4-clique percolation with a weight threshold $l = 1.5$ is the best for CPM. As before, hub percolation gives better results on networks with a high number of overlapping nodes.

2.3 Case-study on economic intersection graphs

In this section, we will examine the community structure of a specific economic network constructed from the Hungarian company register. We will consider a network of companies: each vertex is a special type of company (Ltd.), and the companies are connected if they share a common owner (or member in the case of Ltd.'s). We will call this network as an intersection network, because two vertices are connected, if the sets of owners associated with them have a non-empty intersection. Due to the changes in the regulations governing the company register's construction, there are large amounts of missing and erroneous data. The register has a somewhat unordered structure so the identification of the companies, owners and the construction of the graph itself required the application of several data mining methods, data cleaning and filtering.

The resulting graph is not connected, there is a high number of small disconnected components in it, but fortunately it contains a giant component as well. The small components often cannot be divided into two or more communities, thus they do not provide useful information about the structure of the graph. Therefore, in our analysis we will consider only the giant component. This graph is a small-world network with the previously mentioned properties. It has 239685 vertices and 1423080 edges. Depending on the hub-selection strategy, our method was able to discover the

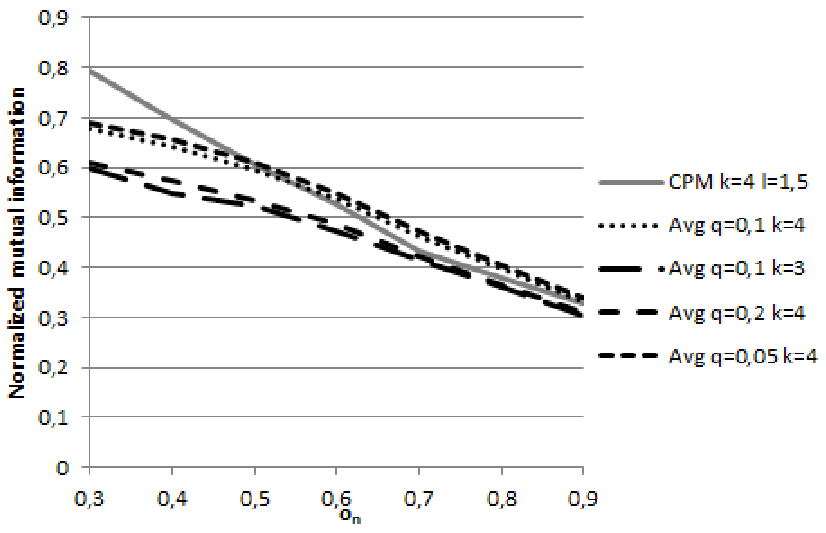

Figure 2.10: The performance of hub percolation compared to CPM with $\mu_t = \mu_w = 0.2$.

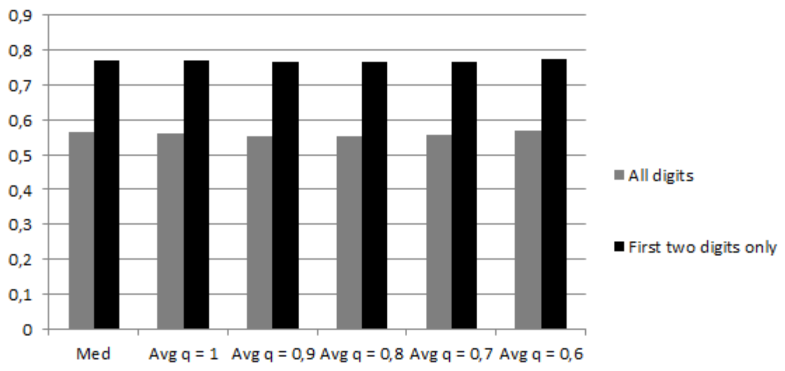

Figure 2.11: The locality of the communities of the intersection network with different hub-selection strategies, using all digits of the zip-code (left) or the first two digits (right).

community structure of this network in 5-7 hours[9]. We have considered comparing our method with CPM on this dataset, but the publicly available[10] implementation was unable to produce results.

There are several points of interests regarding the community structure of the network. Here, we are going to focus on three of them. The first one is the geographical location of these groups and companies inside them. Our main question is, are the communities of the graph local in a geographical sense? Using the register, we can assign zip-codes to the companies, and by counting the number of different zip-codes inside the community – the frequency of individual zip-codes, we can easily address the above question. We can further divide the frequency of the most frequent location by the size of the community, and by averaging this fraction over all of the communities we can represent the locality of these communities as a simple number. The structure of the zip code also allows us to fine-tune the resolution of the analysis. The Hungarian zip-code contains four digits: the first one divides the

[9]On the same hardware as above.

[10]We have used CFinder [100] downloaded from http://cfinder.org/

country into nine large regions, the first two identifies 80 sub-regions. On Figure 11 we can see the computed average locality of the communities. Both the accurate locations – all digits of the zip-code – and the sub-region classification system is used. We can conclude, that the communities are local indeed; in average 77% of companies inside communities belong to the same sub-region, and even in the case of the accurate locations, this percentage does not go below 55%. This implies, than companies owned by the same people tend to stay in the same geographical area. It is important to emphasize, that we are observing a special form of companies: the Ltd.'s. Our results makes sense, because this company form is popular for small companies, that do not have the resources to cover a large area. On Figure 2.11 we can also see a comparison between the different hub-selection strategies[11]. Even though the number of communities and the size of overlap changes according to the observations in the previous section, all strategies gave a similar stable performance.

We can perform the same analysis considering the industrial sectors the companies belong to. Do the communities of the graph belong to similar industrial sectors? The sector classification numbers for the individual companies are available, but due to changes in regulation it is impossible perform a high-resolution scan. On the other hand we can make use of a rough classification system containing 118 different industrial sectors. The method is the same as before: we compute the most frequent sector for each community, and we average the relative frequencies over all communities. As a result we can say, that in average 84% of the companies inside the communities belong to the same industrial sector. The communities are even more "local" to the industrial sector most of their members belong to, than to their geographical location. The reason for this is similar to the previous one: small companies tend to specialize, and it is rare for an owner to have an interest in multiple sectors. Again, we have compared different hub-selection strategies and found, that they have similar performance.

Our last point of interest is the age of the companies. Since the date of establishment is available for all companies, we can ask the question: Were the companies inside the communities of the graph established in a short time period? We can answer this question by computing the standard deviation of the dates of establishments for all companies. The expected value of the standard deviation inside the communities is 5 years. This relatively large value indicates, that the establishment of these companies is spread in time over a considerable interval.

As a conclusion we can say, that our method is capable of identifying communities that share a common geographical location and industrial sector.

2.4 Case-study on word association graphs

In the last case-study of this chapter, we are going to introduce another application of the hub percolation method. We are going to study and compare two word association graphs in different languages. One of them is the well-known network created from the University of South Florida word association norms [91] by Palla et al. [100]. The other one is a Hungarian network constructed by László Kovács using data collected from Internet users [74]. Our main goal is to study the organizing laws behind the formation of communities. Are communities formed around specific

[11] $k = 2$ was used in all experiments.

words and if so, what characteristics do these word have?

The first word association graph we will examine contains English words, and is based on the University of South Florida word association norms [91]. Work on this database began in 1978, and it incorporates almost three-quarters of a million associations from 6000 participants. The participants were presented with a discrete association task. They were given a stimulus word, and they had to respond with the first word that came to their mind that was meaningfully related or strongly associated to the presented word. It is easy to see that this structure is a graph. The nodes represent the words, and a directed edge points from one word to another if there is an association from the first word into the other. It is also possible to assign weights to the edges based on the number of associations. The network studied here was created by Palla et al. [100] for the purpose of testing their community detection algorithm. They have created an undirected network by placing undirected edges between the words if there was a directed edge between them in both directions in the original graph. The weight of the new edge was the sum of the two old ones. Then a weight threshold was applied and only edges with weights above this threshold were kept. The resulting network had 7207 nodes and 31786 edges.

The second network was created from the Internet word association database “Agykapocs.hu" by László Kovács [74]. The database collects word associations in Hungarian continuously since 2008. As before, registered users were presented with discrete association tasks. The network studied here was created by omitting the direction of the connections. It has 25431 nodes and 75584 edges.

2.4.1 Community structure analysis

Our goal is to identify the laws behind the formation of communities in word association graphs with the help of the hub percolation method More precisely, we are going to examine if communities are centered around specific words. We will call these words as “central” and the identification of them will be our first task.

In order to do this we are going to construct a loose ordering of the nodes of the network with more central words at the top of this ordering. We will denote this ordering as the *global ranking*, and the words of the networks will have a rank property based on their place in this ordering. This is a “loose” ranking, because only the relative positions of the words is interesting to us. Small differences in the ranking – e.g. whether they are the 6th or 10th in the word list – are not important for our analysis. After this word list is created some general observations will be made on the global ranking of the vertices of the individual networks.

Finally, we are going to examine if these nodes are indeed central in the communities. After computing the community structure of both networks with the hub percolation method, we are going to construct *local rankings* inside the communities. There are two ways to locally rank the nodes: we can simply order the vertices of the communities according to the global ranking or we can construct an independent ranking on the subgraph induced by the nodes of the community. By comparing the rankings we will show, that the communities found by the hub percolation method are formed around the central nodes of the networks.

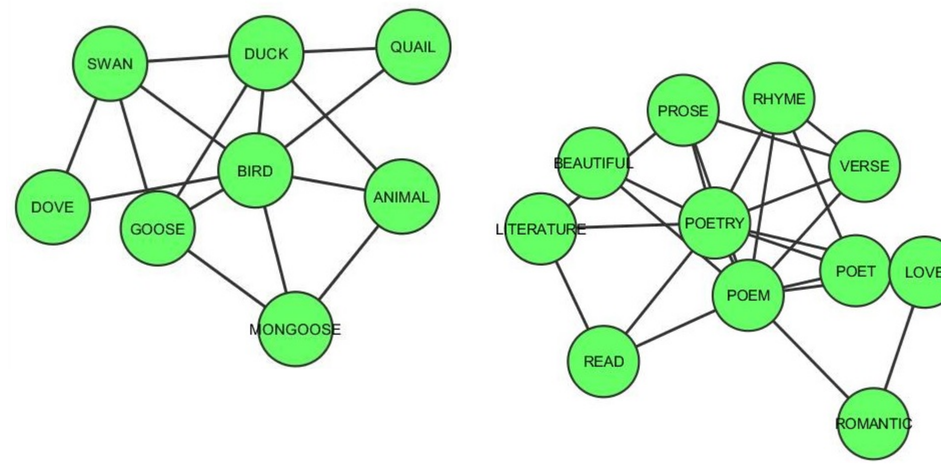

Figure 2.12: Communities of the English word association graph

2.4.2 Global ranking

Two communities of the English network can be seen of Figure 2.12. We can see, that they are centered around category names or collective nouns[12]. Words like these are associated with many other words, therefore it makes sense, that they are the centers of the communities as well. We will adopt this intuition and construct the global ranking according to it.

In order to continue, we must first decide which words are considered to be “central". Manually deciding this for each word in both graphs would be tedious, but fortunately we can make an observation. According to our hypothesis central words are often associated with other words, therefore it is worthwhile to consider measurements like PageRank or simply the degree of a node and order the vertices of the networks according to them, with the highest values at the top. In our works, we have used the PageRank values, but it is well known, that in undirected networks there is a high correlation between these measurements. If we take a look at the top few hundred words of any of these orderings we can see, that they are exactly the words we are looking for: collective nouns, general adjectives, category names, etc. We can see the 12 highest ranking words from both networks in Table 1.

	1	2	3	4	5	6
HUN	Money	Love	Help	Car	Man	Movie
ENG	Food	Water	Money	Car	Bad	Animal
	7	8	9	10	11	12
HUN	Shock	Economy	Manager	Politics	Nature	Advertisement
ENG	Good	Paper	House	Love	Work	Clothes

Table 2.1: The highest ranking words in the global ranking

If these words are central to the communities of the networks they must be present in them. For the English network we can say, that 79 % of the communities

[12]The first one is centered around the words bird and animal, while the second one is centered around poem and poetry.

contain at least one word from the top 300 ranking words. In the Hungarian network the top 100 words are able to cover 95 % of the communities. Despite the differences of the two analyzed networks it is common in them, that the most central words are members of a very large fraction of the communities.

2.4.3 General observations

We can make several observations on the global rankings of both networks. If we examine the types of the top 200 words we can say, that almost two thirds of them are nouns in both networks. The remaining words are either adjectives or verbs. Fortunately just a small number of the words in the lists is a homonym or a polyseme, so there was no need to apply special rules for these words.

We can draw some interesting conclusions just by comparing the highest ranking words in the English and Hungarian networks. For example, there are only 59 common words in the top 200 highest ranking words in both networks. These common words can be grouped together.

- Basic needs: food, money, car, family, child, etc.,
- Everyday activities: work, travel, sleep, etc.,
- Common adjectives: colors, good, bad, expensive, cheap, etc.

This means that 141 words are different. Some of the differences between the English and the Hungarian lists can be explained by the fact, that the words of the networks were collected at different times (e. g. cell phone, email), on different geographical locations (e. g. Hungary, Europe). Structural characteristics of the two languages (English and Hungarian) can account for the variation in the highest ranking words too. Further sources and causes for the differences will be investigated closely in the future.

2.4.4 Local ranking

Previously we have constructed an ordering of the vertices of the network based on how central they are, and found, that the most central vertices are members of almost every community. We did this because we assumed that communities are formed around the frequently associated words. In this section we are going to test this hypothesis. After we have computed the communities of both networks with the hub percolation method, we will examine each community. The global ranking imposes an ordering on the vertices of each community according to their global centrality. We can also construct a *local* ordering by considering the subgraph induced by the vertices of the community, and creating a ranking on the subgraph with the same method.

This way we have two rankings on the nodes of a community: one representing how central they are from a global point of view, another considering only the vertices of the community itself. By comparing these rankings we can decide how well the inner structure of the communities follows our intuition: are the central words of the global ordering also central inside the communities?[13]

[13]That a globally central node has a locally central role too is far from trivial, we can easily consider subgraphs, that do not keep this structure, for example an edge in a star graph.

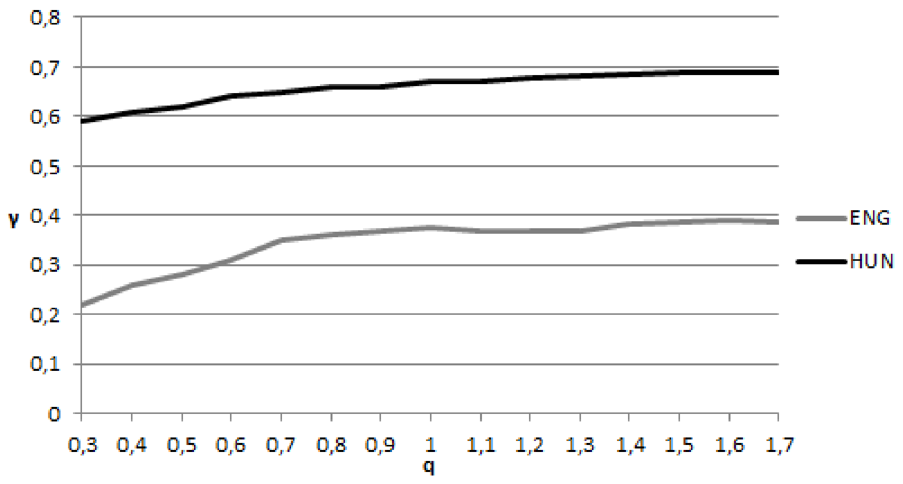

Figure 2.13: Rank correlation (γ) values for both networks with different values for q.

To compare the rankings we are going to use the rank correlation coefficient of Goodman and Kruskal:

$$\gamma = \frac{n_c - n_d}{n_c + n_d},$$

where n_c denotes the number of pairs of words, that are in the same relative position in both rankings, and n_d denotes the number of pairs, that are in the opposite order[14].

At this point we are going to use the customizable nature of the hub percolation method and find the setting, that maximizes this correlation. We can see on Figure 2.13 the averaged rank correlation values for the communities of both networks computed with different values of q, a parameter of the detection method. We can see, that there is a loose positive correlation between the rankings on the English network, while on the Hungarian network this value is much greater. This confirms our hypothesis: although there are differences between the networks, the globally central words are also central to each individual community created by the hub percolation method.

2.5 Generalized community detection framework

The authors of [17] described a general framework for overlapping community detection. In this section we summarize their approach. We define this framework for an undirected simple graph G with vertex (or node) set $V(G)$ and edge set $E(G)$. Edges might have arbitrary weight.

According to the framework introduced in [17], most community detection algorithms consist of two phases. Taking an input graph G, the first phase constructs a hypergraph $\mathcal{F} = (V, \mathcal{H})$, where $V(\mathcal{F}) = V(G)$, and $\mathcal{H} \subset 2^V$. The elements of $\mathcal{H}$ are considered the *building blocks* of communities. The second phase adds a distance function d to set $\mathcal{H}$, creating a metric space $\mathcal{M} = (\mathcal{H}, d)$. Using function d, a

[14]For example consider two orderings: ABC and ACB. AC and AB are in the proper order, but B and C have different relative positions, therefore the correlation between them is 1/3.

clustering algorithm creates a set of clusters $\mathcal{C}$. Finally, the arising clusters are associated to the subsets of V such that $K_i = \cup_{H \in C_i \in \mathcal{C}} H$, where K_i, the ith community corresponds to C_i, the ith cluster and K_i is just the union of the vertex set of those hyperedges that belong to C_i.

It is easy to show, that this framework applies to most community detection algorithms. In the case of the clique percolation method [100], $\mathcal{H}$ contains the k-cliques[15] of the original graph, and function d is:

$$d(K_i, K_j) = \begin{cases} 1, & \text{if } |K_i \cap K_j| = k - 1, \\ \infty, & \text{otherwise} \end{cases}$$

In the same paper [17], the authors have proposed the N^{++} community detection algorithm with a general distance function, where $\mathcal{H}$ is the same as above and $d(K_i, K_j) = 1$ only if $|K_i \cap K_j| \geq c$, where c is a parameter of the algorithm. In other cases $d(K_i, K_j) = \infty$. This method has proven its usefulness in applications [34, 73].

It is easy to see, that the general framework proposed in [17] applies to the hub percolation method. The edges of the hypergraph correspond to the extended communities in C_e, while the distance function is

$$d(K_i, K_j) = \begin{cases} 1, & \text{if } K_i \cap H \subseteq K_j \cap H \text{ or} \\ & K_j \cap H \subseteq K_i \cap H, \\ \infty & \text{otherwise} \end{cases}$$

It is also possible to describe non-clique based methods using this formulation. In the case of COPRA [52], each element of $\mathcal{H}$ initially only contains one unique vertex $v \in V(G)$. In the second phase, these are joined according to a belonging coefficient. A threshold is introduced to provide a lower bound for community membership.

2.6 Conclusions

In this chapter we have summarized the author's work on static community detection. We have proposed the hub-percolation method: a clique-based high-resolution overlapping community detection algorithm. This method is based on two observations. Cliques are the most natural representations of communities, and some vertices are crucial in the birth of communities; they connect different sub-communities together by forming bridges between them. There are multiple ways to select these vertices. The authors have suggested several hub-selection strategies, some of them have tunable parameters. The method also has a filtering parameter k which influences the size and structure of the overlaps between the communities. Adjusting k and the hub-selection strategy allows the user to apply this method to a variety of small-world networks with different densities. It also allows the user to discover several layers of community structure on the same network. We have examined the effect of different parameter choices on several well-known benchmark networks. We have concluded, that the selection strategy should be chosen so that 30-50% of the vertices are selected as hubs and k should be kept low to minimize the number of singletons.

[15]Fully connected subgraphs containing exactly k nodes.

We have used benchmark graphs created with the graph generator of Lancichinetti and Fortunato [76] to evaluate the performance of our method. Using these networks, we have compared our method with the well-known clique percolation algorithm of Palla et al. [100]. We have concluded, that in average the two methods have similar performance, but hub-percolation gives better performance on networks with a high number of overlapping nodes.

A slight adjustment in the hub-selection strategy allows us to handle weighted networks without the need to filter the graph edges according to some possibly non-trivial weight limit. Using the previously mentioned graph generator, we have created weighted benchmark graph, and compared the goodness of our method with those of the weighted clique percolation algorithm. Our conclusions were the same as before: similar performance in average, better results with high overlaps.

We have provided two case-studies as well. One of them was an economical application. An economic intersection networks is a network where the vertices represent companies, or more precisely Ltd.'s, and the companies are connected if they share one or more members. Our method is able to identify communities, that are geographically local and belong to the same industrial sectors in reasonable time considering the size of the network.

We have also used the hub percolation method to discover the community structure of two word association graphs in different languages. We have examined the words, that have high degrees and centralities and found, that they are very often category names, common adjectives or collective nouns. Our main goal was to confirm whether the central nodes of the networks are also central to the communities of them. We did this by constructing a ranking of words both on the whole network and inside communities and then we compared the correlation of these rankings. We have shown, that even though there is a difference between the networks, this correlation exists; it is weaker in the English network and stronger in the Hungarian one.

As a conclusion of our work, we have provided a simple framework to describe various community detection algorithms, and have shown the representation of CPM and the hub percolation method in it.

Chapter 3

Dynamic communities

As a conclusion of our discussion on community detection, we will examine the changes communities go through in time. Based on the first paper the authors published [1], this chapter deals with the concept of dynamic community detection. We have given an introduction into this topic in chapter 1.2.1, but we will review the most basic concepts here as well.

Out motivation for creating this method came from the applications discussed in the end of this chapter. The applications had two distinct requirements not present in available methods. The first one was simply the speed. The graphs to be examined were large real-life networks. They had the properties of complex networks, but they also had many vertices and edges, and it became necessary to develop a method capable of handling such graphs.

The second difference was the kinds of events communities go through during the temporal dynamics of the network. We have decided to use overlapping community detection methods to approach the applications, but we were not satisfied with the clique percolation method because of the reasons discussed previously. Since the method used by Palla et al. in [99] uses a special property of CPM it was also inapplicable to us. We have also found that the basic events they have used are not enough to describe the changes in our networks.

Therefore we have created the method seen in this chapter. We will begin the discussion by giving a short recollection of the end of chapter 1.2.1 on dynamic community detection. We will review the approach of Palla et al.: the events they have used, the available implementation and their results. Then we will motivate and discuss our approach. We will review the extended community events, and the algorithm used to find them. We will give details about the possible implementation and some complexity results. Finally, we will examine the dynamics of the communities in two large real-life networks: a social network and a network of bank transactions.

3.1 Previous works

As we have seen in the introduction, the dynamics of these networks is usually represented as a series of graphs $\{G_i\}_{i\in\mathcal{T}}$, where $\mathcal{T} = \{1, \ldots, \ell\}$ represents a discrete time set. The number of time instances usually depends on two things: the length of the period, in which we observe the network, and the resolution, which governs,

[1]Note, that the hub percolation method had not been developped at this time

that in a given time period, how many "snapshots" do we take from the network. Although one might try to consider the whole history of the graph series in one step, for large graphs this is not feasible. Both Asur et al. [6] and Palla et al. [99] have chosen the path of following the entities from G_i to G_{i+1}, and reconstructing the whole process out of these. Note that Asur et al. deals with clusters, while Palla et al. follows the lifetime of (overlapping) communities. They both took a normative approach to communities: they have decided upon the possible events that might happen to a community. In other words, they have simply listed the basic events of a hypothetical classification. We reproduce some of the relevant work of Palla et al. as follows.

The *basic events* described in [99] are the following:

- Birth, when a new community emerges without predecessor.
- Death, when a community disappears without successor.
- Merge, when several communities join together to form a new community.
- Split, when a community splits into several new communities.
- Growth, when a community gains new members.
- Contraction, when a community loses members.

Assuming the above described events, the problem is reduced to the following. Given the graphs G_1 and G_2, describing the starting and destination graphs, compute the set of communities in both, let those be $\mathcal{K}_1$ and $\mathcal{K}_2$. Then find a relation $\mathcal{R}$ on $\mathcal{K}_1 \times \mathcal{K}_2$ in an "obvious way"; if a $C_1 \in \mathcal{K}_1$ is in no relation, then it is a *death.* If $C_1 \in \mathcal{K}_1$ is in relation with $C_1^2, \ldots, C_\ell^2 \in \mathcal{K}_2$ and $C_i^2 \subset C_1$ for $i \in [1, \ldots, \ell]$, then C_1 *splits into* $C_1^2, \ldots, C_\ell^2$. Similarly, if $C_2 \in \mathcal{K}_2$ is in no relation, then it is a *birth.* If $C_2 \in \mathcal{K}_2$ is in relation with $C_1^1, \ldots, C_\ell^1 \in \mathcal{K}_1$ and $C_i^1 \subset C_2$ for $i \in [1, \ldots, \ell]$, then $C_1^1, \ldots, C_\ell^1$ *merged into* C_2. Finally a related pair (C_1, C_2), $C_i \in \mathcal{K}_i$ for $i \in \{1, 2\}$, should mean a *growth (contraction)* if $C_1 \subset C_2$ $(C_1 \supset C_2)$.

Computing the relation $\mathcal{R}$, assuming that the basic events cover all possible cases, is not hard theoretically. In practice this is different, since for large graphs a naive approach of considering all pairs $(C_1, C_2) \in \mathcal{K}_1 \times \mathcal{K}_2$ is too costly. The way out of this quadratic complexity is an idea due to I. Derényi [99]. They assume that the communities are always given by the clique percolation method (CPM) [1, 99, 100], which is *monotonic.* More exactly, if H and G are graphs on the same vertex set, $E(H) \subset E(G)$ and $C \in \mathcal{K}_H$ then there exists a $C' \in \mathcal{K}_U$, such that $C \subset C'$, where $\mathcal{K}_H$ and $\mathcal{K}_G$ are the sets of communities of H and G, respectively.

Then the *union graph* U is formed such that $V(U) := V(G_1) \cup V(G_2)$, $E(U) := E(G_1) \cup E(G_2)$ and $\mathcal{K}_U$ is computed. Now instead of going through all (C_1, C_2) pairs from $\mathcal{K}_1 \times \mathcal{K}_2$, one needs to check only those pairs for which there is a $C' \in \mathcal{K}_U$, such that $(C_1, C_2) \subset C'$.

By applying this method, they get a convincing results on the dynamics of communities. The most significant results are: the larger a community the older it is. The expected lifetime of a community increases with its size. A small community is more stable if it does not change its members, while it is the opposite for large communities. Let us note that the methodology is crucial, for the definitions, time scale and database the reader should consult the paper [99].

3.1.1 Problems

We have conducted a similar research on two large social networks described in the results chapter, meaning we analyzed the changes in the community structure of the corresponding networks. We tried to adapt the methodology of [99] with little success. To find communities, we used CFinder[2] for the CPM, resulting in the computational issues described in subsection 3.1.1. Then we decided on using the implementation of the N^{++} method[3] developed in [17], which was finally able to handle our networks. We have also found, that the framework they have used is insufficient to describe the community dynamics in our database. One of the reasons for this could be the difference between our database and the one they have used, or the difference between the community detection algorithms. We will discuss this below.

Basic events

In our experience, the description of basic events is not complete. The deviations from the description have at least three main causes.

1. The time scale is too large, and G_1 differs significantly from G_2. This is unavoidable, since in several cases the measurement intervals are given.
2. A community might become extinct not by splitting, but by leaving behind a number of overlapping communities on the same vertex set[4]; we will discuss this in depth in subsection 3.2.
3. In dense real graphs, it happens frequently that a set of overlapping communities change into another set of communities, and the relations cannot be easily mapped.

Computational issues

For large dense graphs the CPM is time inefficient. Deciding the proper value of the parameter k is also problematic. The other problem is, that for some benchmark graphs, e.g. the Zachary, the CPM performs poorly. It is natural to try out other community detection methods, since those might give better predictions in applications [17, 34, 92].

Implementation

The method relies on the monotonicity of CPM. In general, communities do not behave this way, they might split when adding edges to a graph, or merge when deleting edges. This phenomenon makes it harder to map the communities $\mathcal{K}_1$ and $\mathcal{K}_2$.

[2]The software was downloaded from the page http://angel.elte.hu/cfinder/

[3]We would like to express our thanks for obtaining free access to the appropriate softwares.

[4]Let A, B, C and D be cliques. Add a few edges between among those, such that $A \cap B$ and $C \cup D$ are the two resulting communities. Deleting two and adding two edges, $A \cap C$ and $B \cap D$ will be the new communities, which does not fit in the scheme.

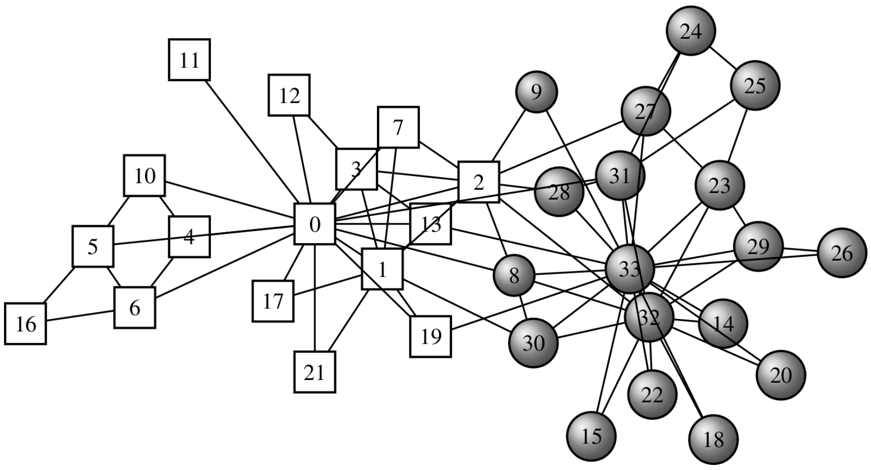

Figure 3.1: Zachary's karate club network. The boxes and naughts indicate the vertices corresponding to the split that has appeared during the original experiment.

3.2 Solutions

To motivate our solution, we analyze a small problem in depth. It shows that the introduction of new classes for community events are unavoidable.

The most obvious way to deal with problem 1 mentioned in subsection 3.1.1 is an artificial refinement of the time scale. That is, we fix the list of the changes that happened between G_i and G_{i+1}, and insert new graphs into the series, such that each new graph differs from the previous one in only one item. However, we will see that changing this order changes both the number and types of appearing community events, which implies, that the artificial refinement of the time scale should be avoided in practice. More importantly, this example will show us, in correspondence with problem 2, that the seven basic events defined above are inadequate when dealing with differences larger than one.

We illustrate the above mentioned problems with Zachary's karate club network [114]. Five edges were removed from the network one by one, in two different sequences[5]. The N^{++} algorithm was used for the purpose of detecting communities; the important parts of the outputs are summarized in Tables 4.1 and 3.2.

In the first sequence, edges a, b, c, d and e are removed in this order. After removing the first two edges, nothing changes. In the next step however, community 10 containing the nodes $\{0, 1, 2, 3, 7, 13, 19\}$ splits into communities 10_a : $\{0, 1, 3, 7\}$ and 10_b : $\{0, 1, 2, 13, 19\}$. In the next step, 10_b splits further into 10_{ba} : $\{1, 2, 19\}$ and 10_{bb} : $\{0, 1, 2, 13\}$. After removing the fifth edge from the network, several communities merge together, namely $7 : \{0, 1, 17, 21\}$, $9 : \{0, 2, 8, 32\}$, 10_{ba} : $\{1, 2, 19\}$ and 10_{bb} : $\{0, 1, 2, 13\}$. Out of these a new community, 13, is born: $\{0, 1, 2, 8, 13, 17, 19, 21, 32\}$.

Thus far, the basic community events defined in [99] were sufficient for the description. What happens however, if we compare the communities from the original graph with the communities of the final graph? Ignoring those communities that did not change, we have number $7, 9$ and 10 in the original graph, and 10_a and 13 in the modified graph. It would seem that community 10 is involved in a contraction event resulting in 10_a. Communities 7 and 9 merge into the new community 13, but

[5]The edges $a = (0, 13)$, $b = (0, 19)$, $c = (2, 3)$, $d = (2, 7)$ and $e = (3, 19)$.

Original	a	b	c	d	Final
7	7	7	7	7	**13**
9	9	9	9	9	**13**
10	10	10	10_a	10_a	10_a
			10_b	10_{ba}	**13**
				10_{bb}	**13**

Table 3.1: The changes in the community structure with the first edge removal sequence. Communities 1 through 6, 8, 11 and 12 do not change, and are omitted.

Original	c	a	e	d	Final
7	7	7	7	7	**13**
9	9	9	9	9	**13**
10	10	10	10_a	10_c	**13**
			10_b	10_d	10_e

Table 3.2: Changes of the communities in the second sequence. Communities 1 through 6, 8, 11 and 12 do not change, and are omitted.

13 also acquires some nodes from community 10, meaning it is also a growth event. This gets further complicated considering the results of the one by one edge removal: It seems that the first event, that took place was a split event. One way to solve this problem is to extend the number of community events used by the algorithm with more complex events that involve multiple basic ones. Before deciding which combinations should be used, let us examine the second edge removal sequence.

In the second sequence, edges c, a, e, d and b are removed in this order. Despite the different sequence, the first few steps are the same until community 10 splits apart. The resulting communities are different however, with 10_a being $\{1, 2, 13\}$ and $10_b : \{0, 1, 2, 3, 7, 19\}$. In the next step, 10_a is involved in a growth event, stealing a node from 10_b, which is in a contraction event, resulting in $10_c : \{1, 2, 13, 19\}$ and $10_d : \{0, 1, 2, 3, 7\}$. It is obvious that the two events are connected, but the original framework does not allow such complex situations.

In the final step, community 10_d loses another node resulting in $10_e : \{0, 1, 3, 7\}$ which corresponds to 10_a from the previous example. A merge event occurs at the same time, joining communities number $7, 9$ and 10_c. It is important to note, that 10_c is not identical to any community in the previous example, not even 10_b.

This example has two important consequences. First, if one tries to refine the time scale by creating an artificial series of graphs by adding or removing edges one by one, the result depends on the order of edges, rendering this approach meaningless. From now on, we will not use this naive idea for solving the problem at hand.

The second consequence is closely tied to the first one. It appears from the example above, that the basic events used to describe the changes of communities are satisfactory only if we change the graph one edge at a time. In all the other cases, more complex events, combinations of the basic events, are required.

Extending the number of community events

Using all possible combinations of the basic events is not necessary, as it would over-complicate the results of the algorithm. The first four events listed below are straightforward, each one is a combination of two basic events. The obscure event represents combinations of possibly more than two events, including merge-split, merge-split-merge, grow-split-merge, and so on. This means, that this event could be divided further into other events, but using the test data available we have found, that these five additional events are sufficient for describing community dynamics. With them, the changes in community structure can be identified reasonably well without unnecessary complications of the algorithm.

Tables 3.3 and 3.4 list the number of different community events found. A more detailed analysis of the results will be given in the results section. In the first one, the number of newly introduced merge and split variants are more numerous than their original counterparts. The number of obscure cases is somewhat higher, but still has the same magnitude. In table 3.4 the newly introduced events are far less numerous, but still relevant. This justifies the introduction of these events, and also implies, that further dividing the obscure cases would result in categories containing a very few number of events.

- Grow-merge, several communities join together, and also absorb some additional members.
- Contraction-merge, several communities join together, but loose some members in the process.
- Grow-split, a community splits into several communities, but these communities absorb additional members.
- Contraction-split, a community splits into several communities, and these communities also loose members.
- Obscure case. Multiple communities are involved from both $\mathcal{K}_1$ and $\mathcal{K}_2$, with their members reordering.

With respect to the above framework, we redefine the concepts of the split and merge events. Given the community sets $\mathcal{K}_1$ and $\mathcal{K}_2$, and a community $C_1 \in \mathcal{K}_1$ which is in relation with communities $C_1^2, \dots, C_\ell^2 \in \mathcal{K}_2$, we define the split event if $|C_1| = |C_1^2 \cup C_2^2 \cup \cdots \cup C_\ell^2|$. If $|C_1| < |C_1^2 \cup C_2^2 \cup \cdots \cup C_\ell^2|$, then we define the grow-split event, and finally if $|C_1| > |C_1^2 \cup C_2^2 \cup \cdots \cup C_\ell^2|$, we define the contraction-split event.

The definition of the merge event is symmetrical: given the community sets $\mathcal{K}_1$ and $\mathcal{K}_2$, and communities $C_1^1, \dots, C_\ell^1 \in \mathcal{K}_1$ which are in relation with a community $C_2 \in \mathcal{K}_2$, if $|C_1^1 \cup C_2^1 \cup \cdots \cup C_\ell^1| = |C_2|$, then we define the merge event. If $|C_1^1 \cup C_2^1 \cup \cdots \cup C_\ell^1| < |C_2|$, then we define the grow-merge event, and finally if $|C_1^1 \cup C_2^1 \cup \cdots \cup C_\ell^1| > |C_2|$, we define the contraction-merge event.

The obscure case is less strictly defined. If there are multiple communities from $\mathcal{K}_1$ and $\mathcal{K}_2$ connected by a relation, without further analysis, we classify this relation as an obscure event.

3.3 Details of the algorithm

The method described in this section follows the idea of I. Derényi [99], with a few important modifications. We will adopt the concept of the union graph U, and use it to solve the originally quadratic problem described in section 3.1 in almost the same way as in [99]. Our additional work centers around two important modifications of the original method.

The first one is abandoning the requirement of monotonicity. This means, that we can no longer assign only one community from $\mathcal{K}_U$ to any community from the original graphs.

The second modification incorporates the detection of the extended community events. The solution to this is fairly straightforward.

We will also describe an optional modification of the original idea of [99]. Instead of using U, we will use I, the *intersection* graph of G_1 and G_2. That is $V(I) := V(G_1) \cap V(G_2)$, $E(I) := E(G_1) \cap E(G_2)$, and $\mathcal{K}_I$ represents the communities of the intersection graph I[6].

3.3.1 Overview

The input of the algorithm consists of three community sets $\mathcal{K}_1, \mathcal{K}_2$ and $\mathcal{K}_U$, where $\mathcal{K}_U$ is the community set of U.

The output of the algorithm is a relation $\mathcal{R}$ on $\mathcal{K}_1 \times \mathcal{K}_2$, corresponding to community events described in [99] and subsection 3.2.

The algorithm can be divided into two phases. The first phase creates a relation $\mathcal{R}_1$ on $\mathcal{K}_1 \times \mathcal{K}_U$ and $\mathcal{R}_2$ on $\mathcal{K}_2 \times \mathcal{K}_U$. The second phase combines $\mathcal{R}_1$ and $\mathcal{R}_2$ into $\mathcal{R}$. Because of the non-monotonic nature of the community detection algorithm, a preprocessing step is required before executing the second phase.

First phase

In the first phase we search for relations among $C_i^1 \in \mathcal{K}_1$ for all i and $C_\ell^u \in \mathcal{K}_U$ for all ℓ. We are looking for two types of relations. The first type is the *exact match:* $C_i^1 = C_\ell^u$. The second is a *contain match:* $C_i^1 \subset C_\ell^u$. Because of the non-monotonic nature of the community detection algorithm $C_\ell^u \subset C_i^1$ might also occur. This case also counts as a contain match.

In order to find these relations, we iterate over the elements of $\mathcal{K}_1$ and compare each C_i^1 to every element of $\mathcal{K}_U$. If we find a contain match, we create a relation $r_1(C_i^1, C_\ell^u)$. If we find an exact match, we also create $r_1(C_i^1, C_\ell^u)$, but we ignore C_ℓ^u in the subsequent computations. This step is repeated for $\mathcal{K}_2$ and $\mathcal{K}_U$.

Intersection approach

Since a community detection algorithm is not necessarily monotonic, there might be elements of $\mathcal{K}_1$ or $\mathcal{K}_2$ that are not in relation with any element of $\mathcal{K}_U$, these were counted as "deaths" before. The intersection approach tries to solve this problem by replacing $\mathcal{K}_U$ with $\mathcal{K}_I$. The only difference in the implementation is, that in the first phase $\mathcal{K}_1$ (and $\mathcal{K}_2$ later) is replaced with $\mathcal{K}_I$, and $\mathcal{K}_U$ is replaced with $\mathcal{K}_1$ (and

[6]The community sets can be created by any, possibly non-monotonic, community detection algorithm.

$\mathcal{K}_2$ later). After the first phase has finished, the algorithm continues after inverting the relations r_1 and r_2.

The intersection method provides almost the same results as the union approach with a few exceptions, that will be noted in the results chapter. The size of the intersection graph I is smaller than or equal (in the special case when $G_1 = G_2$) to the size of the union graph U. From the computational point of view, this implies that the running time of the community detection algorithm should be lower on I. Other than this, the use of the intersection approach is completely optional.

Preprocessing

As we noted before, there may be more than one element of $\mathcal{K}_U$, that is in relation r_1 with a given C_i^1. The same holds for C_j^2 and the relation r_2. To solve this, we put a new, fictitious community C_a^u to $\mathcal{K}_u$, set $r_1(C_i^1, C_a^u)$, and delete all former relations containing C_i^1. The same is done for C_j^2. If C_i^1 and C_j^2 were in relation with the same elements of $\mathcal{K}_u$, then the same C_a^u is used.

Second phase

In this phase we run through the elements of $\mathcal{K}_u$. For each element $C_\ell^u \in \mathcal{K}_U$, let the elements $\mathcal{C}^1 := \{C_1^1, \dots, C_i^1\} \subset \mathcal{K}_1$ and $\mathcal{C}^2 := \{C_1^2, \dots, C_j^2\} \subset \mathcal{K}_2$ be in relation with C_ℓ^u according to r_1 and r_2 respectively. Let $\cup\mathcal{H}$ be the $\cup_{H\in\mathcal{H}} H$ for any set $\mathcal{H}$.

Depending on i and j, and the sizes of the communities involved, we create the relation $r(C_{i'}^1, C_{j'}^2)$ for every i' and j', and we assign community events to these relations.

- If $\mathcal{C}^1 = \emptyset$ and $|\mathcal{C}^2| > 0$, then $r(\emptyset, C_{j'}^2)$ is a *birth* event for every j'.
- If $|\mathcal{C}^1| > 0$ and $\mathcal{C}^2 = \emptyset$, then $r(C_{i'}^1, \emptyset)$ is a *death* event for every i'.
- If $|\mathcal{C}^1| = 1$ and $|\mathcal{C}^2| = 1$, that is $\mathcal{C}^1 = \{C_1^1\}$ and $\mathcal{C}^1 = \{C_1^2\}$, then
  - If $|C_1^1| = |C_1^2|$, $r(C_1^1, C_1^2)$ is an *exact match.*
  - If $|C_1^1| > |C_1^2|$, $r(C_1^1, C_1^2)$ is a *contraction* event.
  - If $|C_1^1| < |C_1^2|$, $r(C_1^1, C_1^2)$ is a *growth* event.
- If $|\mathcal{C}^1| = 1$ and $|\mathcal{C}^2| > 1$, then
  - If $|C_i^1| = |\cup\mathcal{C}^2|$, $r(C_i^1, C_{j'}^2)$ is a *split* event for every j'.
  - If $|C_i^1| < |\cup\mathcal{C}^2|$, $r(C_i^1, C_{j'}^2)$ is a *grow-split* event for every j'.
  - If $|C_i^1| > |\cup\mathcal{C}^2|$, $r(C_i^1, C_{j'}^2)$ is a *contraction-split* event for every j'.
- If $|\mathcal{C}^1| > 1$ and $|\mathcal{C}^2| = 1$, then
  - If $|\cup\mathcal{C}^1| = |C_j^2|$, $r(C_{i'}^1, C_j^2)$ is a *merge* event for every i'.
  - If $|\cup\mathcal{C}^1| < |C_j^2|$, $r(C_{i'}^1, C_j^2)$ is a *grow-merge* event for every i'.
  - If $|\cup\mathcal{C}^1| > |C_j^2|$, $r(C_{i'}^1, C_j^2)$ is a *contraction-merge* event for every i'.
- If $|\mathcal{C}^1| > 1$ and $|\mathcal{C}^2| > 1$ then we introduce an obscure event $r(C_{i'}^1, C_{j'}^2)$ for every i' and j'.

3.3.2 Time complexity

The time complexity of the algorithm will be addressed both from the theoretical and empirical point of view. In the first phase, we compare each community from $\mathcal{K}_1$ to communities from $\mathcal{K}_U$. In the worst case, no exact matches are found resulting in an $O(n * m)$ complexity, where n is the size of $|\mathcal{K}_1|$ and m is the size of $|\mathcal{K}_U|$. An exact match always reduces the number of further computations. Therefore when we find an exact match $r_1(C^1, C^u)$ $(r_2(C^2, C^u))$, we can ignore all other relations involving C^1, C^2 and C^u.

The obscure events are catchier. Note, that the sizes of $\mathcal{K}$'s might be exponential in $|V(G_1)|$, while a complex event could involve any number of communities on both sides, which would result in doubly-exponential running time. Fortunately, usually we have less communities than vertices, and the number of obscure events are small, consisting of only a few sets. So in practice the obscure events have only a negligible effect on the (actual) running time.

In the second phase, if each element from $\mathcal{K}_1$ and $\mathcal{K}_2$ were connected with every element from $\mathcal{K}_U$, we would obtain a worst case complexity of $O(n*m*k)$, $k = |\mathcal{K}_2|$. In practice, a community from $\mathcal{K}_U$ corresponds to only a few communities from $\mathcal{K}_1$ and $\mathcal{K}_2$. This reduces the actual running time of this phase to $O(c * m)$, where c is a small constant.

3.4 Results

To test our algorithm, besides the small examples like Zachary's graph, we have used two large test databases. One came from an international bank [34], while the other one is a large social network. The bank graph is based on a transaction database, and consists of roughly 80000 nodes and 270000 edges, and we were provided with three time instances taken in a six month period. The edges of the social graph were also determined as the output of a data mining project, and the graph contains roughly one million nodes and 1.5 million edges. Here four time instances were taken in a four month period. Due to the size and structure of these networks, the CFinder application fails to provide communities, so the communities were listed by the N^{++} method only.

3.4.1 Observations

Table 3.3 shows the results for the bank dataset for different time intervals. Notice, that the number of death and birth events are very high, about 40% of the communities die. A possible explanations for this is, that the time lapse between the instances is relatively long (three months), and in the last case, where the first and last instances are compared, this is extended to six months. During this long time period, the community structure of a network changes significantly.

Another explanation might be based on a problem referred in 3.1.1. If a dense community changes significantly, it is hard to know what had really happened to it, while the algorithm classifies it as a death event.

The number of unchanged, growing and contracting communities has the same magnitude, which dominates the splitting and merging events.[7]

[7]Since the first types of events involve the exact match events described in 3.3.1, this explains

Events	12u	23u	13u
Deaths	6586	6481	9901
Births	6729	6477	9971
Unchanged	4888	5062	2529
Growths	2185	2084	1629
Contractions	1985	2123	1569
Splits	83	91	45
Grow-splits	154	133	172
Cont-splits	144	149	111
Merges	200	184	117
Grow-merges	356	315	256
Cont-merges	350	326	418
Obscure	531	595	594

Table 3.3: Results on the bank database. The columns show the community events in the following order: first and the second time instances, the second and third time instances, finally the first and the third instances.

It indicates a certain dynamic equilibrium, that the number of growth and contract events, and the death and birth events are balanced. The number of pure split/merge events are less than the newly introduced grow/contraction split/merge events. This justifies the introduction of these events, and underlines the complications in community dynamics.

The running time of our community matching algorithm was 8 seconds, while the search for communities took 5 minutes. The used computer configuration was a machine of two cores, with each processor running at 2.0 GHz and supplied with 2 gigabytes of memory.

Table 3.4 shows the matching results on the social network. In contrast to our previous results, the number of deaths and births are significantly lower, and the number of unchanged communities dominates all other events. The lapse between the time instances is short (one month), but in the last case, the first and last instances are compared resulting in a four months interval. Even in this case, the number of unchanged communities is much larger than the number of deaths, so one concludes that the social graph is more stable than the graph of the bank dataset. It is important to note, that the number of birth events are significantly lower than the number of death events, and the number of split-like events do not balance this by generating more communities. This indicates that the number of communities of the network is decreasing. Indeed, the network has lost around 12.5 percent of its communities in the observed period.

As in the previous case, the number of growth and death events are balanced. Here the number of pure split and merge events outnumber the newly introduced grow/contraction and split/merge events. This also points to the fact, that this network is more stable than the previous.

the low running time.

Events	12u	23u	34u	14u
Deaths	6847	7812	7931	23519
Births	6112	4119	3782	14934
Unchanged	59985	59247	56031	38272
Growths	2999	2161	1963	4343
Contractions	3468	3629	3458	6753
Splits	457	454	383	538
Grow-splits	83	55	37	32
Cont-splits	115	83	102	266
Merges	919	820	785	1030
Grow-merges	232	180	178	438
Cont-merges	142	137	93	79
Obscure	90	92	59	66

Table 3.4: Results for the social network database. The columns show the community events in the following order: first and the second time instances, the second and third time instances, the third and fourth time instances, and finally the first and the fourth instances.

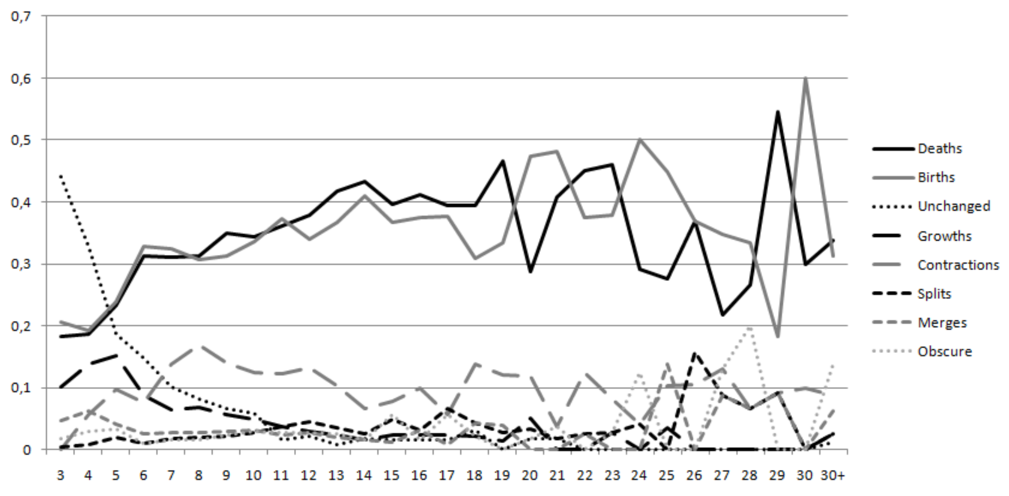

Figure 3.2: The percentage of community events compared to the community sizes for the bank dataset. The results were generated with the union graph approach.

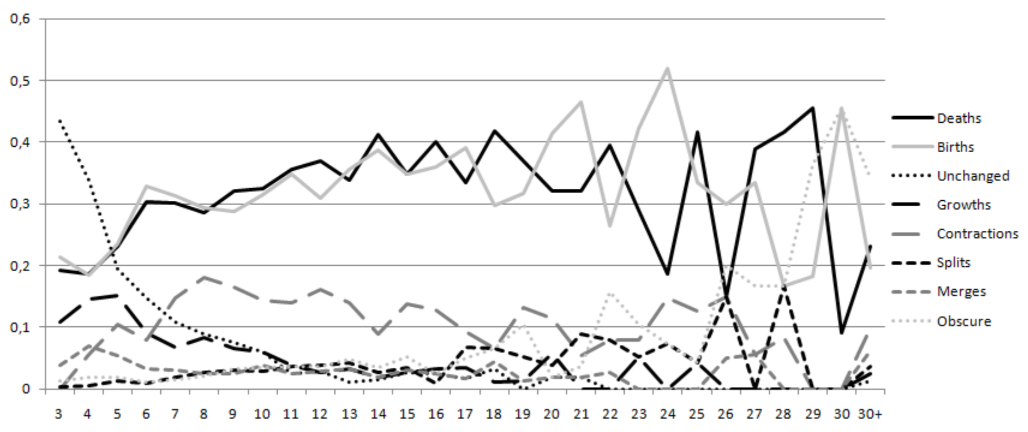

Figure 3.3: The percentage of community events compared to the community sizes for the bank dataset. The results were generated with the intersection graph approach.

Even though this network is larger than the first, the running time of the matching algorithm is about the same due to the very high number of unchanged communities. Note that the N^{++} method needed about 60 seconds for finding the communities, our community matching algorithm took 14 seconds. We have used the same machine as before.

3.4.2 Community evolution

Following the footsteps of Palla et al. [99], we compared the sizes and lifetimes of communities. Figure 3.2 displays our findings on the bank dataset. As we have concluded in the previous section, the community structure of this dataset changes very rapidly. For almost all sizes of communities the number of deaths outweigh all other community events. The only exceptions are very small communities of sizes three and four. It should also be noted, that there is a small spike of the obscure cases for the large communities. Aside from these, it can be said, that the size of a community does not relate to the community event it is involved in. Most importantly, the findings do not confirm the results of [99] that the expected lifetime of a community is a monotone function of its size. For a decisive result, further studies are needed.

If we take the intersection graph approach displayed on Figure 3.3, the results are almost the same, except for the largest communities. These cases are classified as obscure cases in contrast to declaring them dead as before.

The social network shows very different behavior. Figure 3.4 shows results for a one month time lapse. The percentage of death events is very small and constant compared to the community sizes, which is in contrast to the previous dataset. The number of unchanged communities is very high for small sizes, and decreases monotonically. This behavior was also reported in [99].

For large communities the contraction and split events dominate. The number of these events increases monotonically. The number of growth and merge events is small and independent of the sizes, which is somewhat surprising considering their symmetrical counterparts. The number of all other events is small and independent of the community sizes. These results also indicate a more stable nature for this

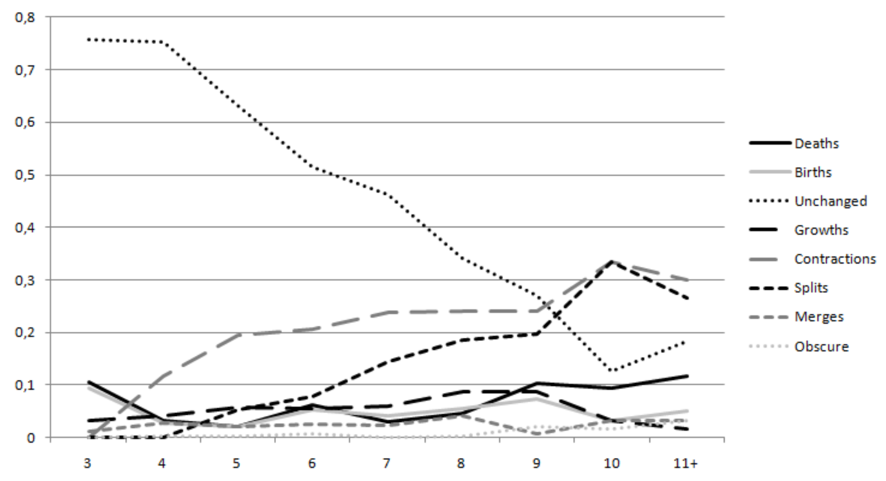

Figure 3.4: The percentage of community events compared to the community sizes for the social graph dataset with one month difference. The results were generated with the union graph approach.

dataset.

Figure 3.5 shows the results for the four months interval. It is clear that in a four month interval, the community structure of a network can change dramatically, yet the difference between the results for the bank dataset and this one is clear. The number of deaths is significantly lower, but still relevant, and except for the smallest communities, independent of size. The number of birth events is lower than the number of death events, this matches with our observations in the previous chapter. The number of unchanged events is high for small communities, and it decreases somewhat faster, than in the previous network. The number of split events is increasing much faster than before, and contraction events are dominant even for medium sized communities. It should also be noted that small communities aside, the number of contraction events is independent from the community sizes. The number of all the other cases is small and constant.

The findings above reveal a strange behavior: while this network is more stable than the banking network, in some sense it is steadily loosing communities. The banking network on the other hand changes more dynamically, but it is in a state of equilibrium.

For the social network, the intersection method provides almost the same results, with the percentage of death events being slightly lower.

3.5 Conclusions

Based on the earlier results of [6, 99] in this chapter we have given a new algorithm and methodology for following the life cycle of communities in dynamic graphs. The methodology successfully extends the incomplete notions used in [99], and results in an algorithm that works based on general community detection methods. The algorithm is very fast, it solves large community matching problems in seconds.

We have worked with two large datasets with fixed observation periods. Our findings indicate, that there is a significant difference between the behavior of the community structures and dynamics of these datasets. One of the networks is more

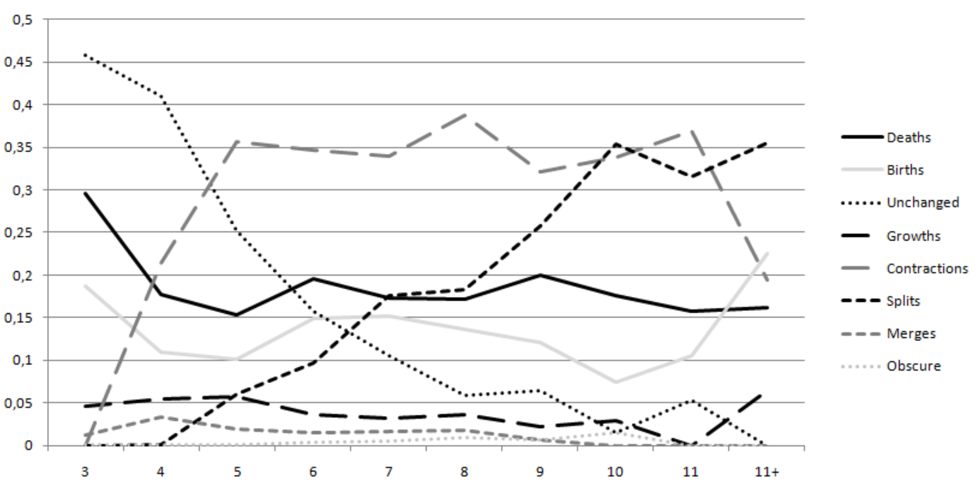

Figure 3.5: The percentage of community events compared to the community sizes for the social graph dataset with four months difference. The results were generated with the union graph approach.

stable, but loses communities steadily, the other network is more dynamical, but maintains its community number. We have also examined the changes in community structure in relation with the sizes of the involved communities. Our findings confirm some of the claims of [99], however not all of them.

Chapter 4

Inverse infection

In the final chapter of this dissertation the authors' work on infection processes is reviewed. This is the chapter, that contains the largest contribution to the authors' scientific work. If we recall our discussion in the end of chapter 1, this field contains many interesting problems: infection models, influence maximization, the estimation of vertex and edge infection values. This chapter touches most of the above topics (with the exception of influence maximization), but the long term goal, that guided the development of several models and algorithms described here, was the prediction of edge infection probabilities: the creation of the Inverse Infection Problem (IIP).

The idea of the Inverse Infection Problem comes from the previous work of Csernenszky et al. [34] published in 2009. This work focused on the application and possible use of infection models on bank transaction networks. A conclusion of this work was, that the precision of current methods for the prediction of credit default can be improved by taking into account the effect members of the network have on each other. This is the so-called network effect: the probability of a node influencing each of its neighbors. This led to idea, that these influence values should be estimated somehow, and over time, to the formulation of the Inverse Infection Problem.

Our approach to the prediction of the influence (or infection) probabilities between nodes is different from the ones existing in literature, although the field is scarcely studied[1]. Compared to these methods, ours does not require information on the individual steps of the infection process. Instead it builds on the experiences gained from the previously mentioned work on bank transaction network, such as estimations of the probabilities of default for individual companies and additional information characterizing the connections between them.

The development of the Inverse Infection Problem required the creation of additional methods and algorithms, and we will begin our discussion of the topic with a generalized infection framework, the Generalized Cascade Model [21]. Based on the Independent Cascade Model, this generalized model uses a probabilistic framework to describe the inputs and outputs of an infection process. Its computation is difficult, so four heuristics methods were also proposed for the sake of efficiency [19]. Then we will describe the Inverse Infection Problem itself. The formulation of the IIP paves the way to the concrete optimization algorithms resulting in the desired infection values [22]. The most successful of these was the Fully Informed Particle Swarm Optimization method of Kennedy and Mendes [67]. We will evaluate the

[1]A description of most of these methods can be found in the end of chapter 1

performance of our model in two different ways. We will use several artificial infection scenarios with different initial infections and infection values to demonstrate the boundaries of our solution. Finally we will return to the source of the idea of the Inverse Infection Problem. We will present a detailed case-study of our model on bank transaction networks.

4.1 Infection Models

The Inverse Infection Problem uses a generalized infection framework, called the Generalized Cascade (GC) model as the infection model. In this section we are going to give a quick refresher on some basic notions already discussed in the end of chapter 1 before we introduce the GC model. The computation complexity of this model is similar to the one in [26], therefore we will propose four heuristics to speed up the process, and evaluate the performance of them in different scenarios.

Infection models assign states to the nodes of the network corresponding to the different phases of infection. The states of the Independent Cascade model and the SIR infection method are the same. A node is susceptible if it is not yet infected, a node is infected, if it is infected and may infect other, and a node is removed, if it is infected but not infectious. Sometimes we will call these states as inactive, newly activated and active.

Any infection model can be described as a process, that has two inputs: the first one is a weighted graph, where the edge weights are real values between 0 and 1: $\forall e \in E(G), 0 \leq w_{u,v} \leq 1$. The second input is the set of initial infectors $A_0 \subset V(G)$. These nodes are considered as infected at the beginning of the process. The infection process happens in discrete time steps. The process terminates at iteration t, and results in the set of infected nodes $A = \bigcup_{i=0}^{t} A_i$. The original formulation of the Independent Cascade (IC) and Linear Treshold (LT) models were for directed graphs, but it is easy to generalize them for undirected ones by assuming, that the edge infection values are symmetrical $w_{u,v} = w_{v,u}$.

The specific way one vertex infects another varies depending on the model. In the case of the IC model [38], let $A_i \subseteq V(G)$ be the set of nodes newly activated in iteration i. In the next iteration $i+1$, each node $u \in A_i$ tries to activate its inactive neighbors $v \in V \setminus \cup_{0 \leq j \leq i} A_j$ according to the edge infection probability $w_{u,v}$, and v becomes active in iteration $i+1$, if the attempt is successful. If more than one node is trying to activate v in the same iteration, the attempts are made independently of each other in an arbitrary order within iteration $i+1$. If $A_t = \emptyset$, the process terminates in iteration t. It is easy to see, that the process always terminates in a finite number of iterations.

4.1.1 Generalized Cascade Model

We can generalize the IC model in the following way [21]. In the Generalized Cascade (GC) model [19] each vertex is assigned a real value p_v between zero and one, that represents the probability of infection before the beginning of the process. We refer to these values as the *a priori distribution.* Vertices are infected independently from each other before the beginning of the process according to their a priori infection probability p_v. This model is capable of summarizing the effect of the a priori infections and the effect of these infections transmitted through the network.

Similarly to the input, the output of the model is given as an *a posteriori distribution*, where values p'_v indicate the probability of being infected during the process for all $v \in V$. The actual way a vertex infects another is the same as in the IC model, although it is possible to use other infection models with the terms of the GC model.

In some applications, an estimate of one or both of the above described probability distributions is available. For example, in the case of the banking application [16, 34] an accurate estimation of the probability of default for each company was given by standard models used by the bank[2]. Another application in telecommunications has estimations for the probability of churn with similar methods. If such estimations are not available we can resort to a crude but effective method. Suppose we can observe the beginning and the end of the infection process k times. By counting the frequencies of infection, for all vertices v, how many times did v belong to A_0 or A we can construct the respective probability distributions. The accuracy of the estimation obviously depends on k, but k does not have to be a large number. We will show in section 4.4.4, that 6-8 observations are enough to produce outputs with acceptable quality.

Based on these remarks and formulations, we can define the Generalized Cascade model [19]:

The Generalized Cascade Model: *Given an appropriately weighted graph G and the a priori infection distribution p_v, the model computes the a posteriori distribution p'_v for all $v \in V(G)$.*

The infection process itself is the IC model, although other models might also be used for different applications. We have chosen the Independent Cascade model as the basis of our method, because it performs well in modeling infection-like processes in business applications [34]. Alternatively, this model can be considered as a general framework of infection.

Unfortunately, the computation of the a posteriori distribution in the IC model is #P-hard. There are several existing heuristics to provide estimations of p'_v [26, 28, 29, 30, 70], and in the next section we are going to introduce four additional ones specifically tailored to the requirements of the GC model.

4.1.2 Heuristics and approximations of the GC model

To provide a way to efficiently compute the GC model, four heuristic methods were proposed in [19]. In this section we will give a detailed description of them. These methods are:

- Complete Simulation is the direct adaptation of the method in [65] for the GC model.
- The Edge Simulation method is a combination of both simulation and exact computations that decreases the standard deviation appearing in other simulations.
- If the infection probabilities are small, then the infections typically do not travel far from the source of infection. Neighborhood Bound Heuristics exploits this property.

[2]The BASEL II default probabilities were computed using vertex attributes.

- The Independent Cascade model itself can be substituted for a similar, but a computationally more tractable model.

The methods presented below fall into three categories. *Simulations* are Monte Carlo generators, i.e. they compute multiple realizations of the probabilistic infection process, and count the relative frequency of vertex infections.

In contrast to this, *heuristics* use an approximation of the GC model, circumventing the #P-hardness, but still dealing with both the a priori and the edge infection probabilities. Finally, *hybrid methods* use some combination of both approaches.

Frequency based simulations

The notion of using Monte Carlo based simulations to compute the a posteriori infection distribution comes from Kempe et al [65]. That is, generate random edges and vertex infections, according to the edge weights and the a priori vertex infections, and then approximate the infection probability of vertices by the relative frequency of vertex infections.

All methods described in this section use the above approach. The original method of Kempe et al. can be adapted to the requirements of the Generalized Cascade model. We will refer to this as *Complete Simulation.* In contrast to this, *Edge Simulation* is a hybrid design using heuristics to improve the performance of the simulation.

Kempe, Kleinberg and Tardos developed a method based on reachability [65] to compute the Independent Cascade model. They construct an unweighted directed graph G' on the same vertex set as G by drawing the edge (u, v) independently of the other edges with probability $w_{u,v}$. The resulting graph G' can be interpreted as a realization of the possible routes of infection. Those vertices that can be reached from an initially infected vertex also become infected. This way the computation of the iterations one by one can be avoided, and the problem of a single simulation step is reduced to a simple (path) searching problem. Note that this formulation helped to show that $f(A)$, the expected infection of a set A is a *submodular function*, that is $f(S \cup \{v\}) - f(S) \geq f(T \cup \{v\}) - f(T)$ for all $v \in V(G)$ and $S \subset T \subset V(G)$.

The process can be applied to G multiple times, resulting in a series of realizations. The desired distribution can be approximated simply by counting the relative frequency of vertex infections.

Complete Simulation

It is easy to adapt the method of Kempe et al. to the needs of the Generalized Cascade model. In addition to the construction of an undirected G', a set $A_0 \subseteq V(G)$ is created, according to the following rule: for all $v \in V(G)$, $v \in A_0$ according to the probability p_v. G' and A_0 is generated multiple times. We will refer to this value as the sample size k. Finally let f_v denote the relative frequency of infection for vertex v.

At line 10, we allow $u = v$. In the main loop, it is indifferent which node in A_0 is the source of the infection. Since a node can be infected only once, any node reachable from a node from A_0 can be left out of further computations. Because of this, the for loops in the algorithm can be interpreted as a single search on G' starting from the nodes in A_0.

4.1. Algorithm. Complete Simulation

Input: Graph G, sample size k

Output: Relative frequency of infection f_v for all $v \in V(G)$

```
1: $j \leftarrow 0$
2: for all $v \in V$: $f_v \leftarrow 0$
3: while $j < k$ do
4:     Generate $G'$
5:     Generate $A_0$
6:     $S \leftarrow \emptyset$
7:     $Q \leftarrow V(G')$
8:     for all $u \in A_0 \cap Q$ do
9:         for all $v \in Q$ do
10:            if there is a path $u, \dots, v$ in graph $G'$ then
11:                $S \leftarrow S \cup \{v\}$
12:            end if
13:        end for
14:        $Q \leftarrow Q \setminus S$
15:    end for
16:    for all $v \in S$: $f_v \leftarrow f_v + 1$
17:    $j \leftarrow j + 1$
18: end while
19: for all $v \in V$: $f_v \leftarrow \frac{f_v}{k}$
```

Before the main loop, f_v is initialized for all vertices. It is easy to see, that the algorithm computes the relative frequencies for all vertices correctly. Since this method is based on frequency counting, its precision and time complexity depends greatly on the sample size k. We will elaborate on this in Section 4.1.3.

Edge Simulation

There is a different way to simulate the generalized model: instead of computing A_0 we can deal directly with the a priori vertex infection probabilities. The notation is the same as before, p_v denotes the a priori infection probability of vertex v. At line 8, we allow $u = v$.

In contrast to the previous method, the source of the infection does matter, since the value p_v is not the same for different vertices. We also do not have any restrictions on the structure of G other than being simple, so any vertex can be a part of a loop. This means, that in order to avoid counting a single p_v multiple times, we have to do a search for each node independently. Obviously the above approach increases the time complexity in general, but if the edge probabilities are low enough in G, then G' has many small components, reducing the running time for each search significantly.

It is clear, that the computational time is greater for ES compared to CS. In exchange, we can expect, that the precision of the approximation is less dependent on the sample size. We will elaborate on this in Section 4.1.3.

4.2. Algorithm. Edge Simulation

Input: Graph G, sample size k

Output: Relative frequency of infection f_v for all $v \in V(G)$

```
 1: $j \leftarrow 0$
 2: for all $v \in V$: $f_v \leftarrow 0$
 3: while $j < k$ do
 4:     Generate $G'$
 5:     for all $v \in V(G)$ do
 6:         $s \leftarrow 1$
 7:         for all $u \in V(G)$ do
 8:             if there is a path $u, \dots, v$ in graph $G'$ then
 9:                 $s \leftarrow s(1 - p_u)$
10:             end if
11:         end for
12:         $f_v \leftarrow f_v + 1 - s$
13:     end for
14:     $j \leftarrow j + 1$
15: end while
16: for all $v \in V$: $f_v \leftarrow \frac{f_v}{k}$
```

Neighborhood Bound Heuristic

There are several existing heuristics for the IC model [26, 28, 29, 30, 70] covering a large area of performance requirements. These methods usually exploit one or more properties of the infection process. Chen's DAG and LDAG algorithm for example focuses on the concept, that the edges with high infection probabilities carry the bulk of the infection process. They propose to construct a local directed acyclic graph containing the relevant edges for each node, and then compute the infection process using the DAG.

The main idea of our method is similar: the construction of a small graph containing all of the possible paths of infection inside a given neighborhood for a given vertex. This graph is created in such a way, that the approximation of the a posteriori infection probability of the corresponding vertex can be easily computed. It is important to emphasize, that the goal of this method is the approximation of the GC model as fast as possible, disregarding requirements for precision.

We are also going to rely on the findings in [17,21,34]. Based on the examination of economic networks, the authors have found, that the edge infection probabilities are small, typically below 0.2. The above observation greatly limits "the travel distance" of an infection event, since even if we consider a path of length two from the source of the infection, the probability of infection is reduced to 0.04 or below.

Based on the remarks above, we propose the Neighborhood Bound Heuristic (NBH). We will define the set containing the in-neighbors of vertex v as $n^-(v)$ like we have done so in the beginning of the first chapter. For all $v \in V(G)$ we are going to construct a weighted, rooted tree T_v with v as the root, and edges pointing towards v. In the first step all vertices $u \in n^-(v)$ are added to T_v, as well as the edges $(u, v) \in E(G)$ for all $u \in n^-(v)$. In the second step we are going to deal with the second neighborhood of v. For all $u \in n^-(v)$ we are going to add the nodes $z \in n^-(u) \setminus \{v\}$ and all of the edges $(z, u) \in E(G)$ to T_v. The subtractions of v is necessary to avoid loops of length two. For all edges in T_v we keep the edge weights from G. Take note, that nothing prohibits the nodes of G (with the exception of v) from appearing multiple times in T_v, this corresponds with our idea of representing all possible infection paths in the second neighborhood of v.

The computation of the a posteriori infection probability of v in T_v is easy. T_v has three levels: the leaves, $n^-(v)$ and the root v. The a posteriori infection probabilities of the leaves are the same as their a priori ones, since they do not have in-neighbors. A node $u \in n^-(v)$ gets infected if one of the leaves connected to it is infected, or becomes infected by itself, meaning $p'_u = 1 - (1 - p_u) \prod_{z \in N^-(u)} (1 - p'_z)$. The computation is executed in the same fashion for v.

The above method is extremely fast, since p'_v can be computed in a single run on the edges of T_v, and if G is sparse, $|E(T_v)|$ is small. The construction of T_v is simple, if we limit the process to the second neighborhood of v in G. If we consider larger neighborhoods, nothing prohibits $|V(T_v)|$ from growing exponentially, and it becomes increasingly difficult to avoid loops. Finally, if we consider the fact, that the edge weights are small, then the loss of precision is still within acceptable bounds.

Aggregated Linear Effect Model

We have also built a model that more or less approximates the mechanism of the Generalized Cascade model, the *Aggregated Linear Effect Model*, shortly ALE model afterward. For a weighted graph G, let the a priori infection of a vertex v be p_v. If one considers only one step of the linear effects at v, it can be defined as $\sum_{u:u \in N(v)} w_{u,v} + p_v$. This is nothing else, but $x + Bx$, where B is the transpose of A, the weighted incidence matrix of G, and x is the vector of a priori infection probabilities.

In the 2nd, 3rd, ... ith steps we may aggregate the effects of the second, third etc. neighborhoods by adding the B^2x, B^3x, ..., B^ix correction terms to the approximation.

Let $B = A^T$, where A is the weighted incidence matrix of graph G, and x is the vector, such that $x_v = p_v$, then the a posteriori effect y is defined as

$$y := (I + B + B^2 + \dots)x,$$

where I is the identity matrix.

Note, that $\sum_{u:u \in N(v)} w_{u,v} + p_v$ is close to the probability of v getting infected independently in one step by itself or by a neighbor. In general, if the weights are small, the error is negligible. Assuming small weights, the infinite series also converges, and we have the more compact form of

$$y := (I + B + B^2 + \dots)x = (I - A)^{-1}x.$$

The values p_v and $w_{u,v}$ do not have to be probabilities any more, we might consider any appropriate scalar functions (perhaps after scaling in order to maintain convergence). Now the following questions arise: How good is the ALE model? How to compute (and later on use) it efficiently?

There are two obvious ways to test the first question. One is to consider some real problems, make a network model out of those, and compute the optimal weights in an ALE model such that the model "prediction" are as close to the real values as possible. However, since the IC models already performed well in such case studies,

Setup	A	B	C	D	E
ℓ_1	0.02	0.05	0.1	0.2	0.5
ℓ_2	0.1	0.1	0.2	0.2	0.5

Table 4.1: Experiment setups.

we might consider here an easier way. We just take a weighted G with some a priori infection x, compute the a posteriori infection y by the associated IC model, and then try to find appropriate new weights such that for the a posteriori effect y' by the ALE model we get $\min ||y - y'||$ in a fixed norm.

Once we have the appropriate weights, that is the matrix B, we can compute y easily. Of course not by inverting $I - B$ but solving the equation $(I - A)y = x$ for y by Gaussian elimination.

4.1.3 Evaluation of the heuristics

For the purpose of evaluation we have used sparse graphs generated with the forest fire model [81], with $|G(V)| = n = 1000, \ldots, 40000$. This allows us to examine how much the computation time of a given algorithm scales with graph size. Since the performance of most methods described here depends on the size of the infection probabilities, we have used five arrangements of edge weights and a priori distributions.

- Each edge weight is drawn independently from an uniform distribution between $(0, \ell_1)$.
- The expected size of the set of random infectors is $n * \ell_2$. For each vertex, there is an a priori infection probability drawn independently from an uniform distribution between $(0, 0.2)$.
- For all other vertices $p_v = 0$.

Using the above process, for each of the unweighted graphs, we have created five weighted ones with a priori probabilities, resulting in $9 \times 5 = 45$ different benchmark networks. The parameters ℓ_1 and ℓ_2 of each arrangement can be seen in table 4.1.

In this section, we are going to evaluate the running time and precision of our methods[3]. While the evaluation of computational time is rather straightforward, we have to make a distinction while measuring performance.

Simulations are Monte Carlo based estimations of the infection process. The goodness of an estimation is governed by the sample size k. From the Law of Large Numbers it follows that if k goes to infinity then the simulated values converge to the infection probability. Depending on k, the output of the simulations differ from each other, characterized by their deviation. It is important to emphasize, that the deviation only depends on k. It does not depend on the graph size or the experiment setup.

On the other hand, heuristics apply a process similar to the infection model to approximate its output in reasonable time. The goodness of the approximation can

[3]We have implemented the methods in JAVA, and we have used a computer with an Intel i7-2630QM processor, and 8 gigabytes of memory.

be measured by comparing it with a simulation computed with a k large enough to minimize deviation. The difference between these can be described by an error function[4]. The error of these heuristics is highly dependent on the sizes of the edge weights as well as the size of the network.

Based on the above facts, we will evaluate the precision of the simulations and the heuristics somewhat differently.

Deviation of the simulations

The most important question of simulation is the relation between the sample size and the accuracy of the simulation. This can be measured as the standard deviation between the a posteriori infections. Increasing k obviously worsens the running time of the simulation, so it is desirable to find a balance between accuracy and complexity.

Let us make some heuristics concerning the expected results of CS. The worst case for the variance is when the characteristic function X_v of the infection of a vertex v follows a Bernoulli distribution. Then the standard deviation of X_v is $\sigma_v = \sqrt{p_v(1-p_v)}$. The standard deviation of the k-element sample is

$$\frac{\sqrt{kp_v(1-p_v)}}{k} = O\left(\frac{1}{\sqrt{k}}\right),$$

that is to get one more correct digit in the outcome one needs one hundred times more iterations.

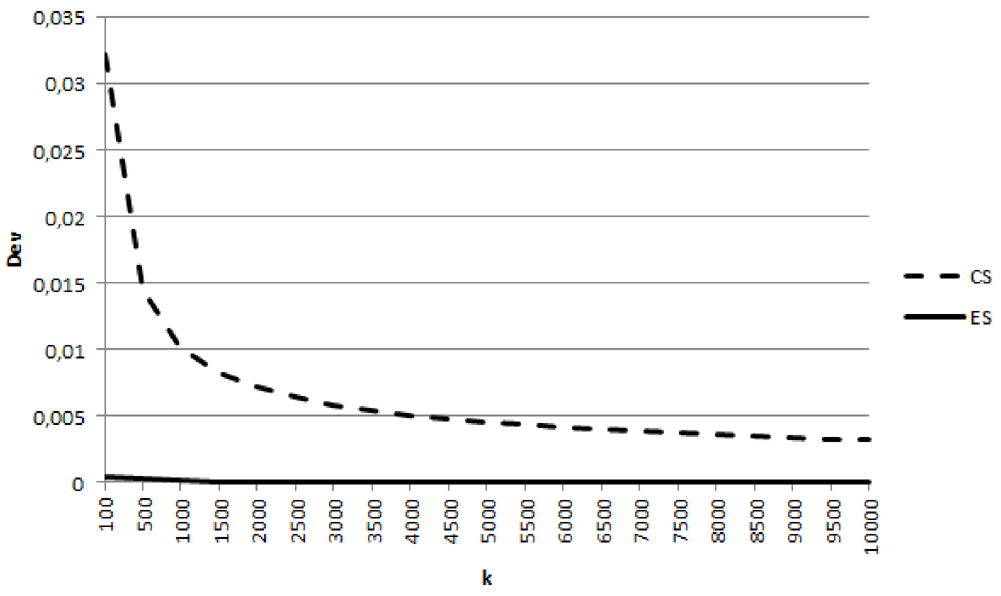

Figure 4.1: The absolute deviation compared to the sample size. In order to evaluate very large values of k, we have used a small benchmark network.

Of course, $X_v = Y_v + Z_v - (Y_v Z_v)$, where Y_v and Z_v are the characteristic functions of the a priori infection and the infection caused by the network, respectively. We might assume Y_v and Z_v independent of each other, and Y_v follows Bernoulli

[4]For this purpose we have used the root mean squared error function (RMSE).

distribution. Now, if we simulate the edges and use the vertex a priori infection probabilities directly, then the approximation of p_v is significantly improved. Indeed, an easy computation gives that

$$Var(X_v) = Var(Y_v) + Var(Z_v) \leq \mathbb{E}[Y_v](1 - \mathbb{E}[Y_v]) + Var(Z_v).$$

However, if we handle Y_v as a constant of value $\mathbb{E}[Y_v]$ then $Var(X_v)$ drops to $Var(Z_v)$. The infection coming for the network is usually much smaller than the a priori infection, and since it is the sum of almost independent variables, the variance of X_v must be even much smaller.

Figure 1 shows the standard deviation compared to the sample size k for both methods. The deviations are averaged for all nodes of the network. It can be seen, that the empirical results for CS roughly correspond to our heuristics. It can also be seen, that even for small values of k ES performs better by magnitudes than CS. In fact, the deviation goes below 10^{-5} for $k = 1500$, while using CS it goes below 10^{-4} only for $k = 100000$.

Precision of the heuristics

In order to measure the accuracy of NBH and ALE, we have used a very accurate Edge Simulation with $k = 5000$ as a benchmark. It can be seen above that the expected error of this simulation is less than 10^{-5}. Root mean squared error is used to measure the difference between the a posteriori distributions.

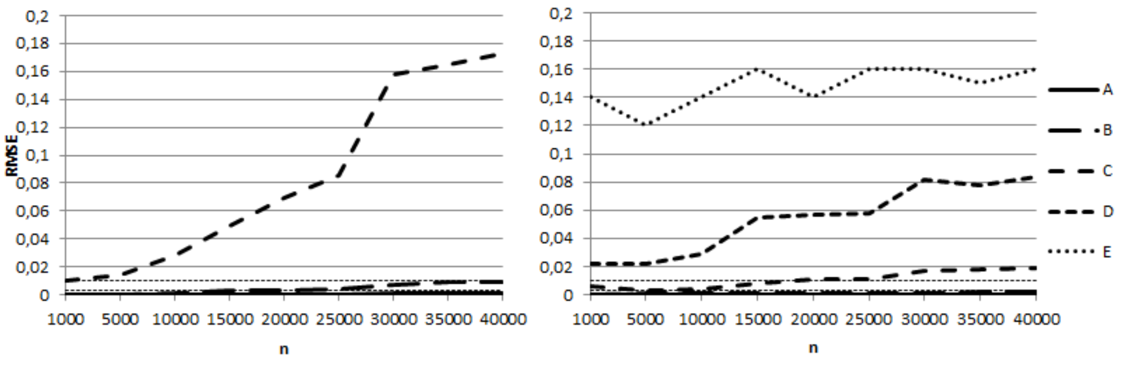

Figure 4.2: The RMSE of ALE (left) and NBH (right) for all experiment setups. The small dashed lines indicate the standard deviation of CS for $k = 10000$ and $k = 1000$.

On the left hand side of Figure 4.5, we can see the performance of ALE compared to the benchmark. If the edge weights are small enough (below 0.1), ALE is able to approximate the GC model with reasonable accuracy. For higher weights the performance of ALE gradually worsens to the point, that for the last two setups it is not able to produce output within acceptable bounds. The accuracy of the method also depends on the size on the graph, although for the first two setups, this is barely noticeable. The situation does not look so grim if we consider our experiences with the edge weights of economic networks, which typically fall into the first two category.

If we take a look at the performance of NBH on the right, we can see, that if the edge weights are below 0.2 (Setup D), the error remains below 0.1. It is easy to see, that lower edge infection probabilities produce more accurate approximations, which confirms our expectation, that if these probabilities are low enough, infections do not travel far. Like ALE, NBH is also slightly dependent on the size of the network. It is also clear, that NBH performs better than ALE in general, producing lower error levels for the same arrangements of infection probabilities.

We can compare these result to the deviation of the simulation based methods. The small dashed lines on both parts of Figure 4.5 indicate the standard deviation of CS with $k = 10000$ (lower) and $k = 1000$ (higher). Take note, that according to Figure 4.4, the deviation of ES is lower, even for $k = 100$. Based on the above fact, we can say, that the performance of both methods is comparable to the simulations only if the infection probabilities are small enough.

Computational time

On Figure 4.6 we can see the computational time of our methods on four probability arrangements. The results for setup A and B were almost the same, so we have left out the first one.

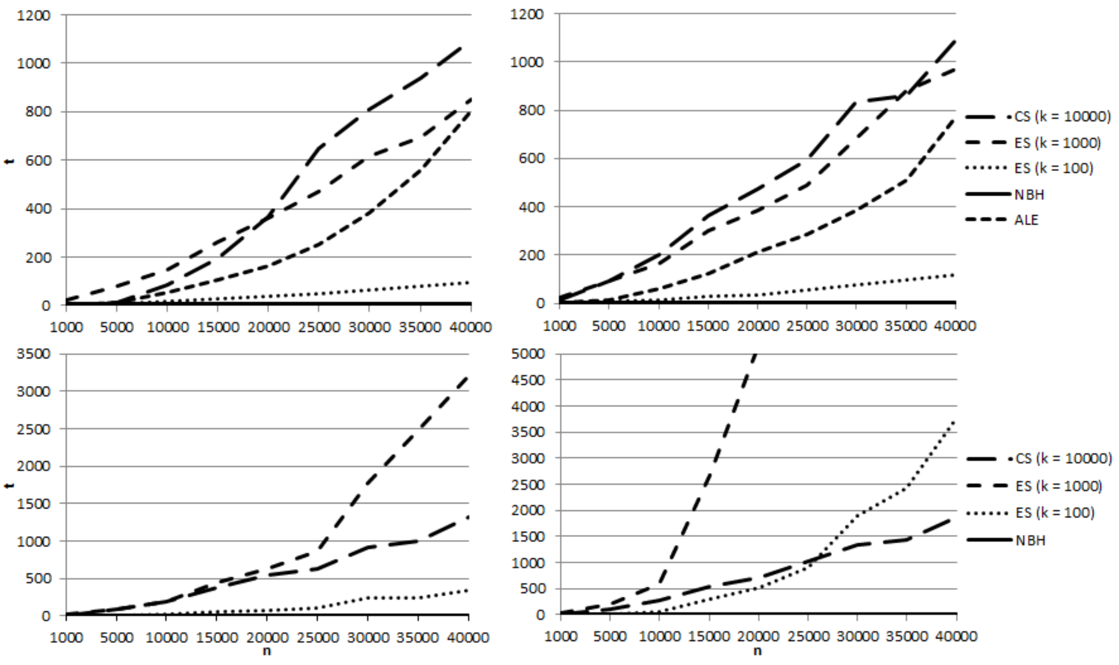

Figure 4.3: The running time of the algorithms measured in seconds, compared to the size of the graphs. Experiment setup B (upper left), setup C (upper right), setup D (lower left) and setup E (lower right).

The running time of CS scales more or less linearly with graph size, independent of probability setups. However, the deviation of CS only decreases below an acceptable level if k is sufficiently high. This means, that even tough the computation of a single G' is fast, a large amount of realizations have to be generated in order to compete with other methods. As a consequence, CS is the slowest of the methods on the first three experiment setups. Its use is only recommended if the probabilities are too high for the other algorithms to tackle.

Unlike CS, the performance of ES is highly dependent on the size of infection probabilities. The computational time gradually worsens with the increase of edge weights, to the point where even $k = 100$ is infeasible. This corresponds with our remarks in Section 4.1.2. Take note though, that the deviation of ES is smaller than CS by magnitudes, meaning an ES with $k = 100$ outperforms a CS with $k = 10000$ in terms of precision. In our area of interest, ES seems to be a reasonable compromise between precision and computational time.

Due to its local nature, NBH is the fastest of the methods. Even on the largest graphs it completes the task in less than two seconds for all experiment setups. As we have seen in the previous section however, this comes at a cost of decreased precision, on all but the smallest probability arrangements. As a consequence, its use is only recommended if the network is large, and the infection probabilities are low.

ALE scales similarly compared to the simulations with a slight increase in performance for the first three probability arrangements, but it is unable to produce meaningful output on the last two setups. If we take the findings of the previous section into account, then we can conclude, that ALE should only be used if the infection probabilities are very low.

4.2 The Inverse Infection Problem

Following the framework of the Generalized Cascade model, an infection model computes the a posteriori infections given a weighted graph and the a priori infections as inputs. In the inverse infection problem the a priori and a posteriori distributions (and an unweighted network) are provided as inputs, and we want to assign edge infection probabilities such that the infection model with the input a priori distribution results in the given a posteriori distribution. Based on this, the Inverse Infection Problem can be defined as in [22]:

Inverse Infection Problem: *Given an unweighted graph G, the a priori and the a posteriori probability distributions p_v and p'_v, compute the edge infection probabilities w_e for all $e \in E(G)$.*

Independently estimating all edge weights of a network is both underdetermined and computationally unfeasible, even if the number of edges is small. Instead, we assume the edge probabilities can be expressed as (normalized) functions of some properties of the edges or nodes that are available in the form of attributes. If there is only one attribute, this function can be expressed as $f(a_1(e))$, where f is the attribute function and $a_1(e)$ represents the attribute of edge e. We have used low-degree polynomials or simply a linear functions like $c_0 + c_1a_1(e)$, where c_0 and c_1 are unknown coefficients. The degree of these polynomials should be low, but we allow different polynomials on different edges, with possibly different degrees. If the maximum degree of these polynomials is f_{max}, then there are at most $(f_{max} + 1)\ell$ coefficients to estimate. If there are multiple attributes it is necessary to summarize the effect of them, and even in the case of a single one, normalization is required to get valid probability values between zero and one. Therefore it is necessary to use two functions, the attribute function is applied to the individual attributes on each edge, then the results of these functions are summarized and normalized: $w_e = g(f(a_1(e)), f(a_2(e)), ..., f(a_n(e)))$, where w_e is the edge weight of e, g is the summarizer-normalizer function, f denotes the attribute function and $a_i(e)$ represents the i-th attribute of edge e. The attribute and the normalizer function is the same for all edges, this way we only have to estimate the coefficients of these functions, and since the number of attributes and coefficients is limited, the problem becomes tractable.

4.3 Learning the edge infection probabilities

The Inverse Infection problem formalizes the requirements of an algorithm capable of estimating the edge infection probabilities. In this section we will define a learning algorithm capable of this task. The problem definition states, that the a posteriori distribution is required as an input of any algorithm. In the case of a learning algorithm, it is considered to be a test or reference dataset. Then the initial coefficients for the edge attribute functions are chosen from reasonable bounds. Given these attribute functions the coefficients and the a priori distribution we can compute an a posteriori distribution corresponding to the chosen coefficients. Finally, an error function calculates the difference between the reference set and the newly calculated infection values. The process aims to minimize this error function by repeatedly adjusting the coefficients. The values of the coefficients and the error together defines an error surface, and our goal is to find the minimum of this surface with respect to the error. This is a typical task for global optimization, i. e. finding the minimum of an unknown multidimensional surface, where the points of this surface can be accurately estimated.

4.3.1 Previous approaches and experiences

We have tried several optimization algorithms, including more simple ones, like grid search, gradient-based methods and meta-heuristics. Our first analysis was performed on banking data where we used grid search for optimization. While this early approach provided some results, it was clear that further refinement of the optimization algorithm was required. Later, we have implemented a multi agent gradient based method, and compared the performance of it with our previous results [21]. The gradient method provided more accurate estimations, but highlighted several unfortunate properties of the problem itself.

Our first observation was that the error function was noisy. This comes from using Monte Carlo methods to approximate the IC model, since the deviation of these simulations makes different runs with the same coefficient values have different results. The noise can be reduced by increasing the frequency parameter of the simulation, but this also increases the time complexity of the method [21].

The second observation was that the problem is underdetermined. Different edge weight configurations can result in the same infection pattern, the same a posteriori distribution. This results in alleys and plateaus on the error surface. In the case of the example on Figure 4.4, the global minimum is in the middle of the alley. Even in this simple example neither algorithms are able to reliably find the best solution.

Grid search had serious performance issues both in finding the global minimum and in time complexity. Due to its search pattern, its precision is simply not enough to tackle with this surface, and it also scales exponentially with the number of coefficients. The gradient method also performs poorly: it easily gets lost on the alleys and plateaus especially if they are noisy as well. As a consequence, it rarely

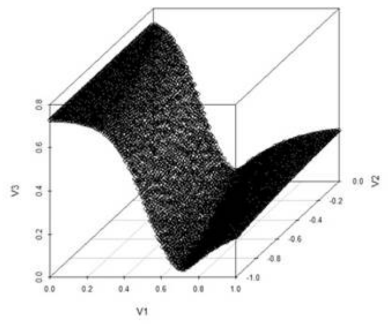

Figure 4.4: The error surface of an IIP with one edge attribute and $f_1(a_1) = c_0 * a_1 + c_1$ as the attribute function. Root mean squared error was used for the evaluation.

finds a solution close to the global minimum, and the number of steps it takes to find a solution at all can be quite high.

We have tried several error functions, mainly vector distance measurements, and ROC evaluation. One of our first experiences was, that the latter is not enough to properly guide the optimization method to the global minimum, so we have shifted our attention to other measurements, and finally settled on the root mean squared error. In this work we are going to use the RMSE as an error measurement, that is we are looking for the minimum of

$$\sqrt{\frac{1}{|V(G)|} \sum_{v \in V(G)} (\vec{p}'_v - \vec{p}_v)^2}, \tag{4.1}$$

where $\vec{p}'_v$ denotes the estimated a posteriori infection of vertex v.

4.3.2 Particle-Swarm Optimization

In order to handle the above mentioned problems, we have decided to implement the particle swarm optimization algorithm of Kennedy [66]. This is an iterative method based on the interaction of multiple agents or "particles". Each agent corresponds to a different coefficient configuration, representing a coordinate in the parameter space of the problem[5].

Apart from the coordinates themselves, the agents also have a velocity. In each iteration the position of an agent is updated by adding its velocity. The velocity of the agent is computed using the best solution the agent has found and the best solutions of the neighboring agents; the goodness of the solution is measured by evaluating the error function on the coordinates visited by the particles. Agents are connected to each other according some topology describing the neighborhood of each agent.

The specific way the velocities of the agents are updated and the topology itself is not fixed: there are various approaches in the literature for specific applications and for more general problem solving. In our work we have followed the recommendations of Kennedy and Mendes [67], and found, that it performs well in finding coefficient configurations close to the global minimum.

[5]Again, the subject of the optimization is the coefficient values of the attribute function(s)

4.3. Algorithm. Particle Swarm Optimization

```
 1: for all $a_i$ do
 2:     Initialize $\vec{x}_i$ for agent $a_i$ within the boundaries of the search space
 3:     Initialize $\vec{v}_i$ for agent $a_i$
 4:     Set $\vec{b}_i \leftarrow \vec{x}_i$
 5:     Select the neighbors of $a_i$ according to the topology
 6: end for
 7: repeat
 8:     for all $a_i$ do
 9:         Update $\vec{v}_i$ according to equation 2
10:         Update $\vec{x}_i$ according to equation 3
11:         Calculate the error function $e(\vec{x}_i)$ in position $\vec{x}_i$
12:         if $e(\vec{x}_i) < e(\vec{b}_i)$ then
13:             $\vec{b}_i \leftarrow \vec{x}_i$
14:         end if
15:     end for
16: until termination criterium is met
```

We have used the Fully Informed Particle Swarm published in [67] with 9 agents in a von Neumann neighborhood[6]. The position and the velocity of the agents are updated according to the following equations:

$$\vec{v}_i \leftarrow \chi \left(\vec{v}_i + \sum_{n=1}^{N_i} \frac{U(0,\varphi)(\vec{b}_{nbr(n)} - \vec{x}_i)}{N_i} \right), \tag{4.2}$$

$$\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i, \tag{4.3}$$

where $\vec{x}_i$ and $\vec{v}_i$ denotes the coordinate and velocity of particle i, $U(min, max)$ is a uniform random number generator, $\vec{b}_i$ is the best location found so far by particle i, N_i is the number of neighbors i has and $nbr(n)$ is the nth neighbor of i. The formula has two parameters: χ is the constriction coefficient and φ is the acceleration constant. Again, we have used the recommendations of Kennedy et al., and set $\chi = 0.7298$ and $\varphi = 4.1$.

At the beginning of the search, the agents are initialized with zero velocities and random starting coordinates within some reasonable bounds of them. Then in each iteration these two vectors are updated according to equations 2 and 3 in a synchronized manner. The search is completed if the global minimum found considering all agents does not change for five consecutive iterations. We have experimented with other values and found, that increasing it does not improve the quality of the results, and decreasing it does not reduce the running time considerably.

4.4 Performance on benchmark networks

The most natural way to evaluate the stability of the optimization method is by counting the average and maximum number of iterations the method takes before

[6]Each agent has four neighbors in a grid, connected to the upper, lower, left and right, while wrapping around the edges.

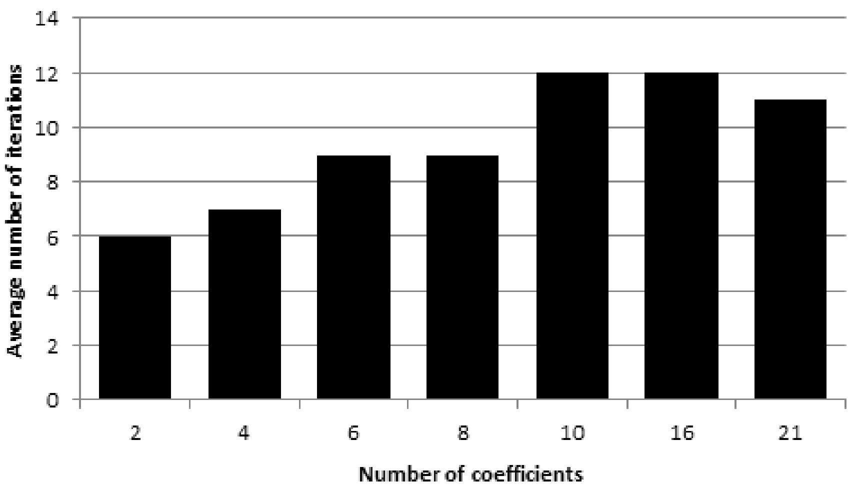

Figure 4.5: The average number of iterations with different function configurations.

it finishes. However, the quality of the solution of the Inverse Infection Problem depends on additional factors; we will discuss three of these. The first one is the choice of the attribute functions. Choosing an appropriate function is important, since depending on the available attributes this function either maps into the $[0, 1]$ interval directly or some additional form of normalization is required. The second one is the choice of heuristics applied for the GC model. These have a serious impact on both the accuracy and the running time of the learning method. The third factor is the number of learning patterns available. In case the exact a priori and a posteriori infection probabilities are not available, the only thing to do is to rely on counting the frequencies of infections. In real life we cannot hope to witness an infection process on any network in more than a handful of times. It is therefore necessary to investigate the sensitivity of our method to low-quality inputs.

As a basis of our analysis we have used graphs generated with the forest fire method of Leskovec et al. [81]. We have created a series of graphs of sizes $n = 1000, \ldots, 100000$, with parameters $p = 0.37$ and $p_b = 0.32$, for forward and backward burning probabilities, respectively. We have assigned a number of edge attributes $a_i, i = 2, \ldots, 10$ to these networks. These attributes are randomly generated: they were drawn independently from a uniform distribution between $[0, 0.5]$. We have also used randomly generated a priori infections. The expected size of the set of initial infectors is $0.3 * n$. If v is selected, then an a priori infection probability was drawn from an uniform distribution between $[0, 0.5]$, otherwise $p_v = 0$. We have used various attribute functions, description of these will be given in section 4.4.2. Finally for each network and each attribute function we have created an a posteriori infection distribution as the reference dataset. For this purpose we have used Compete Simulation with sample size $k = 10000$, because this method gives the best approximation of the original IC model.

4.4.1 Stability of the optimization

The performance of the optimization method itself can be measured in two ways. The distance between the solution found by the method and the global minimum is conveniently measured by the error function itself. However, the precision of the algorithm also depends on the heuristics used to approximate the Generalized Cascade model. Consequently, we will discuss this in the following sections.

The time complexity of the method is the sum of two distinct parts of the algorithm: evaluating points on the error surface and the search method itself. The latter consists of the repeated evaluation of the formula above, after initializing the neighborhood and the starting coordinates. Since the number of agents is small, this part of the algorithm is very fast, and has negligible impact on the running time of the overall method.

In each iteration every agent evaluates the error function. This evaluation is the computation of the GC model using the coordinate - coefficients of the given agent. The time complexity of this step heavily depends on the used heuristic. Altogether, we can say, that the time complexity of a single run is $s * a * h$, where s is the number of iterations, a is the number of agents (a constant) and h is the time complexity of the infection heuristic. This also means, that we can describe the time complexity of the algorithm by measuring the average or maximum number of iterations and multiplying it with the time complexity of the infection method and the number of agents. Breaking the time complexity of the method into two different factors makes sense because of another reason: the individual runs of the GC heuristics may be run on multiple threads simultaneously, significantly improving the speed of the method.

On Figure 4.5 we can see the average number of iterations for different numbers of coefficients. We have used a small network with $|V(G)| = 1000$. The point of interest here is, starting from a simple problem with only two coefficients to more complex ones, the expected number of iterations grows slowly, and stabilizes around 12. The maximum number of iterations remains bounded as well, even in the experiment with 21 coefficients, it does not go beyond 30. The results shown on Figure 4.5 were computed with 9 agents. We have tried this problem with 16 agents as well and got similar results. If we compare the different infection heuristics, they perform similarly, with the non-Monte Carlo methods finishing slightly sooner, usually by 4-5 iterations.

We can conclude, that the Particle-Swarm Optimization method described in this section is able to solve the Inverse Infection Problem with satisfying results. The algorithm is very stable, and even in the worst case, it finishes within 30 iterations. We will evaluate the precision and running times of this method considering different heuristics of the Generalized Cascade model in section 4.1.2. We will also discuss choices for attribute functions, and the number of patterns required to get good estimations of the edge infection probabilities.

4.4.2 Choice of attribute functions

We have seen, that in our model, the edge infection probabilities are computed from some additional information on the edges in the form of edge attributes by so-called attribute functions. The choice of these attribute functions is an important part of our method. A natural requirement of this function is, that it must result in infection probabilities: it must map into the $[0, 1]$ interval.

There are two approaches to this problem: the first one is to construct problem-specific functions, taking into account the structure of the network, the nature of the infection model and the number and domain of the attributes. This way it is

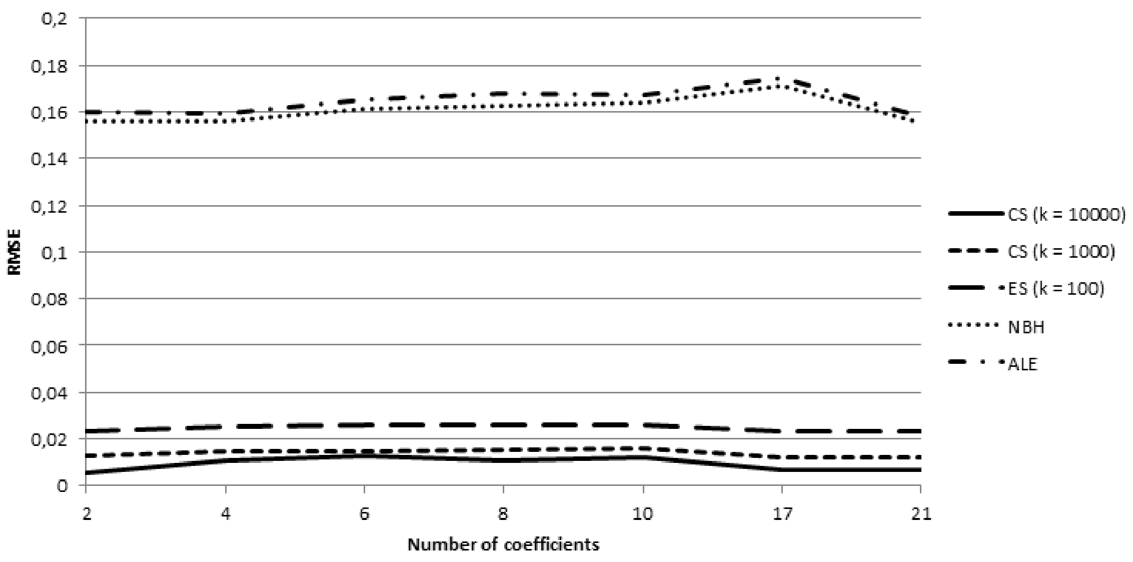

Figure 4.6: The RMSE with different function configurations on a small network with $n = 1000$.

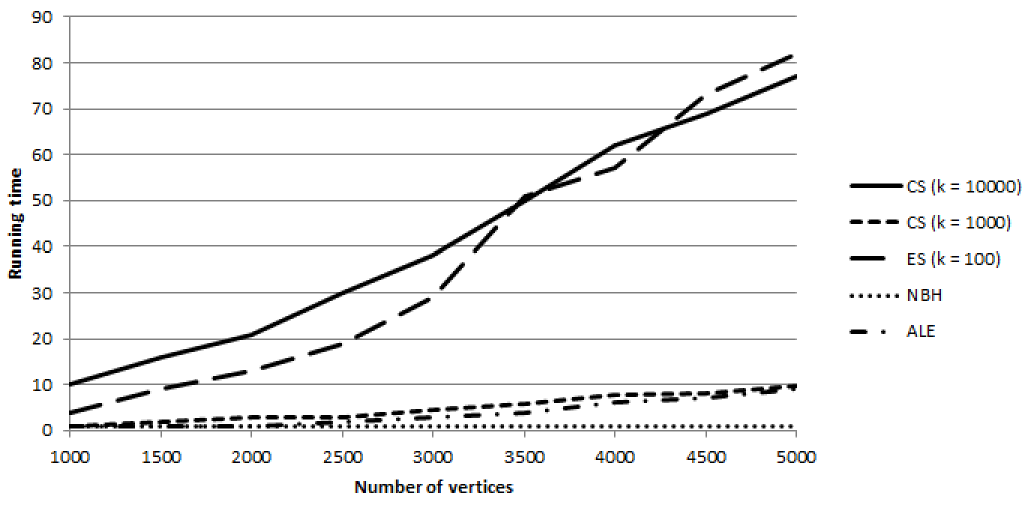

Figure 4.7: The running time of the infection heuristics with different network sizes measured in seconds. Figure used with the permission of the authors [19].

possible to calculate the infection probabilities directly, without any form of additional normalization. This is the obvious choice if the above mentioned information is available.

If we do not have this information, we can try a more user-friendly approach. We can apply functions to the individual attributes, summarize them and finally normalize them. A variety of functions might be considered for this purpose. In our work, we have used low-degree polynomials for the individual attributes and simple addition or multiplication to join them. We have normalized the resulting edge infection probabilities according to

$$\text{norm}(\vec{e}) = \frac{\vec{e} - \min(\vec{e})}{3(\max(\vec{e}) - \min(\vec{e}))}, \tag{4.4}$$

where $\vec{e}$ is a vector containing the infection probabilities for each individual edge. The reason why we have used the multiplier 3 in the denominator is, that according to our findings on banking data, the edge infection probabilities are low [34]. The normalizer function obviously distorts the shape of the individual attribute functions, but in real-life problems a simple weighted, normalized sum of attributes is sufficient to produce acceptable results.

In our works, we have used seven attribute function configurations, $a_i(e)$ denotes attribute i of edge e and c_j denotes coefficient j:

- Weighted sum of two attributes: $c_1a_1(e) + c_2a_2(\text{e})$, two coefficients in total.
- Weighted sum of four attributes: $\sum_i c_ia_i(e), i = 1, 2, 3, 4$, four coefficients in total.
- Weighted sum of six attributes: $\sum_i c_ia_i(e), i = 1, \ldots, 6$, six coefficients in total.
- Weighted sum of eight attributes: $\sum_i c_ia_i(e), i = 1, \ldots, 8$, eight coefficients in total.
- Weighted sum of ten attributes: $\sum_i c_ia_i(e), i = 1, \ldots, 10$, ten coefficients in total.
- Sum of quadratic polynomials with eight attributes $c_1 + \sum_i(c_{2i}a_i(e)^2 + c_{2i+1}a_i(e))$, $i = 1, \ldots, 8$, 17 coefficients in total.
- Sum of quadratic polynomials with ten attributes $c_1 + \sum_i(c_{2i}a_i(e)^2 + c_{2i+1}a_i(e))$, $i = 1, \ldots, 10$, 21 coefficients in total.

We have tested the effect of these function configurations on the stability and accuracy of the optimization method. Details of these can be found in the appropriate subsections.

4.4.3 Accuracy and the choice of heuristics

We have already seen the heuristics of the GC model [19] in section 4.1.2. In this section we will evaluate the performance of them in relation with the learning method described above. Complete Simulation is a direct adaptation of the idea of Kempe et al. [64], it can be considered as the best approximation of the original IC model. Therefore, we will use CS with sample size $k = 10000$ to create an a posteriori

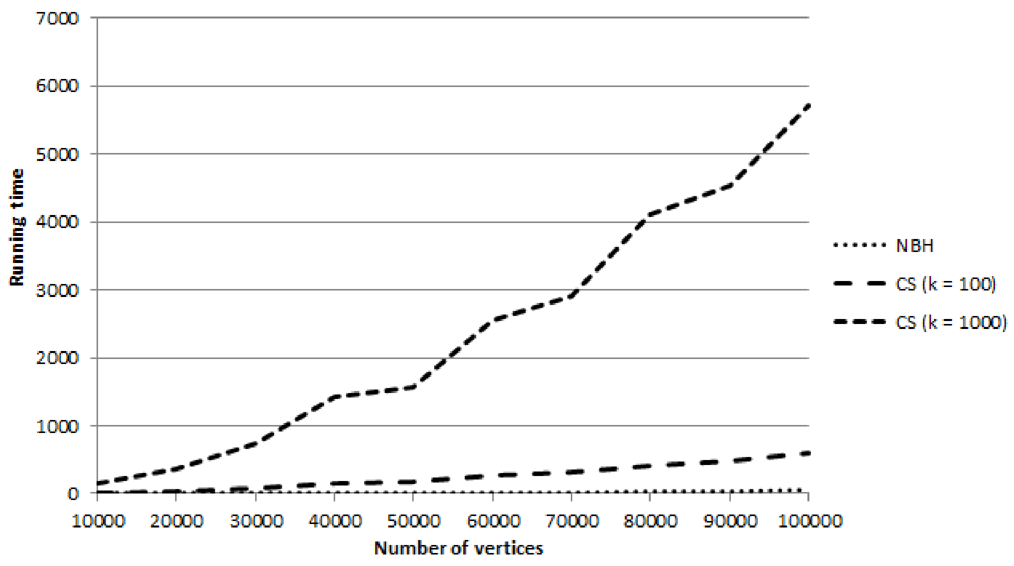

Figure 4.8: The running time of the infection heuristics on large networks measured in seconds.

distribution as a reference set. Then, we will use each heuristic together with the learning method to compute the edge infection probabilities:

- Complete Simulation (CS) with sample size $k = 10000$, a very accurate simulation.
- Complete Simulation (CS) with sample size $k = 1000$, a very fast simulation.
- Edge Simulation (ES) with sample size $k = 100$, a simulation based heuristic.
- Neighborhood Bound Heuristic (NBH), an extremely fast lower approximation.
- Aggregated Linear Effect (ALE) model, a de Groot [36] based simplification of the infection process.

Since the running time and the accuracy of the heuristics are different, we are going to use two different datasets to evaluate their performance. First, small networks with $|V(G)| \leq 5000$ will be used to make general observations, then we will test the more robust heuristics on large networks with $|V(G)| = 10000, \ldots, 100000$. Our largest network has 100000 vertices and 2.3 million edges.

As we can see on Figure 4.6, the Monte Carlo based simulations of the GC model (CS and ES) are able to estimate the reference distribution well, with the measured error between $0.01 - 0.03$. The other two heuristics (NBH and ALE) are tailored to small edge infection probabilities with rare infections, hence they do not perform so well on this dataset. Note, that in some cases even an error of this magnitude is acceptable, and the time complexity of these methods allows them to handle larger networks. If we compare the results computed by using different attribute functions, we can see that they have minimal effect on the accuracy of the methods.

Our results on the running times[7] of these heuristics on small networks correspond with our previous findings [19]. The speed of the simulations are governed by

[7]We have implemented the methods in JAVA, and we have used a computer with an Intel i7-2630QM processor, and 8 gigabytes of memory.

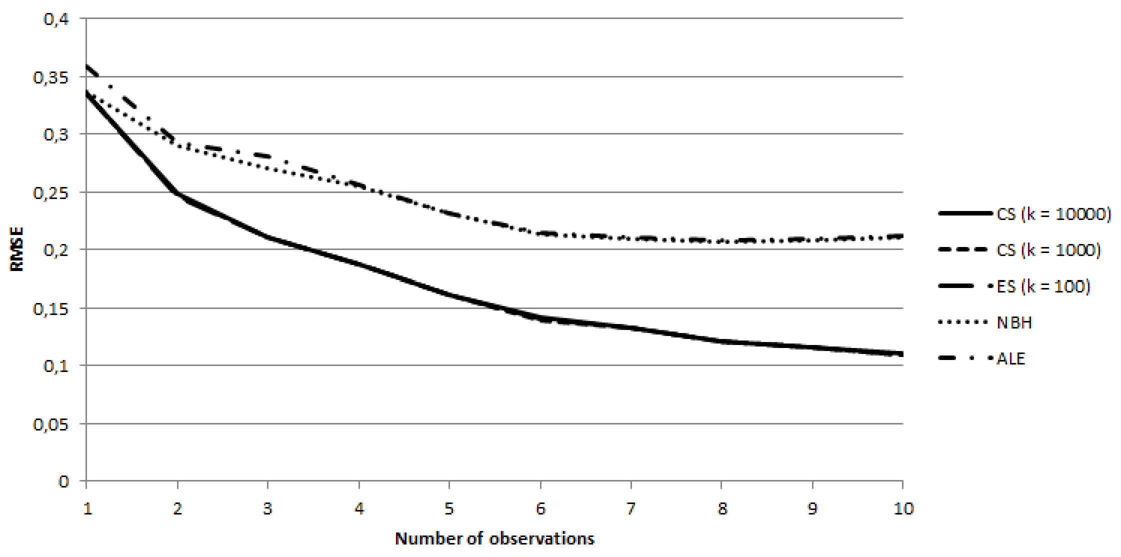

Figure 4.9: The precision of the infection heuristics with a limited number of observations of the reference distribution.

the sample size. Complete Simulation is considerably faster than ES[8] because the latter focuses on the fast computation of smaller infections. By decreasing the sample size CS can tackle larger networks as well. The Neighborhood Bound Heuristic is able to compute the a posteriori infections of the largest networks within a minute, enabling our method to scale upwards and handle real-life datasets and networks with possibly millions of nodes and edges.

We can conclude, that in general, the use of Complete Simulation is recommended. Both its precision and accuracy are good on large graphs. If the infection probabilities are lower than our current dataset, the use of Edge Simulation is also advisable. The Neighborhood Bound Heuristic and the Aggregated Linear Effect model are able to handle even larger networks, yet this comes at the cost of a significantly lower precision.

4.4.4 Number of patterns

In many real-life applications the a priori or a posteriori probabilities of infections are unavailable. In this section we are going to assume, that the initial infection probabilities are given, but we only have a small number of observations on the a posteriori infections. We are going to simulate this on a small network by generating an a posteriori distribution using CS with $k = 1, \dots, 10$, corresponding to $1, \dots, 10$ observations.

We can see, that the proposed method gives a rough estimate of the vertex infection probabilities in only a few iterations. If we consider a threshold of 0.15 as an acceptable estimation, our method only requires 6 observations to reach it. However, it is important to keep in mind, that the method tries to create a posteriori infections close to the reference. The problem is underdetermined even with exact possibilities of vertex infection, with only a handful number of observations many attribute function configurations (and edge weights) may result in the same infection.

[8]Note the sample sizes.

The results in this section only imply, that our method is able to give one of these.

Different infection heuristics are shown on Figure 4.9, one can see, that the simulations have identical performance regardless of the sample size. As before, the non-Monte Carlo based methods perform poorly, their use is not recommended with low-quality inputs.

4.5 Case-study on bank transaction networks

Near the end of this chapter we return to the original problem, that inspired the development of the Inverse Infection Problem. As we have mentioned before, some of the ideas of this method, like representing initial and final infections as distributions and the idea of computing infection probabilities from attributes, came from experience from a previous joint work with the OTP Bank of Hungary[9] on a different project. Existing models did not allow us to go deeper into this problem, therefore we decided to develop our own method. Later, when the preliminary results for the IIP were available, it became possible to test our method on the transaction database, with the task of improving current methods for the estimation of credit default of companies.

The bank has information about the properties of the companies and is able to give an estimation of the individual probability of default for them. The goal of this project was to improve this estimation by taking into consideration the effect companies have on each other. If one of the companies goes into default, how does this change the probability of default of other companies, especially the ones with financial ties to the original one? We can reach this goal by considering the network of companies and the connections between them, and computing the edge infection probabilities between the vertices with the optimization method proposed in [22].

We will begin with the construction of the network itself, the properties of the infection scenario, the required probability distributions and the available edge and vertex attributes. We will describe the form of evaluation we will use to evaluate the result, then we will present them, and draw some general conclusions as well.

It is important to emphasize, that this is an industrial application, therefore many of the details of this case-study are not available to the public either because of privacy reasons or because it is a part of the know-how of the bank. The transaction network itself cannot be published, because the attached attributes often contain information on the neighboring companies that are not public. We were not allowed to release the transaction network even in an anonymous manner. We were also not allowed to present the exact coefficients representing the significance of the attributes. Only the results presented in this section are public.

4.5.1 Infection scenario

In order to deeper understand the behavior of purchaser-supplier connections, a transaction database was created by the bank over a large time period. We can construct a network from this database, where vertices represent corporate clients and edges represent financial connections between clients. Since there are many transactions between the clients, we must decide how to define the edges of the

[9]We will refer to the OTP Bank of Hungary simply as bank from now on.

graph: a filtering process is required during the construction of the network. We have used three criteria to decide whether two nodes should be connected or not:

- Frequency of the transactions: the average number of transactions in a month.
- Amount of transactions: the average transacted amount in a month.
- Relative amount: average incoming amount from one company divided by the total income from all of the partners.

For each criterion above, we have assigned the transactions between the companies into three categories[10]: high, medium and low. We have added an edge between the companies if the transactions between them belong to the high category for each criterion, that is two companies are connected if the transactions between them have high frequency and they are also in the high total and relative amount part of the database. The transactions also have a natural direction to them: they go from the purchaser to the supplier, therefore the graph is directed. Since the business partners of a company may change dynamically, we have built the network considering edges from a one year period: from April 2012 to March 2013. The resulting transaction graph has approximately 68.000 vertices, and 106.000 edges.

Probability distributions

We have selected the time period of our estimations according to the currently used practices of the bank. The task was to make short-time predictions. Which corporate client will be in credit default in the near future? We have created the a priori probabilities considering companies from a three month period starting from January 2013 to March 2013. If the client was in default in this period, it was given an a priori probability of 1, otherwise we have used an estimated probability of default. This estimation was performed by the bank with logistic regression based on a 6 months observation period of its behavioral variables. The a posteriori infection or reference values were constructed from April 2013 to June 2013. Like before, the defaulted companies were given a value of 1.

Attribute functions

The following attributes are available in the database. Some of these are not edge attributes, but vertex attributes that are related to suppliers[11], like its age and type.

1. The number of transactions.
2. The amount of the transaction.
3. Total incoming transactions of the client (supplier).
4. Community information: if the edge belongs to a community[12].
5. Relative traffic, that is the transfer of the edge divided by the sum of all incoming transfers.

[10]Uniform 33% of the cases in each category.

[11]A vertex v is a supplier if it is at the end of a directed edge.

[12]Here we used a N^{++} algorithm for community detection, see [17].

6. The age of the supplier (how old is the company).
7. Unpaid items on the accounts of the supplier.
8. Limit exceeded (for overdrafts)[13].
9. Whether the supplier is a company or a municipality.

In the inverse infection problem the edge weights are computed from edge attributes by the means of attribute functions. In this application we had the above introduced nine attributes : a_i, $i = 1, \ldots, 9$. We have tried several simple attribute functions including polynomials, but have found little difference between them both in terms of accuracy and computational time. For the sake of simplicity, we have chosen a weighted, normalized sum as the attribute function. More formally:

$$w'_e = f(a_i(e)) = \sum_i c_i a_i(e), i = 1, ..., 9,$$

then the resulting values are normalized according to

$$\text{norm}(\vec{e}) = \frac{\vec{e} - \min(\vec{e})}{(\max(\vec{e}) - \min(\vec{e}))}, \tag{4.5}$$

where $\vec{e}$ stands for the unnormalized vector of edge weights. In this formulation the algorithm computes a weighting of the attributes based on their importance in the infection process.

4.5.2 Evaluation

In the previous section we have discussed the application of the Inverse Infection Problem on a bank transaction network, now we are going to present its results. As we have mentioned before, our goal was to improve currently used methods by the bank. Therefore we will use two of these methods for comparison. One of them is a simple logistic regression based on a 6 months observation period of the companies' behavioral variables. The other one uses the same network we have used for the IIP model, but instead of estimating the edge weights it uses a uniform constant value for each edge. The focus of this section is the accuracy of the estimations: how well are we able to predict short time default events. For this purpose we will use well-known performance measurements like ROC evaluation, GINI and RMSE. We will discuss these and the accuracy result in the next section. We will also take a look at the speed of the estimations in a subsequent chapter. We will close the evaluation with some general observations and remarks.

Accuracy of estimations

We have tried various measurements to evaluate the performance of our method. The optimization algorithm itself uses the root mean squared error (RMSE) function to guide the search,

$$\sqrt{\frac{1}{|V(G)|} \sum_{v \in V(G)} (\vec{\tilde{p}}'_v - \vec{p}_v)^2},$$

[13]The actual outstanding is higher then the given credit-limit.

$\frac{\text{TOP \% default rate}}{\text{Average default rate}}$	Regression	Constant	IIP
TOP 1%	7.27	0.79	7.82
TOP 3%	8.11	2.17	8.77
TOP 5%	7.87	2.89	7.97
TOP 10%	4.97	3.16	4.99
Other measurements			
AUC	72.4 %	65.39 %	72.5 %
AUC (lower bound)	69.9 %	62.97 %	70 %
AUC (upper bound)	74.9 %	67.81 %	75 %
GINI	11.2 %	7.69 %	11.24 %
RMSE			0.1661

Table 4.2: Performance of the IIP solution compared to existing methods used by the bank. The first column corresponds to logistic regression, the second stands for the network model with constant weights, the third one for IIP.

where $\vec{p}'_v$ denotes the estimated a posteriori infection of vertex v. However this value is not available for the other models.

The ROC curve was also used to measure performance. We can construct it in the following way: we order the vertices of the graph in a monotone decreasing way by their a posteriori infection values computed by the model. Let $t'_1, \ldots, t'_n$ be the binary values of these vertices given in the reference set, and the function

$$\mathrm{roc}(x) = \frac{\sum_{i \le x} t'_i}{\sum_{i=1}^{n} t'_i}.$$

Now the integral

$$AUC = \int_{x=1}^{n} \mathrm{roc}(x) dx$$

should be maximized. We will also use $GINI = (AUC - 0.5)/2$ to measure performance.

We have also compared the average new default events in the reference period (excluding the ones defaulted before) with those TOP segments of the ordering described above, where the model predicted high default probabilities: these are companies, where default is the most probable. In banking the highest concern is to identify the riskiest clients, and the ones with lower risks are far less important. Therefore this approach is the most natural in this application.

As a general observation we have found, that the AUC and GINI measurements were less effective, because the infection models are more powerful if we consider only the high influence values, and less powerful when considering the whole portfolio, where traditional methods are better.

In Table 4.2, we can see the results of the benchmark methods and the IIP based estimation. The measurements TOP 1-2-5-10%, AUC, GINI and RMSE can be seen. For example the TOP measurements, 7.77 in the first row means, that in the highest one percentage the default rate is 7.77 times higher than the average default rate.

No attributes	Running time (sec)	RMSE
9	166	0,1661
8	152	0,1661
6	140	0,1661
4	123	0,1662
2	107	0,1668
1	102	0,2324

Table 4.3: Running time and error compared to the number of used attributes

We can see in the table that using IIP is far better than the network model with constant edges. Compared to the regression, IIP did not improve the ROC based measurements, but it was able to identify the most risky clients with better precision than the regression. If we take a look at the 1-5% riskiest companies, we can see that our model is able to improve prediction by as much as 10%. Therefore using IIP estimations for short time predictions clearly means an advantage.

Speed of the estimations

We have measured the running time of the algorithm using different number of attributes. We have ordered the attributes by importance, and we have assigned the top 9, 8, 6, 4, 2, 1 most important ones for different runs. Here we found that the computational time of the algorithm can be reduced by decreasing the number of attributes. The measured squared error remains roughly the same during different runs. In Table 4.5.2 we can see the running times and error for different attribute configurations.

Additional observations

We have experimented with omitting the direction of the edges in the network, because our previous studies [34] showed that Basel II default spreads in both directions. This seemingly paradoxical phenomenon comes from the different stability of companies. When a big buyer has cash flow problems, it delays the transfers causing cash flow problems for its supplier, who might go bankrupt. Still the observer, who has no detailed information on the whole situation, sees only that the suppliers bankruptcy spread to the buyer. In this model however, omitting direction decreases performance. The direction of an edge goes from the purchaser to the supplier, implying that the suppliers depend on the purchasers, therefore the bank should estimate the vulnerability of the supplier.

We have also experimented with increasing the number of edges by reducing the criteria for relevant transactions, again this resulted in decreased performance. A company can have many business partners, filtering out the more relevant ones, where the mutual dependency is higher is very important.

The most significant attributes were community information and relative traffic.

4.6 Conclusions

In this chapter we have given a detailed description on the Inverse Infection Problem, its preliminaries and its possible applications. The motivation for the creation of this model comes from applications, namely a previous work on bank transaction networks. The development itself was a huge undertaking and required additional work that resulted in a new infection framework, several heuristics, a learning method for the estimation of the edge infection probabilities, the use of of optimization methods and a final case-study on the network, that gave the inspiration itself.

This chapter introduced the infection framework IIP uses to define influence. This is the Generalized Cascade Model [21]. The model has a priori and a posteriori vertex infection values that represent the probabilities of infection before the beginning and after the end of the process. The computation of the learning method for IIP requires this model to be evaluated many times to give accurate results. To speed up the process we have given four heuristic methods for the GC model qciteacta. Two of these are frequency based simulations (Monte Carlo methods). These methods can be characterized by the number of iterations or frequency used to compute the a posteriori distribution. Increasing the frequency reduces the standard deviation, an unwelcome property of all MC methods. Complete Simulation is a direct adaptation of the method of Kempe et al. [65]. Edge Simulation is a hybrid method combining simulation with exact computations to reduce the standard deviation. The Neighborhood Bound Heuristic limits the spread of infections to the two-neighborhood of vertices. The Aggregated Linear Effect replaces the infection mechanics of GC with a simpler one.

After the GC model and its heuristics we have defined the Inverse Infection Problem [22]. In it, the a priori and a posteriori distributions (and an unweighted network) are provided as inputs, and we want to assign edge infection probabilities such that the infection model with the input a priori distribution results in the given a posteriori distribution. We have built a learning method upon this to estimate the edge infection probabilities. This method translates the problem to a typical task of global optimization, to find the minimum on a multi-dimensional error surface. After a few tries we have selected a Fully Informed Particle Swarm method for the task, and were very satisfied with its performance.

We have used artificial infection scenarios to explore the boundaries of our method. We have tested the stability and accuracy of the optimization, we have given a general approach to choose the correct attribute functions, we have examined the implications of choosing between the heuristics of the GC model and we have tested our method in low-quality inputs as well. The learning method is able to accurately predict the edge infection probabilities in a small number of iterations while the number of attributes and the shape of the attribute functions have only a small effect on this. Our method also handles low-quality inputs well.

Finally, we have presented a case-study of the Inverse Infection Problem on a bank transaction network [16]. The goal of this application was to improve the efficiency of existing models for the prediction of short-time credit default events. Since the creation of IIP was heavily influenced by banking applications, it was quite suited to handle this task. We gave a thorough description of the construction of the transaction network, the a priori and a posteriori infection probabilities and the available edge and vertex attributes.

Concerning our results, Inverse Infection estimations allowed us to find new ways to improve the efficiency of existing models for the prediction of short-time credit default events. The new model has better predictive power than traditional methods: our model can identify the companies where default is most probable better than the previously used models of the bank. We were able to predict the bankruptcies of the riskiest 5% clients 10% more accurately than the regression model. Our model was implemented in August 2013 into the OTP Bank of Hungary's credit monitoring process.

Chapter 5

Summary

The goal of this dissertation is to give an introduction into graph theory and network science, and to summarize the works of the authors in the field of community detection and infection processes. This work can be separated into three closely related topics:

1. The development of a high-resolution overlapping community detection algorithm with customizable parameters.

2. The development of a dynamic community detection algorithm able to handle large real-life networks.

3. The development of a method capable of estimating the edge infection probabilities on bank transaction and other networks.

According to this, the dissertation is organized into four chapters: an introduction into graphs and basic definitions, and the chapters corresponding to the topics of the dissertation.

In the second chapter, right after the introduction, the hub percolation overlapping community detection method is proposed [18]. This method was created to handle diverse real-life networks, with different requirements for the resulting community structure. To demonstrate the usefulness of this method, we have used the community-based graph generator of Lancichinetti [78] to compare the communities given by our method with the ones of the clique percolation method [100]. Furthermore we have given two case-studies, an economic and a linguistic one, each of them has produced satisfying results.

The next topic of the dissertation is closely related to the previous one. Following the works of Palla et al. [99] we have created a method for the identification of dynamic communities [20]. Our improved model introduces new community events, and is not tied to the monotonicity of the community detection algorithm. We have examined and compared the dynamics of two real-life networks, a bank transaction network and a social network, and found that their behavior is quite different.

The final and largest topic of this dissertation deals with infection processes with the goal of building a method that can estimate the influence nodes have on each other. These influence values are given as edge infection probabilities, and the formal definition of the task, the Inverse Infection Problem states, that these values should be estimated given an unweighted network, the initial infectors and an observation on the end of the infection process [22]. The Inverse Infection Problem is based on a probabilistic infection framework, the Generalized Cascade Model [21], and its heuristics [19]. We have evaluated the performance of our method on several artificial infection scenarios [22], and we have provided a detailed case-study on bank transaction networks as well [16].

5.1 Basic definitions

The algorithms and methods in this work are defined on graphs, with vertex (node) and edge sets $V(G)$ and $E(G)$ for a graph G. In almost all cases the graph is considered to be undirected. The graph may contain numerical information stored as weights on the edges, or attributes on both edges and vertices. The hub-percolation method in chapter 2 is able to take into account the weights between the nodes to build communities. The infection models of chapter 4 require real values between 0 and 1 as edge weights as inputs, and the Inverse Infection estimation methods aim to compute these values from additional information on the graph in the form of edge or vertex attributes. In all cases the graph is considered to be connected[1]. The dynamic community detection method in chapter 3 takes place on a series of graphs $\{G_i\}_{i\in\mathcal{T}}$, where $\mathcal{T} = \{1, \dots, \ell\}$ represents a discrete time set. However, in one step only the neighboring graphs in the series are compared.

In addition to the edge weights, the Inverse Infection Problem defines infection values on the vertices between 0 and 1. Vertices are infected independently from each other before the beginning of the process according to their a priori infection probability $p_v, v \in V(G)$. A posteriori infection values $p'_v, v \in V(G)$ denote the probability of being infected during the process.

5.2 The hub-percolation method

We began chapter 2 with the definition of the hub-percolation overlapping community detection method [18]. It is based on two concepts: cliques and hubs. Cliques are fully connected subgraphs, with a given number of nodes: a k-clique is a fully connected subgraph with k nodes. A clique is maximal if it is not a subset of any other cliques. Hubs are the nodes, that hold communities together, they are locally important nodes. The identification of hubs is the task of the hub selection strategy, in chapter 2 we have suggested several of these including one, that is able to handle weighted networks in a natural way. Some of the strategies have customizable parameters.

Description

The hub-percolation method can be divided into the following steps:

1. Find the set of all maximal cliques of size greater or equal than 3.

2. Select the set of hubs according to the hub selection strategy. The strategy may have a parameter q.

[1]This might be the result of some filtering process not mentioned in the dissertation.

3. Identify the k-cliques formed by hubs and extend them with a limited percolation rule.

4. Join communities with the same hubs.

The hub-percolation method provides several ways to influence the computation mechanism. One of them is the filtering parameter k of phase 3, another is the selection strategy and its possible parameter q. This allows our method to handle diverse real-life networks. We have measured the effect these parameters have on the results of the method in several ways: the number of communities, the community size distribution, the average overlap between the communities and the number of nodes left without communities. We have used the benchmark networks of Newman for this purpose [47, 96].

Evaluation on benchmark networks

We have used the community based graph generator of Lancichinetti et al. [78] to compare the performance of our method to a very popular clique-based method, the clique percolation method [100]. We have used mutual information [77] to compare the results of these methods with the natural community structure of the generated networks[2]. We have shown, that our method has better performance if the network has a high number of overlapping nodes.

Case-study on an economic intersection network

We have used our method to examine the community structure of an economic ownership network constructed from the Hungarian company register [18], and we made the following observations. The observed companies were of a special kind (Ltd), so that the communities formed by them are geographically close to each other and typically belong to the same industrial sector. The age of these companies however show a noticeable deviation.

Case-study on two word association graphs

In a more detailed case-study we have compared the community structure of an English and a Hungarian word association graph [23]. We have identified the words central to the formation of the communities, these are category names, common adjectives or collective nouns. Some of these are present in both networks, these words representing basic needs, everyday activities and common adjectives, but there is considerable difference between the networks.

5.3 Dynamic communities

The next topic of this dissertation was the identification of the changes communities go through in time. Our approach is based on the one proposed by Palla et al. in [99], but we made several modifications, that improve the usefulness of the method. We have extended the number of community events, because we have found the ones provided by Palla et al. are not enough to explain the events that occur in the

[2]These were given by the graph generator.

networks we were observing. We have also given a different matching algorithm, that does not depend on the specific properties of the used community detection method.

Description

Our method works by comparing two subsequent graphs in a time series, G_1 and G_2.

1. The communities of G_1 and G_2 are computed by a community detection method.
2. The union of the two graphs G_U is computed, and the community structure of this new network is computed as well.
3. The communities of G_1 and G_2 are matched to the ones of G_U.
4. The results of these matchings are summarized resulting in the community events, that happen between G_1 and G_2.

Case-studies on a social and an economic network

We have used our method to evaluate the dynamics of the community structures of two real-life networks. Our findings indicate, that there is a significant difference between these datasets. One of the networks is more stable, but looses communities steadily, the other network is more dynamical, but maintains its community number.

5.4 Inverse infection

The final topic of this dissertation was the Inverse Infection Problem published in [22] and several other works closely tied to it. The motivation for the development of this model came from a previous work with a bank transaction network, where the prediction of influence between different companies would have been valuable. The led to the creation of several models and algorithms culminating in the aforementioned Inverse Infection Problem. In the last chapter of this work we have given a detailed description of these, starting with the Generalized Cascade Model and its heuristics, and ending in a case-study in economics.

Generalized Cascade Model

The Generalized Cascade model [21] is an infection model and a generalization of the Independent Cascade Model. The Inverse Infection Problem uses this model and its heuristics. In this model each vertex is assigned a real value p_v between zero and one, that represents the probability of infection before the beginning of the process. We refer to these values as the a priori distribution, and these are the inputs of the model. Similarly to the input, the output of the model is given as an a posteriori distribution, where values p'_v indicate the probability of being infected during the process for all $v \in V$.

We can define the Generalized Cascade model in the following way:

The Generalized Cascade Model: Given an appropriately weighted graph G and the a priori infection distribution p_v, the model computes the a posteriori distribution p'_v for all $v \in V(G)$.

Heuristics of the GC model

The computation of the GC model is #P-hard, therefore we have proposed four heuristic methods for it in [19].

- Complete Simulation is the direct adaptation of the method in [65] for the GC model.
- The Edge Simulation method is a combination of both simulation and exact computations that decreases the standard deviation appearing in other simulations.
- If the infection probabilities are small, then the infections typically do not travel far from the source of infection. Neighborhood Bound Heuristics exploits this property.
- The Independent Cascade model itself can be substituted for a similar, but a computationally more tractable model, the ALE model.

The Inverse Infection Problem

The definition of the Inverse Infection Problem [22] is very similar to the GC model:
Inverse Infection Problem: Given an unweighted graph G, the a priori and the a posteriori probability distributions p_v and p'_v, compute the edge infection probabilities w_e for all $e \in E(G)$.

Independently estimating all edge weights of a network is both underdetermined and computationally unfeasible, even if the number of edges is small. Instead, we assume the edge probabilities can be expressed as (normalized) functions of some properties of the edges or nodes that are available in the form of attributes. This way we only have to estimate the coefficients of these functions, and since the number of attributes and coefficients is limited, the problem becomes tractable.

Learning the edge infection probabilities

We can define a learning method based on the formulation above.

- The problem definition states, that the a posteriori distribution is required as an input of any algorithm. In the case of a learning algorithm, it is considered to be a test or reference dataset.
- Then the initial coefficients for the edge attribute functions are chosen from reasonable bounds.
- Given these attribute functions, the coefficients and the a priori distribution we can compute an a posteriori distribution corresponding to the chosen coefficients.

- Finally, an error function calculates the difference between the reference set and the newly calculated infection values. The process aims to minimize this error function by repeatedly adjusting the coefficients.

This is a typical task for global optimization, and after several tries we have selected the Fully Informed Particle Swarm method of Kennedy and Mendes [67].

Evaluation on artificial infection scenarios

We have used artificial infection scenarios to explore the boundaries of our method. We have tested the stability and accuracy of the optimization, we have given a general approach to choose the correct attribute functions, we have examined the implications of choosing between the heuristics of the GC model and we have tested our method in low-quality inputs as well. The learning method is able to accurately predict the edge infection probabilities in a small number of iterations while the number of attributes and the shape of the attribute functions have only a small effect on this even if the quality of the inputs is low.

Case-study on a bank transaction network

Finally, we have presented a case-study of the Inverse Infection Problem on a bank transaction network [16]. The goal of this application was to improve the efficiency of existing models for the prediction of short-time credit default events. Since the creation of IIP was heavily influenced by banking applications, it was quite suited to handle this task. Our model has better predictive power than traditional methods: it can identify the companies where default is most probable better than the previously used models of the bank. Our model was implemented in August 2013 into the OTP Bank of Hungary's credit monitoring process.

5.5 Future works

There are several ways the results if this dissertation can be extended. The most obvious way is to find new applications for the methods, or to improve the performance of them somehow. Let us close this work by examining each topic separately and by giving some new ideas.

We can give additional functionality to the hub-percolation method. The biggest inspiration to do this comes from the word association databases of László Kovács [74] and Nelson et al. [91]. The associations between words in these databases are naturally directed, but the direction is omitted in most community detection applications, because the intuition of communities is naturally undirected. Few directed community detection methods exist [80], but this field is not well studied. Combining the flexibility of hub-percolation with a directed detection mechanism may give an even more useful algorithm.

The association database of László Kovács also gives another application to the dynamic community detection method of chapter 3. The database logs a time stamp for each submitted association, allowing us to examine how word associations of the Hungarian language change in time. The construction of the database began in 2008, and since then several things have happened, that might have had an impact on the thinking of Hungarians: the economic crisis, the policy changes after the

2010 elections, etc. Our method should be able to identify most of these changes by matching the communities formed in different time periods.

The Inverse Infection Problem can be extended both by improving its estimation method, and by providing more applications. The learning framework in chapter 4.3 reduces the problem to global optimization. We were satisfied with the performance of the implemented FPSO method, but this does not mean there isn't an even better algorithm. Another approach would be to change the learning framework itself. While the idea of attribute functions makes our method useful for real-life applications, it might be worthwhile to try and estimate each edge weight separately without the aid of attributes. This way the dimensionality of the problem would be too large for anything but the smallest examples, but this would also shed light on the theoretical bounds of our method. Finally another application is apparent: the economic ownership network of chapter 2.3. The database itself is quite detailed, there is data available on bankruptcy events of the listed companies, therefore we could work out a case-study similar to the banking one.

Összefoglaló

A disszertáció célja, hogy bevezetést nyújtson a gráfelmélet és a hálózatkutatás tudományterületébe, és hogy összefoglalja a szerző munkásságát a közösségkeresés és a fertőzési modellek témaköreiben. A dolgozatot három összefüggő témakörre lehet bontani:

1. Nagy felbontású változtatható paraméterezésű átfedő közösségkereső algoritmus kifejlesztése.
2. Dinamikus közösségkereső algoritmus kifejlesztése, ami alkalmas nagy méretű valós hálózatok kezelésére.
3. Egy olyan módszer kifejlesztése, ami alkalmas élfertőzési valószínűségek becslésére banki tranzakciós hálózatokon.

Ezek alapján a disszertációt négy fejezetre lehet osztani: egy bevezető részre, ami történeti áttekintést ad a témáról és definiálja az alapvető fogalmakat, valamint a fenti pontoknak megfelelő fejezetekre.

Rögtön a bevezető után, a második fejezetben bemutattuk a hub perkolációs (Hub-percolation) átfedő közösségkereső algoritmust [18]. A módszer sajátossága, hogy képes sokrétű valós életből származó hálózatok kezelésére, figyelembe véve ezen alkalmazások változatos igényeit. Hogy demonstráljuk a módszer hasznosságát, Lancichinetti közösségalapú gráfgenerátora segítségével [78] összehasonlítottuk az algoritmusunk eredményeit a népszerű klikk perkolációs módszer (Clique percolation method) [100] eredményeivel. Ezen felül bemutattunk egy gazdasági és egy nyelvészeti esettanulmányt, melyek közül mindkettő kiváló eredményeket adott.

A disszertáció következő témája szorosan kapcsolódott az előzőhöz. Palla et al. munkásságát követve [99] létrehoztunk egy módszert, ami képes dinamikus közösségek azonosítására. Az általunk létrehozott módszer [20] új közösségi eseményeket vezet be, és működésének nem feltétele a statikus közösségkereső algoritmus monotonitása. Megviszgáluk és összehasonlítottuk két valós elétből származó hálózat dinamikáját, ez egyik gazdasági, a másik közösségi háló. Úgy találtuk, hogy a viselkedésük különböző.

A disszertáció utolsó és egyben legnagyobb fejezete fertőzési folyamatokkal foglalkozik azzal a céllal, hogy létrehozzon egy olyan módszert ami képes a pontok közötti befolyás mértékének becslésére. Ezt a mértéket élfertőzési valószínűségek jelzik, és a probléma formális definíciója, az inverz fertőzési probléma (Inverse Infection Problem) azt követeli meg, hogy egy súlyozatlan hálózat, a kezdeti fertőzések és a fertőzési folyamat eredményének ismeretében becsüljük meg ezeket az értékeket [22]. Az inverz fertőzési probléma az általánosított kaszkád (Generalized Cascade) fertőzési modellen [21] és heurisztikáin [19] alapul. A módszerünk használhatóságát bi-

zonyítottuk mesterségesen létrehozott fertőzési feladatokkal [22], valamint egy banki tranzakciós hálózatokkal kapcsolatos részletes esettanulmánnyal [16].

Alapfogalmak és definíciók

A disszertációban használt algoritmusok és módszerek gráfokon alapulnak. Jelölje G gráf pont- és élhalmazait $V(G)$ és $E(G)$. Majdnem minden esetben a gráfot irányítatlannak tekintjük. A gráf numerikus információt tartalmazhat élsúlyok, vagy él- és pontattribútumok formájában. A második fejezetben bevezetett hub-perkolációs módszer képes arra, hogy az élsúlyokat is figyelembe vegye a közösségek építésénél. A negyedik fejezet fertőzési modelljei 0 és 1 közötti élsúlyokat használnak bemenetként, az inverz fertőzési probléma becslései pedig ezeket a súlyokat számolják ki az él- és pontattribútumok segítségével. Minden esetben a gráfot összefüggőnek tekintjük[3]. A harmadik fejezet dinamikus közösségkereső módszere gráfok egy $\{G_i\}_{i \in \mathcal{T}}$ sorozatán alapul, ahol $\mathcal{T} = \{1, \ldots, \ell\}$ egy diszkrét időskálát ír le. Egy lépésben csak az idősor két szomszédos állapotát vizsgáljuk.

Az élfertőzési valószínűségeken kívül az inverz fertőzési probléma a gráf pontjain is értelmez 0 és 1 közötti fertőzési értékeket. A pontok egymástól függetlenül fertőződnek meg a folyamat kezdete előtt $p_v, v \in V(G)$ valószínűséggel. A $p'_v, v \in V(G)$ a posteriori fertőzési valószínűségek pedig azt jelölik, hogy a fertőzési folyamat során milyen valószínűséggel fertőződnek meg a pontok.

A hub perkolációs módszer

A második fejezetet a hub perkolációs közösségkereső módszer [18] ismertetésével kezdtük. Ez a módszer két fogalmon alapul: klikkeken és hub-okon. A klikkek adott k számú ponttal rendelkező teljesen összefüggő részgráfok. Egy klikk maximális, ha nem részhalmaza egyetlen más klikknek sem. A közösségeket összefogó pontokat huboknak nevezzük, ezek lokálisan fontos pontok. A hubok azonosítása a hub kiválasztási stratégia feladata, a második fejezetben javasoltunk néhányat beleértve egy olyat is, amely természetes módon képes az élsúlyok figyelembevételére. A stratégiáknak lehetnek paramétereik.

Leírás

A módszert a következő lépésekre lehet felosztani:

1. Keressük meg a 3-nál nagyobb maximális klikkek halmazát.

2. Válasszuk ki a hub-okat a hub kiválasztási stratégia szerint. Ennek a stratégiának lehet egy q paramétere.

3. Találjuk meg a hub-okból álló k-klikkeket, és terjesszük ki őket egy korlátozott perkolációs szabály szerint.

4. Olvasszuk össze azokat a közösségeket, melyekben ugyanazok a hub-ok szerepelnek.

[3]Ez lehet valamilyen szűrés eredménye, de ez nem tartozik a dolgozat témái közé.

A módszer működését kétféle módon befolyásolhatjuk. Az egyik a k szűrési paraméter, a másik a hub kiválasztási stratégia, és annak q paramétere. Ezek a paraméterek teszik lehetővé módszerünk sokrétű felhasználhatóságát. Azt, hogy a paraméterek milyen hatással vannak a megtalált közösségstruktúrára, sokféleképpen mértük: a közösségek számával, a közösségek méretének eloszlásával, a közösségek közti átlagos átfedéssel és a közösségen kívüli pontok számával. Erre a célra Newman benchmark hálózatait [47, 96] használtuk.

Kiértékelés benchmark hálózatokon

Lancichinetti közösség alapú gráf generátorát [78] használtuk arra, hogy összemérjük módszerünk eredményeit a népszerű klikk perkolációs módszer [100] eredményeivel. A mutual information [77] nevű mérőszámot használtuk arra, hogy összehasonlítsuk a gráfok természetes közösség struktúráját[4] az említett két módszer által találtakkal. Megmutattuk, hogy a módszerünk jobban közelíti a természetes közösségeket, amennyiben ezek között nagyok az átfedések.

Esettanulmány egy gazdasági tulajdonosi hálózaton

Módszerünk segítségével megvizsgáltuk a magyar céginformációs adatbázisból készített tulajdonosi hálózat közösség struktúráját, és a következő megfigyeléseket tettük [18]. A megfigyelt cégfajtára (Kft) jellemző, hogy cégeinek közösségei földrajzilag egymáshoz közel helyezkednek el, és jellemzően ugyanabban a gazdasági szektorban tevékenykednek. A cégek közösségeinek kora azonban szórást mutat.

Esettanulmány szó-asszociációs hálózatokon

Egy részletes esettanulmányban megvizsgáltuk és összehasonlítottuk egy magyar és egy angol szó-asszociációs hálózat közösség struktúráját [23]. Azonosítottuk azokat a szavakat, melyek a közösségek középpontjaiban állnak, ezek tipikusan kategórianevek, gyakori melléknevek vagy kollektív főnevek. A szavak egy része mindkét hálózatban szerepel, ezek jellemzően alapvető igényekhez vagy mindennapi eseményekhez kapcsolódnak, de nagyok a különbségek a két gráf között.

Dinamikus közösségek

A dolgozat következő témája a közösségek időbeni változásának vizsgálata volt. Az általunk használt megközelítés Palla et al. munkáján [99] alapszik, de ezt sok helyen módosítottuk, hogy használhatóbbá tegyük a módszert. Kibővítettük a közösség-események halmazát, mert úgy találtuk a Palla et al. által definiált eseményekkel nem lehet megfelelően leírni a tapasztalatainkat. Ezen felül átalakítottuk az összehasonlítást végző algoritmust, így az már nem függ a felhasznált közösségkereső különleges tulajdonságaitól.

Leírás

A módszerünk egy gráfsorozat két szomszédos G_1 és G_2 elemét hasonlítja össze a következő módon.

[4]Ezeket a gráf generátor adta meg.

1. Határozzuk meg G_1 és G_2 közösségeit egy közösségkereső segítségével.
2. Számoljuk ki G_1 és G_2 unióját G_U-t, majd futtassuk le a közösségkeresőt ezen is.
3. Rendeljük hozzá G_U közösségeit G_1 és G_2 közösségeihez.
4. A hozzárendelések összegzésével megkapjuk a G_1 és G_2 közötti eseményeket.

Esettanulmány egy gazdasági és egy közösségi hálón

A módszerünkkel két valós életből származó hálózat közösség struktúrájának dinamikáját vizsgáltuk. Az eredményeink azt mutatják, hogy lényeges különbség van a két hálózat viselkedése között. Az egyik hálózat stabil, de lassan és folyamatosan veszít a közösségeiből, a másik gyorsan változik, de megtartja közösségeinek számát.

Inverz fertőzés

A disszertáció utolsó nagyobb témája az inverz fertőzési probléma [22] és a hozzá kapcsolódó módszerek és algoritmusok. A modell kifejlesztését az OTP bankkal folytatott korábbi munkánk motiválta, ahol a vállalatok közötti befolyás mértékének a meghatározása hasznos lett volna. Ez különböző más modellek és algoritmusok kifejlesztéséhez vezetett, az inverz fertőzési problémát is beleértve. Az utolsó fejezetben megadtuk ezeknek a módszereknek a részletes leírását, kezdve az általánosított kaszkád modellel és heurisztikáival. A fejezetet egy gazdasági esettanulmánnyal zártuk.

Az általánosított kaszkád modell

Az általánosított kaszkád modell [21] a független kaszkád fertőzési modell általánosítása. Az inverz fertőzési probléma ezen a modellen alapul. Ebben a modellben minden ponthoz egy 0 és 1 közötti p_v érték kapcsolódik, mely azt a valószínűséget jelzi, amivel az adott pont még a folyamat kezdete előtt megfertőződik. Ezeket az értékeket a priori eloszlásnak nevezzük, és ezek képezik a modell bemenetét. Ehhez hasonlóan a modell kimenetét az a posteriori eloszlás adja meg, amely egy p'_v értéket definiál minden $v \in V(G)$ ponthoz. Ezek az értékek azt a valószínűséget jelzik, amivel az adott pont megfertőződik a folyamat során.

A következő módon definiálhatjuk az általánosított kaszkád modellt:

Az általánosított kaszkád modelll: Legyen adott egy megfelelően súlyozott G gráf és a p_v a priori fertőzési értékek, a modell ezek alapján kiszámolja az a posteriori eloszlás p'_v értékeit minden $v \in V(G)$-re.

Az általánosított kaszkád modell heurisztikái

Az általánosított kaszkád modell kiszámítása #P-nehéz, ezért négy heurisztikus módszert adtunk meg a kiszámítására [19].

- A teljes szimulációs módszer (Complete Simulation) Kempe et al. [65] módszerének átalakítása az általánosított kaszkád modell feltételeinek megfelelően.

- Az élszimulációs módszer (Edge Simulation) a szimuláció és az algebrai számítások olyan kombinációja, mely csökkenti a szimulációkra annyira jellemző szórás értékét.
- Ha az élfertőzési valószínűségek kicsik, akkor a fertőzések nem terjednek túl messze a kiindulópontjaiktól. A korlátozott szomszédság heurisztika (Neighborhood Bound Heuristics) ezt a jelenséget használja ki.
- A kaszkád fertőzési folyamatot le lehet cserélni egy könnyebben kiszámolható modellre, ez az ALE heurisztika.

Az inverz fertőzési probléma

Az inverz fertőzési probléma [22] definíciója hasonló az általánosított kaszkád modell definíciójához:

Az inverz fertőzési probléma: Legyen adott egy súlyozatlan G gráf, a p_v a priori és a p'_v a posteriori fertőzési értékek. Számoljuk ki ezekből a w_e élfertőzési valószínűségeket minden $e \in E(G)$-re

Minden egyes élvalószínűséget kiszámolni egyrészt nehéz, másrészt a probléma maga aluldefiniált, még akkor is, ha a hálózat nagyon kicsi. Ehelyett feltételezzük, hogy az éleken lévő értékek a pontokon és éleken levő más attribútumoknak egy (normalizált) függvényeként állnak elő. Így csak ezeknek a függvényeknek az együtthatóit kell kiszámolnunk, és mivel az attribútumok és együtthatók száma korlátozott, a probléma megoldhatóvá válik.

Az élfertőzési valószínűségek tanulása

A fentiek alapján definiálhatunk egy tanuló módszert.

- A definíció szerint az a posteriori eloszlás a probléma bemeneteihez tartozik. Ezt felhasználhatjuk egy tanuló módszer referencia adathalmazaként.
- Az attribútum függvények kezdeti együtthatóit véletlenszerűen választjuk ésszerű korlátok közül.
- Az attribútum függvények, az együtthatók és az a priori eloszlás segítségével kiszámolhatunk egy, az együtthatókhoz kapcsolható a posteriori eloszlást.
- Egy hibafüggvény segítségével meghatározzuk az eltérést a referencia eloszlás és az újonnan számolt a posteriori eloszlás között.

Ez a feladat a globális optimalizálás tipikus esete, a megoldás módszerének pedig néhány egyéb próbálkozás után Kennedy és Mendes FPSO algoritmusát [67] választottuk.

Kiértékelés mesterséges fertőzési feladatokkal

A módszerünk korlátait mesterséges fertőzési feladatokkal fedeztük fel. Megállapítottuk az optimalizálás stabilitását és pontosságát, egy általános megközelítési módot adtunk a megfelelő attribútum függvény kiválasztására, megvizsgáltuk milyen hatásai vannak a használt heurisztikáknak a módszer eredményeire, és kipróbáltuk hogyan viselkedik a módszerünk, ha a bemenetei hiányosak vagy hibásak. A módszer alkalmas arra, hogy kevés lépésszámmal is pontos becslést adjon az élfertőzési valószínűségekre, ezt az attribútumok száma és az attribútum függvények alakja csak kevéssé befolyásolja még pontatlan bemenetek esetében is.

Esettanulmány egy banki tranzakciós hálózaton

A dolgozat végén bemutattuk az inverz fertőzi probléma alkalmazását egy banki tranzakciós hálózaton [16]. Az alkalmazás célja a csődesemények előre jelezetőségének javítása volt. Mivel az inverz fertőzési probléma fejlesztésekor figyelembe vettünk banki alkalmazásokkal kapcsolatos követelményeket, így az esettanulmányt könnyű volt megvalósítani. A módszerünk jobb becsléseket nyújtott, mint a hagyományos módszerek, képes volt arra, hogy azonosítsa a csődre leginkább hajlamos cégeket, és 2013-ban hozzávették az OTP Bank eszközrendszeréhez.

További kutatási irányok

A disszertáció eredményeit többféle módon lehet továbbfejleszteni. A legnyilvánvalóbbak közé az új alkalmazások keresése és az algoritmusok hatékonyságának javítása tartozik. A dolgozatot azzal zárjuk, hogy megvizsgáljuk a tárgyalt témákat, és ötleteket adunk további munkákhoz.

A hub perkolációs módszer könnyen kibővíthető. Ehhez az ötletet Kovács László [74] és Nelson et al. [91] szó-asszociációs adatbázisai adhatják. Az asszociációknak természetes iránya van, ezt azonban a közösségkereső módszerek nem használják, mivel a közösségek elfogadott definíciója irányítatlan. Léteznek irányított közösségkereső módszerek [80], de ez a terület kevéssé kutatott. A hub perkolációs módszer rugalmasságát az irányított közösségek keresésére is ki lehetne terjeszteni, így a jelenleginél is használhatóbb módszert kaphatnánk.

A magyar nyelvű szó-asszociációs adatbázisra a dinamikus közösségkereső módszert is lehetséges alkalmazni. Az adatbázis minden egyes asszociációról tárolja a bevitel idejét, ezzel lehetővé teszi, hogy vizsgáljuk, hogyan változnak ezek az asszociációk az időben. Az adatbázis építése 2008-ban kezdődött, azóta rengeteg olyan dolog történt, amely befolyásolhatja a magyar emberek gondolkodásmódját: a világgazdasági válság, a 2010-es politikai váltás, stb. A módszerünk alkalmas lehet arra, hogy kövesse ezeket a változásokat úgy, hogy a szavakból álló közösségek időbeli változását figyeli.

Az inverz fertőzési problémát mindkét módon ki lehet terjeszteni: a tanuló módszer teljesítménye is javítható, valamint más alkalmazásai is lehetségesek. Az általunk vázolt megközelítés egy globális optimalizálási feladat. Elégedettek voltunk az implementált FPSO módszer teljesítményével, de ez nem jelenti azt, hogy nincs még egy ennél is jobb algoritmus. A tanuló módszert is megváltoztathatjuk. Az attribútum függvények ötlete teszi a módszerünket a gyakorlatban is felhasználhatóvá, megtehetjük azonban, hogy nagyon kis példákon megpróbálkozzunk külön-külön minden egyes élsúly becslésével. Ez segítene megérteni a módszerünk elméleti korlátait. Végül a közösségkereső módszernél ismertetett gazdasági hálózat is alkalmas lehet arra, hogy fertőzési folyamatokat vizsgáljunk rajta. A céginformációs adatbázis eléggé részletes, például csődeseményekről is tartalmaz információt, így a banki tranzakciókhoz hasonló esettanulmányt készíthetnénk.

Acknowledgments

I would like to thank my mentors, András Pluhár and Miklós Krész for their guidance, support and patience.

I would also like to thank my co-authors András Csernenszky and László Kovács for aiding me in my works, and my former tutors, Magda Várterész and Ágnes Achs for introducing me to research, and helping with the first steps.

The figures in the dissertation were made with Cytoscape and Sixtep, I would like to thank these companies for letting me use their software.

Bibliography

[1] B. Adamcsek, G. Palla, I. J. Farkas, I. Derényi, T. Vicsek, CFinder: Locating cliques and overlapping modules in biological networks. *Bioinformatics* **22**, 1021–1023 (2006).

[2] L. A. Adamic, R. M. Lukose, A. R. Puniyani, B. A. Huberman, Search in power-law networks. *Phys. Rev. E* **64**:046135, 2001.

[3] Y-Y. Ahn, J. P. Bagrow, S. Lehman, Link communities reveal multiscale complexity in networks. *Nature*, **466**(7307):761–764, 2010.

[4] W. Aiello, F. Chung, L. Lu, Random evolution of massive graphs. *Handbook of Massive Data*, 97–122, 2002.

[5] R. Albert, H. Jeong, A.-L. Barabási, Attack and error tolerance of complex networks. *Nature* **406**, 378–382, 2000.

[6] S. Asur, S. Parthasarathy and D. Ucar, An event-based framework for characterizing the evolutionary behavior of interaction graphs. *ACM Transactions on Knowledge Discovery from Data* **3**(4), 2009.

[7] A-L. Barabási, R. Albert, Emergence of Scaling in Random Networks. *Science*, **286**(5439):509–512,1999.

[8] E.R. Barnes, An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic and Discrete Methods*, **3**, 541–550, 1982.

[9] M. Bartha, M. Krész, A depth-first algorithm to reduce graphs in linear time, *Proceedings of the 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, IEEE Computer Society, 273–281, 2009.

[10] A. Björklund, Determinant sums for undirected Hamiltonicity. *Proc. 51st IEEE Symposium on Foundations of Computer Science*,173–182, 2010. doi:10.1109/FOCS.2010.24, ISBN 978-1-4244-8525-3.

[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte and E. Lefebre, Fast unfolding of community hierarchies in large networks. arXiv (2008): 0803.0476.

[12] B. Bollobás, *Modern Graph Theory,* Springer, New York (1998).

[13] B. Bollobás and O. Riordan, Clique percolation. *Random Structures Algorithms* **35** (2009), no. 3, 294–322.

[14] M. Boguña, R. Pastor-Satorras, A. Vespignani, Absence of epidemic threshold in scale-free net-works with degree correlations. *Phys. Rev. Lett.* Vol. **90** No 2. (2003) 028701-1–4 .

[15] A. Bóta, Propagation Algorithms for Protein Classification. *Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science*, 9–14, 2010.

[16] A. Bóta, A. Csernenszky, L. Győrffy, Gy. Kovács, M. Krész, A. Pluhár, Applications of the Inverse Infection Problem on bank transaction networks. Accepted for publication in: *Central European Journal of Operations Research.*

[17] A. Bóta, L. Csizmadia, A. Pluhár, Community detection and its use in Real Graphs. *Proceedings of the 2010 Mini-Conference on Applied Theoretical Computer Science*, 95–99, 2010.

[18] A. Bóta, M. Krész, A high resolution clique-based overlapping community detection algorithm for small-world networks. Submitted.

[19] A. Bóta, M. Krész and A. Pluhár, Approximations of the Generalized Cascade Model. *Acta Cybernetica* **21** 37–51, 2013.

[20] A. Bóta, M. Krész, A. Pluhár, Dynamic Communities and their Detection. *Acta Cybernetica* **20** 35–52, 2011.

[21] A. Bóta, M. Krész and A. Pluhár, Systematic learning of edge probabilities in the Domingos-Richardson model. *Int. J. Complex Systems in Science*, **1(2)** (2011) 115–118.

[22] A. Bóta, M. Krész and A. Pluhár, The inverse infection problem. *Proceedings of the 2014 Federated Conference on Computer Science and Information Systems*, 75–83, IEEE, 2014. http://dx.doi.org/10.15439/978-83-60810-58-3.

[23] A. Bóta, L. Kovács, The community structure of word association graphs. Accepted for publication in: *The Proceedings of the 9th International Conference on Applied Informatics*, Eger, 2014

[24] D. Braha, Y. Bar-Yam, Time-Dependent Complex Networks: Dynamic Centrality, Dynamic Motifs, and Cycles of Social Interaction, Adaptive Networks, chapter 3, 2009.

[25] C. Bron, J. Kerbosch, Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM* **16** (9): 575–577, 1973. doi:10.1145/362342.362367.

[26] T. Cao, X. Wu, T. X. Hu, S. Wang, Active Learning of Model Parameters for Influence Maximization. *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, Springer Berlin/Heidelberg, (2011) 280–295.

[27] F. Cazals, C. Karande, A note on the problem of reporting maximal cliques. *Theoretical Computer Science*, **407**(1):564–568, 2008.

[28] W. Chen, Y. Yuan, L. Zhang, Scalable Influence Maximization in Social Networks under the Linear Threshold Model. *Proceedings of the ICDM '10 Proceedings of the 2010 IEEE International Conference on Data Mining*, IEEE Computer Society (2010) 88–97.

[29] W. Chen, Y. Wang, S. Yang, Efficient influence maximization in social networks. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2009) 199–208.

[30] W. Chen, C. Wang, Y. Wang, Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2010) 1029–1038.

[31] J. Cheng, L. Zhu, Y. Ke, S. Chu, Fast algorithms for maximal clique enumeration with limited memory. *Proceedings of the 18th ACM SIGKDD*. ACM, New York, 1240–1248, 2012.

[32] N. Chiba, T. Nishizek (1985), Arboricity and subgraph listing algorithms, *SIAM J. on Computing* **14** (1): 210–223, 1985. doi:10.1137/0214017.

[33] J. E. Cohen, F. Briand, C. M. Newman, *Community food webs: Data and theory*, Springer, New York, 1990.

[34] A. Csernenszky, Gy. Kovács, M. Krész, A. Pluhár, T. Tóth, The use of infection models in accounting and crediting. *Challenges for Analysis of the Economy, the Businesses, and Social Progress* Szeged (2009) pp. 617–623..

[35] A. Decelle, F. Krzakla, C. Moore, L, Zdeborova, Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Phys. Rev. E*, **84**(6):066106, 2011.

[36] M. H. DeGroot Reaching a Consensus. *Journal of the American Statistical Association*, **69** (345): 118–21.

[37] O. Diekmann, J. A. P. Heesterbeek, Mathematical epidemiology of infectious diseases. Model Building, Analysis and Interpretation. *John Wiley & Sons,* 2000.

[38] P. Domingos, M. Richardson, Mining the Network Value of Costumers. *Proceedings of the 7th International Conference on Knowledge Discovery and Data Mining*, ACM (2001) 57–66.

[39] J. P. K. Doye, Network topology of a potential energy landscape: A static scale-free network. *Phys. Rev. Lett.* **88**:238701, 2002.

[40] J. Duch, A. Arenas, Community detection in complex networks using extremal optimization. *Phys. Rev. E* **72**:027104, 2005.

[41] D. Eppstein, D. Strash, Listing all maximal cliques in large sparse real-world graphs. *Experimental Algorithms*, Springer Berlin Heidelberg, 364–375, 2011.

[42] P. Erdős, A. Hajnal, On chromatic number of graphs and set-systems. *Acta Mathematica Hungarica* **17** (1–2): 61–99, 1966. doi:10.1007/BF02020444, MR 0193025.

[43] P. Erdős, A. Rényi, On Random Graphs. *Publicationes Mathematicae* **6**: 290–297, 1959.

[44] T. Evans, R. Lambiotte, Line Graphs, Link Partitions and Overlapping Communities. *Phys. Rev. E*, **80**(2):016105, 2009.

[45] S. Fortunato, Community detection in graphs. *Physics Report*, **486**(3):75–174, 2010.

[46] E. N. Gilbert, (1959), Random Graphs. *Annals of Mathematical Statistics* **30** (4): 1141–1144, 1959. doi:10.1214/aoms/1177706098.

[47] M. Girvan, M. E. J. Newman, Community structure in social and biological networks. *Proc. Natl. Acad. Sci.*, **99**(12):7821–7826, 2002.

[48] P. K. Gopalan, D. M. Blei, Efficient discovery of overlapping communities in massive networks. *PNAS*, **110**(36):14534-14539, 2013.

[49] A. Goyal, F. Bonchi, L.V.S. Lakshmanan, Learning influence probabilities in social networks. *Proceedings of the third ACM International Conference on Web search and data mining.* ACM (2010) 241–250.

[50] M. Granovetter, The Strength of Weak Ties. *American Journal of Sociology* **78(6)** 1360–1380, 1973.

[51] M. Granovetter, Threshold models of collective behavior. *American Journal of Sociology* **83(6)** 1420–1443, 1978.

[52] S. Gregory, Finding overlapping communities in networks by label propagation. *New J. Phys.*, **12**(10):103018, 2010.

[53] E. Griechisch, A. Pluhár, Community Detection by using the Extended Modularity. *Acta Cybernetica*, **10**:69–85, 2011.

[54] N. Guelzim, S. Bottani, P. Bourgine, F. Kepes, Topological and causal structure of the yeast transcriptional regulatory network. *Nature Genetics* **31**, 60–63, 2002.

[55] R. Guimera, L. A. N. Amaral, Functional cartography of complex metabolic networks, *Nature*, **433** (7028), 895-900, 2005.

[56] P. Hajnal, *Graph Theory* (In Hungarian: Gráfelmélet), Typotex, 2003.

[57] T. Hastie, R. Tibshirani, J. Friedman, Hierarchical clustering, *The Elements of Statistical Learning (2nd ed.)*, Springer, New York, 520–528, 2009.

[58] J. Hopcroft, O. Khan, B. Kulis, B. Selman, Tracking Evolving Communities in Large Linked Networks. *Proceedings of the National Academy of Sciences*, **101**, 5249–5253, 2004.

[59] B. A. Huberman, *The Laws of the Web*, MIT Press, Cambridge, MA (2001).

[60] M. O. Jackson, *Social and Economic Networks*, Princeton University Press, 2010.

[61] T. Jeffrey. S. Milgram, 1969. An Experimental Study of the Small World Problem. *Sociometry*, **32** (4), 425–443, 1969.

[62] H. Jeong, S. Mason, A.-L. Barabáasi, Z. N. Oltvai, Lethality and centrality in protein networks. *Nature* **411**, 41–42, 2001.

[63] R. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations*, Plemum Press, New York (1972), 85-104.

[64] D. Kempe, J. Kleinberg, E. Tardos, Maximizing the Spread of Influence though a Social Network. *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2003) 137–146.

[65] D. Kempe, J. Kleinberg, E. Tardos, Influential Nodes in a Diffusion Model for Social Networks. *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP)*, Springer-Verlag (2005) 1127–1138.

[66] J. Kennedy Particle Swarm Optimization. *Encyclopedia of Machine Learning*, Springer US (2010), 760–766.

[67] J. Kennedy, R. Mendes, Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews.* **36** (4) (2006) 515–519.

[68] W. O. Kermack, A. G. McKendrick, A Contribution to the Mathematical Theory of Epidemics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* **115** (772):700, 1927.

[69] B. W. Kernighan, S. Lin, An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal* **49**, 291–307, 1970.

[70] M. Kimura, K. Saito, Tractable models for information diffusion in social networks. *Knowledge Discovery in Databases*, Lecture Notes in Computer Science Springer Berlin / Heidelberg, (2006), 259–271.

[71] J. Kleinberg, An Impossibility Theorem for Clustering. *Advances in Neural Information Processing Systems (NIPS)* **15**, 2002.

[72] J. N. Kleinberg, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, The Web as a graph: Measurements, models and methods. *Proceedings of the International Conference on Combinatorics and Computing*, no. 1627 in Lecture Notes in Computer Science, 1–18, Springer, 1999.

[73] M. Krész and A. Pluhár, Economic Network Analysis based on Infection Models. To appear in *Encyclopedia of Social Network Analysis and Mining*, Springer (2014).

[74] L. Kovács, Conceptual Systems and Lexical Networks in the Mental Lexicon, (In Hungarian: Fogalmi rendszerek és lexikai hálózatok a mentális lexikonban) Tinta Könyvkiadó, Budapest, 2013.

[75] J. M. Kumpula, M. Kivela, K. Kaski, J. Saramaki, Sequential algorithm for fast clique percolation. *Phys. Rev. E*, **78**(2):026109, 2008.

[76] A. Lancichinetti, S. Fortunato, Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Phys. Rev. E*, **80**(1):016118, 2009.

[77] A. Lancichinetti, S. Fortunato, J. Kertész Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, **11**(3):033015, 2009.

[78] A. Lancichietti, F. Radicchi, J. J. Ramasco, S. Fortunato, Finding statistically significant communities in networks. *PLoS One*, **6**(4):e18961, 2011.

[79] C. Lee, F. Reed, A. McDaid, N. Hurley, Detecting highly overlapping community structure by greedy clique expansion. *Preprint*, 2010.

[80] S. Lehmann, M. Schwartz, L. K. Hansen, Biclique communities. *Phys. Rev. E* **78**:016108, 2008.

[81] J. Leskovec, J. Kleinberg, C. Faloutsos, Graphs over time: densification laws, shrinking diameters and possible explanations. *Proceedings of the1th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM (2005) 177–187.

[82] D. R. Lick, A. T. White(1970), k-degenerate graphs. *Canadian Journal of Mathematics* **22**: 1082–1096, 1970. doi:10.4153/CJM-1970-125-1.

[83] T. Luczak, Sparse random graphs with a given degree sequence. *Random graphs* (Poznań, 1989) Wiley, New York, 165–182, 1992.

[84] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Slooten, S. M. Dawson, The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, **54**(4):396-405, 2003.

[85] U. Von Luxburg, A tutorial on spectral clustering. *Statistics and computing* **17**(4), 395–416, 2006.

[86] J. B. MacQueen, Some Methods for classification and Analysis of Multivariate Observations. *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, 281–297, 1967.

[87] V. Maniezzo, R. Battiti, J-P Watson, On Effectively Finding Maximal Quasi-cliques in Graphs. In *Learning and Intelligent Optimization*, Lecture Notes in Computer Science (5313) 41–55, Springer Berlin Heidelberg, 2008.

[88] D. W. Matula, L. L. Beck, Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM* **30** (3): 417–427, 1983. doi:10.1145/2402.322385, MR 0709826.

[89] S. Milgram, The Small World Problem. *Psychology Today*, **1**(1), 60–67, 1967.

[90] J. Moon, L. Moser, On cliques in graphs. *Israel Journal of Mathematics*, **3**(1):23-28, 1965.

[91] D. L. Nelson, C. L. McEvoy, T. A. Schreiber, The university of south Florida word association, rhyme, and word fragment norms, http://w3.usf.edu/FreeAssociation/, 1998.

[92] T. Népusz, A. Petróczi, L. Négyessy, F. Bazsó, Fuzzy communities and the concept of bridgeness in complex networks. *Phys. Rev. E*, **77**(1):016107, 2008.

[93] M. E. J. Newman, Detecting community structure in networks. *Eur. Phys. J. B* **38**, 321–330 (2004).

[94] M. E. J. Newman, Finding community structure in networks using the eigenvectors of matrices, *Phys. Rev. E* **74**:036104, 2006.

[95] M. E. J. Newman, The structure and function of complex networks. *SIAM review*, SIAM, 2003.

[96] M. E. J. Newman, The structure of scientific collaboration networks. *PNAS*, **98**(2):404–409, 2001.

[97] M. E. J. Newman, M. Girvan, Finding and evaluating community structure in networks. *Phys. Rev. E*, **69**(2):026113, 2004.

[98] V. Nicosia, G. Mangioni, C. Carchiolo, M. Malgeri, Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, **3**:P03024, 2009.

[99] G. Palla, A.-L. Barabási and T. Vicsek, Quantifying social group evolution. *Nature* **446**, 664–667 (2007).

[100] G. Palla, I. Derényi, I. Farkas, T. Vicsek, Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, **435**(7043):814–818, 2005.

[101] C. H. Papadimitriou, K. Steiglitz, *Compinatorial Optimization: Algorithms and Complexity.* Prentice Hall, 1982.

[102] A. Pluhár On the clusters of social graphs based on telephone log-files. (in Hungarian) Research Report 2001.

[103] A. Pluhár Determination of communities in social graphs by fast algorithms. (in Hungarian) Research Report 2002.

[104] I. de S. Pool, M. Kochen, Contacts and influence, *Social Networks* **1**, 1–48, 1978.

[105] D. J. de S. Price, A general theory of bibliometric and other cumulative advantage processes. *J. Amer. Soc. Inform. Sci.* **27**, 292–306, 1976.

[106] S. Redner, How popular is your paper? An empirical study of the citation distribution. *Eur. Phys. J.* B **4**, 131–134, (1998).

[107] N. Robertson, D. Sanders, P. Seymour, R. Thomas (1996), Efficiently four-coloring planar graphs, *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, ACM Press, pp. 571–575, 1996.

[108] T. C. Schelling, Dynamic models of segregation. *The Journal of Mathematical Sociology*, **1**(2), 1971.

[109] E. Tomita, A. Tanaka, H. Takahashi, The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, **363**(1):28–42, 2006.

[110] G. Thilo, S. Hiroki, (Eds.) *Adaptive Networks.* Springer Verlag, New York, 2009.

[111] S. Tsukiyama, M. Ide, I. Ariyoshi, I. Shirakawa, A new algorithm for generating all the maximal independent sets. *SIAM Journal on Computing* **6** (3): 505–517, 1977. doi:10.1137/0206036.

[112] D. J. Watts, S. H. Strogatz, Collective dynamics of small-world networks *Nature*, **393**(6684):440–442, 1998.

[113] D. B. West, *Introduction to Graph Theory*, Prentice Hall, 2001.

[114] W. W. Zachary, An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 452–473, 1977.

[115] S. Zhang, R. S. Wang, X. S. Zhang, Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A*, **374**, 483, 2007.