Adaptive Decentralized Routing and Detection of Overlapping Communities

by

Oleg Bakun

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2011 by the
Graduate Supervisory Committee:

Goran Konjevod, Co-Chair
Andrea Richa, Co-Chair
Violet R. Syrotiuk
Andrzej Czygrinow

ARIZONA STATE UNIVERSITY

May 2011

ABSTRACT

This dissertation studies routing in small-world networks such as grids plus long-range edges and real networks. Kleinberg showed that geography-based greedy routing in a grid-based network takes an expected number of steps polylogarithmic in the network size, thus justifying empirical efficiency observed beginning with Milgram. A counterpart for the grid-based model is provided; it creates all edges deterministically and shows an asymptotically matching upper bound on the route length.

The main goal is to improve greedy routing through a decentralized machine learning process. Two considered methods are based on weighted majority and an algorithm of de Farias and Megiddo, both learning from feedback using ensembles of experts. Tests are run on both artificial and real networks, with decentralized spectral graph embedding supplying geometric information for real networks where it is not intrinsically available.

An important measure analyzed in this work is overpayment, the difference between the cost of the method and that of the shortest path. Adaptive routing overtakes greedy after about a hundred or fewer searches per node, consistently across different network sizes and types. Learning stabilizes, typically at overpayment of a third to a half of that by greedy. The problem is made more difficult by eliminating the knowledge of neighbors' locations or by introducing uncooperative nodes. Even under these conditions, the learned routes are usually better than the greedy routes.

The second part of the dissertation is related to the community structure of unannotated networks. A modularity-based algorithm of Newman is extended to work with overlapping communities (including considerably overlapping communities), where each node locally makes decisions to which potential communities it belongs. To measure quality of a cover of overlapping communities, a notion of a node contribution to modularity is introduced, and subsequently the notion of modularity is extended from partitions to covers.

The final part considers a problem of network anonymization, mostly by the means of edge deletion. The point of interest is utility preservation. It is shown that a concentration on the preservation of routing abilities might damage the preservation of community structure, and vice versa.

ACKNOWLEDGEMENTS

I want to thank Goran Konjevod for his supervision and collaboration, which made this work possible.

I also want to thank the committee for the provided feedback and suggestions.

This work was supported in part by NSF grant 0830791.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

We consider problems related to various aspects of man-made networks. Many of these networks are social. Their creation may be regulated by some rules, but is often controlled by actions from many people who act independently and in a decentralized manner. Quite often, the so-called small-world phenomenon (which assumes an abundance of short paths between many nodes) emerges in man made networks. The networks are often presented as graphs with possible additional annotation. For further aspects of social networks, see the monograph by Easley and Kleinberg [18].

We are particularly interested in the decentralized routing and detection of overlapping communities in man-made networks. We believe that there exists an interplay between these two problems. The adaptive routing that we present in this dissertation is fully decentralized and based on the ability of an entity (a person for example) to evaluate its immediate neighborhood and make a judgment on which of its neighbors is the most useful in forwarding a message to a particular target. When it comes to the community detection algorithm that we present in this dissertation, it is crucial that each node evaluates to what communities its neighbors belong, and by that, to what communities it itself belongs. It is important that people in real life can make these kinds of decisions independently, because social networks are formed by independent decentralized actors (although other stages of our community detection algorithm are not decentralized). In our study of networks, we found that there is a difference between two types of networks: networks that have hubs (high node degrees) and networks that are more regular (having similar node degrees). These two types exhibited different behavior in the learning curves demonstrated for the adaptive routing. The same two types of networks were affected to different magnitudes in the increase of the shortest path distance during a network anonymization procedure (another problem that we considered).

In this work, we study the structure and feasibility of routing in man-made networks. In both cases, it is extremely important for a node to realize its place in the network and has a sense of direction and distance between the nodes. This sense of distance is provided by using the same algorithm that embeds a network into a low-dimensional space. This algorithm exploits spectral properties of the transition matrix of a random walk on the graph (in particular, it relies on the eigenvectors and eigenvalues of this matrix to build the embedding). While talking about decentralized routing, we will consider the grid-based model that was introduced by Kleinberg [28] and for which a mechanism allowing expected small-world navigation was shown. This model classifies its edges as "local" and "long-range". We can consider local edges as those that form communities, and the long-range edges as bridges that connect communities. The edges that connect communities are extremely important in the creation of short paths between seemingly unrelated nodes. Should we draw a network (not necessarily grid-based) by using Koren's spectral graph drawing technique [32], we would see that nodes belonging to the same community are often placed close to each other and that edges that are internal to communities are often shorter than edges that connect communities.

We recently presented an abbreviated version of the routing chapter of this dissertation in [9]. Full details of the work on adaptive decentralized routing and detection of overlapping communities are also available in the technical reports [7] and [8] respectively.

1.1 Scope

This dissertation is organized in the following way. Chapter 2 reviews the work by Milgram [36] that describes the small-world phenomenon and the grid-based model by Kleinberg [28] that simulates this phenomenon and gives an algorithmic justification for it. We provide a deterministic counterpart for this model. Chapter 3 presents an adaptive decentralized algorithm that employs a learning technique to increase node's knowledge about its immediate neighbors with respect to navigation to different parts of a network. Experiments in which this algorithm outperforms the greedy algorithm are conducted. Chapter 4 defines the notion of a node contribution to the partition modularity and extends the notion of mod-

ularity from partitions to covers. An algorithm capable of discovering significantly overlapping communities is presented. Chapter 5 looks into network perturbations performed during network anonymization in order to preserve privacy. It points out how this leads to reduction of the usability of anonymized networks. Chapter 6 provides the conclusion and present directions of the future work.

Chapter 2 is written around the grid-based model introduced by Kleinberg [28]. In this model, each node has a known location in the two-dimensional space. Local edges of the nodes are created deterministically between pairs of closely located nodes, but the long-range edges are created probabilistically according to an inversely proportional distribution. The expected path length in the greedy routing has a polylogarithmic upper bound. We modify the long-range edge generation to be deterministic. The upper bound on the path length remains the same, but becomes guaranteed for any source-target pair of nodes (there is nothing probabilistic in the model anymore). The deterministic model allows shorter average long-range edges relatively to the expected length of the long-range edges in the probabilistic model.

Chapter 3 covers the subject of adaptive decentralized routing. We start with the greedy algorithm, but improve results by considering the edges as local experts. In grid-based networks, each node knows its location; real world networks can be embedded into a low-dimensional space. We embed them into two-dimensional space by using Koren's graph drawing algorithm [32]. Each node learns about its network by developing a sense of the direction and distance to other nodes (or regions to which these nodes belong). It allows the node to partition the space into different regions (independently of the partitions created be other nodes) and correlate search targets with these regions.

We follow an idea of Awerbuch and Kleinberg [4] to consider an outgoing edge as an expert always advising to use its edge. We use these experts by employing machine learning techniques (either the Multiplicative Weights Update Method by Arora, Hazan, and Kale [3] or the Exploration-Exploitation Experts Method by de Farias and Megiddo [15]).

Even when 10% of the nodes are "lazy" (partially cooperative), the rest eventually learns how to avoid the lazy nodes and produce better results than greedy. We had partial success in another adverse environment where all nodes are "blind" (not aware about locations of their neighbors), and thus do not have any initial guess about values of their experts.

Chapter 4 deals with the problem of community detection in networks. While there are many algorithms dedicated to the problem, only few of them consider overlapping communities. Often it is assumed that an overlapping community has a core consisting of nodes exclusively belonging to the community and a small fringe that consist of nodes of the community that are shared with other communities. Our goal is to allow discovering communities in which most of the nodes belong to multiple communities.

While there is no agreed single measure that evaluates quality of a community structure, a notion of modularity suggested by Newman and Girvan [41] seems to be the most popular when one considers partitions (a collection of communities in which each node belongs to exactly one community). We consider a contribution that a node makes to the modularity of a partition. This allows to extended the notion of modularity to covers (a collection of communities in which a node may belong to many communities).

Newman [40] suggested an algorithm that attempts to maximize the modularity. We add a parameter to this algorithm that allows splitting a community into smaller sub-communities, which is especially important for the case of overlapping communities, because two or more communities that significantly overlap, or are just well connected between themselves, can be mistakenly considered as a single community. We follow an idea by Gregory [26] to replace an original node by several clones and thus allow the node to participate in several communities. We introduce an assumption that the node is capable of forming its own ideas to what communities its neighbors belong. The overall communities of the network are a result of the impressions about the network that are made by each node.

The absence of real-world data (that would include ground truth communities) forces us to introduce a model that constructs artificial networks with overlapping commu-

nities. We show that our algorithm can discover the ground truth communities with a good precision even when those communities are strongly connected or significantly overlap.

In Chapter 5, we look into usability preservation issues related to the identity protection during the process of network anonymization. A company might want to release data related to its clients that can be represented as a graph. In order to protect clients' identity, the data might be anonymized. That is, the names of nodes (and edges) are replaced by meaningless identifiers. There are no attributes associated with the nodes or the edges. However, that might be not enough because an adversary might know some structural data about its target (the target's degree for example or some additional information). In this case, a perturbation of the network's topology is required as well. We will concentrate on a particular way of perturbation — edge deletion.

In order to preserve routing ability of the perturbed network one might want to preserve "odd" edges that connect otherwise distant parts of the network. We define the stretch of an edge as the length of a shortest backup path connecting the edge's endpoints. In this case, edges with a high stretch (that keep many nodes close to each other) should be preserved and edges with a low stretch deleted. However, this makes the community structure of the network less obvious. The opposite action, of deleting high stretch edges, leads to the preservation of the community structure but affects routing abilities. In general, it seems that anonymity preservation demands loss of network usability, because unusualness can be a reason for an anonymity breach.

1.2 Major related work

Here we will mention the most important previous work that influenced this dissertation. More complete information about the previous work can be found in the corresponding chapters. The book by Easley and Kleinberg [18] covers different aspects of social networks. It considers ways in which information spreads in social networks and what is the interplay with the network structure. It studies small-world phenomenon and routing in small-world networks. It is interested in the dynamics of the network development, with

special emphasis on social aspects of man-made networks. It also studies many macroeconomic problems considered form the point-of-view of the game theory.

The paper by Kleinberg [28] that introduced the grid-based model simulating the small-world phenomenon is the most influential single source that led to this dissertation. This model demonstrated the necessity of at least an approximate knowledge of the underlying space in which the network lies (not all dimensions of the space have to be based on the geography). It also showed the necessity to have better information about the immediate surroundings of the target. The grid-based model was later studied by many scientists. Kleinberg also wrote a survey [29] related to the construction of and to searching in small-world networks.

The necessity to have a good understanding of the surroundings of a node and its general place in the network required a good and fast algorithm that provides with this information. That was done by Koren [32] who developed an algorithm for the graph drawing purposes. We used this algorithm to embed a network into a low-dimensional space to help with routing and community detection problems that are considered in this dissertation.

This work also uses concepts of machine learning and game theory. Different variants of the weighted majority algorithm are widely used across different fields. A survey of them was published by Arora, Hazan, and Kale [3]. We used a variant of this famous algorithm but achieved better results with a variant of the method designed to induce cooperation in a setting from the game theory. This method was designed by de Farias and Megiddo [15]; it requires evaluation of only one advice per stage and fits our setting much better. Finally, the idea to consider an edge as a constant expert that gives an advice "choose me" on all the stages of the game comes from Awerbuch and Kleinberg [4].

We adapt an algorithm devised by Newman [40] that targets to maximize the modularity (a measure that is used by many to evaluate the quality of a partition) and extends it to a cover. We use an idea of Gregory [26] that a node might be replaced by its clones and this will allow creation of overlapping communities by the means of an algorithm that normally

detects non-overlapping communities. A survey dedicated to the community detection was published recently by Fortunato [20].

A paper by Backstrom et al. [5] showed that active attacks (where an adversary plants several nodes into the network prior to the anonymization) could be successful. The adversary can find the planted nodes in the released anonymized network by knowing basically just the degrees of the planted nodes. A survey by Zhou et al. [51] considers ways to protect the user anonymity by means of perturbation of the network topology; it also deals in part with the usability preservation, which is also a subject of this dissertation.

Chapter 2

GRID-BASED MODELS

2.1 Introduction

In this chapter,[1] we review the grid-based model that was introduced by Kleinberg [28] and is extensively used to simulate the small-world phenomenon. This phenomenon was described by Milgram in nineteen sixties (see Section 2.1.1) and it exposed the fact that many people are connected to each other by short chains of common acquaintances and that people can discover these chains locally and without using an exhaustive search.

The grid-based model by Kleinberg (see Section 2.2) creates local edges (between nodes that are close to each other in the underlying space) deterministically. They provide connectivity of the network but cannot connect nodes that are far away from each other in the underlying space by short paths. Edges of another type (so-called long-range edges) are created probabilistically and they connect nodes that can be far away and create the small-world effect. These edges are created according to a probability distribution that depends on the dimension of the underlying grid.

We created a deterministic counterpart of the grid-based model (see Section 2.3). The long-range edges are here also created deterministically. Each edge corresponds to exactly one dimension of the underlying grid and their creation does not depend on the dimension of the grid.

2.1.1 The small-world phenomenon in social networks

The psychologist Milgram [36] conducted a series of experiments in the nineteen sixties that exhibited counterintuitive results that collectively were named the "small-world phenomenon". In these experiments, a person (*source*) was given a name, occupation, and location of another person (*target*) with whom the source was not in general acquainted. Some additional information, such as the family status and age of the target might be provided as well. Both the source and the target were located in the U.S., but lived in different states. A current message holder (initially the source) had to forward a message (together

[1] A preliminary version of this chapter was a part of the author's M.S. Thesis [6].

with all the known information about the target) to the target, but one could not contact the target directly (unless one knew the target personally). Every participant in these experiments could contact only a person who was an acquaintance on a first-name basis. The objective was to form a path from the source to the target involving as few intermediaries as possible. Importantly, the current message holder could not survey part or all of one's acquaintances prior to the message forwarding. Therefore, the current message holder had to choose only one acquaintance and forward all the information to this person with a request to participate in the experiment. The message holder had to choose an acquaintance who had a better chance of having a short path to the target and was likely to participate in the experiment.

The rate of successful experiments was quite low — 20 to 25 percent. However, the average path length was extremely short — normally only 5 to 6 people sufficed for the message forwarding. The observed occurrences of the existence of the short paths between randomly selected people were termed the small-world phenomenon. A few aspects of the phenomenon have been studied by psychologists and sociologists: the high number of acquaintances that an average person has, the person's ability to choose the right acquaintance to forward the message to (usually this choice was based on geography and occupation), the rate of refusal to participate in the experiment and so on.

Numerous mathematical models were suggested to explain this phenomenon. Many of them share some common restrictions:

- The number of contacts actually used by each participant is limited — a participant cannot forward a message to all acquaintances (and thus overload the network).
- A participant does not know the exact global structure of the network.
- A participant has some inexact way to estimate the likelihood that an acquaintance has a short path to the target.

2.1.2 *Related work*

Newman writes in the survey [39] about first attempts to explain the small-world phenomenon. The high degree of the nodes in the graph (the high number of acquaintances in a social network) would suggest a short diameter in the graph representing the social network. However, it does not explain the ability of the person to navigate the global graph while knowing only about their immediate surroundings. It also was noticed that the real world social networks exhibited a clustering effect: a friend of my friend is likely to be my friend (high rate of cliques in subgraphs of the global graph).

Watts and Strogatz introduced a one-dimensional grid model [49]. All nodes are spread out uniformly over a circle. Each node is connected to a fixed number of its closest neighbors (the same number of neighbors in the clockwise and counterclockwise directions) also known as local contacts. In addition to this, some other nodes (long-range contacts) can be randomly chosen to be connected. Under certain conditions, a greedy algorithm will find with high probability a path from the source to the target in a polylogarithmic number of steps. The ability to find a path between a pair of randomly selected nodes in a polylogarithmic number of steps is considered to be the primary characteristic of the small-world phenomenon.

The work of Watts and Strogatz can be considered as a precursor for the grid-based model that was described by Kleinberg [28] and which we review in Section 2.2. This model gave a push to the research in this area. We want to note a work by Fraigniaud et al. [21] that gave an augmentation scheme for an arbitrary graph that leads to an upper bound of $\widetilde{O}(n^{1/3})$ on the diameter of the graph. A survey by Kleinberg [29] looks into the decentralized search, models that simulate the small-world phenomenon, and problems of graph reconstruction.

2.2 Grid model by Kleinberg

2.2.1 Probabilistic grid model

Kleinberg [28] suggested a model supporting the small-world phenomenon that has become a subject of study by many researchers. The underlying grid can be considered as a square-shaped lattice, where nodes are located at intersections of n columns and n rows. Throughout this chapter, when we talk about a distance, we talk about the lattice distance, unless explicitly stated otherwise. Thus, the grid can be described by the following parameters:

- The number of rows and columns in the grid, denoted by n. There is a node u at any location (i, j), where i and j are integers from the range $[1, n]$. There are no other nodes.
- A constant $p \geq 1$, that is, the local contact threshold. For any node u, there is a directed edge from u to every node v such that the lattice distance from u to v is no more than p. v is called a *local contact* of u.
- A constant r, that is, an exponent of the inverse r-th power distribution that defines the probability of one node to become a long-range contact of another node. The lattice length of a long-range contact is bounded by $2n$ because of the size of the grid. That is, the chance that node v becomes a *long-range contact* of u is proportional to $1/d(u,v)^r$, where $d(u,v)$ is the lattice distance between u and v.
- A constant q, that is, the number of long-range contacts per node. The q contacts are generated independently of each other.

Figure 2.1 and Figure 2.2 are taken from the Kleinberg's paper [28]. In Figure 2.1, we see local contacts only, in a network with p equal to 1. In Figure 2.2, we see all contacts of node u: four local contacts (p is equal to 1) and two long-range contacts (q is equal to 2). The local contacts form a regular underlying grid, which makes the graph connected.

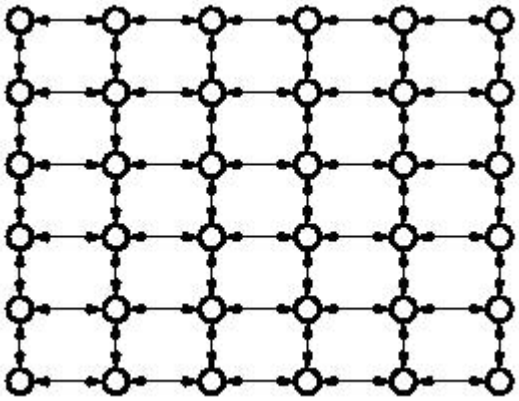

Figure 2.1: Local contacts (from [28]).

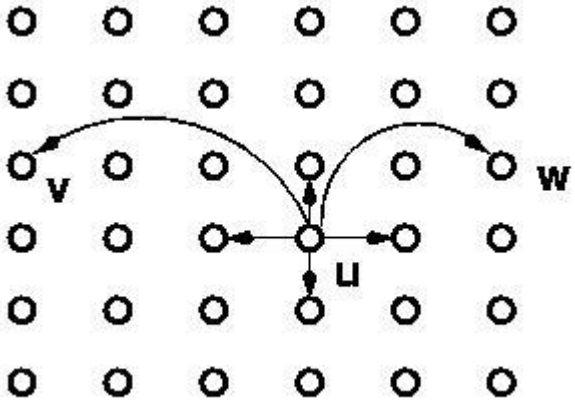

Figure 2.2: All contacts of a single node (from [28]).

The long-range contacts give an opportunity for a pair of remote nodes to be connected by a short path. At each step, the current message holder makes a choice based on the greedy algorithm. The next message holder is chosen based on one criterion only — it should be the neighbor closest to the target (from the point of view of the underlying geography). It is not beneficial to the current message holder to know the choices that were made by its predecessors, because the algorithm is greedy and the path will never cycle, due to the fact that the current message holder always has a local contact that is geographically closer to the target than the current message holder is. All this makes the algorithm decentralized.

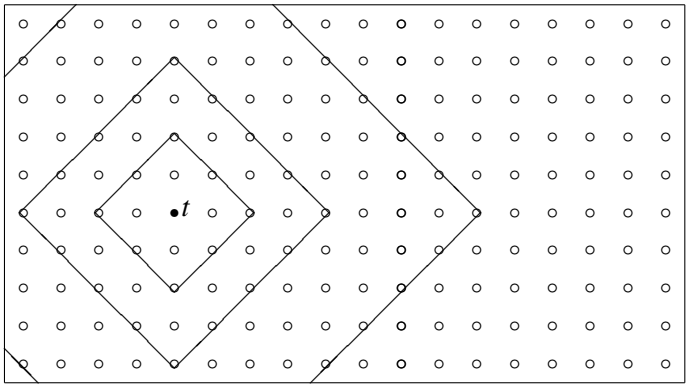

Figure 2.3: Areas of the grid defined with respect to the target t.

2.2.2 Overview of Kleinberg's results

Kleinberg proved that the parameters p and q (as long as they are constants and greater than 0) do not affect whether the small-world phenomenon will be observed or not. The only parameter that is important in this matter is r. The small-world phenomenon is expected to be observed if and only if r is equal to 2. In all other cases of r, the expected path length has an expected polynomial lower bound.

Lemma 1. *(Positive results from [28].) When r is equal to 2, the expected delivery time by the greedy algorithm is* $O(\log^2 n)$.

The grid is divided into the logarithmic number of *areas* with respect to the target t. The nodes at the (lattice) distance no more than 2 from the target form area 0. Area j (with $j > 0$) is formed by the nodes at distance d from the target, where $2^j < d \leq 2^{j+1}$. See an example depicting a part of the grid and the boundaries of the areas defined with respect to the target t in Figure 2.3. When the current message holder is in area j, then the process is in *phase* j. There are $\lceil \log n + 1 \rceil$ phases. The greedy process starts at the phase defined by the location of the source node and advances towards smaller-index phases. Kleinberg showed that the expected number of steps necessary to leave the phase j is $O(\ln n)$. Thus, overall expected time to reach the target is $O(\log^2 n)$.

Lemma 2. *(Negative results from [28].) When $0 \leq r < 2$, the expected delivery time by the greedy algorithm is $\Omega(n^{(2-r)/3})$.*

In this case, the long-range contacts are chosen almost uniformly at random, the corresponding edges are too long and become useless when the message is close to the target, but not so close that the routing can be done in the sub-polynomial time by using local contacts only. There will be many cases when the source and the target are at such distance. The long long-range edges often would have taken the message away from the target in this situation and are useless. However r is small, and there are too few long-range edges that are relatively short. Therefore, it is difficult to bump into a node at such distance from the target, which has a long-range contact that is closer to the target.

Lemma 3. *(Negative results from [28].) When r is greater than 2, the expected delivery time by the greedy algorithm is $\Omega(n^{(r-2)/(r-1)})$.*

In this case, the difficulties arise in the initial (high index) phases. Too many of the long-range edges are relatively short. If the source is far enough from the target, although many of the long-range edges are useful, they are too short to bring the message to the target in the sub-polynomial time.

Kleinberg also suggested that the exponent r should match the dimension of the grid in order to achieve the small-world phenomenon for networks of higher dimensions.

2.3 Deterministic generation of long-range contacts

In the grid model described by Kleinberg [28], the long-range contacts are chosen at random according to the inverse power distribution. In this chapter, we describe a model in which the long-range contacts are assigned deterministically. We will see that it is possible to create a small-world network deterministically. Because the network is deterministic, it has guaranteed (and not expected) poly-logarithmic delivery time.

2.3.1 Network setup

Let us consider the one-dimensional case of the Kleinberg model. We will refer to such networks as *chains*. Later we will go back to higher dimensions. We make the following assumptions:

- n, that is, the number of nodes in the chain is a power of 2; $n = 2^k$. This assumption will be abolished in Section 2.3.2.1.
- p, that is, the local contact threshold is equal to 1. Thus, in the one-dimensional case, two "dead end" nodes have one local contact and all the other nodes have two.
- q, that is, the number of long-range contacts per node is equal to 1. The length of the long-range edges is always a power of 2. The length is limited by the diameter of the space. The overall distribution of lengths of the long-range edges follows an inversely proportional distribution. Note that a long-range contact may (and often will) coincide with a local contact.

This setting can be represented by the following graph $G = (V, E)$. G is considered to be laid out in a chain. Any references to directions are references to the directions in the chain. $V = \{v_1, v_2, \cdots, v_n\}$. $|V| = n = 2^k$. We will say that an edge is *right-oriented* if the index of its tail is smaller than the index of its head. Similarly, we will that say an edge is *left-oriented* if the index of its tail is greater than the index of its head. We index each edge by two indices, the first index is equal to the index of the tail of the edge, and the second index expresses a type of the edge. There are four types of edges: the first type is for the right-oriented local edges, the second type is for the left-oriented local edges, the third type is for the right-oriented long-range edges, the fourth type is for the left-oriented long-range edges.

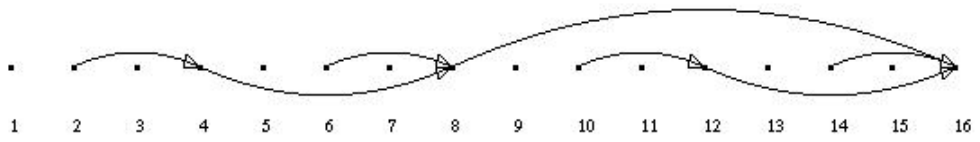

Figure 2.4: G_{16}. Right-oriented long-range edges.

There are $2n-2$ directed edges representing local contacts. For each i from 1 to $n-1$, there is a right-oriented local directed edge $e_{i,1} = (v_i, v_{i+1})$. For each i from 2 to n, there is a left-oriented local directed edge $e_{i,2} = (v_i, v_{i-1})$.

Edges that represent long-range contacts of length 1 coincide with edges representing local contacts; a set of coinciding edges that share the same head and the same tail are merged into one edge. Only edges connecting long-range contacts at distance longer than 1 are added. There is one right-oriented edge of length $n/2$: $e_{n/2,3} = (v_{n/2}, v_n)$. This is the only right-oriented long-range edge that connects the left and right halves of the chain. There are two right-oriented edges of length $n/4$: $e_{n/4,3} = (v_{n/4}, v_{n/2})$ and $e_{3n/4,3} = (v_{3n/4}, v_n)$. The former is the only right-oriented long-range edge that connects the first and second quarters of the chain, and the latter is the only right-oriented long-range edge that connects the third and fourth quarters of the chain. And so forth, until $n/4$ right-oriented edges of length 2 are created. As an example, let G_{16} be the chain on 16 nodes. See Figure 2.4 for the all right-oriented edges no shorter than 2. See Figure 2.5 for the pseudo code generating the long-range edges.

Left-oriented long-range edges are defined in a similar way, but their tails are shifted one unit to the right from the tails of their right-oriented counterparts. So, the longest left-oriented edge is $e_{n/2+1,4} = (v_{n/2+1}, v_1)$ and so on. See the six longest edges of G_{16} in Figure 2.6. The right-oriented edges are $e_{8,3} = (v_8, v_{16})$, $e_{4,3} = (v_4, v_8)$ and $e_{12,3} = (v_{12}, v_{16})$; they are shown in the upper half of the figure. The left-oriented edges are $e_{9,4} = (v_9, v_1)$, $e_{13,4} = (v_{13}, v_9)$ and $e_{5,4} = (v_5, v_1)$; they are shown in the lower half of the figure.

```
// activate the generation of the long-range edges by
// LongRangeEdges(1,n) call
void LongRangeEdges( int LeftEnd, int RightEnd )
{
    if( RightEnd − LeftEnd ≤ 2 )
    {
        return;
    }
    else
    {
        Middle = ⌊(LeftEnd + RightEnd)/2⌋;
        // create right-oriented long-range edge e_{Middle,3}
        if(RightEnd − Middle ≥ 2)
            CreateEdge( Middle, RightEnd );
        // create left-oriented long-range edge e_{Middle+1,4}
        if(Middle + 1 − LeftEnd ≥ 2)
            CreateEdge( Middle + 1, LeftEnd );
        LongRangeEdges( LeftEnd, Middle );
        LongRangeEdges( Middle + 1, RightEnd );
        return;
    }
}
```

Figure 2.5: Generating long-range edges.

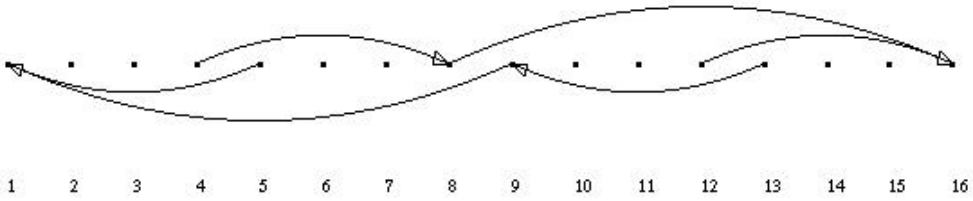

Figure 2.6: G_{16}. Long long-range edges (no shorter than 4).

If we remove all four edges that connect the left and the right halves of G_{16} (see Figure 2.7), we will receive two isomorphic subgraphs. These subgraphs are also isomorphic to graph G_8 — the chain on 8 nodes. This is an important property of the suggested generation model. Graph G_n includes 2 non-intersecting subgraphs isomorphic to graph $G_{n/2}$; includes 4 non-intersecting subgraphs isomorphic to graph $G_{n/4}$; includes 8 non-intersecting subgraphs isomorphic to graph $G_{n/8}$; and so on. This will allow splitting the search algorithm into $\log_2 n$ phases, where every following phase is a replica of the previous phase on a smaller scale.

2.3.2 Routing algorithm

As an input, in addition to the chain described above we will receive the locations of the source — node s, and of the target — node t. We will use a greedy algorithm: if we are currently at the location u, then we choose an outgoing edge $e = (u, v)$ that brings us as close to the target t as possible, (we use the lattice distance to t in order to estimate the shortest path distance to the target and as the greedy criterion). In the case of a tie, the longest edge among the best candidates is chosen. The number of the outgoing edges per any node cannot exceed 3: the right-oriented local edge, the left-oriented local edge, and no more than one long-range edge (either right-oriented or left-oriented; see the creation of outgoing edges of the nodes $Middle$ and $Middle + 1$ in Figure 2.5). At any point of the process, no more than two edges can be considered: these will be right-oriented if the target t is to the right of the current message holder, and left-oriented otherwise.

Let us define phase one as the part of the algorithm that brings the current location u for the first time to the half of the chain to which the target belongs. Without loss of generality, we can suggest that the target t is located in the right half of the chain, that is, the target t belongs to $\{v_{n/2+1}, v_{n/2+2}, \cdots, v_n\}$. How long will it take to finish phase one? If the source s happened to be on the same half as the target t, then phase one ends immediately. Otherwise, this will happen only if at least one of the two edges $e_{n/2,1} = (v_{n/2}, v_{n/2+1})$ or $e_{n/2,3} = (v_{n/2}, v_n)$ is used, since those are the only edges that start in the left half of the chain

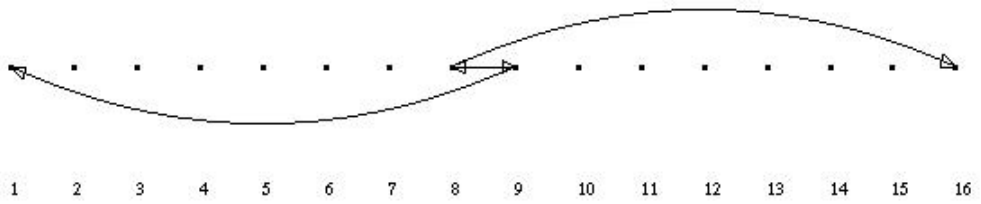

Figure 2.7: All edges connecting the left and the right halves of G_{16}.

and finish in the right half. See Figure 2.7 for all edges connecting the left and the right halves of G_{16}.

Is it possible that the current location u will ever move from the right half to the left half and thus use one of the only two edges that can enable this: $e_{n/2+1,2} = (v_{n/2+1}, v_{n/2})$ or $e_{n/2+1,4} = (v_{n/2+1}, v_1)$? In both of the cases, the message should be located at $v_{n/2+1}$ — the leftmost node of the right half of the chain. However, the target t is located somewhere else in the right half of the chain and thus to the right of u. Choosing $e_{n/2+1,2}$ or $e_{n/2+1,4}$ means going away from the target t, and that contradicts the greedy nature of the algorithm. We infer that the algorithm will never move from the half of the chain containing the target to the other half of the chain.

By completing phase one, we achieve an important goal, we have the initial problem but on a smaller scale: the number of the nodes has decreased by a factor of two and we will never deal with the other half of the nodes again. In the phase two, we need to bring the current location u to the (third or fourth) quarter, which includes the target t. Continuing in the same manner after $\log_2 n$ phases, we will end up at distance no more that two from the target t and finish the process. If we prove that the first phase takes no more than $\log_2 n$ steps, then the overall process will not take more than $\log_2^2 n$ steps, which asymptotically matches the order of the upper bound on the performance of the greedy algorithm over a grid with randomly generated long-range edges described by Kleinberg [28].

Let us evaluate the number of steps that are necessary to finish the first phase. The target node t is in the right half and the source node s is in the first or second quarter.

The only right-oriented edges connecting the left half of the chain to the right half of the chain are $e_{n/2,1} = (v_{n/2}, v_{n/2+1})$ and $e_{n/2,3} = (v_{n/2}, v_n)$, see Figure 2.7. Thus, in both cases $v_{n/2}$ is visited at one step before the last step of the first phase. The algorithm uses only right-oriented edges during the first phase, due to its greedy nature. Importantly, by the construction of the chain, if two right-oriented long-range edges overlap, then one of them "covers" the other edge entirely. That is, if $e_1 = (v_i, v_j)$ and $e_2 = (v_k, v_l)$ are two right-oriented long-range edges and $i < k < j$, then $l \leq j$. That means that if the source s is in the first quarter, then at some point the algorithm will visit $v_{n/4}$ and will skip the whole second quarter in one step by using $e_{n/4,3} = (v_{n/4}, v_{n/2})$, because $e_{n/4,3}$ is the best outgoing edge of $v_{n/4}$ when the target t is in the right half of the chain. The maximum lattice distance that should be covered before reaching $v_{n/2}$ is $n/2-1$, and if the source s is at lattice distance more than $n/4$ from $v_{n/2}$, then the distance will shorten by at least the half in one step. Similarly, the shorter long-ranges edges will be reducing the initial part of the path, and the whole phase cannot take more than $\log_2 n + 1$ steps.

It was shown that the first phase takes $O(\log_2 n)$ steps. Any following phase takes $O(\log_2 n)$ as well, because it is a replica of the first phase on a smaller scale. The total number of the phases is $\log_2 n$. Thus, the overall time is $O(\log_2^2 n)$.

2.3.2.1 Arbitrary number of nodes

If n is not a power of 2 then the routing algorithm remains unchanged. The algorithm generating the long-range edges is also essentially the same. We still need to choose a pair of neighboring nodes in the middle of the chain that will be tails of all four edges connecting the left and the right halves of the chain. We need to continue the same process on the smaller scale until the long-range edges are two units long. This setup ensures that once the algorithm exits a routing phase, it will never return there. This also ensures that if the algorithm is making several consecutive steps in the same direction then all long-range edges (that are part of this stretch) will be included in the path, unless they are covered by longer edges included in the path. The lengths of the edges are not powers of two anymore, but they still form a hierarchy and the length of the long-range edge of the next higher level

is roughly twice the length of the long-range edge from the lower level. The pseudo code generating the long-range edges still works for the chains with an arbitrary number of nodes (see Figure 2.5).

2.3.3 Higher dimensions

Let us consider the two-dimensional $n \times n$ grid. Note that n is the number of rows (columns) in the grid. Only for the one-dimensional case, n is also the number of nodes in the network. We can naturally define two *basic directions* for each dimension: left and right (for the horizontal dimension), up and down (for the vertical dimension). The local contacts for the two-dimensional grid are created the same way it was done for the chain, their maximum number per node increases to four. The number of long-range edges per node (q) is increased to two to allow a node to have an edge for each of the two dimensions. The generation of the long-range edges requires some changes. In the deterministic model we limit the long-range edges to four types; each type will be aligned (parallel and having the same direction) with only one of the four basic directions. That is, either row or column indices of the head and the tail of each edge coincide. Only one or two types of the long-range contacts can be useful at each step (which is completely determined by how the current location u is positioned relatively to the target t).

As in the case of the chain, many long-range edges will be of length one, thus they coincide with local edges and can be omitted. Consider a node u that is located at the intersection of i-th row and j-th column. To determine the length and the direction of the left-oriented or the right-oriented long-range edge, originated from the node u, consider only the index of the column, j. The index of the row for the left-oriented or right-oriented long-range contact is already determined by the node u — they are equal to i. The index of the column for the left-oriented or the right-oriented long-range contact of u is defined by the rules described for the chain, where the whole row i of the two-dimensional grid is considered as a chain. Similarly, to determine the up-oriented or the down-oriented long-range contacts of the node u, consider column j as another chain.

Two facts are important in this edge generation procedure. Firstly, determining the exact location of horizontally oriented edges and vertically oriented edges are two independent processes (conducted according to the same rules). Secondly, the heads of the all left-oriented (right-oriented) long-range edges originating from a single column will belong to some other single column; the heads of all up-oriented (down-oriented) long-range edges originating from a single row will belong to some other single row.

The greedy algorithm for the routing from the source node s to the target node t in the two-dimensional grid is composed of two interleaved processes: one process is responsible for the horizontal movement and the other for the vertical movement. The exact order of interleaving of horizontal and vertical steps is not important. The key is that the sub-sequence of horizontal (vertical) steps is identical to the sequence defined for the chain, obtained by the projection of the two-dimensional grid onto the one-dimensional line.

The algorithm can be expanded to higher dimensional grids: the order of complexity and the order of the inverse distribution remains the same. For the k-dimensional grid, the running time becomes $O(k(\log^2 n))$. The number of local and long-range contacts per node grows linearly with the dimension of the grid — two local and one long-range edges per each dimension. However, it is reasonable to suggest that the dimension of grid k is incompatibly smaller than the number of rows in the grid. Then the dimension k can be considered as a constant and removed from the bound on the time complexity and from the bound on node degree.

2.3.4 The average and expected edge lengths

The order of complexity of the greedy routing for one- and two-dimensional grids of the deterministic model is the same $O(\log^2 n)$. Surprisingly, the same inverse linear distribution is used to generate the long-range edges, which at first glance appears to contradict the results of Kleinberg for the probabilistic model that the exponent of the inverse distribution should match the dimension of the grid. However, there are other differences between the two settings. If we look at the frequency with which edges of a certain length are created in

the deterministic model, we see that a length, which is twice as short, is used twice as often. The result looks like the inverse linear distribution, but some lengths (everything that is not a power of 2) are not used at all. The average length of the long-range edges can be a better property to consider. In the one-dimensional case, we have $n/2$ edges of the length 1, $n/4$ edges of the length 2, and so on. This makes the sum of all lengths about $n(\log n)/2$, and the average length is $(\log n)/2$. The average length remains the same for higher dimensions.

In the probabilistic model of generation of the long-range edges for the two-dimensional grids, the expected length $E[l(u)]$ of a long-range edge originated from node u is the sum (over all the possible distances in the grid) of the probability p_d that node v which is at the distance d from the node u becomes a long-range contact of u, multiplied by the number n_d of nodes at the distance d from the node u, multiplied by the distance d between u and v:

$$E[l(u)] = \sum_d p_d \cdot n_d \cdot d = \sum_d \frac{1/d^2}{C(u)} \cdot n_d \cdot d = \frac{1}{C(u)} \sum_d n_d/d \tag{2.1}$$

This is based on the fact that Kleinberg sets p_d to be inversely proportional to d^2:

$$p_d = \frac{1/d^2}{C(u)},$$

where $C(u)$ is the normalizing constant defined with respect to the node u and is equal to

$$C(u) = \sum_{v \neq u} 1/d^2 = \sum_{d=1}^{\infty} n_d \cdot \frac{1}{d^2} \tag{2.2}$$

The expected length of long-range edge depends on the location of the tail of the edge. In Lemmas 4 and 5, we will show that the edges with the longest and shortest expected length are of the same asymptotic order: $\Theta(n/\ln n)$. Consequently, all the other long-range edges are of the same order as well.

Thus, we see that the expected length of a long-range edge in the probabilistic model is $\Theta(n/(\log n))$, while in the deterministic model it is only $\Theta(\log n)$. This can be caused by the fact that in the deterministic algorithm we had a chance to align the long-range contacts in certain (natural) directions. Another important factor is that, in the probabilistic model, decisions about long-range contacts are made independently of each other.

2.3.4.1 Corner node

The nodes that are located in the corners of the grid have the longest expected length of long-range edge, because all the other nodes (which are the candidates to be the head of the edge) are filling only a quarter of the two-dimensional space. The other three quarters of the space are not a part of the grid (the origin of the space partitioning is the considered node in the corner).

Lemma 4. *The expected length of the long-range edge originated from a corner of the two-dimensional probabilistic grid-based graph satisfies the following asymptotic relation:*

$$E[l(corner)] = \Theta\left(\frac{n}{\ln n}\right)$$

Proof. When $d \in [1, n-1]$, the number of nodes that are at distance d: $n_d = d+1$ (when $d = n-1$ those n nodes form a diagonal of the grid). When $d \in [n, 2n-2]$, the number of nodes that are at distance d: $n_d = 2n-d-1$. According to Formula (2.1), the expected edge length:

$$\begin{aligned}
E[l(corner)] &= \frac{1}{C(corner)} \sum_d \frac{n_d}{d} = \frac{1}{C(corner)} \left[\sum_{d=1}^{n-1} \frac{d+1}{d} + \sum_{d=n}^{2n-2} \frac{2n-d-1}{d} \right] \\
&= \frac{1}{C(corner)} \left[\sum_{d=1}^{n-1} 1 + \sum_{d=1}^{n-1} \frac{1}{d} + 2n \sum_{d=n}^{2n-2} \frac{1}{d} - \sum_{d=n}^{2n-2} 1 - \sum_{d=n}^{2n-2} \frac{1}{d} \right] \\
&= \frac{1}{C(corner)} \left[(n-1) + \sum_{d=1}^{n-1} \frac{1}{d} + 2n \left(\sum_{d=1}^{2n-2} \frac{1}{d} - \sum_{d=1}^{n-1} \frac{1}{d} \right) - (n-1) - \left(\sum_{d=1}^{2n-2} \frac{1}{d} - \sum_{d=1}^{n-1} \frac{1}{d} \right) \right] \\
&= \frac{1}{C(corner)} \left[2n \left(\sum_{d=1}^{2n-2} \frac{1}{d} - \sum_{d=1}^{n-1} \frac{1}{d} \right) - \sum_{d=1}^{2n-2} \frac{1}{d} + 2 \sum_{d=1}^{n-1} \frac{1}{d} \right] \qquad (2.3)
\end{aligned}$$

From Formula (2.2), we can bound $C(corner)$:

$$C(corner) = \sum_{d=1}^{\infty} n_d \cdot \frac{1}{d^2} = \sum_{d=1}^{n-1} \frac{d+1}{d^2} + \sum_{d=n}^{2n-2} \frac{2n-d-1}{d^2}$$

The last summation is positive; its terms decrease with d. The largest term is $(2n-n-1)/n^2$ (when $d = n$). Thus,

$$0 < \sum_{d=n}^{2n-2} \frac{2n-d-1}{d^2} \leq \sum_{d=n}^{2n-2} \frac{2n-n-1}{n^2} = (n-1)\frac{n-1}{n^2} < 1$$

The sum is bounded from both sides by constants, so it is $o(\ln n)$ as well. Now we can bound $C(corner)$ as n tends to infinity:

$$C(corner) = \sum_{d=1}^{n-1} \frac{d+1}{d^2} + o(\ln n) = \sum_{d=1}^{n-1} \frac{1}{d} + \sum_{d=1}^{n-1} \frac{1}{d^2} + o(\ln n)$$

$$= \zeta(1) + \zeta(2) + o(\ln n) = \ln n + \gamma + \Theta(1/n) + \frac{\pi^2}{6} + o(\ln n) = \ln n + o(\ln n),$$

where $\zeta(i) = \lim_{n\to\infty} 1/d^i$ is the Riemann zeta function, $\zeta(1) = \ln n + \gamma + \Theta(1/n)$, $\gamma = 0.577\ldots$ is the Euler-Mascheroni constant, and $\zeta(2) = \pi^2/6$.

We use this by going back to Equation (2.3):

$$E[l(corner)] = \Theta\left(\frac{1}{\ln n}\right)\left[2n\left(\sum_{d=1}^{2n-2}\frac{1}{d} - \sum_{d=1}^{n-1}\frac{1}{d}\right) - \sum_{d=1}^{2n-2}\frac{1}{d} + 2\sum_{d=1}^{n-1}\frac{1}{d}\right]$$

$$= \Theta\left(\frac{1}{\ln n}\right)\left[(2n-1)\sum_{d=1}^{2n-2}\frac{1}{d} - (2n-2)\sum_{d=1}^{n-1}\frac{1}{d}\right]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[(2n-1)(\ln(2n-2)+\gamma+\Theta(1/2n)) - (2n-2)(\ln(n-1)+\gamma+\Theta(1/n))]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[(2n-1)(\ln(2n-2) - \ln(n-1) + \Theta(1/n)) + \ln(n-1) + \Theta(1)]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[(2n-1)\Theta(1) + \ln(n-1) + \Theta(1)] = \Theta\left(\frac{n}{\ln n}\right)$$

To summarize:

$$E[l(corner)] = \Theta\left(\frac{n}{\ln n}\right)$$

□

2.3.4.2 Central node

The node that is located in the center of the grid has the shortest expected length of long-range edge, because all the other nodes (which are the candidates to be the head of the edge) are spread out evenly an all four directions. We will consider here grids with an odd number of the rows, so there is only one central node in the grid.

Lemma 5. *The expected length of the long-range edge originated from the central node of the two-dimensional probabilistic grid-based graph with an odd number of the rows satisfies the following asymptotic relation:*

$$E[l(center)] = \Theta\left(\frac{n}{\ln n}\right)$$

Proof. When $d \in [1,(n-1)/2]$, the number of nodes that are at distance d: $n_d = 4d$. When $d \in [(n+1)/2, n-1]$, the number of nodes that are at distance d: $n_d = 4(n-d)$. According to Formula (2.1), the expected edge length:

$$\begin{aligned} E[l(center)] &= \frac{1}{C(center)} \sum_d \frac{n_d}{d} = \frac{1}{C(center)} \left[\sum_{d=1}^{(n-1)/2} \frac{4d}{d} + \sum_{d=(n+1)/2}^{n-1} \frac{4(n-d)}{d} \right] \\ &= \frac{4}{C(center)} \left[\sum_{d=1}^{(n-1)/2} 1 + \sum_{d=(n+1)/2}^{(n-1)} \frac{n}{d} - \sum_{d=(n+1)/2}^{(n-1)} 1 \right] \\ &= \frac{4}{C(center)} \left[\left(\frac{n-1}{2}\right) + \sum_{d=1}^{n-1} \frac{n}{d} - \sum_{d=1}^{(n-1)/2} \frac{n}{d} - \left(\frac{n-1}{2}\right) \right] \\ &= \frac{4}{C(center)} \left[\sum_{d=1}^{n-1} \frac{n}{d} - \sum_{d=1}^{(n-1)/2} \frac{n}{d} \right] \end{aligned} \tag{2.4}$$

From Formula (2.2), we can bound $C(center)$:

$$C(center) = \sum_{d=1}^{\infty} n_d \cdot \frac{1}{d^2} = \sum_{d=1}^{(n-1)/2} \frac{4}{d} + \sum_{d=(n+1)/2}^{n-1} \frac{4(n-d)}{d^2}$$

The last summation is positive; its terms decrease with d. The largest term is $(4(n-(n+1)/2)/((n+1)/2)^2$ (when $d = (n+1)/2$). Thus,

$$0 < \sum_{d=(n+1)/2)}^{n-1} \frac{4(n-d)}{d^2} \leq \sum_{d=(n+1)/2)}^{n-1} \frac{4(n-(n+1)/2)}{((n+1)/2)^2} = \frac{n-1}{2} \times \frac{4((n-1)/2)}{((n+1)/2)^2}$$

$$= 4\frac{((n-1)/2)^2}{((n+1)/2)^2} = 4\left(\frac{n-1}{n+1}\right)^2 < 4$$

The sum is bounded from both sides by constants, so it is $o(\ln n)$ as well. Now we can bound $C(center)$ as n tends to infinity:

$$C(center) = \sum_{d=1}^{(n-1)/2} \frac{4}{d} + o(\ln n) = 4(\ln n + \gamma) + o(\ln n) = 4\ln n + o(\ln n),$$

where $\gamma = 0.577\ldots$ is the Euler-Mascheroni constant.

We use this by going back to Equation (2.4):

$$E[l(center)] = \Theta\left(\frac{1}{\ln n}\right)\left[\sum_{d=1}^{n-1} \frac{n}{d} - \sum_{d=1}^{(n-1)/2} \frac{n}{d}\right]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[n(\ln n + \gamma + \Theta(1/n)) - n(\ln(n/2) + \gamma + \Theta(2/n))]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[n(\ln n + \gamma - \ln n + \ln 2 - \gamma + \Theta(1/n))]$$

$$= \Theta\left(\frac{1}{\ln n}\right)[n(\ln 2 + \Theta(1/n))] = \Theta\left(\frac{n}{\ln n}\right)$$

To summarize:

$$E[l(center)] = \Theta\left(\frac{n}{\ln n}\right)$$

□

2.3.5 Summary

We built a deterministic grid-based model. The long-range edges were created iteratively, creating a length-based hierarchy. The idea for edge generation was dictated by an attempt to simulate the search process in the probabilistic grid-based model divided into a logarithmic number of phases. The freedom in creation of the long-range edges allowed generating long-range edges that have asymptotically shorter average length than the corresponding expected length in the probabilistic model. This can be useful in an environment where the cost of edge creation increases monotonically with the edges length. Another difference from the probabilistic model is that each long-range edge is aligned with only one dimension of the grid and the edge length distribution is independent of the dimension of the grid.

Chapter 3

ADAPTIVE DECENTRALIZED ROUTING IN SMALL-WORLD NETWORKS

3.1 Introduction

Graphs are widely used to model real-world networks and so it is of interest to have a model describing the structure of a random network. While the classical Erdős-Rényi model [12] has been well studied, its properties do not match those observed in real-world networks. Observation of real-world networks has led to several models being proposed, many of which are based on explicitly using geometric properties such as an underlying metric space where nodes are located, and making a distinction between "local" and "long-range" connections. One of the most prominent such models is Kleinberg's grid-based model [28], motivated by the earlier work of Watts and Strogatz [49]. This is the same model, which we studied in Chapter 2. In addition to a generative definition of the network as a regular grid of local contacts together with a bounded number of long-range contacts chosen randomly from a particular probability distribution, Kleinberg offered an algorithmic justification for the small-world phenomenon. This justification comes from considering the decentralized greedy routing algorithm in which every node forwards the message to the neighbor who is closest to the destination in terms of some notion of distance based on the geometry of the underlying space. While in certain networks constructed from real world data there may not be an intrinsic basis for a geometric interpretation, one may be postulated through an embedding of the network into a low-dimensional space using any of a number of algorithms developed either for graph drawing or specifically for low-dimensional embeddings. Kleinberg proved that in his grid-based model, the expected delivery time of greedy routing, measured in the number of hops, is polylogarithmic in the size of the network. This provides some theoretical explanation for empirical results found through many studies of processes similar to greedy routing, starting from the early work of Milgram [36], which showed that many arbitrarily chosen pairs of strangers were linked through very short chains of acquaintances. In this chapter, we attempt to examine possible improvements to greedy routing through a decentralized learning process.[1] One way to view this is as the natural

[1] Bakun and Konjevod recently presented an abbreviated version of this chapter in [9].

development of a routing strategy: each node works completely independently and bases its routing decisions only on its past experience.

3.1.1 Related work

3.1.1.1 Greedy routing and variants

Of many papers on greedy routing, we only mention those most closely related to our work. Fraigniaud et al. [22] improve greedy routing if long-range links of nodes in the vicinity are known. Lebhar and Schabanel [33] grow a (small) search tree in every routing step, examining a total of $\Theta(\log^2 n)$ nodes to improve the routes (again, in a $\sqrt{n} \times \sqrt{n}$ grid) to expected length $O(\log n \log\log n)$.

3.1.1.2 Edges as experts

Using a model similar to ours, Awerbuch and Kleinberg [4] propose an online shortest path algorithm that considers each edge in the graph as an expert. However, their online shortest path algorithm knows the network topology but not the edge costs. In addition, they seek to optimize the cost for a fixed source-target pair. We, on the contrary, group target nodes into regions that become smaller and smaller the closer they are to the target, thus incrementally navigating towards the target and allowing an emergence of the routing scheme.

3.1.1.3 Compact routing

A perhaps more practical motivation for our work comes from studying compact routing schemes for distributed networks. A routing scheme is simply an algorithm that each node in a network runs to determine how to forward a message. Each message is given a header that may contain information about its destination and possibly its path so far, and this header, together with the routing tables present at the node, is all that the node uses to determine the next hop for the message. In compact routing, the message header and the routing table at each node are restricted in size to a number of bits polylogarithmic in the size of the network. Intuitively, this may be thought of as allowing the header and the routing table to store some information about a small number of other nodes; just numbering the nodes consecutively will lead to names of size logarithmic in the network size. Thus, compactness

is a relatively strict requirement, but it is one that arguably leads to scalable performance, since the storage necessary at each node grows slowly with the network. There has been a lot of work on compact routing in the distributed algorithms community, a large amount of it is focusing on the tradeoff between two mutually incompatible objectives. The first of these is the *stretch* of a routing scheme, as measured in terms of the worst-case, over all source-destination pairs, ratio between the length of a routing path and the shortest path. The second objective is the memory overhead: the amount of storage allowed for routing tables and message headers. There exists by now a large literature on compact routing. General surveys are available [44, 23, 24, 17]. It is known that in general networks either the storage must grow polynomially with the size of the network or some routing paths will require more than a constant stretch [47, 1]. The most general class of networks, that are known to allow constant stretch for compact routing, are networks of low doubling dimension [31].

We can claim that our routing is a name-dependent compact routing scheme for networks with a polylogarithmic bound on the node's outdegree. It has to be name-dependent, because we need to know targets' locations. However, we cannot claim any stretch because we cannot guarantee the message delivery. While we do not focus on theoretical results in this dissertation, we do note that we are not aware of any such work on compact routing schemes designed specifically for small-world networks. In fact, special properties common to most of the proposed models for small-world networks, and also the intuitive motivation for them, for example the existence of multiple short paths between most nodes in the network, give hope that the situation here may be better than for general networks and, for example, allow for a constant stretch with compact storage requirements.

3.1.2 Summary of results

We describe a decentralized learning process designed to improve on the routes found by the greedy routing algorithm. We use online machine learning techniques to construct search paths that are shorter than paths constructed by the greedy algorithm. We apply our adaptive decentralized routing to real networks and to networks generated from the grid-based

model. In order to supply geometric information not present in their definition, we embed real networks into two-dimensional space using a spectral graph-drawing algorithm by Koren [32].

Our algorithm is adaptive and resilient. In separate sets of experiments, we change the environment by depriving nodes of the knowledge about their neighbors' locations, or by assuming that a significant fraction of nodes is only partially cooperative. Even in the presence of these added difficulties, the learning process works almost as well, if somewhat more slowly.

The first method we examine is a variation of the classical weighted majority algorithm [35] (described in Section 3.3.1). We refer to it as MWU, after the Multiplicative Weights Update method by Arora, Hazan, and Kale, whose survey [3] collects applications of the method in various fields. The second learning method we propose is a variation of the Exploration-Exploitation Experts method (EEE) suggested recently by de Farias and Megiddo [15]. They describe this as an artificial intelligence method whose goal is to enforce cooperation in multistage games in reactive environments. These methods use an ensemble of experts advising on actions for each stage of the learning process. In both cases, the cumulative result achieved by the method will be in the limit arbitrarily close to the result achieved by the best of the experts.

If, after a period of learning, we "deactivate" those steps in our algorithm that relate to learning, that is, to changing the behavior according to the received feedback, we are left with an algorithm that behaves greedily, albeit with the greedy decisions influenced by its knowledge of the network accumulated through past feedback.

Finally, by showing similar behavior in synthetic grid networks as in spectral embeddings of those based on real-life data, our work also provides an independent confirmation of the quality of embeddings produced by Koren's spectral embedding algorithm.

3.2 Adaptive decentralized routing

Greedy search studied by Kleinberg [28] assumes that each node only knows the exact location of its neighbors and of the target. The node does not know the complete topology of the network, so it chooses the neighbor that is closest to the target in the terms of the lattice distance of the underlying grid. We make the following assumptions that go beyond those required by Kleinberg's algorithm:

- Each node knows its own location. (In Kleinberg's greedy routing, each node relies on its knowledge of its neighbors' locations, but, strictly speaking, does not need to know its own location, even though in the standard grid model this can be easily derived from the neighbors' locations.)
- Each node has access to locally stored unshared memory of size polylogarithmic in the size of the network times the outdegree of the node. There is a logarithmic number of the ensembles of experts, and an outgoing edge serves as an expert in each of the ensembles.
- Each message has a header of size polylogarithmic in the size of the network. It has to store its route in order to provide the feedback.
- Each successfully delivered message is followed by an acknowledgement receipt, which is sent back to the source using the original path in the reverse order. (This requires the message header to store the path the message has traversed. In our experiments, we restrict all actual routing paths to a polylogarithmic number of hops by declaring failure if the maximum allowed length is exceeded, and thus we may still claim that the message header we use is compact.)

We cannot claim that our routing is compact, because the local memory of the node is proportional to the node's outdegree (we need to maintain a counter and weight for each expert). Thus, we can make the claim about compactness only for networks with a

polylogarithmic bound on the node's outdegree. It is a name-dependent routing because we require the node's location to be a part of the node's name. However, we cannot claim any stretch because we cannot guarantee the message delivery.

As usual with a machine learning method, the results improve over time and a large number of routing queries are run before the method provides the best results. Although in our experiments, it took only about a hundred source-destination pairs per node to improve on the performance of the greedy algorithm.

We now give two building blocks of the algorithm (regions in Section 3.2.1 and constant experts in Section 3.2.2), and then present the routing scheme (Section 3.2.3). We coach this in terms of the grid-based model and use geometry of the two-dimensional grid for the algorithm description. In those cases where the network does not come with a natural geometric structure and coordinate information, we first embed it into two-dimensional space as described in Section 3.5.1.

3.2.1 Regions

Our search algorithm is still fully decentralized, but it is not greedy any more. Each node's goal is to deliver the message not to the target itself, but to a region of the network to which the target is known to belong. Each node partitions the space into disjoint regions, defined with respect to the node's location. See an example of the region partitioning in Figure 3.1, on the left. The idea to split the whole space into regions comes from the proof of the upper bound on the expected path length (Lemma 1) in the grid-based model. We will make adjustments for others graphs by evolving the described here static partitioning into the dynamic partitioning later (in Section 3.5.2). Kleinberg considers the routing from the point of view of the target and divides the process into a logarithmic number of phases. The phases in effect divide the whole space into the corresponding smaller areas (see Figure 2.3). Although the sizes of the areas are remarkably different, each phase is expected to last no more than a logarithmic number of steps. This indicates that the navigation in the immediate vicinity is disproportionally important (relatively to the size of the vicinity). In the case

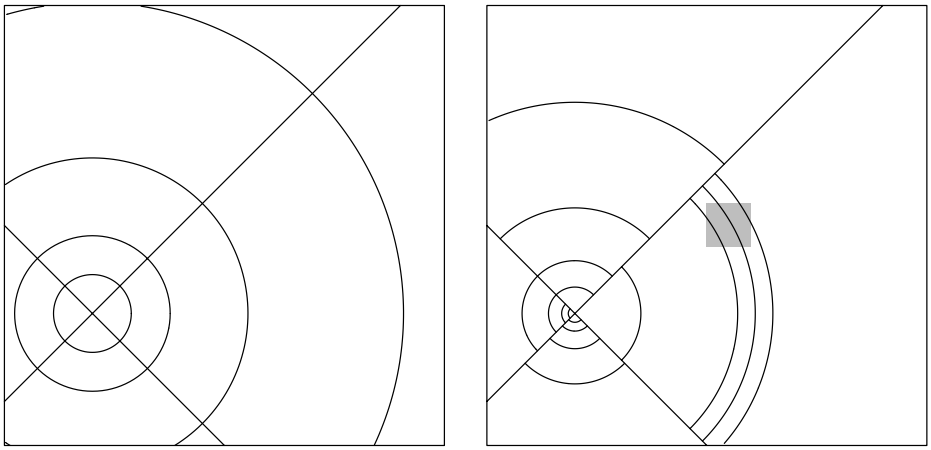

Figure 3.1: Examples of region partitioning. Original static partitioning (left). Dynamic partitioning (right). High node density area is painted grey. It draws regions to increase the node coverage in the area.

when node u has to dispatch messages to different targets, we need to define the areas (regions) with respect to the location of node u. We want to have regions closer to the node u to be small. In addition, we want to introduce direction (two per dimension), because in our case u sends messages away from itself along different directions, and not attracting the messages toward itself.

In order to navigate, instead of the actual coordinates of the target, the node uses the direction and the Euclidean distance to the target. In the two-dimensional network, there are four natural directions: left, right, up and down. Let $k = \lfloor \log_2(2\sqrt{n}) \rfloor$ be the number of regions in a single (out of four natural) direction, where n is the number of nodes in the network, and let D_{space} be the diameter of the metric space. The boundaries between the regions of the same directions are circular arcs. Region $j = 2, 3, \cdots, k$ will cover area of the corresponding direction at distance $D_{space}/2^{k-j+1}$ to $D_{space}/2^{k-j}$ from the corresponding node, for $j = 1$ it is the area at distance up to $D_{space}/2^{k-1}$.

As the message nears the target, the regions become smaller and finally converge to the target itself. This way the regions closest to the *current message holder* (CMH) are very small; and the farthest regions might cover up to 30% of the space (assuming a square shape

of the entire space). Being able to effectively dispatch messages to the close-by targets is extremely important, because the message holder makes the same kind of decision whether it is the original source itself or an intermediary, and because if the current node is close to the target then it is likely to be a frequent intermediary for this target. In this way, each node defines $4\lfloor\log_2(2\sqrt{n})\rfloor$ regions and stores all related information locally without sharing it with its neighbors. There is a total of $4n\lfloor\log_2(2\sqrt{n})\rfloor$ regions in the network.

3.2.2 Constant experts

In order to decide where to forward a message, each region has its own ensemble (set) of experts ("constant experts" in our case). A *constant expert* (or just *expert*) here is essentially an outgoing edge of the corresponding CMH that gives the same advice "choose me" in all stages of the protocol. The number of experts in the ensemble is equal to the outdegree of the corresponding node. An ensemble is responsible for dispatching any message that has a target, which is a node of the corresponding region. The best expert (edge) of the same ensemble may vary depending on which of the multiple nodes belonging to the region is the target in this stage and what choices will be made by the remaining intermediaries between the CMH and the target. We will discuss machine learning methods in Section 3.3.

Let e be an outgoing edge of node u. The node u has $4\lfloor\log_2(2\sqrt{n})\rfloor$ regions; and it has $4\lfloor\log_2(2\sqrt{n})\rfloor$ experts that advise choosing the edge e. The rating of an expert advising to choose e is determined by the performance of the expert with respect to its ensemble, and is independent of a performance by any other expert advising to choose e (because this other expert belongs to a different ensemble and attempts to solve a different problem — navigation to a different region).

3.2.3 Routing scheme

The online algorithm works in the following way:

- At each step the CMH (initially the source node):
  - Determines the region that contains the target, based on the target's coordinates.

  - Chooses an expert according to the adopted machine learning method (either EEE or MWU, see Section 3.3.1). The chosen expert is often (but not always) the best expert so far.
  - Adds CMH's name, the chosen expert, and the step number to the header of the message.
  - Forwards the message to the corresponding neighbor.

- Exceptions to the above:
  - If the target is one of the CMH's neighbors, then the corresponding edge is chosen automatically.
  - The message cannot be returned to the immediate previous message holder (that is, no cycles of length two are allowed), unless this is the only neighbor. This operation is cheap, and is done because if the best experts of two nodes want to forward a message to each other, it might take long until some other expert will be followed to break the cycle.

- If the target is not found in the number of steps equal to $\log_2^2 n$, the search is halted and declared as unsuccessful. No feedback to the participating nodes is provided in this case.

- If the target is found then an acknowledgement receipt including the route of the message and the total number of steps is formed. The receipt is sent to the source along the original route, but in the reverse order. At each intermediary node, including the source, the counters and weights of the corresponding ensemble are updated according to the actual length of the part of the path from the node to the target. This way a node learns not only if it is a source, but also if it is an intermediary.

We compare this adaptive routing protocol to the greedy algorithm (which uses Euclidean distance) and the breadth first search (which provides the shortest path distance). Unlike the other two, adaptive routing requires sending acknowledgement receipts. We

ignore this additional cost. There are at least two justifications. One is that when the network is static (no new nodes, no new edges) we can stop sending the receipts and stop learning once the desired performance is achieved. The other is that the receipts can be sent when the network load is low and the delivery is cheaper (though this can somewhat slow down the learning process).

3.3 Machine learning methods

In Section 3.3.1, we start with a description of our variant of the multiplicative weights update method (a generalized version of the well known weighted majority algorithm). This method was later used in the experiments with artificial grid-based networks only (Section 3.4). Then we continue with the exploration-exploitation experts method, which better suits our needs to learn the network topology. This method demonstrated better results and was used in the both artificial (Section 3.4) and real networks (Section 3.5).

For both methods, each region of each node requires its own ensemble; each outgoing edge of the node is an expert of the ensemble.

3.3.1 Multiplicative Weights Update method

Variants of *multiplicative weights update* (MWU), which is a generalization of the weighted majority algorithm, are widely used in various fields of computer science. Several experts form an ensemble. All the experts advise on the course of action at every stage. It is assumed that it is easy to observe outcomes of their advices. The method (MWU in this case) does not know at the beginning which experts are good and which are not. It observes the outcomes of past advice (of all the experts at all the stages) and tries to learn the best expert. The *best expert* is defined as someone who would accumulate the smallest penalty (the largest reward), if we always followed this expert at all the stages. The best expert is defined with respect to a sequence of problems presented in front of the ensemble. MWU provides us with a strategy how to learn about the best expert and how, by following the best expert most of the time, to average regret that converges to the average regret of the best expert. Here, regret of an expert is the difference between the sum of all penalties

incurred by the expert and the sum of all penalties that can be incurred if the machine learning follows the best advice at each stage of the game.

The method associates a weight with each expert. After each stage the weights of all the experts are updated using the formula:

$$w_i = \begin{cases} w_i(1-\varepsilon)^{+M(i,j)/\rho} & \text{if } M(i,j) \geq 0 \\ w_i(1+\varepsilon)^{-M(i,j)/\rho} & \text{otherwise} \end{cases} \tag{3.1}$$

Here i denotes an expert, j is an outcome, $M(i,j)$ is the corresponding penalty, all penalties are in the range $[-\rho,\rho]$, and ε is a constant that defines how strong is the impact of the last outcome. At each stage, MWU chooses an expert i at random with probability $w_i/\sum w_k$. For more details see the overview by Arora et al. [3].

The direct adoption of this algorithm is not possible in our environment; in this case, we would have to examine exponentially many paths in order to know the outcome of following a particular expert. Therefore, we will update the weight of only one expert per ensemble at a time — the weight of the chosen expert. In addition, we use a one-dimensional penalty matrix $M(j)$, because there is no point to discriminate against any edge. The matrix $M(j)$ is a matrix with dynamic penalties, because it is difficult to estimate from the start, what would be a good path length between a CMH and its region. The average outcome seen by the ensemble so far is considered as neutral (penalty of zero). For more details on the penalty matrix, see Appendix A.

3.3.2 Exploration-exploitation experts method

The original *Exploration-Exploitation Experts* (EEE) learning method [15] induces cooperation between players in a game theory setting. For example, it helps in the Prisoner's Dilemma game when the best-case scenario for two suspects (players) is to cooperate (with each other and not with the police) and deny any wrong-doing. However, it requires trust between the suspects. Without trust, the cooperation seems irrational, because cooperation is strictly dominated by defection (giving out your partner and reducing by this your own punishment). On the over hand, when this stage of the game is played repeatedly, one of the

suspects may induce cooperation with the other suspect by repeatedly choosing the dominated strategy to indicate that one is ready to act irrationally. It is impossible to make this kind of indications in a single stage game, where rationality is a manifestation of selfishness limited by the short-term horizon.

In order to induce cooperation a player employs the EEE method. EEE is conducted in *phases*; a phase in turn consists of several consecutive stages where the method follows the same expert. For an expert e, a counter, which is telling for how many phases e has been chosen so far, is maintained. When the expert e is chosen for a new phase, its counter is incremented by 1 and the phase will run for the number of stages equal to the counter. This way, the length of the phase is defined by the "popularity" of the chosen expert: at some point late in the game, a length of a phase of a popular expert will be very long, and a length of a phase of an unpopular expert will be very short. The idea is simple: for each of the experts, EEE checks if the cooperation (with the environment or nature) can be induced by gradually increasing the number of stages in which the same expert is consecutively chosen. In addition to the counter, the method maintains a weight of each expert. When the feedback arrives, only the weight of the corresponding expert (that was chosen for the forward routing in this stage) is updated in such a way that it is an average of all the received feedback for the expert.

Now we need to decide how the expert is chosen for a new phase. Suppose that we are at beginning of the phase j, that is, j is equal to the sum of the counters of all the experts of the ensemble. Each phase is either an exploration or an exploitation phase. The main version of EEE suggested by de Farias and Megiddo assumes that at the beginning of phase j, with the probability $p_j = 1/j$, the phase is declared an exploration phase and the expert that will be used throughout the phase is chosen uniformly at random. Otherwise, this is an exploitation phase, and the expert that has shown the best results so far (the expert that has the highest weight) is chosen.

De Farias and Megiddo studied the competitiveness of the method with respect to the best expert as well. The situation here is different form the situation with MWU, because here we do not know outcomes of following the advice of each of the experts, and cannot learn from these outcomes. The expected average reward of EEE is infinitely close to the average outcome of the best expert (when the number of stages grows to the infinity). However, the average outcome for the best expert is calculated only for the stages in which this expert was actually chosen.

When the intensive learning is over, there will often be one leading expert, chosen for almost all exploitation phases. Its counter will be greater that the sum of all the other counters, and so the probability that stage i is an exploration stage that does not utilize the best expert will be roughly $1/i^2$ (i is counted from the beginning, regardless of what expert is chosen, the same way it is done for j — the counter of phases). This means that the frequency of the learning stages decays fast. That can be undesirable in dynamic environments (for example in our case, when edges can be added or deleted over the time). To overcome this problem de Farias and Megiddo suggested a different version of EEE, where the probability of phase j to become an exploration phase is a constant and does not depend on the value of j.

Our version of EEE has the following specifications and alterations. In our case, a player is a region of a node, which routes incoming messages to all targets of the region (an expert is not a player; an expert (outgoing edge) is a strategy). Each time a player receives a message, it is a new stage in the game of routing. We use the reciprocal path length as the feedback (outcome of the stage). At the beginning (when the first target of the corresponding region arrives), we initialize the weight of each expert with the reciprocal of the Euclidean distance between the corresponding neighbor and the target. De Farias and Megiddo were concerned with a reactive environment and introduced phases (choosing one expert for several consecutive stages) in order to overcome cyclic behavior of the environment and to encourage cooperation among agents. While this may indicate further research possibilities, we do not currently examine such situations and abandon phases all together.

Stage i becomes an exploration stage with probability $1/\sqrt{i+1}$, which allows more exploration stages. We have received better results with this low exponent (in comparison to $1/i^2$). This may be caused by the fact that we are mostly concerned with the initial stages and because our environment is highly dynamic at the beginning (every region starts as an agent in its own initial stages and affects the environment more strongly because it has many exploration stages of its own). Even if a node u has to route to the same target several times, the best edge can be different for different stages, because no one knows what decision will be made later by other nodes (of the remaining parts of the paths to the target) in cases when the nodes are entering into new phases.

3.4 Grid-based networks

3.4.1 Review of the grid-based model

The small-world network model we use in our experiments is the grid-based model that was suggested by Kleinberg [28]. This is the same model that we studied in Chapter 2; however, the notation is slightly different. To recap, it is a regular two-dimensional grid (square $l \times l$ lattice) augmented with a small number of long-range edges. Throughout this chapter, we denote the number of nodes in the network by n; then for the grid $n = l^2$. The network contains edges between every pair of nodes at lattice distance at most p. These edges are said to link local contacts. Additionally, each node has q outgoing links to some nodes that may be far away from each other. Each of these long-range contacts of u is chosen randomly, so that for any node $v \neq u$, the probability of v becoming a long-range contact of u is proportional to $1/d^r(u,v)$, where $d(x,y)$ is the lattice distance between x and y. See a more detailed description of the model in Section 2.2.1.

3.4.2 Experimental setup

Thus, a grid can be described by the parameters l (size of the grid), $p \geq 1$ (local contact range), r (distribution exponent), and q (the number of long-range contacts). In all our experiments p is one, q is one, and r is two (this is the only value of r that leads to the emergence of the small-world network for the two-dimensional grid). That is, most of

the nodes have four local contacts (border nodes have three and corner nodes have two). Most nodes have one long-range contact (some long-range contacts may coincide with local contacts). Note that, although long-range edges are unidirectional, we assume that the acknowledgement receipt (but not the message itself) can be sent in the opposite direction of the edge.

For each search, we choose both the source and the target uniformly at random. We run the adaptive decentralized routing described above, and send the acknowledgement receipts upon success. Each new source-target pair is chosen independently of the previous ones.

To evaluate the performance of the adaptive routing algorithm in comparison to the performance of the greedy routing algorithm, we define the *overpayment rate*, as

$$\zeta = (d_{ad} - SPD)/(d_{gr} - SPD) \tag{3.2}$$

Here, *SPD* is the average shortest path distance over all source-target pairs in the network; d_{gr} denotes the average path length of the paths constructed by greedy routing (that uses Euclidean distance) for the first 100,000 source-target pairs (the expected path length for greedy routing does not change much with time); and d_{ad} denotes the average path length of the paths constructed by the adaptive routing for the latest bucket. A *bucket* (initially) consists of a hundred (or a thousand) consecutive source-target pairs. We use non-overlapping buckets of exponentially growing size in order not to overcrowd plots with logarithmic scales. The average path length takes into account both successful and unsuccessful (over $\log_2^2 n$ steps) path lengths of the adaptive routes. In this way, an overpayment rate of 0 means that the adaptive routing shows the shortest path distance results and overpayment rate of 1 means that it is similar to greedy routing.

We conducted experiments for the grids of the following sizes: 32×32, 64×64, 128×128, and 256×256. The corresponding maximum path length $d_{\max} = \log_2^2 n$ varied from 100 to 256. Although the maximum path length to nodes ratio $d_{\max}/n$ varied from 0.004 to 0.098, $d_{\max}$ is not high at all. In fact, $d_{\max}$ is only 6.44 to 7.93 times longer than

the corresponding average greedy path length. In Appendix B, we will consider the same grids but with dynamic long-range edges, where networks change with time.

3.4.3 Experiments with EEE

The fraction of unsuccessful searches (when the target is not found in $d_{\max}$ steps) might be high at the start, but decreases to less than 10^{-3} after 50 source-target pairs per node on average. As long as the fraction is small (and it is), the unsuccessful searches are not a real problem at all. The fraction of unsuccessful searches in the last bucket and $d_{\max}/n$ decrease with grid size, even as the results get better. We could have changed the algorithm as the following (but we did not). Upon realizing that the search cannot be completed in $d_{\max}$ number of steps, we would allow the search to continue from the current location greedily (we still would consider this search as a failure and would not return the acknowledgement receipt).

In Figure 3.2, we see that the overpayment rate (and thus the average path length) exhibits the same type of behavior for grids of different sizes, improving slightly with the grid size. The horizontal axis shows the number of source-target pairs per node on a logarithmic scale. The vertical axis shows the overpayment rate (per bucket). EEE starts poorly with many unsuccessful searches, partially due to the high level of exploration rate at early stages that does not allow a full utilization of good initial weights of the experts. It is followed by a sharp improvement that happens at about the same point for all networks. We see that it takes about 42 to 119 source-target pairs per node to reach the results of the greedy algorithm. After 300 pairs per node EEE is "overpaying" 55%–82% of what the greedy is "overpaying" comparing to the SPD, and by 3000 pairs per node, this decreases to 39%–55%. Because of the worst start, the largest network has the largest overall fraction of unsuccessful searches of 0.0042, which is still low. The vast majority of the failures were at the beginning of the learning process. All greedy searches were successful. The results produced for the last bucket of the experiments are shown in Table 3.1.

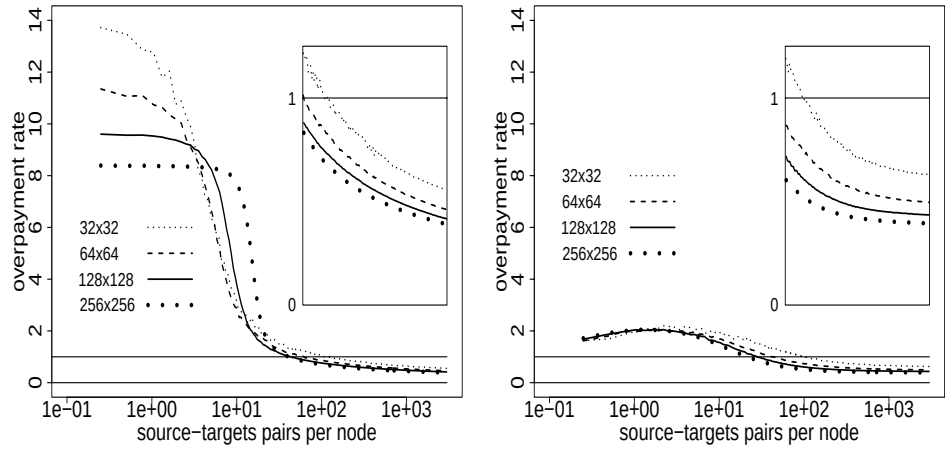

Figure 3.2: Overpayment rate progress for the grid-based networks: EEE (left), MWU (right). Inserts show detail of crowded graph regions.

network	regions per node	$d_{\max}$	average path length				overpayment rate	
			SPD	EEE	MWU	greedy	EEE	MWU
32×32	24	100	6.83	10.03	10.45	12.61	0.55	0.63
64×64	28	144	8.12	13.43	13.82	19.61	0.46	0.50
128×128	32	196	9.31	17.31	17.67	28.50	0.42	0.44
256×256	36	256	10.51	21.91	22.02	39.72	0.39	0.39

Table 3.1: Routing in the grid-based networks. For the adaptive search, the data is given for the last bucket.

3.4.4 Experiments with MWU

The overpayment rate (and thus the average path length) exhibits the same type of behavior for the grids of different sizes, with a slight improvement as the grid size increases, see Figure 3.2. There are no exploration steps in MWU, therefore MWU immediately benefits from good weight initialization based on the Euclidean distance between the CMH and the target, and the failure rate is not high at the start. We see that it takes about 22 to 105 source-target pairs per node to reach the results of the greedy algorithm. After 300 pairs per node EEE is "overpaying" 43%–75% of what the greedy is "overpaying" comparing to the SPD, and by 3000 pairs per node, this decreases to 39%–63%. From the point of view of the average path length, MWU results were very similar to EEE results for the larger networks.

The largest network has the largest overall fraction of unsuccessful searches 0.00014, which is lower than for EEE. However, the failure rate MWU does not change much with time, and eventually EEE shows better current failure rate toward the end. The results shown at the end of the experiments are shown in Table 3.1.

3.5 Real embedded networks

In Section 3.4, we considered artificial networks created according to the grid-based model. Although those networks demonstrate some characteristics of small-world networks (existence of short paths between most pairs of nodes and ability to find them in a decentralized way), they also have characteristics that cannot be expected in real networks. In the grid-based model:

- The network is embedded into (in fact explicitly based upon) the metric space.
- The nodes are distributed uniformly within the minimal bounding box of the network.
- Greedy routing guarantees to make progress at each step (CMH has at least one neighbor, which is closer to the target than CMH is).
- The node outdegrees are within a narrow range (this does not fit the power law degree distribution often associated with the small-world networks).

In this section, we conduct experiments that are similar to the experiments of Section 3.4, but for real networks coming from several different sources. We are not pursuing any concrete application, but evaluating the performance of the adaptive search for realistic small-world network topologies.

We describe the spectral graph embedding that we rely on in Section 3.5.1. We describe further improvements of the adaptive routing protocol in Sections 3.5.2 and 3.5.3. We describe our embedded networks and conduct experiments similar to the experiments of Section 3.4 in Section 3.5.4. We will be using the EEE method only, because it produces better results. In Sections 3.5.5 and 3.5.6 we describe experiments conducted in more adverse environments.

3.5.1 Spectral graph embeddings

The problem of reconstructing the node locations in a grid-based small-world network, given only the graph structure of the network, was posed by Kleinberg [29]. Sandberg [45] studied reconstruction of and also routing in an anonymized grid-based small-world network, by using an algorithm based on the Markov Chain Monte Carlo method. Koren [32] proposed a spectral graph-drawing algorithm, using eigenvectors of the transition matrix of a random walk on the graph, to assign coordinates for the nodes. Similar ideas go back to the original elastic spring model of Tutte [48]. Dell'Amico [16] embedded the OpenPGP web of trust into a metric space for the purpose of greedy routing by using the spectral graph-drawing algorithm. He also suggested a decentralized version of the Koren's algorithm that performs comparatively well. In Appendix C we describe our decentralized version of Koren's algorithm that matches the original algorithm exactly, but requires more communication between the nodes than the version by Dell'Amico. We will not simulate the decentralized versions and use the original Koren's spectral graph-drawing algorithm to embed several real networks into two-dimensional Euclidean space. See Appendix D for the details on the spectral graph embeddings and the plots of the embeddings.

3.5.2 Dynamic partitioning

To be able to deal with non-uniformly distributed nodes we will allow the adaptive algorithm to change dynamically the partitioning of the regions (in the experiments of Section 3.4 the regions were static). It will update the region partitioning of each direction of the node of each node independently from the partitioning of the other directions and nodes. Each node supports the same four directions within the partitioning, and keeps the number of regions in each direction unchanged. A node learns about the distribution of the network's nodes in the space and about their popularity, and then it adjusts the region boundaries (the arcs dividing neighboring regions of the same direction) by splitting overused and merging underused regions. This way, the nodes will be sensibly distributed between regions even if they are not evenly distributed within the metric space. As in experiments of

Section 3.4, the closest nodes (that are likely to include many neighboring nodes) are more important than the other nodes for the purpose of routing and they are contained in smaller cardinality regions. See an example of the dynamic partitioning in Figure 3.1, on the right. For more details on dynamic partitioning, see Appendix E.

3.5.3 Avoiding cycles and dead ends

Due to the topology, there are no dead ends in grid-based networks. There are also no cycles in the greedy route because CMH always has at least one neighbor closer to the target than CMH. To diminish the severity of the cycling problem in adaptive routing it was enough to explicitly prohibit cycles of length two and implicitly break larger cycles by means of exploration when a random expert is chosen anyway. In embedded networks, dead ends do exist and the absence of experts that bring immediate improvement is frequent. To overcome these difficulties both greedy and adaptive routing should be enhanced.

For greedy routing, the algorithm keeps in the message header all previously visited nodes of the path and check against them at every step. If all neighbors of CMH have been visited, the message backtracks until it reaches a node with at least one unvisited node. In other words, a node can be revisited only as part of the backtracking process. This is probably not an exhaustive search, because the minimal lookahead in the forward search is used.

For adaptive routing, we check at each node if it has been visited before (this information is available in the header). If it has, we will conduct an exploration step with probability 0.5. On the one hand, this probability is high enough to have a good chance of getting to a new neighbor; on the other hand, if there is only one neighbor that leads to the target we will not be avoiding it all of the time.

3.5.4 Embedded networks

We studied five embedded networks. Four of them are real data networks, and one of them is the anonymized grid-based (GB) network considered for the sake of comparison. All the real data sets come from GML files accessible on the web page [37] supported

by Newman. All directed edges are converted to undirected ones (with the exception of the grid-based network). Koren's algorithm can manage weights; it will treat them as a measure of similarity. Adaptive routing then should treat them in a different way: the reciprocal of an edge weight should be considered as edge contribution to the path length. However, we would then have difficulty comparing the results of adaptive routing to the results of greedy routing. Therefore, we decided to ignore edge weights to keep the analysis straightforward.

The political blogs network (PB) was constructed by Adamić and Glance [2] and depicts connections between political weblogs covering the 2004 US elections. The power grid network (PG) was constructed by Watts and Strogatz [49] and depicts the power grid of the western U.S. states. The autonomous systems network (AS) is a snapshot constructed by Newman [37] from GPB tables of University of Oregon Route Views Project for July 22, 2006. The condensed materials coauthorship network (CM) was constructed by Newman [38] and updated in 2005.

Some characteristics of the largest components of the networks are given in Table 3.2. The average SPD is calculated over all pairs of nodes of the network. The *coefficient of variation* is a ratio of the standard deviation to the mean; we consider it for the nodes' outdegrees. The effective diameter of 0.9, denoted as $D_{0.9}$, is defined by Tauro et al. [46] as a minimal threshold sufficient for 90% of the source target pairs of the connected component to be connected by the paths of length at most $D_{0.9}$. The presence of nodes of high degree and high standard deviation of node outdegrees (PB, AS and CM) makes networks look like star graphs, and wastes most of the "space" in the embedded space, see Figure D.1. When the maximum node degree in the graph is close to the average node degree (PG and GB), the nodes are spread out more uniformly across the space. We will see that learning curves for these two types of network are different as well.

We want to evaluate how well the adaptive routing performs. It is not enough to consider the overpayment rate only for real world networks. These networks are not as regular as the grid-based model networks, and the embeddings are estimated by the graph-

network	nodes n	average outdegree	maximum outdegree	outdegree standard deviation	coefficient of variation	average SPD	effective diameter $D_{0.9}$	diameter
GB	4096	4.68	5	0.51	0.108	8.12	9.83	17
PG	4941	2.67	19	1.79	0.671	18.99	26.87	46
CM	36458	9.42	278	13.19	1.400	5.50	6.68	18
PB	1222	27.36	351	38.40	1.404	2.74	3.33	8
AS	22963	4.22	2390	32.94	7.809	3.84	4.63	11

Table 3.2: Statistics of the largest connected components. The networks are sorted by the coefficients of variation (ratio of the standard deviation to the mean) of the nodes' outdegrees.

drawing algorithm. As a result, many more searches will fail because of reaching dead-ends or cycling (very few adaptive searches for the grid-based networks failed). We have to evaluate how many of the adaptive searches failed compared to the greedy ones. Therefore, instead of plotting progress of the failure rates we will introduce *failure-success deviation* (FSD) defined as:

$$FSD = \begin{cases} (f_{ad} - f_{gr})/(1 - f_{gr}) & \text{if } f_{ad} > f_{gr} \\ (f_{ad} - f_{gr})/f_{gr} & \text{otherwise} \end{cases} \tag{3.3}$$

Here, f_{gr} is a failure rate demonstrated by the greedy search for the first 100 thousand source-target pairs (the expected failure rate for the greedy routing does not change much with time); and f_{ad} is a failure rate demonstrated by the adaptive searches for the latest bucket. In other words, FSD shows how far f_{ad} deviated from f_{gr} towards the extremes. If all adaptive searches failed $FSD = 1$; if they all succeeded $FSD = -1$; and if f_{ad} is equal to f_{gr} then $FSD = 0$.

The progress of the overpayment rates and of FSD for the embedded networks is shown in the top row of Figure 3.3. The corresponding data for the averages after 3000 source-target pairs per node are shown in Table 3.3. PG and GB had sharp changes in the behavior similar to what was shown in the case of grid-based networks in Section 3.4, but the other networks (with high outdegree standard deviation) showed a more steady improvement. Adaptive routing showed improvement over greedy routing in the embedded networks with the overpayment rates comparable to (or even better than) the corresponding rates for grid-based networks, where the "real" coordinates of nodes are known. But, as

network	regions per node	maximum path length $d_{\max}$	average path length		ζ	failure rate		FSD
			adaptive	greedy		adaptive	greedy	
GB	28	144	19.85	47.09	0.30	0.0139	0.0746	-0.814
PG	28	150	27.52	59.19	0.21	0.0063	0.1378	-0.954
CM	32	229	76.30	180.9	0.40	0.1809	0.6459	-0.720
PB	24	105	6.84	19.11	0.25	0.0068	0.0585	-0.884
AS	32	209	43.30	77.98	0.53	0.1226	0.2215	-0.447

Table 3.3: Routing in the embedded networks. For the adaptive search, the data is given for the last bucket.

expected, the average path length and the rate of failure for the embedded networks are much higher than for the grids. The failure rates for adaptive routing were considerably better than for the greedy routing in 4 out of 5 embedded networks. Even for AS (the remaining fifth network) the failure rate for the adaptive was almost twice smaller than for the greedy algorithm.

3.5.5 Lazy nodes

In this section, we introduce the notion of lazy nodes and conduct experiments in a more adverse environment to find out the adaptive routing is still possible. A *lazy node* is a node unwilling to rate (or incapable of rating) its experts. As a result, the lazy node always behaves as if the current step is an exploration step and chooses a random neighbor to be the next CMH. However, it executes all the other actions that an ordinary node does: the lazy node forwards the message to the target immediately if the target is its neighbor, it accurately updates the message header with the routing information, and so on.

In the experiments of this section, each node is declared as a lazy with probability 0.1 independently at random. If we were conducting simple greedy routing in such networks, we would expect to see about 10% of the intermediate nodes on the path to be lazy nodes. However, in the adaptive search the fraction of lazy nodes seen decreases to the range of 5.7% to 8.7% (see Table 3.4). This is due to the fact that the neighbors of the lazy nodes detect the ineffectiveness of the lazy nodes and try to avoid them.

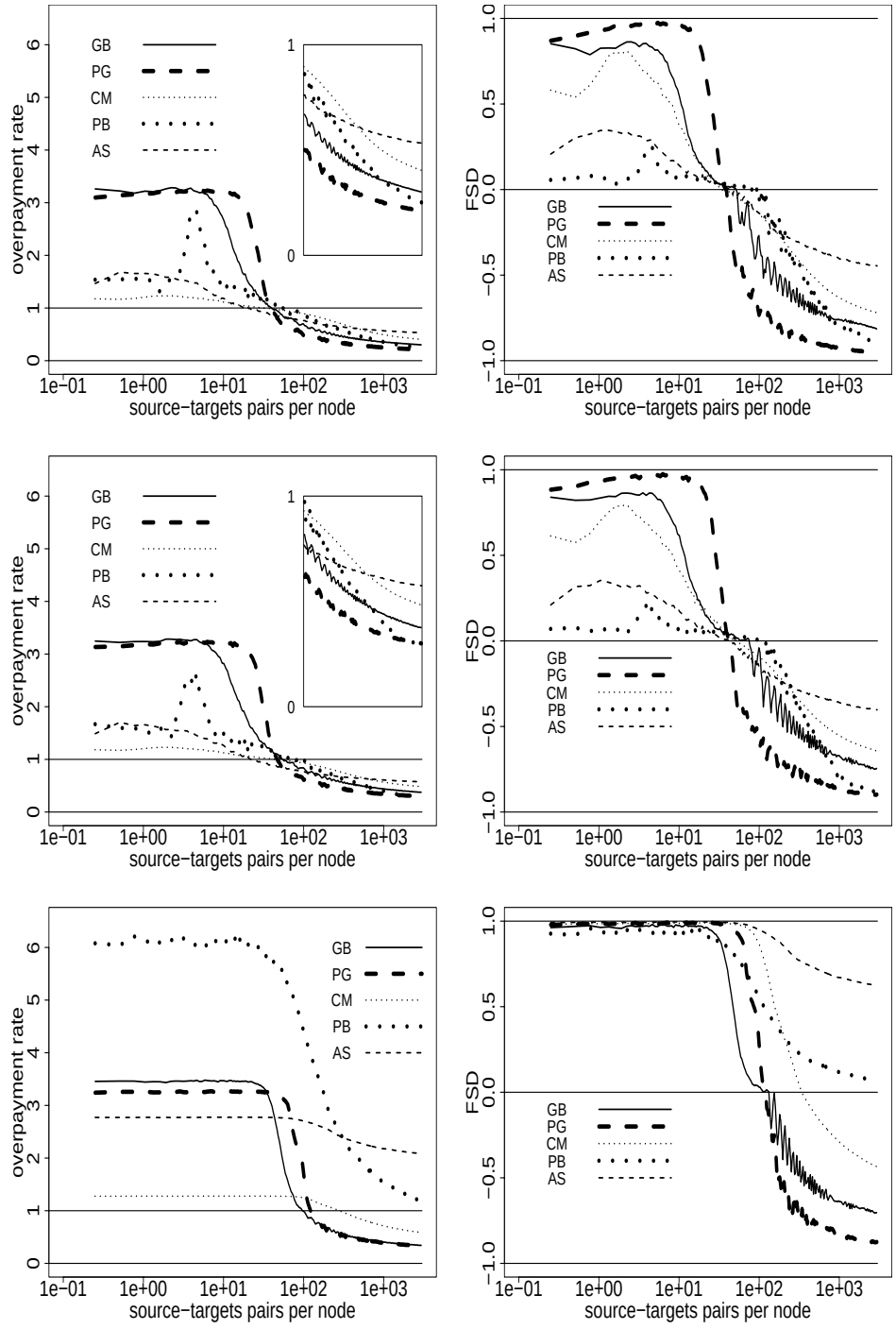

Figure 3.3: Overpayment rate ζ (left column) and FSD (right column) in the embedded networks. Regular setup (top), with lazy nodes (middle), with blind nodes (bottom). Inserts show details of crowded graph regions.

network	average path length				overpayment rate			lazy nodes usage rate
	greedy	adaptive						
		regular	lazy	blind	regular	lazy	blind	
GB	47.09	19.85	22.77	21.45	0.30	0.38	0.34	0.059
PG	59.19	27.52	31.07	32.68	0.21	0.30	0.34	0.057
CM	180.9	76.30	90.29	108.29	0.40	0.48	0.59	0.066
PB	19.11	6.84	7.45	21.97	0.25	0.29	1.18	0.087
AS	77.98	43.30	46.45	158.26	0.53	0.57	2.08	0.082

Table 3.4: Average path lengths and overpayment rates in the embedded networks: lazy and blind nodes. For the adaptive search, the data is given for the last bucket. The greedy searches are conducted on the networks without lazy and blind nodes.

Table 3.4 shows a comparison between the path lengths and overpayment rates for the networks with and without lazy nodes. Table 3.5 shows the corresponding data for the failure rates and FSD. In this section and the following section on the blind nodes, we refer to the searches without lazy nodes as regular adaptive searches; they are identical to the searches of Section 3.5.4. We gave a significant advantage to the greedy searches by conducting them without lazy nodes. PB and AS (the star-like networks, with the highest ratio of maximum node degree to the number of nodes) suffered the least significant impact in the overpayment and failure rates (compared to the regular data), even though they have the highest lazy nodes usage rate. The plots for the regular and lazy adaptive results in Figure 3.3 show that the corresponding curves are similar, with only change being, that the improvements for the networks with the lazy nodes happen slightly later and the final results for the networks for with lazy nodes are slightly worse. The data of Tables 3.4 and 3.5 demonstrates resilience of the adaptive routing and the absence of any sharp deterioration in the adaptive routing performance in the face of the substantial presence of lazy nodes.

3.5.6 Blind nodes

In the previous experiments, we were assuming that each node knew the exact locations of its neighbors. Now we assume that it does not, and only knows its own location and that of the target. This brings three disadvantages:

network	failure rate				*FSD*		
	greedy	adaptive					
		regular	lazy	blind	regular	lazy	blind
GB	0.0746	0.0139	0.0190	0.0221	-0.814	-0.745	-0.704
PG	0.1378	0.0063	0.0141	0.0172	-0.954	-0.898	-0.875
CM	0.6459	0.1809	0.2311	0.3647	-0.720	-0.642	-0.435
PB	0.0585	0.0068	0.0075	0.1233	-0.884	-0.872	0.069
AS	0.2215	0.1226	0.1323	0.7066	-0.447	-0.403	0.623

Table 3.5: Failure rates and *FSD* in the embedded networks: lazy and blind nodes. For the adaptive search, the data is given for the last bucket. The greedy routing searches are always conducted on the networks without lazy and blind nodes.

- Firstly, when CMH is one step away from the target it cannot detect this and send the message to the target automatically without choosing an expert. This has only a limited impact on large networks.

- The second disadvantage is that in the case of failure we cannot default to the greedy algorithm, which does not work in this environment at all. (Defaulting to the greedy algorithm, in case that the adaptive algorithm cannot reach the target in the maximum permitted number of steps, is an option that we did not use in any of our experiments.)

- The third disadvantage is much more severe. We cannot initialize the experts' weights, because their Euclidean distances to the target are unknown. Thus, all experts have an equal weight at the beginning and our search is similar to the random walk at first stages of the learning process.

We conducted experiments for the same networks as before, but all of the nodes were blind for the adaptive searches. It is impossible to conduct greedy routing for blind nodes, so we again used the regular environment for the greedy algorithm. As a result of the mentioned above difficulties (mostly of the inability to initialize the weights of the experts), almost all the initial searches fail. It takes a pair with the target close to the source or some luck for the search to succeed. However, once this happens, the resulting path updates the weights of the participating experts and it will help the following searches. Experiments indicate a critical number of successful searches after which there is a (sometimes sharp)

improvement in the rate of success. The networks with the lowest coefficients of variation of the node outdegrees (PG and GB) suffered the least from the blind nodes and eventually showed results compatible to the results of the regular environment. The star-like networks (AS and PB) suffered the most from the blind nodes and could not reach the performance of the greedy routing in the regular environment (at least in the experiments of 3000 source-target pairs per node). Average path length and failure rate progress for adaptive routing with blind nodes are shown in Figure 3.3 and Tables 3.4 and 3.5.

3.6 Summary

Our online adaptive decentralized routing performs well both in the studied real world networks and networks built according to the grid-based model. It outperforms the greedy algorithm and shows results somewhere in the middle between the shortest path distance and the greedy algorithm results. Both EEE and MWU models show similar behavior that scales well with the grid size for the grid-based networks. However, EEE performs better in the real world networks, which may be related to the fact that the original MWU should update all experts after each stage, and we can do it only for the chosen expert. The search performs reasonably well in the environments that contain either lazy or blind nodes. In the lazy nodes environment, the neighbors of lazy nodes automatically respond to the ineffectiveness of lazy nodes by avoiding them. In the blind nodes environment, networks with the lowest coefficients of variation eventually show results comparable to the results of the regular environment. We could overcome the difficulties presented by non-uniformly distributed nodes by using dynamically defined regions. Providing any bounds on the number of the source-target pairs necessary for the algorithm to converge seems difficult, because of the highly dynamic environment in the first stages of the algorithm, caused by the fact that each region of each node has its own ensemble of experts.

Chapter 4

COVER MODULARITY — OVERLAPPING COMMUNITIES

4.1 Introduction

Social networks are a result of human interaction. It is reasonable to assume that some of the networks are based on a particular type of relationship (a hobby, for an example). The increasing usage of the Internet gives an easy way to build and maintain social networks. The websites that are specially oriented to support such networks, such as MySpace and Facebook, enable creation (and monitoring) of gigantic networks with a complex structure. Closely related entities in social networks are called *communities*. The detection of communities can be useful in the achievement of a wide range of objectives starting from pure scientific interests to marketing to security surveillance. Many algorithms for community detection were suggested in the last decade (see the survey by Fortunato [20]), however only few of them can detect overlapping communities. We consider overlapping communities not as a border effect, where some of the entities from the fringe of the community might also belong to a fringe of another closely related community. We consider overlapping communities as a natural product of a network covering many types of relationships. This assumes that an entity might participate in many communities and an overlap between two communities might be significant, especially if the communities are interdependent.

4.1.1 Community

In the community-uncovering problem, we try to identify communities, that is, closely related sets of nodes in the network. The network is usually presented as a graph, and thus can be fully described by its adjacency matrix (edge weights might be allowed as well). We might have some additional general information about the network, such as how the data was collected, what the nodes and the edges represent, how many communities are expected to be found, and so on. However, we normally do not have any meaningful additional information about properties of specific nodes or edges, other than what is presented by a graph.

Identifying communities in a network presented as a graph is a difficult task. The problem begins with the fact that the notion of the community is in the eye of the beholder; it might be defined in different ways, or not be well defined at all. For example, one might see a community as something node-centric: a node (the center of the community) and nodes of the graph that are at up to a certain distance to the center. One might see a community as a non-extendable set of nodes that are up to a certain distance from each other. Finally, one might see a community as a set of nodes that are forming a non-extendable clique in the graph (or almost a clique, however "almost" might be defined).

The lack of clarity of the definition of what community is leads to the lack of clarity of the evaluation metrics for communities produced by a community detection algorithm. Given a network and its proposed communities, can we say how much we like them? This question is separate from that of whether these communities are the best possible ones for the particular network or not. Should the "quality" of a community be measured by the average node degree that counts only edges internal to the community? Should it be measured by the ratio of the community size (either the number of nodes or the number of edges) to the diameter of the community? One measure that appears to be reasonable is the conductance. *Conductance* of the community is the ratio of the number of edges that connect the community to the rest of the graph to the sum of the degrees of the community nodes. The measure was used by Leskovec et al. [34] to demonstrate that communities of size around 100 are the most pronounced communities in the real world networks.

4.1.2 The big picture

However, evaluating a single community of the network might be not enough. It is reasonable to assume that there are several communities in real world networks, and then we would like to be able to evaluate the quality of a collection of communities that are suggested for the network. The most popular measure for evaluating communities is called modularity (see Section 4.2.1); it was introduced by Newman and Girvan [41] [40]. It is defined for partitions (each node belongs to exactly one set), we will expand this notion to a more general notion, namely covers (see Section 4.2.2).

Newman [40] suggested an algorithm that puts initially all nodes in a single community and then recursively evaluates (based on the modularity) whether a community should be split into two sub-communities. Importantly, the algorithm decides by itself when to stop further splitting and does not require a predefined number of communities or their maximum size as an input. We will use our variant of this algorithm as a building block of the proposed later algorithm for the detection of overlapping communities.

Our algorithm for the detection of overlapping communities uses the idea of Gregory [26] that a node can be replaced by several clones that will represent the node in different communities. In our algorithm, the decision of how the clones are created is made locally by each node, thus reflecting its views on how its neighbors are organized in communities.

The absence of real world networks with known overlapping communities, forces us to introduce a model that generates a network with known overlapping communities ("ground truth groups") in Section 4.4.1. We will compare these communities against the communities produced by our algorithm by the means of Jaccard index (see Section 4.2.3).

To summarize, in Section 4.2, we will examine how to evaluate the quality of suggested communities. In Section 4.3, we will propose an algorithm that detects overlapping communities. In Section 4.4, we will propose a model that generates an artificial network with known overlapping communities, and we will examine results produced by the proposed algorithm.

4.1.3 Other previous work

A few algorithms for detecting overlapping covers of communities were suggested. Gregory [25] extended the notion of edge betweenness of Newman and Girvan [41] to the split betweenness, allowing not only the deletion of edges but also splitting nodes. At each iteration, the number of the shortest paths between all pairs of nodes going through each edge (*edge betweenness*) and going through each node (*split betweenness*) is calculated. The busiest element of the network is detected and transformed in the following way. If it is an

edge, it is removed. If it is a node, it is split into two clones. Gregory also suggested splitting nodes based on the split betweenness in Peacock [26], and then using any algorithm for the non-overlapping partitioning as a black box. Our approach is similar to Peacock, see discussion in Section 4.3.4.

Palla et al. [43] introduced one of the first (if not the first) algorithms for the detection of overlapping communities. They define a community as a continuum of neighboring cliques. To be more precise, two cliques of size k are neighboring cliques if their share $k-1$ nodes. Two k-cliques A and B belong to the community C if and only if there is a chain of neighboring k-cliques that leads from clique A to clique B. The community C should be non-extendable, that is, there is no k-clique D that does not belong to the community C and is a neighboring clique to any k-clique that belongs to the community C. The method has several problems:

- The notion of a clique is too restrictive.
- There is no way to fine-tune k, small changes of k lead to completely different results.
- k has to be a small number (normally from 3 to 6), not only because big cliques are rare in real world networks, but also because of the prohibitively high computational cost.
- k is the same constant across the network and it can lead to a problem if the degrees of nodes are coming from a wide range (a node with a low degree will never be accepted into a community if k is high, even if all of its edges are connected to the nodes of the same clique).

Baumes et al. [11] build communities bottom-up using their LA-IS2 algorithm, which optimizes the ratio of the number of edges internal to the community to the number of nodes of the community. It is unclear if it can detect a community with all nodes belonging to multiple communities. It seems like they consider the overlap as a small group

of boundary nodes that usually belong to two neighboring communities. However, the reality might be quite different; and in social networks, a node might participate in many communities (some of which can be unrelated to each other).

The fuzzy c-means algorithm by Zhang et al. [50] extends the idea of k-means clustering to overlapping communities by assigning a node to a community with some probability. The algorithm suffers from the same problems as those related to the standard k-means algorithm (k should be known, need of good seeds, clusters tend to be of the similar size).

4.2 Definitions

4.2.1 Modularity

Newman and Girvan [41] introduced a notion of *modularity* for a specific family of collections of communities, namely partitions. A collection (set) of communities forms a *partition* if each node of the network appears in exactly one community.

Definition 1. *Let P be a partition of a graph. Let i and j be communities of the partition P. Denote by e_{ii} the fraction of edges that are internal to the community i. Denote by e_{ij} the fraction of edges that connect the nodes of community i to the nodes of community j. The modularity Q of the partition P is defined as:*

$$Q(P) = \sum_{i \in P} \left[e_{ii} - \left[\sum_{j \in P} e_{ij} \right]^2 \right] \tag{4.1}$$

Newman used this notion later again [40] and considered a community as a collection of nodes that have more internal edges than they expected to have based on their degrees. He characterized a good partition as one which has "fewer than expected edges between communities". Newman [40] gave an equivalent definition of the modularity for a partition of a graph:

Definition 2. *The modularity Q of a partition P is defined as:*

$$Q(P) = \frac{1}{4m} \sum_{u \in V, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv}, \tag{4.2}$$

where m is the number of edges in the graph, A_{uv} are the elements of the adjacency matrix, k_u is the degree of the node u, σ_{uv} is 1 if the nodes u and v belong to the same community and is -1 otherwise.

The intuition behind this definition is easier to comprehend, and we will be using this definition throughout this dissertation.

Consider the case where we are dealing with a weighted undirected graph where each pair of nodes is connected and all loops are present, that is there are $n(n+1)/2$ edges. The nodes' degrees (reflecting weights of adjacent edges) are known upfront and assigned to the nodes. However, we do not know the weight of any individual edge. Our task is to assign the weights proportionally to the nodes' degrees. Consider edges adjacent to the node u. Let k_u be the degree of node u. Let k_v be the degree of node v, which is obviously a neighbor of u (it is a complete graph). Let m be the sum of all the edge weights in the graph. Edge $e = (u,v)$ should get v's share of u's degree. v's share is equal to $k_v/(\sum_{w \in V} k_w) = k_v/2m$. Applying v's share to u's degree, we get $k_u k_v/2m$. Note that we get the same term if we consider the edge e from the point of view of node v. This term appears as the expected weight of edge e in Formula (4.2).

For an unweighted undirected graph, $k_u k_v/2m$ becomes the likelihood of existence of edge $e = (u,v)$. It is not a probability, because $k_u k_v/2m$ is greater than 1, if k_u and k_v are greater than $\sqrt{2m}$. Many consider real world networks as sparse networks. It is reasonable to assume that a node's degree in such networks is bounded by $\sqrt{2m}$. Then the term $A_{uv} - k_u k_v/2m$ from Formula (4.2) is positive when the edge e exists, and is negative otherwise. That incentivizes assigning nodes connected by an edge to the same community, and assigning nodes not connected by an edge to different communities. The modularity also punishes when two nodes connected by an edge are in different communities, or when two nodes not connected by an edge are in the same community. Note, that a pair of low degree nodes connected by an edge (or a pair of high degree nodes not connected by an edge) is especially valuable, because of the higher absolute value of the term $(A_{uv} -$

$k_u k_v/2m$). The multiplier $1/4m$ was added to Formula (4.2) to match the original definition of modularity by Newman and Girvan [41] (see Formula (4.1)).

We will introduce the notion of *contribution of a node to the partition modularity* in order to understand better the notion of modularity, and extend it from partitions to covers in Section 4.2.2.

Definition 3. *The contribution of node u to the modularity of the partition P is defined as:*

$$Q(u,P) = \frac{1}{4m} \sum_{v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv} \tag{4.3}$$

It follows that:

Lemma 6. *The modularity of the partition P of the network $G = (V,E)$ is equal to the sum of the contributions of the nodes of V.*

$$Q(P) = \sum_{u \in V} Q(u,P)$$

Proof.

$$Q(P) = \frac{1}{4m} \sum_{u \in V, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv} = \sum_{u \in V} \left[\frac{1}{4m} \sum_{v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv} \right] = \sum_{u \in V} Q(u,P)$$

□

4.2.1.1 Bounds on the partition modularity

In the following, we will investigate the bounds of the partition modularity.

Lemma 7. *The upper bound on the partition modularity is 1. The lower bound on the partition modularity is -1.*

Proof. We will start by bounding the node contribution to the partition modularity from above. Taking into account that σ_{uv} is equal to either 1 or -1, and $A_{uv} \geq 0$, and $k_u k_v/2m \geq 0$:

$$Q(u,P) = \frac{1}{4m} \sum_{v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv} \leq \frac{1}{4m} \sum_{v \in V} (A_{uv} + \frac{k_u k_v}{2m})$$

$$= \frac{1}{4m}(\sum_{v\in V} A_{uv} + \sum_{v\in V} \frac{k_u k_v}{2m}) = \frac{1}{4m}(k_u + \frac{k_u}{2m}\sum_{v\in V} k_v) = \frac{1}{4m}(k_u + k_u) = \frac{k_u}{2m}$$

Then, for the partition:

$$Q(P) = \sum_{u\in V} Q(u,P) \leq \sum_{u\in V} \frac{k_u}{2m} = \frac{2m}{2m} = 1$$

Similarly, for the lower bound:

$$Q(u,P) = \frac{1}{4m}\sum_{v\in V}(A_{uv} - \frac{k_u k_v}{2m})\sigma_{uv} \geq \frac{1}{4m}\sum_{v\in V}(-A_{uv} - \frac{k_u k_v}{2m}) = -\frac{k_u}{2m}$$

$$Q(P) = \sum_{u\in V} Q(u,P) \geq \sum_{u\in V} -\frac{k_u}{2m} = -\frac{2m}{2m} = -1$$

□

The following lemma shows that the upper bound proven in Lemma 7 is asymptotically tight.

Lemma 8. *There is a family of graphs, which have partitions with the partition modularity converging to 1.*

Proof. Let k be a constant. Consider a family of graphs $G(k,l)$ that consists of l cliques, each clique is of size k, and there are no edges between nodes belonging to different cliques. Each clique will be represented by a single community. Let n be the number of nodes in the graph, $n = kl$. All nodes will have the same degree of $k-1$. Let m be the number of the edges in the graph, $m = (k-1)n/2$.

There are $k(k-1)\times n/k = (k-1)n$ ordered pairs (u,v) of distinct $(u \neq v)$ nodes that have an edge between them and belong to the same community (pairs are ordered because A_{uv} and A_{vu} are considered separately in Formula (4.2)). For each of them $\sigma_{uv} = 1$ and

$$A_{uv} - \frac{k_u k_v}{2m} = 1 - \frac{(k-1)^2}{(k-1)n} = \frac{n-k+1}{n}$$

Each node is also in the same community with itself $(u = v)$, but it does not have a loop. There are n nodes, for each of them $\sigma_{uv} = 1$ and

$$A_{uv} - \frac{k_u k_v}{2m} = 0 - \frac{(k-1)^2}{(k-1)n} = -\frac{k-1}{n}$$

There remaining $n^2 - kn = (n-k)n$ ordered pairs of nodes, which do not belong to the same community and do not have an edge between them (pairs are ordered because A_{uv} and A_{vu} are considered separately in Formula (4.2)). For each of them $\sigma_{uv} = -1$ and:

$$A_{uv} - \frac{k_u k_v}{2m} = 0 - \frac{(k-1)^2}{(k-1)n} = -\frac{k-1}{n}$$

Combining all three types of them, we have:

$$\begin{aligned}
Q(P) &= \frac{1}{4m}\left[(k-1)n\frac{n-k+1}{n} \times 1 - n\frac{k-1}{n} \times 1 - (n-k)n\frac{k-1}{n} \times -1\right] \\
&= \frac{1}{2(k-1)n}\left[(k-1)n\frac{n-k+1}{n} - n\frac{k-1}{n} + (n-k)n\frac{k-1}{n}\right] \\
&= \frac{1}{2(k-1)n}\left[(k-1)(n-k+1) - (k-1) + (n-k)(k-1)\right] \\
&= \frac{k-1}{2(k-1)n}(n-k+1-1+n-k) \\
&= \frac{1}{2n}(2n-2k) = 1 - \frac{k}{n} = 1 - \frac{1}{l}
\end{aligned}$$

So, for the family of graphs $G(k,l)$ and the corresponding "natural" partitions

$$\lim_{l\to\infty} Q(P) = 1$$

□

Lemma 9. *The lower bound on the partition modularity is* $-1/2$.

This better lower bound on the partition modularity came from Brandes et al. [13]. They also proved that finding a partition with maximum modularity is an NP-hard problem. In both cases they were using Formula (4.1) for the modularity.

Lemma 10. *For any graph, the partition that consists of a single community has modularity of 0.*

Proof. For any pair of nodes u and v, $\sigma_{uv} = 1$. Then from Definition 3:

$$Q(u,P) = \frac{1}{4m}\sum_{v\in V}(A_{uv} - \frac{k_u k_v}{2m})\sigma_{uv} = \frac{1}{4m}\left[\sum_{v\in V} A_{uv} - \sum_{v\in V}\frac{k_u k_v}{2m}\right]$$

$$= \frac{1}{4m}\left[k_u - k_u \sum_{v \in V} \frac{k_v}{2m}\right] = \frac{1}{4m}(k_u - k_u) = 0$$

From Lemma 6:

$$Q(P) = \sum_{u \in V} Q(u, P) = 0$$

□

Lemma 11. *There exists a graph that does not have a partition with positive modularity.*

Proof. Consider a triangle graph $G = (V, E)$, where $V = \{u, v, w\}$ and $E = \{e_1 = (u, v), e_2 = (u, w), e_3 = (v, w))\}$. Each node has degree 2 and $m = 3$, so $k_u k_v / 2m = 2/3$. There are three possible partitions.

Partition P_1: all nodes belong to the same community. According to Lemma 10, $Q(P_1) = 0$.

Partition P_2: each node has its own community. There are 6 ordered pairs of nodes connected by an edge and 3 nodes that do not have loops.

$$Q(P_2) = \frac{1}{4m} \sum_{u \in V, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv} = \frac{1}{12}\left[6(1 - 2/3) \times -1 + 3(0 - 2/3) \times 1\right]$$

$$= \frac{1}{12}(6 \times -1/3 + 3 \times -2/3) = \frac{1}{12} \times -4 = -1/3$$

Partition P_3 has two nodes in one community and the remaining node in another community. Only one edge is an internal edge.

$$Q(P_3) = \frac{1}{4m} \sum_{u \in V, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv}$$

$$= \frac{1}{12}\left[4(1 - 2/3) \times -1 + 2(1 - 2/3) \times 1 + 3(0 - 2/3) \times 1\right]$$

$$= \frac{1}{12}(4 \times -1/3 + 2 \times 1/3 + 3 \times -2/3) = \frac{1}{12}(-4/3 + 2/3 + -6/3) = \frac{1}{12}(-8/3) = -2/9$$

□

To summarize, the modularity enables comparison of partitions of the same graph, but in general, it is unclear whether a specific partition produces modularity that is close to

the best possible modularity of the network. The partition modularity of 0 may mean that the partition is meaningless, but it also may mean that the graph does not have a pronounced community structure. The random graphs created according to the Erdős-Rényi model [19] are unlikely to have a good community structure and are likely to have highest partition modularities close to 0. The maximum possible modularity cannot be higher than 1, but this level is not achievable for most networks. Newman and Girvan [41] report that real world networks with a community structure usually have partitions that produce modularities from 0.3 to 0.7.

4.2.2 Modularity and covers

It is not necessarily the case that the communities of a network have to form a partition. Consider a network where a node represents a person and an edge represents a relationship based on employment and/or a hobby. It might be preferable to create two partitions rather than one: one partition would be based on the employment relationships and another based on the hobby relationships. Reasonable communities might not form partitions at all; a person might be unemployed and have a few hobbies. It is unclear how to adapt the definition of the modularity to the *community cover* or just *cover* (a collection of communities that is not necessary a partition).

Suppose that node u belongs to the community C of the partition P. The contribution of node u to the modularity of the partition P depends on the community C only and does not depend on other communities of the partition P (see Formula 4.3 and the definition of σ). We would like to define the contribution of node u to the community C. However, the node u may belong to multiple communities in a cover. Therefore, we will have to introduce some definitions that may look excessive for the case of partitions, but are necessary for the case of covers. We will abuse notation and define σ_{uvC}. Let $C \subseteq V$, and $u \in C$, and $v \in V$. Then $\sigma_{uvC} = 1$ if $v \in C$, and $\sigma_{uvC} = -1$ if $v \notin C$.

We are now ready to define *the contribution of a node to the community modularity*.

Definition 4. *Let node $u \in C$. The contribution of the node u to the modularity of the community C*

$$Q(u,C) = \frac{1}{4m} \sum_{v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uvC} \tag{4.4}$$

Note, that for partitions $\sigma_{uvC} = \sigma_{uv}$, and as a result $Q(u,C) = Q(u,P)$.

Definition 5. *The modularity of the community C is the sum of the contributions of the nodes of the community to the community modularity:*

$$Q(C) = \sum_{u \in C} Q(u,C) = \frac{1}{4m} \sum_{u \in C, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uvC} \tag{4.5}$$

Lemma 12. *The modularity of the partition P is equal to the sum of the modularities of its communities:*

$$Q(P) = \sum_{C \in P} Q(C) \tag{4.6}$$

Proof. By Lemma 6

$$Q(P) = \sum_{u \in V} Q(u,P) = \sum_{u \in V} Q(u,C) = \sum_{C \in P} \sum_{u \in C} Q(u,C) = \sum_{C \in P} Q(C)$$

□

It is unclear how to proceed with modularity for a cover, which is not a partition. Consider a case where the community cover CVR consists of the communities of two partitions: P_1 and P_2. Each node u participates in two communities $C_1(u)$ of P_1 and $C_2(u)$ of P_2. We can calculate the contributions of the node u to the community modularities: $Q(u,C_1(u))$ and $Q(u,C_2(u))$, and define the modularity of the node in the cover CVR as the mean of the node's contributions to the community modularities (all communities in which the node u participates): $Q(u,CVR) = (Q(u,C_1(u)) + Q(u,C_2(u)))/2$. Without loss of generality assume that $Q(P_1) > Q(P_2)$. In this case, $Q(P_1) > Q(P_1 \cup P_2)$. Therefore, modularity of one of the partitions is greater than modularity of the cover consisting of the communities of the two partitions, as long as $Q(P_1) \neq Q(P_2)$.

Thus, the suggested definition of modularity will not allow a blind usage of the modularity maximization as a perfect tool for cover detection and evaluation. However, we do not have a better proposal for the node contribution to the cover modularity.

Definition 6. *The contribution of the node u to the modularity of the cover CVR is the mean of the contributions of the node u to the community modularities over all communities of the cover CVR to which the node u belongs:*

$$Q(u,CVR) = \frac{1}{|\{C : u \in C\}|} \sum_{C:u\in C} Q(u,C) \tag{4.7}$$

Definition 7. *The modularity of cover CVR is the sum of the contributions of the nodes of the graph to the cover modularity:*

$$Q(CVR) = \sum_{u\in V} Q(u,CVR) = \sum_{u\in V} \left[\frac{1}{|\{C : u \in C\}|} \sum_{C:u\in C} Q(u,C) \right] \tag{4.8}$$

Note that we use the same definitions of the community modularity (4.5) and of the node contribution to the community modularity (4.4) for partitions and for covers. The cover modularity and the node contribution to the cover modularity are defined in such way that partitions are just special cases of covers.

Nicosia et al. [42] used a genetic algorithm (with crossovers and mutations) to detect overlapping communities in networks with directed edges. To allow the overlap, a vector of nonnegative "belonging factors" is associated with a node, and the sum of the elements of the vector is equal to 1. The size of the vector is equal to a predefined (maximum) number of the communities in the network. An edge has a belonging factor as well, which is a (user defined) function of the corresponding belonging factors of the edge endpoints. More interestingly, they introduced a notion of modularity for overlapping communities based on the belonging factors as a fitness function:

$$Q_{ov} = \frac{1}{m} \sum_{c\in C} \sum_{u,v\in V} \left[\beta_{e(u,v),c} A_{ij} - \frac{\beta^{out}_{e(u,v),c} k^{out}_u \beta^{in}_{e(u,v),c} k^{in}_v}{m} \right],$$

where the β's are different belonging factors. Our definition of $Q(CVR)$ does not appear to be a special case of the above formula for Q_{ov}, but they both attempt to resolve the same issue.

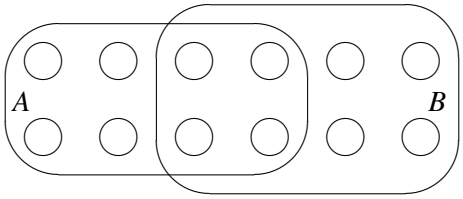

Figure 4.1: Jaccard index. $|A \cap B|/|A| = |A \cap B|/|B| = 1/2$, but $J(A,B) = 1/3$.

4.2.3 Jaccard index

Another measure that we consider to evaluate set (community) similarity is the *Jaccard index*, a generalization of which (known as cosine similarity) is widely used in text mining. We will use this measure to evaluate the quality of a proposed community, assuming that a ground truth community is known. However, it cannot be used as an objective function.

Definition 8. *The Jaccard index between two sets is a ratio of the cardinality of the intersection of the sets to the cardinality of their union, that is*

$$J(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}.$$

See an example in Figure 4.1.

We denote the cover of ground truth communities as $GTCVR$, and the cover of communities detected by the algorithm as $ACVR$. Ideally, we want both covers to have the same cardinality. Then we can define a bijective function f that maps $GTCVR$ to $ACVR$. However, $ACVR$ may have larger cardinality than $GTCVR$. Therefore, we will require f to be injective instead (we can always add empty communities to $ACVR$ if its cardinality is smaller than the cardinality of $GTCVR$). Now we can extend the notion of a Jaccard index from sets to covers.

Definition 9. *The Jaccard index between the ordered pair of sets* $(GTCVR, ACVR)$ *is*

$$J(GTCVR, ACVR) = \max_{injective f} \sum_{C \in GTCVR} w_C \times J(C, f(C)), \tag{4.9}$$

where w_C is the weight of the community C of the cover GTCVR with respect to the cover GTCVR: $w_C = |C| / \sum_{D \in GTCVR} |D|$.

Finding such an f can be done in the polynomial time (see Section 4.2.3.1), but instead we will be using upper and lower bounds that are easier to find.

It is much easier to find a function g from $GTCVR$ to $ACVR$ that maximizes the Jaccard index for every community of $GTCVR$: $g(C) = \arg\max_{D \in ACVR} J(C,D)$. In fact, f gives a solution that is inferior to g, because f is more restrictive (it has to be injective): $J(GTCVR,ACVR) \leq \sum_{C \in GTCVR} w_C \times J(C, g(C))$. The deficiencies of g are that multiple communities of $GTCVR$ can correspond to the same community of $ACVR$, and there is no penalty for having communities in $ACVR$ that do not correspond to any community of $GTCVR$ (the last fact is also a deficiency of f).

It is also computation-wise easy to find some injective function h from $GTCVR$ to $ACVR$, that does not necessarily produce the optimal Jaccard index, but tries to do it greedily. If the number of communities in $GTCVR$ is known, we will select the same number of communities from $ACVR$ and make h bijective (see Section 4.3.2.5 for the details on h). h is suboptimal, so: $J(GTCVR,ACVR) \geq \sum_{C \in GTCVR} w_C \times J(C, h(C))$. Thus, we can bound the Jaccard index for the covers from both sides:

$$\sum_{C \in GTCVR} w_C \times J(C, h(C)) \leq J(GTCVR,ACVR) \leq \sum_{C \in GTCVR} w_C \times J(C, g(C)), \qquad (4.10)$$

where h is injective and $g(C) = \arg\max_{D \in ACVR} J(C,D)$.

4.2.3.1 Exact matching

To recap, $G = (V,E)$ is the original graph, $n = |V|$, $m = |E|$. $GTCVR$ is a collection of the ground truth communities; we do not control this collection. Let $k = |GTCVR|$. In theory, $GTCVR$ can be the power set of V, but then the whole idea of the ground truth communities is pointless. We will assume that k is polynomial in n (although most reasonable covers will be $O(n)$). $ACVR$ is a collection of communities produced by the algorithm. The cardinality of $ACVR$ cannot be greater than the number of node clones produced by the algorithm, so

$|ACVR| \leq 2m$ (the algorithm will be described is Section 4.3). We need to find an injective function f from $GTCVR$ to $ACVR$. If $|ACVR|$ is less than k (which is unlikely), then we can add empty sets to $ACVR$. We are looking for f such that

$$J(GTCVR, ACVR) = \max_{injective f} \sum_{C \in GTCVR} w_C \times J(C, f(C)),$$

where w_C is the weight of the community C with respect to its cover:
$w_C = |C| / \sum_{D \in GTCVR} |D|$.

Let $l = \max(|GTCVR|, |ACVR|)$. If $|GTCVR| \neq |ACVR|$ we will add to the smallest collection the number of empty sets required to satisfy the $|GTCVR| = |ACVR|$ condition. We need to build a complete bipartite graph $K_{l,l}$. Each node in one part will represent a unique community of $GTCVR$. Each node in the other part will represent a unique community of $ACVR$. Each edge $e = (u, v)$ will have weight $w_{C_1} \times J(C_1, C_2)$, where u represents $C_1 \in GTCVR$, and v represents $C_2 \in ACVR$. Considering the facts that $|GTCVR|$ is polynomial in n, $|ACVR| \leq 2m$, and each community is no larger than n, the process of construction of $K_{l,l}$ can be done in polynomial time.

In order to solve Formula (4.9) and find f, it is enough to find a maximum matching in the bipartite graph $K_{l,l}$. This is a known problem (see it in Section 7.13 of the book by Kleinberg and Tardos [30]); the solution can be found in the polynomial time. That can be done in time polynomial in the size of the parts and thus polynomial in n. One part is connected to a newly introduced source node s, and the other part to a newly introduced sink node t. The exact maximum matching is constructed by finding a sequence of augmenting paths.

4.3 Algorithm

4.3.1 Overview

We suggest an approach that is somewhat similar to Peacock [26], in the sense that it uses a *non-overlapping algorithm (NOLA)* as a black box. We receive a network as an input,

which is represented by a graph (referenced as the *original graph* $G = (V, E)$ from now on). Our algorithm proceeds in the following stages:

1. *The vicinity graphs stage.* For each node, we create a vicinity graph (defined below). The vicinity graph of node u is split into non-overlapping communities according to NOLA. The communities are formed according how the node u sees the world. Our expectation is that if the node u participates in several communities of the original graph, then each of these communities will be represented by at least one community of the vicinity graph of u.

2. *The extended graph stage.* We build the extended graph (defined below) based on the communities produced in the previous stage. For each of the communities of the vicinity graph of node u, we create a clone of u. We run NOLA on the extended graph, and receive a community partition of the extended graph.

3. *The conversion stage.* We map the clones (the nodes of the extended graph) back to the nodes of the original graph, thus creating an overlapping cover of the communities instead of the partition.

4. *The refinement stage.* We adjust each of the communities locally, regardless of the rest of the cover.

5. *The selection stage.* We rank communities and might remove some of them from the final cover.

MACSIN (abbreviated after the title "Modularity and community structure in networks") is the modularity-based algorithm introduced by Newman [40]. We will be using a version of MACSIN as NOLA in all our experiments. See more about the original and our version of MACSIN in Section 4.3.3.

4.3.2 Full description

Let $G = (V, E)$ be the original undirected unweighted graph in which we attempt to detect potentially overlapping communities.

4.3.2.1 Vicinity graphs stage

Consider a social network where nodes represent people. A person knows only one's immediate acquaintances (comprising a small portion of the network), and some of the relationships between the acquaintances. Based on this information, the person might identify several types of relationships in which he participates and classify (label) each of the acquaintances accordingly. Each of the types of relationships might be considered as one's personal view on potential communities of the network in which the person participates. The view is usually partial, because the entire communities are unlikely to be known to an individual. However, as long as those views, although partial, are somewhat accurate, there is a chance of detecting these communities.

During the vicinity graphs stage, we create clones for each node. Each node is considered independently of other nodes. For each node u, we construct a *vicinity graph* $G_u = (V_u, E_u)$, where V_u is a set of neighbors of u in G (but not u itself); and there is an edge between two nodes $v \in V_u$ and $w \in V_u$ in E_u if and only if there is an edge between v and w in G. An example of a vicinity graph is shown in the second picture of Figure 4.2. Note that G_u can be a disconnected graph.

We can execute this stage of the algorithm in a parallel (distributed) manner, because we use the local information only and communities of one vicinity graph do not affect communities of other vicinity graphs. In general, we can build non-overlapping communities of G_u according to any NOLA. Vicinity graphs are much smaller than the original graph, so decomposition of a vicinity graph into communities is fast. In all our experiments, we are using our version of MACSIN [40].

Our expectation is that if a node participates in several communities, then its neighbors that belong to the same community are likely to have edges among themselves, and its neighbors that belong to different communities do not. Of course, if two communities are strongly overlapping, then it is difficult to make such distinction. Then again, if there is a

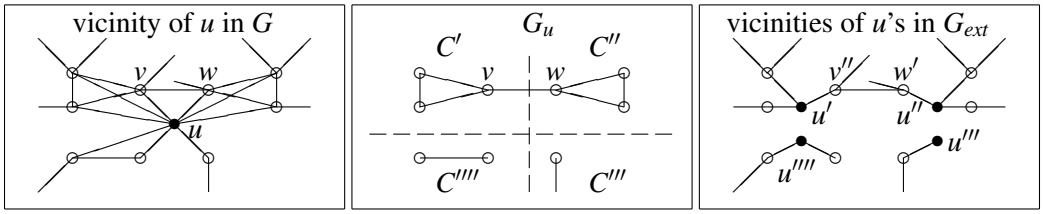

Figure 4.2: Vicinity and extended graphs. Node u and its clones in G_{ext} are drawn as filled disks. C', C'', C''', and C'''' are communities of G_u. They are respectively represented by nodes u', u'', u''', and u'''' in G_{ext}. Having the distinct nodes u' and u'' in G_{ext}, and not one node instead shows that the node u sees the nodes v and w as nodes from different communities. However, edges $e(u', v'')$, $e(v'', w')$, and $e(w', u'')$ show that the node v sees the nodes u and w as nodes of the same community, and the node w sees the nodes u and v as nodes of the same community. There can be other edges of G_{ext} that connect the clones of the node u; they existence depends on the vicinity graph of the node u and the vicinity graphs of nodes other than u.

non-annotated edge between a pair of nodes that participates in more than one community, how can we attribute the edge to one community or another (or several of them)?

4.3.2.2 Extended graph stage

During the extended graph stage, when we know (whether and) how the neighborhood of a node in G was partitioned into smaller communities, we can build G_{ext} — the *extended graph* of the graph G. Each node u of G is replaced by its clones in G_{ext}. The number of clones of u is equal to the number of communities into which we have partitioned V_u in the previous stage. Each clone of u represents a particular community of G_u and inherits only those adjacent edges of u, which connect u to its neighbors of this community. In this way, G_{ext} has more nodes than G, but the number of the edges remains the same. An example of clones of a single (original) node and their adjacent edges is given in the last picture of Figure 4.2. At this point, we can run any NOLA on G_{ext}. In all our experiments, we are using our version of MACSIN. After completion of the algorithm, we receive non-overlapping communities in terms of clones of the original nodes.

4.3.2.3 Conversion stage

At the conversion stage, we transform the communities of the extended graph to the communities of the original graph by mapping the clones of the original nodes back to the original

nodes. Importantly, this stage allows usage of the results of NOLA to create an overlapping cover instead of a partition. If two or more clones of the same node are present in the same community, they are merged into a single node. Thus, the number of communities in which a node participates is not completely defined by the vicinity graphs stage, which relies on local information only; the vicinity graphs stage just sets an upper bound on this number. For example, in the last picture of Figure 4.2, the nodes u' and u'' of G_{ext} (clones of the original node u) have a good chance to be merged back into a single node due to the short path through the nodes v'' and w'. The conversion stage is straightforward and fast.

4.3.2.4 Refinement stage

Starting this moment we are dealing with the original graph G only (for example, in the modularity calculations), and not with the extended graph G_{ext}. We refine a community based on the modularity of the community (see Equation (4.5)), which does not depend on other communities of the cover. For community C, we check whether its modularity per node can be improved by removing a single member of C, and remove all such members simultaneously (that is, remove $u \in C$ if $[Q(C \setminus \{u\})/(|C|-1)] > [Q(C)/|C|]$. We repeat this process until no improvement is possible.

Alternatively, we could refine communities by assuming that each member of the community should be connected to a relatively large number of other members of the community. We compare the *edge density* of the community (the ratio of the internal edges of the community to the cardinality of the community) to the edge density of its subcommunities. We run the following iterative process: at each iteration, we create a subcommunity by removing a node with the current lowest internal degree. At the end we replace the original community by its subcommunity with the highest (internal) edge density (unless the original community has the highest edge density). This process is a 2-approximation algorithm for finding a subgraph with the highest edge density [14].

In both cases of the refinement, we are considering a community contraction, not a community expansion. This is done not in order to limit the number of explorative branches,

but because we are dealing with overlapping communities. We expect a node to have edges that have nothing to do with a particular community of the node. We are likely to have more such edges in the case of overlapping communities than in the case of non-overlapping communities; those edges might belong to a different community of the node and lead to a false community expansion (and even to a merge of two overlapping communities into a single community). In all the experiments of Section 4.4, we refine communities based on the maximization of the modularity per node criterion.

At this point, given the ground truth communities, we can calculate the upper bound on the Jaccard index for the (ground truth and algorithm provided) covers at the end of the refinement stage according to Formula (4.10).

4.3.2.5 Selection stage

As in Section 4.2.3, we denote the collection of the algorithm produced communities as $ACVR$, and the ground truth cover as $GTCVR$. We rank the communities of $ACVR$ in decreasing order based on the community modularity per node $Q(C)/|C|$ (the same measure we used in the refinement stage). Given the ground truth communities, we could now calculate the lower bound on the Jaccard index for the (ground truth and algorithm produced) covers according to Formula (4.10). However, we need to find an injective function from $GTCVR$ to $ACVR$ first. Let us denote the number of communities in $GTCVR$ by N. We compare the highest-ranking unpaired community of $ACVR$ against all unpaired communities of $GTCVR$ and pair the best match (from the point of view of the Jaccard index between two sets). We continue the process until all communities of $GTCVR$ are paired. Thus, we actually create a bijective function from the N highest-ranking communities of $ACVR$ to $GTCVR$. (To get a corresponding injective function from $GTCVR$ to $ACVR$ we just need to change the direction of the correspondence.)

It would have made more sense to rank communities of $GTCVR$ instead. Nevertheless, we do not do this because in the real world we might not know $GTCVR$ itself, but know only the expected number of the communities in $GTCVR$ (that is N). Obviously, we cannot

calculate the lower bound for the Jaccard index in this case, but we can give the ranked list of only N communities as the answer. If $GTCVR$ becomes known later, a bijective function will be constructed in the same way it was done previously.

4.3.3 Details on MACSIN

4.3.3.1 The original version of MACSIN

First we will review the original version of the MACSIN algorithm, which was presented by Newman [40] (see Figure 4.3). MACSIN seeks to partition the nodes of the given graph $G = (V,E)$ in a way that maximizes the modularity. The MACSIN algorithm initially puts all the nodes into a single community, and then iteratively attempts to split any previously formed community that has not yet been split into two. This iterative process creates a binary tree. The leaves of the tree form the current partition. At each step of the algorithm, it is checked if a leaf of the tree should become an internal node (in other words, if a community of the current partition should be split). The iterative process stops when all the leaves were considered and rejected to be split. The split happens only if the modularity of the new potential partition is higher than the modularity of the current partition. This decision is local; it does not depend on the communities of the current partition except the one that is under consideration to be split (an independent proof of it comes from the discussion in Section 4.2.2, where it is shown in Lemma 12, that $Q(P) = \sum_{C \in P} Q(C)$ for partitions). Because of this, the order in which the leaves are considered for the split is not important, and each leaf should be considered only once. Finding a best possible split of a community is computationally expensive (finding a partition that provides maximum modularity is an NP-hard problem [13]). Instead, MACSIN looks for an approximate solution for each community.

Let us check how it works for the first split (when all the nodes start in the same community). Recall the formula for the partition modularity (4.2):

$$Q(P) = \frac{1}{4m} \sum_{u \in V, v \in V} (A_{uv} - \frac{k_u k_v}{2m}) \sigma_{uv}$$

Input: Graph $G = (V, E)$
create community C_0, add each node of V to C_0
create an empty partition P
add C_0 to P
unmark C_0 // it is a leaf
while (there is an unmarked community in P)
{
    pick an unmarked community $Leaf$
    find the potential split of $Leaf$ into communities $Child_A$ and $Child_B$
    // the potential split is based on the spectral qualities of the graph G
    form a candidate partition $P_{cand} = P$
    remove $Leaf$ from P_{cand}
    add $Child_A$ and $Child_B$ to P_{cand}
    if $(Q(P_{cand}) > Q(P))$
    // this is equivalent to **if** $((Q(Child_A) + Q(Child_B) - Q(Leaf)) > 0)$
    {
        remove $Leaf$ from P
        unmark $Child_A$ and $Child_B$ // they are leaves
        add $Child_A$ and $Child_B$ to P
    }
    else
    {
        mark $Leaf$ // it is a permanent member of P, should not be split
    }
}
Output: the partition P

Figure 4.3: The original version of MACSIN.

Newman uses the terms of the formula to form the matrix B, where $B_{uv} = (A_{uv} - k_u k_v/2m)$. He shows that the modularity satisfies

$$Q = \sum_i (w_i^T \cdot s)^2 \beta_i, \tag{4.11}$$

where w_i is i-th normalized eigenvector of B, β_i is its eigenvalue, and s is a vector whose components are either 1 or -1, depending to which of the two communities their corresponding nodes belong. Newman ignores all eigenvectors except w_1 (the largest one). Then he aligns s with w_1 in order to maximize $w_1^T \cdot s$ by assigning nodes with positive coordinates in w_1 to one community and the rest of the nodes to the other community. The split happens only if the modularity of the suggested partition is greater than 0. Or equivalently, the split happens only if β_1 in Formula (4.11) is positive.

This is exactly what is done in MACSIN. One only needs to switch to the subgraph induced by the community under consideration. In order to do so, the algorithm calculates the eigenvector corresponding to the largest eigenvalue of the matrix B with its elements defined as:

$$B_{uv} = A_{uv} - \frac{k_u k_v}{2m} - \delta_{uv}[k_u^{(g)} - k_u \frac{d_g}{2m}], \tag{4.12}$$

where g is the subgraph induced by the community under consideration for the further split, u and v are nodes of g, $k_u^{(g)}$ is the internal degree of u (only edges internal to g count), d_g is the sum of all node degrees (only edges internal to g count), δ_{uv} is the Kronecker's delta, and the rest is the same as in Equation (4.2). Note, that for the highest level (when the subgraph g is the whole graph), $B_{uv} = A_{uv} - k_u k_v/2m$.

4.3.3.2 Our version of MACSIN

We will be using our version of MACSIN as a NOLA (non-overlapping algorithm) in all of our experiments of Section 4.4. The main differences between our version and the original version of the algorithm are (in order of decreasing importance):

- We introduce a decomposition rate (defined below) to manage the progress of MACSIN (that is, to force the further split of communities).

- We use a transition matrix instead of matrix B.
- We consider three candidate splits instead of one.
- We do not do the final fine-tuning as part of NOLA, because the original fine-tuning was designed for partitions. We do community refinement as described in Section 4.3.2.4 instead. See [40] for the original final fine-tuning details.

4.3.3.2.1 Decomposition rate. In many of our experiments with artificially constructed networks and their ground truth communities, our algorithm often produces fewer than expected communities, and they are larger in size. The original version of MACSIN algorithm is unusual in the sense that the recursive process of the network decomposition stops automatically when the further partitioning does not increase the modularity, and the algorithm does not require an additional parameter that regulates the number or the size of the communities. Most of the community detection algorithms require such a parameter, but also claim that this allows multiresolution (see the discussion in [20]). Our version of MACSIN has such a parameter as well; unfortunately, we do not have a good intuitive description of it. We introduce a *decomposition rate* r (a real number that is usually around 0), that effectively allows the creation of a larger number of smaller communities. Consider a situation where we decide whether a parent community should be split into two nonintersecting subcommunities A and B. We split the parent community if $(Q(A)+Q(B))/Q(A\cup B)-1\geq r$. For the original MACSIN [40] the decomposition rate r would be 0 (MACSIN does not use the coefficient), but if it is less than 0 then a larger number of smaller communities is created. The decomposition rates for the vicinity graphs and the extended graph do not have to be the same.

Networks with overlapping communities have a legitimate reason to have a negative decomposition rate. When a node u participates in several communities, each of the communities has adjacent to u edges that are internal to one of the communities. This creates additional forces that try to keep the neighbors of u together. A negative decomposition rate helps to overcome these forces.

4.3.3.2.2 Transition matrix. Koren [32] used the *transition matrix* on random walks $D^{-1}A$ for graph drawing purposes, where D is a diagonal matrix where the diagonal entries are filled with the degrees of the corresponding nodes (the same matrix is used in the chapter on routing). We use the transition matrix instead of matrix B in order to assign coordinates to the nodes for sake of convenience. The transition matrix has an additional advantage due to the fact that real world networks are usually sparse. In this case, the transition matrix is sparse too, but matrix B is not (due to the term $k_u k_v/2m$ and other terms of Formula (4.12)).

4.3.3.2.3 Three candidate splits. For each community, we produce three candidate splits and chose the best of them (if any). The split is evaluated by how much further it increases the modularity. In the original MACSIN algorithm, the split will actually happen only if the split would increase the overall partition modularity, or in other words, the marginal modularity is positive. We choose that split (out of the three) which produces the largest marginal modularity (but it does not have to be positive anymore, because of the decomposition rate). All of the three splits are based on the coordinates of the eigenvector corresponding to the largest eigenvalue. The first candidate split is similar to the original MACSIN algorithm's split: nodes associated with the negative coordinates form one potential sub-community and nodes associated with the positive coordinates form the other community. In the second candidate split, the range of nodes' coordinates is partitioned into 10 equal intervals. The split will happen on the border of the least populated interval and its less populated neighbor. In the third candidate split, we split by the largest gap between the sorted coordinates of the nodes.

The first candidate split usually creates two sub-communities of comparable size. Let us consider a situation where we have a large network with one relatively small but obvious community (for example, the community is connected to the rest of the network by a single simple path) and many other communities that are interconnected between themselves and are therefore much less obvious. The coordinates of the nodes of the obvious community are likely to be concentrated on the one end of the range of nodes' coordinates.

The rest of the nodes will be much closer to the origin. The first candidate split might produce a bad solution because the nodes of the non-obvious communities will be affected by the nodes of the obvious community. It comes as a result of the strong repulsive forces between the obvious community and the rest of the network, which will shift most of the nodes away from the obvious community and some of the nodes that are very close to the origin will change their signs (while still being close to the origin). Therefore, it is useful to release the obvious community first and then proceed with the rest of the network. The second and third candidate splits are targeting to solve this scenario.

4.3.4 Comparison to Peacock

Gregory [26] proposed Peacock algorithm. It finds the node u with the highest split betweenness (the number of the shortest paths between all the pairs of nodes going through a single node), replaces it by two clones connected by a new edge, and distributes the original edges adjacent to u between the clones. It was not clear how the edges are distributed between the clones. Thus, the network is continuously expanding until the ratio of the maximum split betweenness to the maximum edge betweenness (the number of the shortest paths between all the pairs of nodes going through a single edge) is lower than a certain user-defined limit. When the network expansion is over, any NOLA runs on the extended graph. Both, the maximum split betweenness and the maximum edge betweenness are global numbers for the whole network and should be recalculated after each cloning.

Our algorithm is different from Peacock in several aspects. We list them here in order of decreasing importance:

- We consider only the immediate neighborhood of the node u in order to decide whether it should be cloned. That is, the cloning of any other node (including a neighbor of u) has no effect on the cloning of u. Our version is more distributed, initially each node judges by itself in which prospective communities it participates.
- We use the modularity instead of the split betweenness and the edge betweenness. The extent of the iterative split procedure is controlled by the decomposition rate.

- We use the same NOLA for the vicinity graphs and the extended graph stages.
- We do not connect the clones of the same node with new edges. We see a node participating in several communities as an actor playing several roles. Keeping these roles connected does not seem to make sense, because they can be orthogonal. This option was considered in Peacock, but was rejected because the global network was becoming disconnected quickly, and that was seen as a disadvantage (maybe because the original network was gradually expanding and the ratio of the maximum split betweenness to the maximum edge betweenness does not have meaning once the network becomes disconnected).
- We also introduce a selection stage, which ranks the communities and might filter out insignificant ones.

4.4 Experiments

For all the networks, which we consider in this section, we face three problems:

- Choosing the decomposition rate for the vicinity graphs.
- Choosing the decomposition rate for the extended graph.
- Deciding how many communities survive throughout the selection stage.

These problems do not allow us to claim that this process is truly unsupervised learning. The last problem is less severe, because we might be given an (approximate) number of the desired communities upfront, or we can claim that we provide a ranked (by the modularity per node) list of communities and the ground truth has several versions of resolution, so the user makes the final cut.

However, the notion of the decomposition rate is unlikely to make intuitive sense to the user. As a result of that, we will only show in our experiments that there are levels of decomposition rates that produce good results, but we do not know how to choose them without actual knowledge of the ground truth.

There is an alternative that we have tried, but did not pursue systematically. In this setup, the decomposition rates for the vicinity and the extended graphs are the same. The quality of results was not affected seriously, but they were slightly worse (in the terms of the Jaccard index). We could try several levels for the decomposition rates with the step of 0.01 and let the user choose which of the solutions is the best. The user is involved to a larger extent, but we do not need to know anything about the ground truth.

4.4.1 Community generation model — independent dimensions network

There are a few small networks with known ground truth community structure. However, these communities are not overlapping and they cannot be used as test cases to evaluate our algorithm. Below we introduce a model that generates data with known ground truth. We assume that a network describes a limited number of types of relationships; and that the types of these relationships are independent of each other. A node represents a subject participating in all these types of relationships; and an edge between two nodes is evidence that these nodes are connected in one or more type of the relationships. However, an edge can also connect two nodes that do not participate in any community together; those edges make the community detection more difficult.

We assume that the network is embedded in a low dimensional space, and that each dimension corresponds to exactly one type of relationship. Let dim be the dimension of the space. The dimensions are independent of each other. An *underlying group* (or just a *group*) is a set of nodes that participate in one type of relationship. That is, a group corresponds to exactly one dimension. It will also play a role of a ground truth community in the a posteriori analysis of the quality of results produced by the algorithm. Let N be the number of groups per dimension. For each of the groups, we assign a unique integral coordinate from the interval $[0, N-1]$. Each node is assigned to exactly one of these groups (of the same dimension) independently at random according to the uniform distribution. A node assumes its underlying group's coordinate as its own coordinate for the corresponding dimension. Let n be the number of nodes in the network, then the expected size of a group is n/N.

For each dimension, two types of edges might be created. An *intragroup edge* — an edge between two nodes of the same underlying group is created with the *intragroup probability* p_{intra}, independently of other edges. High intragroup probability allows formation of dense subgraphs representing groups. An *intergroup edge* — an edge between two nodes of different underlying groups (of the same dimension) is created with the *intergroup probability* p_{inter}, independently of other edges. Two nodes connected due to an intergroup edge of one dimension still can belong to the same group in another dimension (but it will be impossible to say due to which dimension an edge was created by looking at the resulting graph). Probability p_{inter} might depend on the distance between the coordinates of the underlying groups. High intergroup probability models close connection between two or few groups. Intergroup edges also can be used to model a noise.

The process is repeated for each dimension independently of other dimensions. We will call a network created according to the described above procedure as $IDN(dim, N, n/N, p_{intra}, p_{inter})$, named after *independent dimensions network*. Note that all edges between two groups of the same dimension, which were created due to all other dimensions, can be deemed as noise (as long as the dimensions are independent of each other). Multiple edges between a pair of nodes are merged into one edge.

The information about the chosen model and its parameters is hidden, unless explicitly stated otherwise. Ideally, the algorithm knows only the graph that represents the network. In reality, unsupervised learning is rarely successful.

We could have generalized the model in several ways, but we will not be using more complex constructions in most of our experiments. For example, we might have allowed an underlying group to be based on multiple types of relationships. Or, we might have allowed a node not to participate in an arbitrary number of types of relationships. Later we will allow some types of relationships to be interdependent, see Section 4.4.4.2.

We will evaluate the results of the experiments of this section in two ways. We will assume knowledge of the ground truth communities that form partitions(s). We will

calculate the lower and upper bounds on Jaccard similarity the way it was described in Section 4.3.2. We will also calculate partition modularities of the ground truth community and compare them to the modularities of the covers that produced the lower bounds on Jaccard similarity.

4.4.2 One-dimensional networks

4.4.2.1 One dimension, 50% uniform noise

4.4.2.1.1 Setup. The artificial network consists of 400 nodes. It has edges that represent one type of relationship. We create 20 underlying groups corresponding to the type of relationship. The distances between the coordinates of the nodes do not play any role in this experiment; the coordinates are only necessary to distinguish between the groups. Each node is assigned to one of these groups independently at random according to the uniform distribution. The intragroup probability p_{intra} is 0.5. The intergroup probability p_{inter} is a constant c_1 that is set to a level sufficient to obtain the number of intergroup edges equal to 50% of all the edges.

We created an instance of $IDN(1, 20, 20, 0.5, c_1)$. The groups' sizes varied from 12 to 29. If one is interested in identifying the underlying groups, then 50% of the edges in the network can be considered as uniformly random noise. The resulting average node degree was 20.6. The modularity calculated by Formula (4.6) for the partition matching the underlying groups was 0.443. Ideally, we want exactly 20 communities forming a partition to be selected.

4.4.2.1.2 Experiment. The decomposition rate in the vicinity and extended graphs was set to 0, the same as if it was used in the MACSIN algorithm. 3347 clones were created at the end of the vicinity graphs stage (8.37 per node). 961 communities with 2386 clones (5.97 per node) were created at the end of the refinement stage. But only 31 communities were of size 4 or larger. Now we can calculate the upper bound on the Jaccard index according to Formula (4.10); it was equal to 0.887. However, we can calculate the lower

network	average degree	modularity by groups		cover modularity	bounds on Jaccard index	
		dim A	dim B		lower	upper
1D 50% uniform noise	20.6	0.443	N/A	0.423	0.941	0.961
1D 50% intraedges	20.8	0.438	N/A	0.385	0.747	0.797
2D 100% intraedges	19.7	0.469	0.456	0.465	0.980	0.980
2D 67% intraedges	30.3	0.291	0.294	0.289	0.898	0.905
2D 50% intraedges	41.0	0.204	0.207	0.192	0.667	0.692

Table 4.1: Single tests data. Except of the first test, all intergroup edges are created with the probability inversely proportional to the distance between the groups.

bound only after the selection stage, because we need to create a bijective function and thus, use our knowledge of the number of groups. The lower bound was equal to 0.855. The modularity of the cover that produced the lower bound was calculated according to Formula (4.8) and equal to 0.397, which is not so far away from 0.443, calculated for the partition matching the underlying groups.

This network is relatively easy to decompose, because all the underlying groups are equally apart from each other and the intergroup edges are randomly created. This allowed good results without tuning the decomposition rates, and staying close to the MACSIN algorithm. Trying various decomposition rates allowed getting even better results. When the decomposition rates were set for the vicinity graphs to 0.1, and for the extended graph to -0.00945, lower and upper bounds on the Jaccard indices were 0.941 and 0.961; the modularity of the cover that produced the lower bound was 0.423. A summary of this and other single tests is in Table 4.1. Note, that the underlying groups are normally unknown and we only make the claim that levels of the decomposition rates that lead to good results exist.

4.4.2.2 One dimension, 50% intraedges

4.4.2.2.1 Setup. The setup is similar to the setup of Section 4.4.2.1 with one exception. p_{inter} is not a constant anymore, but is different for different pairs of nodes and depends on the distance between the groups (to which the nodes belong). The intergroup probability p_{inter} is such, that two nodes from different underlying groups have a chance to be

connected by an edge with probability inversely proportional to the distance between the corresponding groups. The total number of intergroup edges is at least the total number of intragroup edges.

We created an instance of $IDN(1,20,20,0.5,c_2/d)$. Where d is the distance between a pair of nodes, and c_2 is set to a level sufficient to obtain the number of intergroup edges equal to 50% of all the edges. This setup increases the number of edges between the neighboring (at distance one) groups and makes it difficult to separate them. There are 19 pairs of neighboring groups in the network. For each of the pairs, we compared the sum of the community modularities of the two groups with the modularity of the community comprised of the two groups. In all 19 cases, the former was smaller. That is, if we were using the MACSIN algorithm for a community formed by the union of any two neighboring groups, it would not split it into two communities corresponding to the groups. This is an example, that usage of decomposition rates different from 0 is necessary even for non-overlapping communities. For the overlapping communities, the necessity might be even higher, because intraedges created due to the two overlapping communities can be falsely treated as intraedges of a non-existing community (or a super-community), which is a union of the two communities.

To summarize, we created an instance of $IDN(1,20,20,0.5,c_2/d)$. The groups' sizes varied from 12 to 29. Almost 50% of the edges were intragroup edges, 18% of the edges connected nodes of the neighboring groups. The resulting average node degree was 20.8. The modularity calculated by Formula (4.6) for the partition matching the underlying groups was 0.438. Ideally, we want exactly 20 communities forming a partition to be selected.

4.4.2.2.2 Experiment. For the reasons mentioned above, we could not get good results when the decomposition rates for the vicinity and extended graphs were set to 0. To somewhat simplify the situation we kept the decomposition rate for the vicinity graphs set to 0 at first. In this case, the best results were shown when the decomposition rate for the extended

graph was set to -0.0594. 2545 clones were created at the end of the vicinity graphs stage (6.36 per node). 676 communities with 2031 clones (5.08 per node) were created at the end of the refinement stage. But only 67 communities were of size 5 or larger. The lower and upper bounds on the Jaccard index were 0.747 and 0.797. The modularity of the cover that produced the lower bound was calculated according to Formula (4.8) and equal to 0.385, which is not so far away from 0.438, calculated for the partition matching the underlying groups. Trying various decomposition rates for the vicinity graphs did not lead to better results.

4.4.3 Two-dimensional networks

4.4.3.1 Two dimensions, 100% intraedges

4.4.3.1.1 Setup. The artificial network consists of 400 nodes. It has edges that represent two types of relationships: A and B. In order to assign edges representing the relationships of A we create 20 underlying groups corresponding to this type of relationship. The intragroup probability p_{intra} is 0.5. The intergroup probability p_{inter} is 0. We repeat the same procedure for the relationships of B. Outcomes of the trials related to the relationships of A has no effect on the outcomes related to the relationships of B.

To summarize, we created an instance of $IDN(2,20,20,0.5,0)$. The groups' sizes varied from 10 to 30. When we are looking at nodes of some group of type A, we see that about a half of the edges are intraedges of the group and about a half of the edges are intraedges of the groups of type B. The relationships of B has nothing to do with the relationships of A; and we might think that the intraedges of the groups of type B are a uniformly random noise with respect to the group of relationships of type A, just like in Section 4.4.2.1. This is not exactly true (but very close to it), because the intraedges of the relationships of B have a structure.

The expected probability that two nodes belong to the same two underlying groups and have two edges (that will be merged to one edge) is about 1.25%. The resulting average node degree was 19.65. The modularity calculated by Formula (4.6) for the partitions

matching the underlying groups of a single dimension was 0.469 and 0.456. Ideally, we want exactly 40 communities forming two partitions to be selected. Those partitions would create overlapping communities.

4.4.3.1.2 Experiment. We kept the decomposition rate for the vicinity graphs set to 0 at first. In this case, the best results were shown when the decomposition rate for the extended graph was set to -0.02176. 1195 clones were created at the end of the vicinity graphs stage (2.99 per node). 81 communities with 806 clones (2.02 per node) were created at the end of the refinement stage. Only 37 communities were of size 4 or larger; at least three underlying groups were not covered well. The fact that the number of clones (806) is close to the number of the nodes times two (800) was an indicator that the selection stage would be easy. The lower and upper bounds on the Jaccard index were 0.912 and 0.916. The modularity of the cover was equal to 0.477, which is even higher than 0.469 and 0.456, calculated for the partitions matching the underlying groups. That can be partially explained by the fact that the underlying groups are defined before the edges are added to the network and this is a probabilistic process. The communities are identified after the edges are known.

Trying various decomposition rates allowed getting even better results. When the decomposition rates were set for the vicinity graphs to -0.01, and for the extended graph to -0.01042, the lower and upper bounds on Jaccard index were the same 0.98, the cover modularity was 0.465.

4.4.3.2 Discussion.

The three experiments that we have conducted so far have many things in common. They have underlying groups of the same expected size; they have the same resulting intragroup probability. About 50% of the edges adjacent to the nodes of any group connect the group to other groups. That explains the similarity in the average node degree and in the modularities of the partitions based on the underlying groups (see the first three lines of Table 4.1). Unlike the other two experiments, the middle experiment (where the neighboring groups are

strongly connected) represents a more difficult case. In this experiment, the cover modularity of the discovered (by the algorithm) communities and the Jaccard index were slightly lower. The problem of this experiment is that the neighboring groups are more interconnected; and it is difficult (if not impossible) to find the exact splits between the groups, even when the splits are forced by an appropriate value of the decomposition rate. In the following, we will conduct two experiments of similar setup, where the strong interconnection between the neighboring groups of the same dimension is combined with the presence of two dimensions in the networks.

4.4.3.3 Two dimensions, 67% intraedges

4.4.3.3.1 Setup. The setup is similar to the setup of Section 4.4.3.1 with one exception. The intergroup probability p_{inter} is equal to c_3/d. Here d is the distance between two groups corresponding to the pair of nodes (other dimensions do not affect d); and c_3 is set to a level sufficient to obtain the number of intergroup edges equal to 33.3% of all edges of the corresponding dimension.

To summarize, we created an instance of $IDN(2,20,20,0.5,c_3/d)$. There were 40 underlying groups; each node belonged to two underlying groups (one per each of the dimensions). If one is interested in identifying communities (corresponding to the underlying groups) of type A, then only $1/3$ of the edges in the network can be considered as the intragroup edges, $1/6$ as the distance-based intergroup edges, and $1/2$ as random noise (they are created due to the relationships of type B). The resulting average node degree was 30.25. The modularity calculated for the partitions matching the underlying groups of one of the dimensions was 0.291 and 0.294. Ideally, we want exactly 40 communities forming two partitions to be selected. Those partitions would create overlapping communities.

4.4.3.3.2 Experiment. To somewhat simplify the situation we kept only the decomposition rate for the vicinity graphs set to 0 at first. In this case, the best results were shown when the decomposition rate for the extended graph was set to -0.05783. 1938 clones were created at the end of the vicinity graphs stage (4.85 per node). 392 communities with 1528

clones (3.82 per node) were created at the end of the refinement stage. Only 60 communities were of size 5 or larger. The lower and upper bounds on the Jaccard index were 0.779 and 0.804. The modularity of the cover was equal to 0.266, which is close to 0.291 and 0.294, calculated for the partitions matching the underlying groups.

Trying various decomposition rates allowed getting even better results. When the decomposition rates were set for the vicinity graphs to -0.05, and for the extended graph to -0.04366, the lower and upper bounds on the Jaccard indices were 0.898 and 0.905; the cover modularity was 0.289.

4.4.3.4 Two dimensions, 50% intraedges

4.4.3.4.1 Setup. The setup is similar to the setup of Section 4.4.3.3 with one exception; there are even more intergroup distance-based edges. The intergroup probability p_{inter} is equal to c_4/d, where c_4 is set to a level sufficient to obtain the number of intergroup edges equal to 50% of all edges of the corresponding dimension.

To summarize, we created an instance of $IDN(2,20,20,0.5,c_4/d)$. There were 40 underlying groups; each node belonged to two underlying groups (one per each of the dimensions). If one is interested in identifying communities (corresponding to the underlying groups) of type A, then only $1/4$ of the edges in the network can be considered as the intragroup edges, $1/4$ as the distance-based intergroup edges, and $1/2$ as random noise (they are created due to the relationships of type B). The resulting average node degree grew (relatively to the previous experiment) to 41.01. The modularity calculated for the partitions matching the underlying groups of one of the dimensions went down to 0.204 and 0.207.

4.4.3.4.2 Experiment. Not surprisingly, the results were worse than the results of the previous experiment, because this network is more difficult. We could receive the best results that when the decomposition rates were set for the vicinity graphs to -0.03, and for the extended graph to -0.09744. In this case, the lower and upper bounds on the Jaccard indices were 0.667 and 0.692; the cover modularity was 0.192.

4.4.4 Serial tests

What are the limitations on the settings in which the proposed algorithm works for the proposed network generation model? If an average node of the community has too few edges that connect it to the other nodes of the community then it is difficult to identify the community. The larger the community the larger number of such (internal to the community) edges per node is expected. Another important factor is the contrast between the edges belonging to the community and the edges that connect the community to the rest of the network (*bridges*). It is not only a question of the numerical ratio. Increasing the number of bridges does not cause serious problems as long as these bridges lead to the far communities. There seems to be two main scenarios causing problems in community detection: two communities have many bridges between themselves, and two communities have a significant overlap (share many nodes). In both cases, it is difficult to decide whether there are two communities or just one.

4.4.4.1 Independent dimensions

We will consider here the first scenario: two communities have many bridges between themselves. In this section, p_{intra} is always a constant and p_{inter} is a function inversely proportional to the distance between the corresponding groups (these functions might be different in different experiments). In order to make some things obvious, we will be using p_{neigh} instead of p_{inter}. p_{neigh} is the probability that two nodes belonging to neighboring groups (the groups at distance 1) are connected by an edge, we call such edge a *neighbor-edge*. It can be defined alternatively as $p_{inter} = p_{neigh}/d$, where d is the distance between two groups corresponding to the pair of nodes (other dimensions do not affect d). Thus, we will work with networks of $IDN(dim, N, n/N, p_{intra}, p_{neigh}/d)$ family.

Consider for a moment that we are dealing with the one-dimensional case. Therefore, if we have two networks A and B with the same expected size of groups, the same p_{neigh} and the same p_{inter}, but the network A has a half the number of the groups of B. Then it will look like we took two networks of the size of A and added some additional edges to

obtain the network of the size of B. Alternatively, we can think that the network A grows into the network B, all this without changing p_{neigh} and p_{inter}, and without reassigning any existing edges. Of course, this will lead to the increase of the average node degree, and should make the community detection somewhat harder, because the fraction of intraedges is reduced.

If we have more than one dimension, then edges created due to the one dimension are created independently of the other dimensions, the same way it was done before. This will lead to the situation that the growth of the network by adding new groups in one dimension requires rewiring of the edges created due to the other dimensions. As a result, the average (node) overlap between two groups of different dimensions will be smaller. But anything else will mean that the dimensions are not independent.

In the following experiments, the number of dimensions is 2. The expected group sizes were 20, 40, and 80. The number of groups per dimension was 10, 20, 40, 80, and 160. We kept p_{intra}/p_{neigh} equal to 5. The number of groups per dimension, the number of dimensions and p_{intra}/p_{neigh} determine the resulting fractions of intragroup edges and neighbor-edges, and independent of the expected group size. We will denote the actual ratio of the number of the intragroup edges to the number of the neighbor-edges as r_{ion} (Intragroup edges Over Neighbor-edges). Thus, we created instances of $IDN(2,N,n/N,c,c/5d)$ family, where c is a monotonically decreasing function of the expected group size n/N.

We were targeting to receive similar high results for the Jaccard index, while keeping p_{intra}/p_{neigh} equal to 5, and saw that larger groups require a higher number of intraedges per node. However, it does not have to be a constant fraction of the group size. As the size of the groups grew, we could use smaller p_{intra}. See the settings of the related parameters in Table 4.2.

Table 4.3 gives ranges of the resulting fractions of the intragroup edges and the neighbor-edges across different networks. These fractions were calculated with respect to a specific dimension (that is, for a two-dimensional network we have two scores for the

group size	p_{intra}	p_{neigh}	expected number of intraedges created due to the corresponding dimension
20	0.50	0.10	9.5
40	0.35	0.07	13.65
80	0.25	0.05	19.75

Table 4.2: Settings that are independent of the number of groups. $p_{intra}/p_{neigh} = 5$.

groups per dimension	fraction of intragroup edges	fraction of neighbor-edges	ratio r_{ion}
10	0.316 - 0.330	0.180 - 0.198	1.64 - 1.83
20	0.265 - 0.269	0.131 - 0.142	1.89 - 2.05
40	0.224 - 0.233	0.106 - 0.111	2.07 - 2.18
80	0.194 - 0.201	0.087 - 0.091	2.21 - 2.26
160	0.175 - 0.178	0.076 - 0.077	2.30 - 2.35

Table 4.3: Data that is dependent on the number of groups.

expected group size	groups per dimension				
	10	20	40	80	160
20	0.786 - 0.798	0.836 - 0.840	0.791 - 0.818	0.791 - 0.818	0.786 - 0.805
40	0.812 - 0.820	0.793 - 0.830	0.787 - 0.802	0.803 - 0.819	0.814 - 0.828
80	0.826 - 0.887	0.852 - 0.875	0.834 - 0.849	0.842 - 0.850	0.815 - 0.823

Table 4.4: Bounds of Jaccard indices.

intragroup edges and the neighbor-edges). The resulting ratio of the intragroup edges to the neighbor-edges r_{ion} is significantly smaller than 5 (p_{intra}/p_{neigh}), because most of the groups have two neighboring groups in each dimension, and because edges created due to the other dimensions affect r_{ion} as well. However, the importance of the second factor diminishes to zero as the number of groups per dimension increases to infinity.

Table 4.4 shows the upper and lower bounds on the Jaccard index. For the smaller networks, we tried several decomposition thresholds, for the larger ones just few or one. The table shows that we could achieve similar results for the Jaccard indices for the different expected sizes of a group, the different numbers of groups per dimension by varying p_{intra} while keeping p_{intra}/p_{neigh} the same.

4.4.4.1.1 One parameter change series. We also conducted a series of experiments in which only p_{neigh} was different. The increase of p_{neigh} caused an increase of the number of

p_{neigh}	0.04	0.05	0.06	0.07	0.08	0.09	0.10
ratio r_{ion}	2.94	2.49	2.17	1.92	1.71	1.55	1.42
lower bound on J	0.928	0.879	0.831	0.793	0.774	0.706	0.645
upper bound on J	0.941	0.891	0.857	0.830	0.791	0.731	0.658

Table 4.5: Dependencies on p_{neigh}. $IDN(2,20,40,0.35,p_{neigh}/d)$.

p_{intra}	0.25	0.30	0.35	0.40	0.45
ratio r_{ion}	1.46	1.70	1.92	2.12	2.32
lower bound on J	0.368	0.643	0.793	0.916	0.931
upper bound on J	0.402	0.669	0.830	0.923	0.941

Table 4.6: Dependencies on p_{intra}. $IDN(2,20,40,p_{intra},0.07/d)$.

the intergroup edges and the nodes' degrees. As a result, it caused decreases in p_{intra}/p_{neigh} and r_{ion}. That makes the community detection more difficult and lowers the Jaccard index J. Some data can be found in Table 4.5. In all the networks: p_{intra} was 0.35, the number of dimensions was 2, the expected group size was 40, and the number of groups per dimension was 20. It was $IDN(2,20,40,0.35,p_{neigh}/d)$.

In a different series, we achieved a similar phenomenon of change of p_{intra}/p_{neigh} by changing only the p_{intra} parameter. The results are given in Table 4.6. In all the networks: p_{neigh} was 0.07, the number of dimensions was 2, the expected group size was 40, and the number of groups per dimension was 20. It was $IDN(2,20,40,p_{intra},0.07/d)$.

In both series we saw that higher r_{ion} produces a better Jaccard index, but we should remember that only one parameter was changing in each series. It cannot be used as a sole indicator of how difficult it is to detect communities. However, it confirms the role of the contrast between the intraedges and the neighbor-edges.

4.4.4.2 Codependent dimensions

In this section, we consider the case where a significant node overlap between communities makes it difficult to distinguish them as two communities, and not as a single community comprised of them. The *IDN* model makes it impossible to have a node overlap between groups corresponding to the same dimension, because they form a partition. Groups corresponding to the different dimensions are likely to overlap, but the overlap is insignifi-

cant unless the number of groups corresponding to a dimension is small, but the last fact makes the network less interesting. We will solve the problem by introducing a dependency between the dimensions and defining a *codependent dimensions network* (*CDN*). $CDN(dim, N, s, n/N, p_{intra}, p_{inter})$ has the same parameters as $IDN(dim, N, n/N, p_{intra}, p_{inter})$ with an addition of span s, whose role is defined below. In *CDN*, we consider exactly one dimension as a *base dimension* and distribute nodes among the group of the base dimension exactly the same way as it was done for *IDN*. Let s be the *span* of *CDN* and i be the coordinate of node v in the base dimension, then the coordinate of the node v in the dimension, which is not the base dimension, is an integer drawn independently at random from the uniform distribution on the interval $[max(i-(s-1)/2, 0), min(i+(s-1)/2, N-1)]$. We will call this interval the *actual span* of the group i. It is convenient (although not necessary) to limit s to the odd positive numbers. *CDN* can be considered as a generalization of *IDN*. Indeed, when $s \geq 2N-1$ the actual span of any group of the base dimension is $[0, N-1]$.

See Figure 4.4 for the illustrative example of the expected overlap for the span equal to 5. The size of a rectangle corresponds to the expected cardinality of the overlap between two groups of different dimensions. The actual spans for the first (and last) $(s+1)/2$ groups are smaller than s. This affects the expected size of the overlap for those groups, but not of the groups that are in the middle. Note, that expected sizes of the first (and last) groups of a non-base dimension are not n/N, but the rest of the groups have the same expected size as the groups of the base dimension. Small spans ensure large (usually about $1/s$) overlaps between a group of the base dimension and up to s groups of a non-base dimension.

We want to check how the algorithm works for networks with significant node overlaps. Therefore, we want to mitigate against the impact of the problem studied in Section 4.4.4.1, where the neighboring groups had many bridges. We can do so by decreasing p_{neigh} comparing to what it was set for in the experiments described by Table 4.2. p_{neigh} should be low enough to ensure that *IDN* produces a very high score for the Jaccard index, so the algorithm will have a chance at least for the cases with the high spans. We consider a series of networks that are produced by $CDN(2, 40, s, 40, 0.35, 0.05/d)$ for different

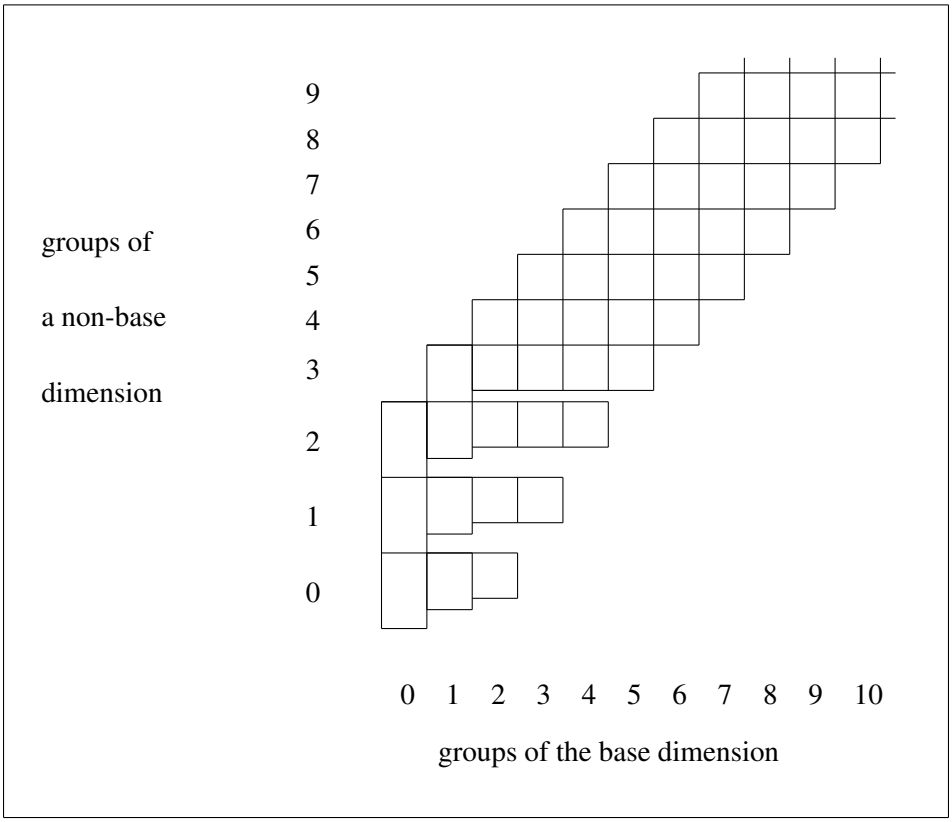

Figure 4.4: Expected cardinality of the overlap between groups of *CDN* with span 5. Size of a rectangle corresponds to the expected cardinality of the overlap.

values of s. The results are given in Table 4.7. We conducted experiments for only one decomposition threshold for the vicinity graphs for all networks. The average overlap line refers to the observed average overlap between the i-th group of the base dimension and the i-th group of the non-base dimension. It is close to $1/s$, as expected. The ratio r_{ion} is less important in this case; we are more interested in the node overlap between groups of different dimensions. Most of the edges between the nodes of the neighboring groups of one dimension were created as intraedges of the groups of the other dimension. There is a significant average overlap of 0.116 for the network with span of 11, and still the lower bound on Jaccard index is high (0.753). Even when more than a third of nodes of a group on average is shared with some other group (which corresponds to a span of 3), the lower bound on the Jaccard index is 0.488.

span s	*IDN*	21	11	7	5	3
average overlap	0.019	0.075	0.116	0.158	0.219	0.361
ratio r_{ion}	2.20	2.36	2.00	1.74	1.62	1.48
lower bound on J	0.883	0.849	0.753	0.643	0.556	0.488
upper bound on J	0.897	0.870	0.769	0.668	0.591	0.519

Table 4.7: Different spans for $CDN(2,40,s,40,0.35,0.05/d)$.

4.5 Summary

We consider MACSIN algorithm as an important building block of our algorithm for the detection of overlapping communities, but found it beneficial to extend MACSIN by adding a parameter (the decomposition rate) that regulates the degree of the iterative splitting of communities. This is true even in a network with non-overlapping communities. Consider the case where there are two obvious communities. Then start adding edges that connect these communities until at some point it becomes clear that the original communities become a single community. Somewhere in the middle of the process, there is a gray area where it is not clear if there are two communities or one. The MACSIN algorithm in this case says exactly when the switch happens. However, it seems that this decision should be subjective (or application dependent), and the decomposition rate is one mechanism that allows it.

In the case of overlapping communities, the situation becomes even more difficult. A pair of close but non-overlapping communities may be connected not only by the edges that were created because of the closeness of the communities (whatever close means), but also because some nodes of these two communities belong to a third community that overlaps with the first two communities. These nodes may be connected by additional edges that are legitimate intraedges of the third community. Thus, we see even a stronger than expected connection between the first two communities and it is difficult to separate them without using the decomposition rate.

We developed an algorithm that detects overlapping communities. The first stage of the algorithm that builds the vicinity graphs is crucial. It is based on locally made deci-

sions. We assume that an entity (node) may play different roles in the network; these roles lead the entity to different communities. The node is capable of evaluating its immediate surrounding and makes a decision to what potential communities itself and its neighbors belong. The final communities of the overall network are a composition of the views on the communities that are made by each node individually.

We conducted tests on many networks, including networks with non-obvious community structure. Even for these tests, we could get high Jaccard indices for the collections detected by the algorithm and the ground truth communities. As mentioned before, we do not know how to choose the best decomposition rates and actually think that it is subjective; we just show that there are rates that produce good results. We extended the notion of modularity from partitions to covers and saw that when Jaccard indices were high, the corresponding cover modularity (discovered by the algorithm) was similar to the partition modularities of the ground truth communities. We conducted serial tests and saw that our algorithm scales well. Larger communities require higher internal node degree (node degree that counts only intraedges), but the degree does not have to grow as fast as the size of the community. Finally, we conducted tests with a high pairwise overlap and still could see high Jaccard indices.

Chapter 5

EFFECTS OF EDGE REMOVAL ON NETWORK UTILITY

5.1 Network anonymization

We consider a problem of network anonymization while preserving utility. A company (*curator*) might be releasing information about its clients to subcontractors in order to analyze the data. However, the analysis should be performed based on the already collected data; and it does not require the knowledge of real identities that described in the data to be released. That allows the curator to protect the privacy of those who mentioned in the data. Nevertheless, an attacker dedicated to deanonymization may overcome this kind of protection because people tend to leave unique marks in their surroundings. It might be enough for the attacker to know a unique subset of attributes of a person in question (*target*) and find the target in the anonymized data. This was actually done in AOL case [10].

As a result, the curator minimizes the amount of information that is released, as long as the omitted data is inconsequential in the eyes of the curator. For example, the curator can release information which of the entities contacted each other through their email accounts, but the exact data stamps might be omitted. One of the most conservative ways of releasing information is to represent it as an anonymized graph without any annotation. For example, a node represents an email account and an edge is evidence that there was an email exchange between the corresponding accounts. Backstrom et al. [5] showed that even this scant information is prone to an attack. The suggested walk-based attack is not the main subject of the discussion, and we just notice that it is partially based on the detection of a path in the graph with a unique sequence of the nodes' degrees.

5.1.1 Utility

A possible way of protecting against this kind of attack might be perturbations in the topology of the graph. Those perturbations however affect most of the nodes (because we do not know what nodes are likely to be attacked). Even if the perturbations do not change the nodes' degrees significantly, they might however affect some other qualities of the network.

As an extreme example, the perturbation can turn a connected graph into a disconnected one.

Hay et al. [27] considered attacks based on the knowledge of the degree of the targeted node (or on the knowledge of the neighbors' degrees). They perturbed networks by deleting a number of edges uniformly at random and then adding the same number of edges choosing uniformly at random among pairs of nodes not connected by an edge. When 5–10% of the edges were replaced by other edges, network anonymity significantly increased (the adversary knowing the target node degree would find more suitable candidates in the perturbed network). They also reported change in network characteristics such as diameter, clustering coefficient, and so on, which affects the network utility.

Edge deletion is one of the major approaches to network perturbation. The perturbations should be done in ways that do not affect specific properties of the network. The problem is however, that other properties of the network might be seriously affected. We will not try to define what the utility of a perturbed network is. The utility seems to depend on a particular application in which the perturbed network is used. Even in this case, it is not easy to specify all qualities of the network that should be preserved. Releasing several perturbed networks of the same original network for different applications, creates a new source of a potential breach if an adversary tries to derive knowledge about the original network by detecting correspondence between nodes of the perturbed networks that represent the same node in the original network.

In the following, we will show that non-uniform deletion of a part of the edges can affect a utility of the anonymized network. More specifically, we consider edge deletion as a means of the network perturbation and use the stretch of the edge as the main factor in the decision whether an edge should be selected for deletion. We rate each existing edge by how much it "deserves" to appear in the network based on the network topology. Let edge e of graph $G = (V, E)$ be the edge under consideration. We construct graph $G_e = (V, E \setminus \{e\})$, which is identical to the original graph with the exception that the edge e is missing. For

edge $e = (u, v) \in G$, we calculate the length of the shortest path between the nodes u and v in G_e (that is, the *stretch* of e in G) and the number of such unique paths (any pair of these paths can overlap, but only partially). The edges that have shorter stretches have more "rights" to be in the graph G than the edges that have longer stretches. In case of the tie, the edges that have more unique paths supporting their stretch have more "rights" to be in the graph G than the edges that have less such paths.

We have chosen the stretch as a measure of edge oddness due to the simplicity of the calculations necessary to obtain the measure (we use a BFS-like approach). In Section 5.3.5, we will compare this measure to the commute time and see that their effects are quite similar. Throughout this chapter, with the exception of Section 5.3.5, we always assume that edge ratings are based on the stretch.

Consider a situation where the curator of the network asks a subcontractor to identify a set of nodes that play an important role in the routing, for example in the setting that is similar to the problem of routing in Section 3. Edges with high stretches are important in the routing because they often belong to many shortest paths. The edges with the highest stretch (that is, the "odd" edges) make the small-world networks small-world. Removal of edge $e = (u, v)$ with a high stretch is likely to lead to excluding of the nodes u and/or v from the list of the important nodes. We will run experiments on the real world networks of Section 3.5.4 to show that if we concern with the preservation of the short paths then the edges with the long stretch should be preserved during the edge removal.

On the other hand, if a subcontractor is asked to identify communities similarly to the problem in Section 4 (but not necessarily overlapping communities) then the edges with high stretch are not so important. Edges within a community are likely to have low stretch, otherwise the community is not as tightly knit as expected [18]. Thus, the edges with high stretches are likely to connect different communities; and their removal or perturbation will not change the community structure. If anything, it will make the community detection easier.

We will take a two-dimensional network from Section 4 and a one-dimensional network, those with a clear structure of communities (for which the algorithm could produce covers with Jaccard index of about 0.8 against the ground truth). We will delete either random edges, or edges with the lowest stretch, or edges with the highest stretch. We will see (at least when 10% of edges are deleted), that deletion of the edges with the lowest stretch (that is, the "natural" edges) leads to the deterioration of the community structure and should be used if we want to conceal it. If we want to preserve the structure then the edges with the highest stretch edges can be deleted harmlessly.

5.2 Intent

We do not make assumptions about powers of the adversary, and about what kind of prior knowledge the adversary has about the target. All we say is that edge deletion is a popular way of the network anonymization. In addition, we need to care about the preservation of utility while trying to induce anonymization [27].

It is difficult to talk about utility in general. When we concentrate on preserving one quality (by using the stretch as a device to make preference which edges can be most harmlessly deleted), we can hurt another quality more that expected.

One of methods of the network perturbation is the edge deletion. In this section, we suggest versions of a reasonable procedure to perform it. We will check the results of running this algorithm on different networks in the following sections.

5.2.1 Edge deletion procedure

If the deletion of an edge (but not its endpoints) transforms its connected component into two connected components then the edge is a *bridge*. An edge has an infinite stretch if and only if it is a bridge. We will call a node *depleted* if it has lost a half or more of its original edges. The *edge deletion procedure* is given in Figure 5.1.

Exclusion of bridges is introduced to prevent an immediate and unavoidable creation of new disconnected components. As a (partial) result, nodes with a degree of one never lose their edges. A new disconnected component can be considered as a major al-

```
// input: k_max — number of edges to be deleted
create EdgeList — the ranking list of the edges of the largest connected component
remove bridges from EdgeList // Exclusion of bridges
k = 0
while k < k_max
    edge e = (u, v) is a top of EdgeList
    remove e from EdgeList
    // Exclusion of edges of depleted nodes
    if deletion of e makes neither u nor v a depleted node
    then
        delete edge e
        k = k + 1
    end if
end while
```

Figure 5.1: Edge deletion procedure.

ternation of the network topology. Also we do not want to reduce the size of the largest component because we want to deal with largest connected components in the original and its perturbed networks of approximately the same number of nodes when we are comparing different metrics (for example, the average *SPD*). Although we do not allow removal of bridges, it does not mean that the connected component would not be split under any circumstances. That may happen because the stretch is calculated with respect to the original network and not with respect to the current network where some of the edges were already removed. Considering the stretch of the current network would be a better approach, but it requires additional computations.

Exclusion of edges of depleted nodes is introduced to prevent a significant change of the network topology around a single node. That is, a half or more of adjacent edges of any node must survive the deletion. That means (among other things) that nodes of the degree two do not lose any edges. Making this threshold (of 50%) of the surviving edges

substantially higher would mean that only an edge adjacent from both of its sides to nodes with higher degrees could be deleted.

The ranking in our experiments will be done in one of three orders: ascending, descending, or random. The descending order gives the highest rank to an edge that connects seemingly the most unrelated nodes. The ascending ranking is the reverse of the descending ranking. The random order associates with an edge a number uniformly drawn from the same range, and ranks the edges according to these numbers. The following is a description of the descending order:

- Rank the edges according to their stretch. Longer stretch means higher rank.
- In the case of the tie, rank the edges by the number of the unique paths supporting the stretch. Fewer paths means higher rank.
- Break the remaining ties randomly.

When we talk about the descending and ascending orders of deletion, we always imply the edge rating based on the stretch, unless it is explicitly stated otherwise, like in Section 5.3.5.

5.3 Impact on the shortest paths

To affect the routing opportunities in the graph the most, we should concentrate on the edges with the highest stretch (these edges are deleted first when the descending order of deletion is used). Those edges are expected to participate in many shortest paths of the graph and should play an important role in the formation of the small-world phenomenon. In the grid-based model, for example, some of the long-range edges play this role. The proof of the upper bound on the expected path length for the greedy routing is based on the expectation to encounter a useful long-range contact (mentioned in Lemma 1 of Section 2.2.2, proved in [28]).

We use the embedded networks from Section 3.5.4 for our experiments. See the description of the networks in Table 5.1. It repeats Table 3.2 with one exception. The grid-based network was converted to an undirected graph by adding the necessary edges. The

network	nodes n	average degree	maximum degree	degree standard deviation	coefficient of variation	average SPD	effective diameter $D_{0.9}$	diam-eter
GB	4096	5.42	11	1.03	0.190	6.68	7.86	12
PG	4941	2.67	19	1.79	0.671	18.99	26.87	46
CM	36458	9.42	278	13.19	1.400	5.50	6.68	18
PB	1222	27.36	351	38.40	1.404	2.74	3.33	8
AS	22963	4.22	2390	32.94	7.809	3.84	4.63	11

Table 5.1: Statistics of the largest connected components. Undirected graphs. The networks are sorted by the coefficients of variation (ratio of the standard deviation to the mean) of the nodes' degrees.

rest of the networks were already undirected and remained unchanged; their data is repeated here for the sake of convenience.

5.3.1 Grid-based network

We start with the *grid-based* network (GB), which was converted to an undirected graph. In GB, any pair of nodes has at least two simple edge-disjoint paths connecting them. Most of the nodes of the network have a degree of five or six, and the standard deviation of the node degree is very small. The local edges have the stretch of three at most, and the stretch of the long-range edges may vary considerably.

We remove a fixed fraction of the edges from the network according to the edge deletion procedure. We conduct three experiments for each of the chosen fractions. In the first experiment, we will choose edges for the deletion uniformly at random regardless of their stretch; this will serve as a reference point. In the second experiment, we will delete edges with in the descending order (highest stretch first), and in the last experiment we delete edges in the ascending order. For each experiment, we calculate an average *shortest path distance* (SPD) for a hundred thousand pairs of nodes chosen uniformly at random.

The results are given in Table 5.2. The first line shows the results for the original network (with no deleted edges, so the results for the all orders of deletion are identical). Let SPD_0 be the average *SPD* for the original network, and SPD_{desc} (SPD_{asc}, SPD_{rand}) be the average *SPD* for the network perturbed in the descending (ascending, random) order. The *increase of SPD* columns show by how much the average *SPD* grew after the edge deletion

($SPD_{desc}/SPD_0 - 1$ for the descending order, for example). There is an obvious difference in the increase of *SPD* between the random and the descending orders of edge removal for these sets of experiments. The *ratio* columns show this difference. The *ratio of descending order* or r_{desc} column is a ratio of increases of *SPD* between the descending and random orders $r_{desc} = (SPD_{desc}/SPD_0 - 1)/(SPD_{rand}/SPD_0 - 1) = (SPD_{desc} - SPD_0)/(SPD_{rand} - SPD_0)$, because the random order serves as a reference point. Similarly, *ratio of ascending order* or r_{asc} column is defined as $r_{asc} = (SPD_{asc} - SPD_0)/(SPD_{rand} - SPD_0)$.

In general, we expect the function r_{desc} of fraction of deleted edges to be close to a concave function. When too few edges are deleted, then *SPD* is not seriously affected by the order of deletion, because the number of shortest paths in which these edges participate is small anyway. If too many edges are deleted, then we might be dealing with similar sets of deleted edges (although they do not have to be identical, because of the interplay between an order of deletion and the exclusion of edges of depleted nodes).

We will be mostly talking about r_{desc} and only notice that r_{asc} was always less that 1 for GB, which means that the difference between the descending and ascending orders is even larger. When 10% of the edges were removed in the descending order *SPD* grew up by almost 65% and r_{desc} was above 10. We received this high ratio because most of the edges are local edges and they extremely bad suited for the creation of the small-world network; and many of the long-range edges have a high stretch.

The *max* row of Table 5.2 shows what happens if we delete all qualified edges (they should not satisfy the aforementioned exclusions). The real fraction of the deleted edges cannot be higher than 50%, because of the exclusion of edges of depleted nodes. This row can be used as a vague upper bound of what can be achieved by the edge deletion procedure.

5.3.2 Power grid network

The two exclusions in the edge deletion algorithm (prohibiting deletion of bridges and of edges of depleted nodes) did not affect seriously the sets of deleted edges for GB network.

fraction of deleted edges	average *SPD*			increase of *SPD*			ratio	
	asc.	rand.	desc.	asc.	rand.	desc.	asc.	desc.
0	6.679	6.679	6.679	N.A.	N.A.	N.A.	N.A.	N.A.
0.01	6.693	6.720	6.939	0.002	0.006	0.039	0.348	6.364
0.03	6.731	6.793	7.532	0.008	0.017	0.128	0.453	7.437
0.1	6.920	7.095	11.14	0.036	0.062	0.668	0.581	10.72
max	8.511	9.294	28.74	0.274	0.392	3.303	0.701	8.434

Table 5.2: Grid-based network perturbation.

This was not a problem because of the abundance of the backup paths in GB and because most of the nodes had four local edges with a low stretch and high stretch edges were relatively evenly distributed. This is not true for real world networks. Even with the exclusions, the size of the largest component of *power grid* network (PG) changed considerably in the case of in the descending order of deletion. Only 2749 out of 4941 nodes survived when 10% of edges were deleted in the descending order. (In other networks, we never lost more than 3% of nodes.) Cases like this may happen, because we calculate all the stretches in the original network only, and not in the current network. We could have checked whether the edge which is about to be deleted creates a new component online, but it would cost $O(m)$ per edge, and $O(m^2)$ totally, and we decided not to do it. An idea that gives even more precise data is to recalculate all stretches after each edge deletion, but it would cost even more. Partially, the loss of the component size for PG can be related to a very low average node degree in the original network, which is equal to 2.67.

The results of experiments are shown in Table 5.3. Again, we see significant differences in the increase of *SPD* between the random and the descending orders of the edge removal. Moreover, it strongly affects *SPD* even with a low fraction of deleted edges. Even when only 1% of the edges were removed in the descending order and 97% of the nodes did not lose any adjacent edges, *SPD* grew by 18.4% (versus 1.2% in the random order case). Again, r_{asc} was always less that 1 for PG, which means that the difference between the descending and ascending orders is even larger.

fraction of deleted edges	average *SPD*			increase of *SPD*			ratio	
	asc.	rand.	desc.	asc.	rand.	desc.	asc.	desc.
0	18.99	18.99	18.99	N.A.	N.A.	N.A.	N.A.	N.A.
0.01	19.01	19.21	22.48	0.001	0.012	0.184	0.107	15.76
0.03	19.20	20.34	26.33	0.011	0.071	0.387	0.158	5.430
0.1	22.51	25.32	34.04	0.185	0.333	0.793	0.556	2.379
max	36.10	39.88	40.63	0.901	1.100	1.140	0.819	1.036

Table 5.3: Power grid network perturbation.

5.3.3 Networks with hubs

Here we consider the remaining three networks: the *condensed materials* coauthorship network (CM), the *autonomous systems* network (AS), and the *political blogs* network (PB). We got the striking difference in the increase of *SPD* between the random and descending orders of the edge deletion for these networks. However, the magnitude of the increase was smaller, although for two of them (CM and AS) the difference was still significant. The key to the problem may lie in the presence of *hubs* (nodes with high degrees) in these networks (there were no hubs in GB and PG).

Hubs also present a problem in the anonymization process; they are easily identifiable in an unlabeled network based on the node degree. Moreover, the node degree is often considered as the most accessible information (to the adversary), for example in [27]. Being able to identify hubs in the anonymized network may lead to the identification of many other nodes, assuming that the adversary may have knowledge about the presence of edges between the target nodes and the hubs. If there are many hubs, the connections from the hubs to a node may often work as a unique signature of the node.

To be more specific, we might say that if a node has a degree higher than a *hub threshold* then it is a hub. It makes sense in this case to convert hubs into a regular node overtly at the beginning of the anonymization process and only then apply the edge deletion procedure of Section 5.2.1. The *hub conversion procedure* is conducted as the following. We remove an edge between a node with the highest degree and its neighbor with the

while at least one hub remains in the network
    identify node u with the highest node degree in the network.
    identify node v, a neighbor of u with the highest node degree.
    delete edge $e = (u, v)$.
end while

Figure 5.2: Hub conversion procedure.

highest degree iteratively; until no hubs are remained in the network, see Figure 5.2. We considered two alternatives to this procedure, but they had shortcomings. A more drastic alternative would be to remove all the hubs with all their edges. However, this can lead to a severe reduction in the size of the largest component (more that a half of nodes were lost in one case). A smaller impact alternative would only delete the edges between the hubs, but preserve the edges between the hubs and the rest of the nodes. However, that resulted in preservation of some of the hubs (they preserved most of their edges), which defeats the purpose of the hub conversion procedure.

The question what constitutes the hub threshold is arguable, but it seems that if a node is connected to the square root (or more) of the nodes in the largest connected component then it is definitely a hub. In this case, the hub threshold for CM is 190, for AS is 151, and for PB is 34. The conversion of hubs led to the decrease of the largest component of the networks and of their average node degree. Some data for the largest components of the original networks (annotated as having the hub threshold equal to ∞) and networks formed by the conversion of hubs into regular nodes is given in Table 5.4. The conversion of hubs affected PB the most, which is result of the very high level of the average node degree (more than 27) relatively to the small number of nodes (1222).

Alternatively, a hub should be defined as a node that that have more neighbors than the average node degree plus some number of standard errors of the node degree. This definition might be more suitable for PB. In the following sections were using the first definition.

network network	hub threshold	number of hubs	largest component size	average degree	average *SPD*
CM	∞	0	36458	9.42103	5.49924
	190	11	36458	9.39936	5.50617
	100	115	36458	9.22102	5.59007
AS	∞	0	22963	4.21861	3.83999
	151	52	18489	3.18178	5.94757
PB	∞	0	1222	27.3552	2.73706
	34	321	1222	14.2111	3.43892

Table 5.4: Networks before and after removal of hubs

hub thresh- old	fraction of deleted edges	average *SPD*			increase of *SPD*			ratio	
		asc.	rand.	desc.	asc.	rand.	desc.	asc.	desc.
∞	0	5.499	5.499	5.499	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	5.586	5.687	6.229	0.016	0.034	0.133	0.465	3.890
	max	6.631	6.923	7.731	0.206	0.259	0.406	0.795	1.568
190	0	5.506	5.506	5.506	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	5.589	5.699	6.246	0.015	0.035	0.134	0.432	3.846
	max	6.666	6.931	7.739	0.211	0.259	0.406	0.814	1.567
100	0	5.590	5.590	5.590	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	5.670	5.789	6.300	0.014	0.036	0.134	0.400	3.574
	max	6.799	7.040	7.917	0.216	0.259	0.416	0.834	1.605

Table 5.5: Condensed materials coauthorship network perturbation.

5.3.3.1 Condensed materials coauthorship network

For the results in the original CM network, see the upper section of Table 5.5 (annotated as having hub threshold ∞). When 10% of the edges were removed with the descending order, the average *SPD* grew by 13% and r_{desc} was 3.89. Although the ratio and the increase are significant, the increase is much lower than for PG and GB. The *max* row indicates that the possible room for improvement is limited and achieving results of GB and PG is very unlikely.

Here and for the two remaining networks, we calculate the increases of *SPD* and the ratios with respect to the networks received after the hub conversion (if applicable). Remember, the hub conversion is defined as an overt action by the network curator. Only

11 out of more than 36 thousand nodes of CM were identified as hubs (see the row of Table 5.4 annotated as CM network having a hub threshold of 190). Therefore, there is no much difference in the results shown for the original network and the networks received after removal of the hubs (compare different sections of Table 5.5). We lowered the hub threshold to 100 and still could not get significantly larger increase of the average *SPD*. This remains an example of not well-understood network resilience, although partially it might be related to the fact that when we lower the hub threshold we treat regular nodes as hubs and this does not give us the desired effect. Note that the average node degree does not change much when the hub threshold goes down (see Table 5.4). Again, r_{asc} was always less that 1 for CM networks, which means that the difference between the descending and ascending orders is even larger.

5.3.3.2 Political blogs network

The results for the smallest network, which is a network of political blogs, are given in Table 5.6. The reason for the resilience of the original PB network might be a combination of a very high average node degree (more than 27) and a small network size (1222). Even when we removed all removable edges (about 43% of the edges), the average *SPD* was 3.165 (increase of only 0.156).

Due to the aforementioned combination, about a fourth of the nodes were qualified as hubs with the hub threshold of 34 ($\approx \sqrt{1222}$). Many of the edges were deleted during the hub conversion, and the results became much better. When 10% of the edges were removed with the descending order, the average *SPD* grew by 10% and r_{desc} was 3.87. Again, r_{asc} was always less that 1 for all PB networks, which means that the difference between the descending and ascending orders is even larger.

5.3.3.3 Autonomous systems network

The results for the original autonomous systems network are in the upper section of Table 5.7 and look similar to the results shown for the original CM. When 10% of the edges were removed with the descending order, the average *SPD* grew by 10% and r_{desc} was 2.04.

hub thresh-old	fraction of deleted edges	average *SPD*			increase of *SPD*			ratio	
		asc.	rand.	desc.	asc.	rand.	desc.	asc.	desc.
∞	0	2.737	2.737	2.737	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	2.778	2.796	2.852	0.015	0.022	0.042	0.693	1.945
	max	3.028	3.113	3.165	0.106	0.137	0.156	0.774	1.138
34	0	3.439	3.439	3.439	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	3.480	3.530	3.791	0.012	0.026	0.102	0.447	3.870
	max	3.900	3.983	4.339	0.134	0.158	0.262	0.847	1.655

Table 5.6: Political blogs network perturbation.

Surprisingly, the average *SPD* for the ascending order of edge deletion was even higher than the average *SPD* for the descending order (for all other networks the average *SPD* for the ascending order was the lowest among *SPD*s for the three orders). We believe this is related to the important role that hubs play in this network. AS has nodes with very high degree and the coefficient of variation of the node degree is the highest among the considered networks (see Table 5.1). Consider two hubs with very high degrees that are connected by an edge. This edge might be a part of many shortest paths. It is also likely to have a high ranking in the ascending order, because many nodes might be connected to the two hubs. Deletion of the edge between the hubs will increase many *SPD*s, although just by one. And that might explain why the ascending order of deletion is so powerfull in the original AS.

AS network had slightly higher fraction of hubs than CM (52 hubs out of almost 23 thousand nodes, see Table 5.4). There was a significant difference in the increase of *SPD*, after the hub conversion (see Table 5.7). The increase of *SPD* went up from 10% to 20% for the descending order. r_{desc} reduced from 2.04 to 1.72, but it is still significant. The average *SPD* for the ascending order was still higher than the average *SPD* for the random order, but less than the average *SPD* for the descending order ($r_{desc} = 1.72$ and $r_{asc} = 1.27$).

5.3.4 SPD summary

If we consider the original networks (that is, do not convert hubs) and look at the result of removal of 10% of the edges, r_{desc} was always high (between 1.9 and 3.8 for the real networks, and 10.7 for GB), which shows a big difference between the descending and the

hub thresh-old	fraction of deleted edges	average *SPD*			increase of *SPD*			ratio	
		asc.	rand.	desc.	asc.	rand.	desc.	asc.	desc.
∞	0	3.840	3.840	3.840	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	4.567	4.040	4.247	0.189	0.052	0.106	3.638	2.038
	max	5.393	5.291	5.614	0.404	0.378	0.462	1.070	1.223
151	0	5.948	5.948	5.948	N.A.	N.A.	N.A.	N.A.	N.A.
	0.1	6.851	6.657	7.170	0.152	0.119	0.206	1.274	1.723
	max	7.795	7.945	8.438	0.311	0.335	0.419	0.925	1.247

Table 5.7: Autonomous systems network perturbation.

random orders of the edge deletion. With the exception of AS, the same applies to the difference between the descending and the ascending orders of the edge deletion (r_{asc} was less than 1 for the other four networks). In the terms of the magnitude of increase of the average *SPD* for the descending order, the results were mixed. CM and AS had increases of above 10%, which is significant. GB and PG had increases of over 66%, which is huge (even when only 1% of the edges was deleted, the increase was over 12% for these networks). PB was resilient (combination of the small network size and the high average node degree) and had an increase of only 4%.

The introduction of the hub conversion brought increase in the magnitude of increase of the average *SPD* for the descending order for AS and more importantly for PB. CM remained unaffected, may be because it had only few hubs.

5.3.5 Commute time

In this section, we consider commute time as a measure by which the edges are rated. The *commute time* of a random walk between two nodes on a graph is defined as the expected time (number of steps) for the random walk to depart from one of the nodes, reach the other node, and return to the first node. Instead of calculating the commute time between the endpoints of an edge exactly, we estimate it. For each edge, we conducted 100 random walks emanating from one of its endpoints (the commute time is symmetric) and measured the average commute time. A walk can be quite long, so in order to save the time we also set a maximum walk limit. If a walk reaches the limit, the walk is terminated and the limit is

network	commute success rate	average *SPD*				
		ascending stretch	ascending comm. time	random	descending comm. time	descending stretch
GB	0.688	6.920	7.022	7.095	7.219	11.14
PG	0.785	22.51	20.69	25.32	39.70	34.04
CM	0.086	5.586	5.656	5.687	5.758	6.229
PB	0.963	2.778	2.840	2.796	2.806	2.852
AS	0.320	4.567	4.835	4.040	4.085	4.247

Table 5.8: Commute time versus stretch.

assumed as the walk length. For the largest network (CM), the limit was set to 1000; for the rest of the networks, it was set to 10000. The success rates (of the commute completion) and the average *SPD*s for five orders of deletion are given in Table 5.8. The fraction of deleted edges was set to 0.1. Three of the orders were already discussed before and their results are repeated here for the sake of convenience. The two new orders (the descending and ascending orders of the edge deletion based on the commute time) rate edges based on the commute time solely.

The stretch can be considered as a local measure (at least for the edges with a short stretch), but all edges of the network affect the commute time for any pair of nodes. Addition of an edge to the network may lead only to the reduction of some of the stretches, but the commute time might be increased for some pairs of nodes and decreased for some other pairs. For all five networks that we considered here, usage of a descending order of deletion (based on either the stretch or the commute time) led to the increase of the *SPD* (comparatively to the random order of deletion). For GB, PG, and CM, usage of both ascending orders of deletion led to the decrease of the *SPD*. For AS, they led to the increase of the *SPD*. Only for the combination of PB and the ascending orders, there was a switch; the stretch based deletion decreased the *SPD*, the commute time based deletion increased the *SPD* (all comparatively to the random order of deletion). Thus, in most of the cases, the stretch based and the commute time based ratings caused impacts of a same character but of a different magnitude (stretch based orders had usually a stronger impact).

5.4 Impact on the community structure

5.4.1 Evaluation of the orders of edge removal

In this section, we consider networks that have an underlying community structure. Internal edges of a community build the community, and we expect to see many of them. Consider an edge that is internal to some community. It is reasonable to expect that the edge has many short paths connecting its endpoints. Therefore, internal edges are likely to have low stretch and might have many paths that support this stretch. On the other hand, an edge that connects two communities is likely to have a large stretch. Otherwise, one should consider including its endpoints into the same community.

We can evaluate the obviousness of a community structure of an original and its perturbed network by calculating their partition modularities (with the partition as defined by the known ground truth communities) and comparing them with each other. However, that measure might be difficult to comprehend. To overcome this problem we will assume that our community detection algorithm works well and will use it to produce communities that are supposed to match the ground truth communities. We will calculate Jaccard indices between the ground truth communities and the communities discovered by algorithm.

5.4.1.1 Two-dimensional network

We will borrow a network that we considered in Section 4.4.4.1. The network is $IDN(2, 20, 40, 0.35, 0.07/d)$. We chose this network because the community detection algorithm showed good results for it. Modularity by groups columns of Table 5.9 show the modularity of partitions formed by the ground truth communities of the original and perturbed networks. The random order of the edge deletion did not affect the modularity much, while the ascending order led to the decrease of modularity, and the descending order led to the increase of modularity. The last three columns are based on the results produced by the community detection algorithm. The cover modularity column shows the modularity for the cover that produced the lower bound on Jaccard index. When 10% of the edges are

fraction of of deleted edges	order of deletion	modularity by groups		cover modularity	bounds on Jaccard index	
		dim A	dim B		lower	upper
0	N.A	0.2155	0.2143	0.2074	0.7931	0.8296
0.03	ascend.	0.2037	0.2047	0.1831	0.7201	0.7553
0.03	random	0.2157	0.2146	0.2006	0.7739	0.7857
0.03	descend.	0.2234	0.2217	0.2160	0.8153	0.8265
0.1	ascend.	0.1833	0.1854	0.1449	0.5110	0.5342
0.1	random	0.2141	0.2143	0.2027	0.7744	0.7866
0.1	descend.	0.2425	0.2404	0.2364	0.8394	0.8448

Table 5.9: Perturbation of two-dimensional network.

fraction of of deleted edges	order of deletion	modularity by groups	cover modularity	bounds on Jaccard index	
				lower	upper
0	N.A	0.4445	0.4004	0.8369	0.8489
0.03	ascend.	0.4295	0.3850	0.7530	0.7659
0.03	random	0.4432	0.3960	0.8214	0.8289
0.03	descend.	0.4594	0.4177	0.8525	0.8609
0.1	ascend.	0.3927	0.2291	0.4976	0.5253
0.1	random	0.4420	0.3811	0.7687	0.7859
0.1	descend.	0.4985	0.4620	0.8402	0.8474

Table 5.10: Perturbation of one-dimensional network.

deleted, the ascending order of deletion caused a significant damage, the random order caused a small damage, and the descending order actually improved the results. When 3% of the edges are deleted, the tendencies are the same but the amplitudes of the changes are smaller.

5.4.1.2 One-dimensional network

We also tried the edge deletion for a one-dimensional network, for which we could produce Jaccard index of just above 80%. It was a $IDN(1,20,40,0.25,0.05/d)$ network. We kept the ratio $r_{ion} = p_{intra}/p_{neigh}$ equal to 5, similarly to the two-dimensional case. We could lower p_{intra} and thus the number of internal edges, because the network is one-dimensional and there is no noise caused by the edges of the missing dimension. The results are given in Table 5.10 and they are very similar to the results of the two-dimensional case.

5.4.2 Why one might want not to preserve the community structure

Here we want to introduce a scenario where the preservation of communities during the anonymization process can be undesirable. It might go against the general idea of Section 5.4 that we want to preserve the community structure of the network, but this should be considered anyway. Consider an attack that is similar to the active attack described by Backstrom et al. [5]. The attacker plants several nodes prior to the network anonymization. These nodes are connected between themselves and also to legitimate nodes of the network, some of which are the target nodes (which the attacker wants to identify in the anonymized network). When the anonymized network is released, the attacker identifies the planted nodes based on the connections between the planted nodes and knowledge of the exact node degree of the planted nodes. After that, the identification of the target nodes is possible.

The problem of identification of a planted subgraph in an anonymized network is closely related to the problem of subgraph isomorphism, which is a known NP-hard problem. Backstrom et al. [5] make this problem easier by introducing a special path connecting the planted nodes that takes into account the nodes' degrees. They claim that detection of the planted nodes can be easily done in the real networks. At the same time, the curator of the network that does the anonymization might perturb the network and make the anonymization more difficult. Without knowing any specifics of the work that is done by the adversary and the curator, we cannot prove anything and can only assume that detection of the planted nodes is still hard (whatever hard might mean).

Suppose, for the sake of argument, that the adversary wants to identify the planted nodes based on the subgraph isomorphism. The whole network will be much larger than the planted subgraph, and the size of the network affects the time necessary to find the isomorphism. Suppose that the adversary creates links between the planted nodes in such way that they form community in the network. It is a reasonable assumption because if the adversary wants to create a link between two planted nodes then he has full control over

both endpoints of the edge. If the anonymization process preserves the network communities, then the adversary might run a community identification process on the anonymized network. Even if the planted community is not identified exactly by the community identification process, it would suffice that all the planted nodes end up in one of the identified communities. Now the adversary has to solve the subgraph isomorphism problem many times (one per each identified community), but the size of each instance of the problem is much smaller.

5.5 Summary

We have demonstrated a difficulty of the utility preservation in the process of network anonymization. Concentrating on achieving the best results in order to preserve one quality of the network might damage another quality. When we prefer the deletion of edges with a high stretch as a part of the network perturbation, we gain in the preservation of the community structure of the network, but we worsen routing abilities in the network. When we prefer the deletion of edges with a low stretch, we preserve many short paths in the network, but destroy its community structure.

If we look at the edge stretch as a measure whether the edge deserves to exist, then edges with a long stretch can be deemed as "odd" edges. The number of "odd" edges should be small and it is not a surprise that their removal affect some property (shortest path distance in our case). Edges with a short stretch should be deemed as "normal" and there should be a plenty of them. We expected that it would require removing much more of them in order to see a change (destruction of the community structure in our case). However, it took about the same fraction of edges to be deleted in order to see the changes.

Chapter 6

CONCLUSION AND FUTURE WORK

A few aspects related to small-world networks were considered in this dissertation. The grid-based model introduced by Kleinberg possesses two main characteristics that pertained to small-world networks: the existence of short paths between many randomly chosen pairs of nodes of the network and the ability of the nodes to find these paths in a decentralized fashion. The routing in this model is based on the ability of a node to understand its own place in the network, having good understanding of the locations and the roles of its immediate neighbors, and having at least an approximate understanding where the targets are located.

The proof on the upper bound of the expected path length in the grid-based model was a source of our idea for how a deterministic grid-based model should be organized. We saw that in the probabilistic model, the search was divided into a logarithmic number of phases and later phases were responsible to route in exponentially smaller areas around the target. In the deterministic model, we do not know where the target will be at the time when the grid is created, but we can assign the long-range edges so that we have the same logarithmic number of phases and later phases are responsible to navigate in smaller areas similarly to the probabilistic model. The difference is that the areas corresponding to the phases in the deterministic model are defined at the stage when the grid is created and the target is unknown.

We proved that the average length of the long-range edges in the deterministic model is asymptotically shorter than the expected edge length of a long-range edge in the probabilistic model. It can be useful in an environment where the deterministic generation of long-range edges is allowed and the cost of connection is increases with the geographical distance of the underlying space.

A long-range edge in the deterministic model is always aligned with a direction in a single dimension. It would be interesting to know if a path length can be decreased if an

edge lies in the subspace spanned on more than one dimension. For example, if some edges will be connected to the center of the multidimensional grid and not to the center of the chain (one-dimensional grid comprising a grid of higher dimension).

Greedy routing in grid-based model is memoryless. Humans are capable of learning, and our adaptive decentralized search capitalizes on the experience of previous searches. We borrow a lot from the grid base-model: we still partition the space into logarithmic number of regions and initialize expert weights based on the Euclidean distance in the underlying space. However, this time each node is an active learning agent and defines these regions with respect to its own location. Our adaptive routing improves upon greedy routing because each node learns about the network beyond its immediate neighborhood. The learning happens not because a node learns explicitly about some additional nodes to which it is not connected by an edge. It learns about the underlying space of the network and learns which of its neighbors can be more useful in routing to a particular region.

We saw that the adaptive routing outperforms greedy routing in grid-based networks and real world networks. It scales well, showing better results for larger networks. It is also capable of managing dynamic networks as long as changes in the network approximately match the underlying space (it cannot deal with randomly created edges). We saw that the adaptive routing learns at different speeds. Networks that are regular (with a low deviation of the nodes outdegrees) have something similar to a phase transition. The overpayment rates dropped significantly in a short period of time. However, networks with hubs showed an improvement that is more gradual. We do not know the reason for this phenomenon and this can be a source of the future research.

Another strength of our adaptive routing is its resilience, which is particularly valuable for decentralized systems where the assumption of failure of part of the nodes is reasonable. We saw very good results in networks with 10% of lazy nodes. It would be interesting to see how the adaptive routing works for different fractions of lazy nodes in the network. In

particular, it is interesting to check whether the transition phase still happens in the learning curves of more regular networks when the fraction of lazy nodes becomes larger.

The other adverse environment we considered was an environment with blind nodes. We considered the extreme situation when all the nodes were blind. Even in this environment, some networks outperformed the greedy routing. It would be interesting to see how the adaptive routing behaves in the situation when only a part of the nodes is blind. It is possible to imagine that there will be a significant improvement when even a small fraction of nodes is not blind. The non-blind nodes are likely to play an important role in shaping the first successful searches. With time, the difference between blind and non-blind nodes should diminish.

Yet much more challenging will be situation where a blind node does not know not only coordinates of its neighbors but also of itself. The whole idea of partitioning is based on the assumption that a node forms regions with respect to its own location, and now it is deprived of this ability. The solution might lay in a different kind of dynamic partitioning. A node splits the whole space into four equal quadrants at first. Then it finds out which quadrant is overused and splits it into four smaller quadrants. This way, it can "zoom in" to the most populated areas. We did not try this approach and cannot say if this will indeed work.

Another area for future work can be the integration of the network embedding and the message routing into a single decentralized process. We provided a decentralized version of the network embedding. However, we cannot unite it with the adaptive decentralized routing because we need to know the node locations in order to route, and they become known only in the middle of the extended flow. It will be interesting to know if there are special cases of graphs for which a node is also used as a location service that knows how to route to a small number of other nodes.

We considered the problem of community detection in networks. This problem is especially difficult when we try to detect overlapping communities. We considered the

modularity; this measure is widely used to evaluate quality of the network partition into communities. We defined a notion of the contribution of a node to the partition modularity. This allowed us to extend of the notion of the modularity to covers (a partition is a special case of a cover). The definition of cover modularity is not perfect (we demonstrated it on the example of a cover consisting of two partitions; it would be interesting to find out an alternative definition that does not suffer from this problem). However, we saw that the suggested cover modularity correlates well with the Jaccard index in our experiments (the Jaccard index is defined with respect to the ground truth communities and expected to reflect the quality of the cover well).

We developed an algorithm for detection of the overlapping communities. At the first stage of the algorithm, each node makes its own decision to how many communities it belongs and how its neighbors are distributed between the communities. Importantly, this stage is conducted by each node locally independently of other nodes. Later these local decisions are combined to form final communities. Interestingly, we build our communities and rate them based on the notions of modularity and contribution to modularity.

The existing problem is that our algorithm relies on the decomposition rates (a tuning parameter that allows us to regulate how deep the iterative process of a community splitting should go). We do not know how to find these rates and only show (in our experiments) that such rates exist. An interesting question is what should be done in order to find these rates, or at least how to suggest a limited number of possible solutions.

Due to the absence of real world networks with known overlapping communities, we had to design independent dimensions network model. This model not only allows to create overlapping communities, but also to introduce distance-based relationship between non-overlapping communities. Later we extended this model to codependent dimensions network model that allows a significant overlap between communities.

We intentionally targeted the case that all nodes of the network belong to multiple communities. However, in our community generation models each node participates in the

same number of communities. It will be interesting to see what happens when the models allow nodes to participate in different numbers of communities.

Finally, we considered the problem of preservation of network utility while protecting the user anonymity. The definition of a network utility is application driven. Efforts to preserve a particular utility better may lead to a significant decrease in the level of preservation of another utility. We demonstrated this on the example of two characteristics of the network that were considered in this dissertation before: routing opportunities versus community structure.

We considered one of the simplest ways of network anonymization — edge deletion. It would be interesting to see how things change when edge replacement is used, for example.

An area for future work can be related to the case when several perturbed networks preserving different properties of the same original network are released. It might be easier to match nodes of the released networks than to find planted nodes of an active attack. It is enough for the adversary to match any pair of nodes (hubs for example) and then to extend to other nodes.

Another area for future research might be related to the understanding of the role of hubs in the preservation of network utility. Similarly to the routing experiments, we saw that networks with hubs and networks that are more regular exhibit different types of behavior.

[1] I. Abraham, C. Gavoille, and D. Malkhi. On space-stretch trade-offs: Lower bounds. In *Proceedings of the 18th ACM Symposium on Parallel Algorithms and Architecture*, pages 207–216, 2006.

[2] L. A. Adamic and N. Glance. The political blogosphere and the 2004 u.s. election: divided they blog. In *LinkKDD '05: Proceedings of the 3rd international workshop on Link discovery*, pages 36–43, New York, NY, USA, 2005. ACM Press.

[3] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta algorithm and applications. Unpublished survey. Available at `www.cs.princeton.edu/~arora/pubs/MWsurvey.pdf`.

[4] B. Awerbuch and R. Kleinberg. Online linear optimization and adaptive routing. *J. Comput. Syst. Sci.*, 74(1):97–114, 2008.

[5] L. Backstrom, C. Dwork, and J. Kleinberg. Wherefore art thou r3579x?: anonymized social networks, hidden patterns, and structural steganography. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 181–190, New York, NY, USA, 2007. ACM.

[6] O. Bakun. On some questions related to the small-world phenomenon. 2007. Thesis (M.S.). Available at ASU Libraries.

[7] O. Bakun. Technical report: Adaptive decentralized routing in small-world networks. 2011. Technical report. Released to the committee.

[8] O. Bakun. Technical report: Cover modularity — overlapping communities. 2011. Technical report. Released to the committee.

[9] O. Bakun and G. Konjevod. Adaptive decentralized routing in small-world networks. In *INFOCOM IEEE Conference on Computer Communications Workshops , 2010*, pages 1–6, 2010.

[10] M. Barbaro and T. Zeller, Jr. A face is exposed for AOL searcher no. 4417749. The New York Times, 9 Aug. 2006.

[11] J. Baumes, M. K. Goldberg, and M. Magdon-Ismail. Efficient identification of overlapping communities. In P. B. Kantor, G. Muresan, F. Roberts, D. D. Zeng, F.-Y. Wang, H. Chen, and R. C. Merkle, editors, *ISI*, volume 3495 of *Lecture Notes in Computer Science*, pages 27–36. Springer, 2005.

[12] B. Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.

[13] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner. On modularity - np-completeness and beyond. Manuscript available at http://digbib.ubka.uni-karlsruhe.de/volltexte/documents/3255.

[14] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *APPROX '00: Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 84–95, London, UK, 2000. Springer-Verlag.

[15] D. P. de Farias and N. Megiddo. Combining expert advice in reactive environments. *J. ACM*, 53(5):762–799, 2006.

[16] M. Dell'Amico. Mapping small worlds. In *Peer-to-Peer Computing*, pages 219–228, 2007.

[17] M. Dom. Compact routing. In D. Wagner and R. Wattenhofer, editors, *Algorithms for Sensor and Ad Hoc Networks*, volume 4621 of *Lecture Notes in Computer Science*, pages 187–202. Springer Berlin / Heidelberg, 2007.

[18] D. Easley and J. Kleinberg. Networks, crowds, and markets: Reasoning about a highly connected world. Draft version is also available at http://www.cs.cornell.edu/home/kleinber/networks-book/networks-book.pdf, 2010.

[19] P. Erdös and A. Rényi. On random graphs, I. *Publicationes Mathematicae (Debrecen)*, 6:290–297, 1959.

[20] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[21] P. Fraigniaud, C. Gavoille, A. Kosowski, E. Lebhar, and Z. Lotker. Universal augmentation schemes for network navigability: overcoming the √n-barrier. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, SPAA '07, pages 1–7, New York, NY, USA, 2007. ACM.

[22] P. Fraigniaud, C. Gavoille, and C. Paul. Eclecticism shrinks even small worlds. In *PODC*, pages 169–178, 2004.

[23] C. Gavoille. Routing in distributed networks: Overview and open problems. *ACM SIGACT News - Distributed Computing Column*, 32(1):36–52, 2001.

[24] C. Gavoille and D. Peleg. Compact and localized distributed data structures. *Journal of Distributed Computing*, 16(2-3):111–120, 2003.

[25] S. Gregory. An algorithm to find overlapping community structure in networks. In *PKDD 2007: Proceedings of the 11th European conference on Principles and Practice of Knowledge Discovery in Databases*, pages 91–102, Berlin, Heidelberg, 2007. Springer-Verlag.

[26] S. Gregory. Finding overlapping communities using disjoint community detection algorithms. In *Complex Networks: CompleNet 2009*, pages 47–61. Springer-Verlag, May 2009.

[27] M. Hay, G. Miklau, D. Jensen, P. Weis, and S. Srivastava. Anonymizing social networks. Technical report, University of Massachusetts Amherst, 2007.

[28] J. Kleinberg. The small-world phenomenon: an algorithm perspective. In *Proceedings of the 32nd ACM Symposium on Theory of Computing*, pages 163–170, 2000.

[29] J. Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians*, 2006.

[30] J. Kleinberg and Éva Tardos. *Algorithm Design*. Addison-Wesley, Boston, MA, 2005.

[31] G. Konjevod, A. W. Richa, and D. Xia. Optimal scale-free compact routing schemes in networks of low doubling dimension. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms*, pages 939–948, 2007.

[32] Y. Koren. On spectral graph drawing. In *Proc. 9th Inter. Computing and Combinatorics Conference (COCOON03), LNCS 2697*, pages 496–508. Springer-Verlag, 2002.

[33] E. Lebhar and N. Schabanel. Almost optimal decentralized routing in long-range contact networks. In *ICALP*, pages 894–905, 2004.

[34] J. Leskovec, K. J. Lang, and M. Mahoney. Empirical comparison of algorithms for network community detection. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 631–640, New York, NY, USA, 2010. ACM.

[35] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. In *FOCS*, pages 256–261, 1989.

[36] S. Milgram. The small world problem. *Psychology Today*, 1(1):60–67, 1967.

[37] M. E. J. Newman. Network data. Web page at `www-personal.umich.edu/~mejn/netdata/`.

[38] M. E. J. Newman. The structure of scientific collaboration networks. *PROC.NATL.ACAD.SCI.USA*, 98:404–409, 2001.

[39] M. E. J. Newman. The Structure and Function of Complex Networks. *SIAM Review*, 45(2):167–256, 2003.

[40] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

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

[42] V. Nicosia, G. Mangioni, V. Carchiolo, and M. Malgeri. Extending the definition of modularity to directed graphs with overlapping communities. *Journal of Statistical Mechanics: Theory and Experiment*, 2009(03):P03024, 2009.

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

[44] D. Peleg. *Distributed Computing: A Locality-Sensitive Approach*. Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, 2000.

[45] O. Sandberg. Distributed routing in small-world networks. In *Proceedings of the 9th ALENEX*, pages 144–155, 2006.

[46] S. Tauro, C. Palmer, G. Siganos, and M. Faloutsos. A simple conceptual model for the Internet topology. In *IEEE Global Telecommunications Conference, 2001. GLOBECOM'01.*, 2001.

[47] M. Thorup and U. Zwick. Compact routing schemes. In *Proceedings of the 13th ACM Symposium on Parallel Algorithms and Architecture*, pages 1–10, 2001.

[48] W. T. Tutte. How to draw a graph. *Proceedings of the London Mathematical Society*, s3–13(1):743–768, 1963.

[49] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, April 1998.

[50] S. Zhang, R.-S. Wang, and X.-S. Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483 – 490, 2007.

[51] B. Zhou, J. Pei, and W. Luk. A brief survey on anonymization techniques for privacy preserving publishing of social network data. *SIGKDD Explor. Newsl.*, 10:12–22, December 2008.

Appendix A

DETAILS ON THE PENALTY MATRIX

We do not know at the outset which outcomes (path lengths) should be considered as good ones, in other words what the penalty matrix $M(i,j)$ should look like. To overcome this difficulty we consider the average outcome seen so far by the ensemble (denoted as d_{aver}) as a neutral outcome (with zero penalty). We treat all experts equally, therefore each region has its own dynamic one-dimensional penalty matrix $M(j)$, and j is an outcome, i.e., the length of the portion of the path from the node (corresponding to the ensemble) to the target. Thus, Formula (3.1) for the weight update of the expert i becomes

$$w_i = \begin{cases} w_i(1-\varepsilon)^{+M(j)/\rho} & \text{if } M(j) \geq 0 \\ w_i(1+\varepsilon)^{-M(j)/\rho} & \text{otherwise} \end{cases} \tag{A.1}$$

If the new outcome d is longer than the average one we will use $min(4, d/d_{aver} - 1)$ as a penalty; otherwise when the new outcome d is shorter that the average one we will use $max(-4, -d_{aver}/d + 1)$ as a penalty (a negative penalty is a reward). These two ratios (d/d_{aver} and d_{aver}/d) are always greater than or equal to 1 when they are used. So we subtract (add) 1 from (to) the corresponding ratio in order to prevent the (-1,1) gap in the range of penalties. If $d = d_{aver}$ the penalty is equal to zero (there is no weight update). By limiting the penalty range to the interval [-4,4], we are setting ρ to 4; the smaller is ρ, the faster is the convergence (as long as the path lengths are not extremely far away from the limit of the average path length). We set ε in our experiments to 0.3. Finally, in order to speed up the convergence, we initialize the weights of an expert i before the first stage with the reciprocal of the Euclidean distance between the corresponding neighbor and the first target that was assigned to the ensemble to the power of ten: $1/l_2^{10}$. The reason for the so high exponent is that we want that the ensemble will start in the greedy-like style and we want a clear distinction between the neighbors with the closest Euclidean distance to the target and the rest. Exact values for all these constants were chosen experimentally.

Appendix B

DYNAMIC LONG-RANGE EDGES

In this section, we consider networks that change with the time. We rarely but consistently reassign heads of some of the edges to different nodes. This way the number of outgoing edges and thus the experts in the ensembles do not change. However, experts that correspond to the reassigned edges become poorly rated. We cannot make reasonable assumptions how real-world networks will change with the time unless we have an expertise in their domains. Because of that, we concentrate here only on the grid-based networks. For these networks, we can reassign long-range edges without hurting their quality of being able to represent small-world networks.

To be more precise we create a grid-based network with a chosen number of rows (columns), and specified parameters p, q, and r (see their definitions in Section 2.2.1). With some frequency, we reassign one edge at a time. We choose an edge for the reassignment uniformly at random between all long-range edges that are longer than 1. Only these long-range edges do not coincide with the local edges and have their own experts. When the edge is chosen, a new corresponding long-range contact (the head of the edge) is chosen from exactly the same distribution that was used for the original network (it is defined by the exponent r). If the length of the newly proposed long-range edge is 1, the process of the search for a new contact for the same tail of the edge continues immediately until a longer edge is found. This way we are dealing with an ordered sequence of grid-based networks. Each network is very similar to its predecessor and successor. It would have made more sense not to reassign the long-range edge from the old contact to the new contact at once, but move it slowly along the local edges. Although that would also make the learning process easier, we did not want to complicate the model with the speed rate at which the contact should travel.

When an edge changes its contact, the corresponding experts are likely to have outdated weights in EEE method. However, we could learn the correct weights the first time. Why cannot we do it again? The problem lies in the exploration rate, in the beginning it is high, but soon becomes very low. (The probability of stage i to become an exploration stage is $1/\sqrt{i+1}$.) The reason is that we do not want to waste many stages on the learning

when we succeeded to rate the experts well enough. This works in static environments, but backfires in dynamic ones.

In our experiments with the grid-based networks from Section 3.4.3, we conducted 3000 searches per node. We repeat plots of the progress of the overpayment rate for these experiments with EEE on the left part of Figure B.1. We show the average path lengths and overpayment rates for the last buckets in these experiments and experiments with the dynamic long-range edges in Table B.1.

We start with exactly the same networks and conduct the same number of searches in our experiments with the dynamic long-range edges. We reassign a random long-range edge every 750 searches. That is, an average long-range edge changes its contact four times. When the reassignment happens, the tail of the node is not informed about it, so it cannot reinitialize the weight of the affected experts based on the Euclidean distances from the new contact to the regions. However, CMH still can detect when it is one hop away from the target. The progress of the overpayment rate is shown in the middle of Figure B.1. Obviously, the overpayment rate becomes higher because the environment is more dynamic now. However more significantly, the overpayment rate reaches it minimum around 140 to 600 searches per node and then starts to grow. At this point, the fraction of the exploration stages becomes so low that it cannot keep up with the changes caused by the reassignment of the dynamic edges. The results are becoming quite bad; it is likely that eventually the experts of the long-range edges will end up with low weights and the routing will be conducted by using mostly the local edges. We increased the number of searches for the smallest network to 30000 per node, and the overpayment rate for the last bucket was 5.46 and almost a quarter of the searches failed (the failure rate was 0.246).

De Farias and Megiddo considered a different scheme to make a decision whether a phase should be an explorative or exploitative. They suggested that a phase becomes an explorative phase with a constant probability that does not change with the time. That is likely to hurt somewhat in environments that are more static, but helps with dynamic ones.

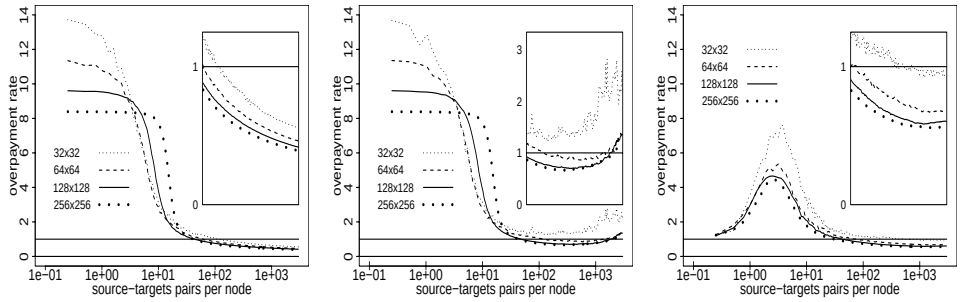

Figure B.1: Dynamic long-range edges. Overpayment rates.

network	average path length			overpayment rate		
	static env.	dynamic env.	constant exploration rate	static env.	dynamic env.	constant exploration rate
32×32	10.03	19.71	12.24	0.55	2.23	0.94
64×64	13.43	23.21	16.02	0.46	1.31	0.69
128×128	17.31	35.15	20.86	0.42	1.35	0.60
256×256	21.91	44.68	26.69	0.39	1.17	0.55

Table B.1: Dynamic long-range edges. The results are given for the networks with static long-range edges, dynamic long-range edges, and dynamic long-range edges with a constant exploration rate. The data is given for the last bucket.

On top of that, we can speed up the learning process about the moving edges by discounting an old feedback. Up until now, all feedbacks to an expert had the same importance. With a constant exploration rate, we will pretend that the weight of the constant expert is a result of no more than 19 feedbacks. This way a new feedback has greater importance (at least of 5 percent of the current weight is formed by the latest feedback). We conducted the same experiments with the dynamic long-range edges and received better results. We used the exploration rates of 0.03. We used this exploration rate and the importance 0.05 of the last feedback because it gave good results for the smallest network, and we used these constants for the rest of the networks. The overpayment rates are shown on the left part of Figure B.1, it shows that the overpayment rates are steadily decreasing and the results for the larger networks become slightly better (the method scales well).

Appendix C

DECENTRALIZED EMBEDDING

It was suggested by the committee to consider decentralization of the extended flow, which includes both the embedding and the routing. Although, the decentralized version for the embedding was found and will be presented below, we cannot claim that the whole flow is a name-independent decentralized algorithm. Although the embedding algorithm does not depend on the names of the nodes, the routing does. In order to route the message, CMH should know the target's location and we require it to be a part of the name. We do the embedding only because we do not know the true locations of the nodes, so they cannot be incorporated into the names (unless we allow name change during the preprocessing, that is, the embedding stage). We are not aware about any specific class of graphs for which we can make the extended flow decentralized.

C.1 Koren's algorithm.

Koren [32] uses the transition matrix of a random walk on the graph $D^{-1}A$ for the purposes of graph visualization. The main idea is to put closely related nodes close to each other. Koren motivates his method by an attempt to minimize $\sum_{e=(i,j)} w_{ij}(x_i - x_j)^2$, which can be rewritten as $x^T Lx$. Where $L = (D - A)$ is the Laplacian, A is the adjacency matrix, D is a diagonal matrix with the diagonal filled with the nodes' degrees, the components of x are the coordinates of the nodes, and w_{ij} is the weight of the edge between the nodes i and j. This leads to

$$\min_x x^T Lx$$

given:

$$x^T Dx = 1$$

$$x^T D1_n = 0$$

Where the last two lines make sure that the components of x are centered at 0, and they are not collapsing to the origin. Both of them are normalized with respect to D in order to take care of the variation of the nodes' degrees. The solution for the above becomes a search for the generalized eigenvectors of (L, D) (that is, $Lu = \mu Du$, where u and μ are a generalized eigen-pair of (L, D)). They are equivalent to the eigenvectors of $D^{-1}A$, which is exactly what is used for the drawing by Koren by running an iterative process (see Figure C.1). The

```
1  Function SpectralDrawing (G — the input graph, k — dimension)
2  // This function computes u_2, ..., u_k,
3  // the top (non-degenerate) eigenvectors of D^{-1}A.
4      const ε ← 10^{-7} // tolerance
5      for i = 2 to k do
6          û_i ← random // random initialization
7          û_i ← û_i / ||û_i||
8          do
9              u_i ← û_i
10             // D-Orthogonalize against previous eigenvectors:
11             for j = 1 to i − 1 do
12                 u_i ← u_i − (u_i^T D u_j)/(u_j^T D u_j) × u_j
13             end for
14             // multiply with 1/2(I + D^{-1}A):
15             for j = 1 to n do
16                 û_i(j) ← 1/2 × (u_i(j) + (Σ_{k∈N(j)} w_jk u_i(k))/deg(j))
17             end for
18             û_i ← û_i / ||û_i|| // normalization
19         while û_i · u_i < 1 − ε // halt when direction change is negligible
20         u_i ← û_i
21     end for
22     return u_2, ..., u_k
```

Figure C.1: The algorithm for computing degree-normalized eigenvectors by Koren [32].

line 16 of this figure is the crux of the algorithm. The next value of the node's coordinate is the average of the current value of the node's coordinate and the average of the coordinates of the nodes' neighbors.

C.2 Dell'Amico's algorithm.

The algorithm provided by Koren [32] is centralized. Dell'Amico [16] used the spectral graph drawing to embed the OpenPGP web of trust into a metric space for the purpose of greedy routing and also suggested a decentralized version of the Koren's algorithm. We used Koren's version in our experiments because of its straightforwardness and because

we do not see it as a part of our adaptive routing algorithm per se. Nevertheless, it would add to the value of the distributed routing if the mandatory input might be produced in the distributed fashion as well.

The central operation of the algorithm (the line 16 of Figure C.1) is a calculation of the next location of the node based on its current location and the current locations of its neighbors. This is definitely can be done in the distributed fashion. $\Theta(m)$ messages of the constant size are sent to facilitate the calculations at every iteration (one message in each direction of each edge). There is a delay of $\Theta(1)$ at each of the iterations for the coordinate exchange, because it can be done simultaneously.

However, for each larger iteration of the vector calculations (the lines 8 – 19 of Figure C.1), there are operations that require a global effort of the network: D-orthogonalization, normalization, and correctness of the stoppage criterion.

- Dell'Amico suggests that D-orthogonalization (the line 12 of Figure C.1) is redundant and abandons it in his experiments. Koren also wrote that it was added for the sake of numerical stability.

- Dell'Amico claims that the underflow/overflow of the variables storing the coordinates of the nodes was not an issue in his experiments, and does the normalization (the line 18 of Figure C.1) only at the end of the algorithm instead of doing it at each iteration. He suggests a distributed normalization that is imprecise, but it might work.

- Instead of checking the conversion of the coordinates as the stoppage criterion (the line 19 of Figure C.1), Dell'Amico runs the algorithm for a fixed number of iterations, which explains in part why there is no need in the normalization after each iteration. He suggests trying different numbers of the iterations, running a simulation of an application that uses the results of the embedding, and deciding whether the embedding was good based on the results from the application. Dell'Amico used 200 iterations in his experiments. This low number might be an explanation why the algorithm did

not run into the underflow/overflow problem without the frequent normalizations, but a different network may require a higher number of the iterations.

Overall, the algorithm of Dell'Amico does not require much overhead (in the terms of the time and the number of messages) related to the transition from the centralized to the distributed version, but there are questions related to the precision of the solutions.

C.3 New distributed algorithm.

We propose here an implementation of the Koren's algorithm in a distributed manner without any relaxation, but the overhead will be more significant than in Dell'Amico's case. As mentioned before, the main iterative operation (the line 16 of Figure C.1) recalculates coordinates of a node based on the coordinates of its neighbors and can be done in the immediate vicinity of each node. Consider the normalization (the lines 7 and 18 of Figure C.1) and the stop criteria (the line 19 of Figure C.1). They both are essentially dot products; and these particular dot products are linear combinations of the locally computable terms (because each node needs to multiply its own coordinates). The remaining work to do is D-orthogonalization (the line 12 of Figure C.1). $u_i^T Du_j$ and $u_j^T Du_j$ are very similar to the dot product because D is diagonal; they are linear combination of the locally computable terms as well. All we need for the transition from the centralized to the distributed version is to execute the summation of the locally calculated terms and propagate the results (the length of the current vector or some other real number) back to every node.

We can assume that each node has a unique ID. We need to choose a root r (with the largest ID, for example) and build a spanning tree that will be used for the data collection and the result propagation. It can be a tree matching a breadth-first search tree. Each node u (except r) initializes d_u (its distance from r) to infinity, initializes its immediate ancestor to NULL, and creates an empty list of its descendants. r sends to all its neighbors a message that they are at distance 1 from r. When the node u receives a message that it might be at distance d from r, it compares d to d_u. If $d_u > d$, then u:

- stores the last sender as the current immediate ancestor of u (this neighbor is the only immediate ancestor of u)
- notifies the last sender, that u is a descendant of the sender;
- notifies u's previous immediate ancestor, that u is not its descendant anymore;
- updates d_u: $d_u = d$;
- notifies all neighbors of u that they might be at distance $d+1$ from the root r.

If $d_u \leq d$, then nothing is done by u. This way, the node u knows its (exactly) one immediate ancestor and the list of descendants of u. Edges that connect each node to its immediate ancestor form a spanning tree. The height of this rooted at r tree cannot be greater than D — the diameter of the network.

In order to produce a linear combination of the locally computable terms, each node u computes its term locally and adds it to the data received by u from its descendants. Only when all the descendants of u have sent the data to u, u can send the sum to its immediate ancestor. When the root r receives the data, it can calculate the linear combination and start the result propagation (it is either the length of the vector or some other real number). Each node propagates the result to its descendants.

Let k be the dimension of the space (that is the number of eigenvectors that we want to produce). We need to do D-orthogonalization to all previously calculated eigenvectors. At each iteration of the vector's calculations (the lines 8–19 of Figure C.1), $\Theta(kn)$ messages of the constant size are sent to accommodate D-orthogonalization. There are also $\Theta(m)$ messages of the constant size to be sent in the line 16 of Figure C.1. Therefore, there are totally $\Theta(kn+m)$ messages of the constant size are sent at each iteration of the loop. There is a delay of $O(kD)$ at each of the iterations, because the data propagation cannot be done simultaneously.

In the previous paragraph, we bounded the number of the messages and the delay for one iteration of the loop of the lines 8–19 of Figure C.1. In order to produce the bounds

for all k eigenvectors, assume that the number of iterations for one vector is bounded from above by a number N. Then in order to produce all the vectors, we will need $O(kN(kn+m))$ messages of the constant size and the related delay of $O(k^2ND)$.

Appendix D

DETAILS ON THE SPECTRAL GRAPH EMBEDDINGS

The description of Koren's algorithm and the related discussion is given in Appendix C. Here we discuss how it was used for the networks that we considered for the routing.

The spectral graph-drawing algorithm requires specifying two parameters in addition to the graph topology. One of them is the dimension of the desired metric space, and we set it to two in all our experiments for the sake of simplicity. We do not attempt to assign a geographic interpretation to the resulting coordinates. Tolerance is the second parameter that indirectly determines the number of iterations necessary for the iterative process of eigenvector calculation. The lower the tolerance, the smaller the changes in the same eigenvector between two consecutive iterations are required, and the better the drawing should be.

However, we run into the problem that most of the space available for the drawing remains unused. The main idea behind spectral drawing is that similar nodes are pulled close to each other (an edge is an evidence of similarity), and nodes that are not similar are pulled apart. When many nodes of a graph have very high degree, most of them can be considered as similar to each other and will be drawn within a very small portion of the space, while the rest of the space will be used to demonstrate the remoteness of the few least connected nodes (see Figure D.1). The lower the tolerance is, the smaller the portion of the space where most of the nodes are concentrated. In order to prevent this we will not always be using the lowest (time-wise possible) tolerance, but the (negative) power of 10 that leads to the best results for the greedy routing. That is, we will compare the results of the adaptive algorithm to the best observed results of the greedy routing.

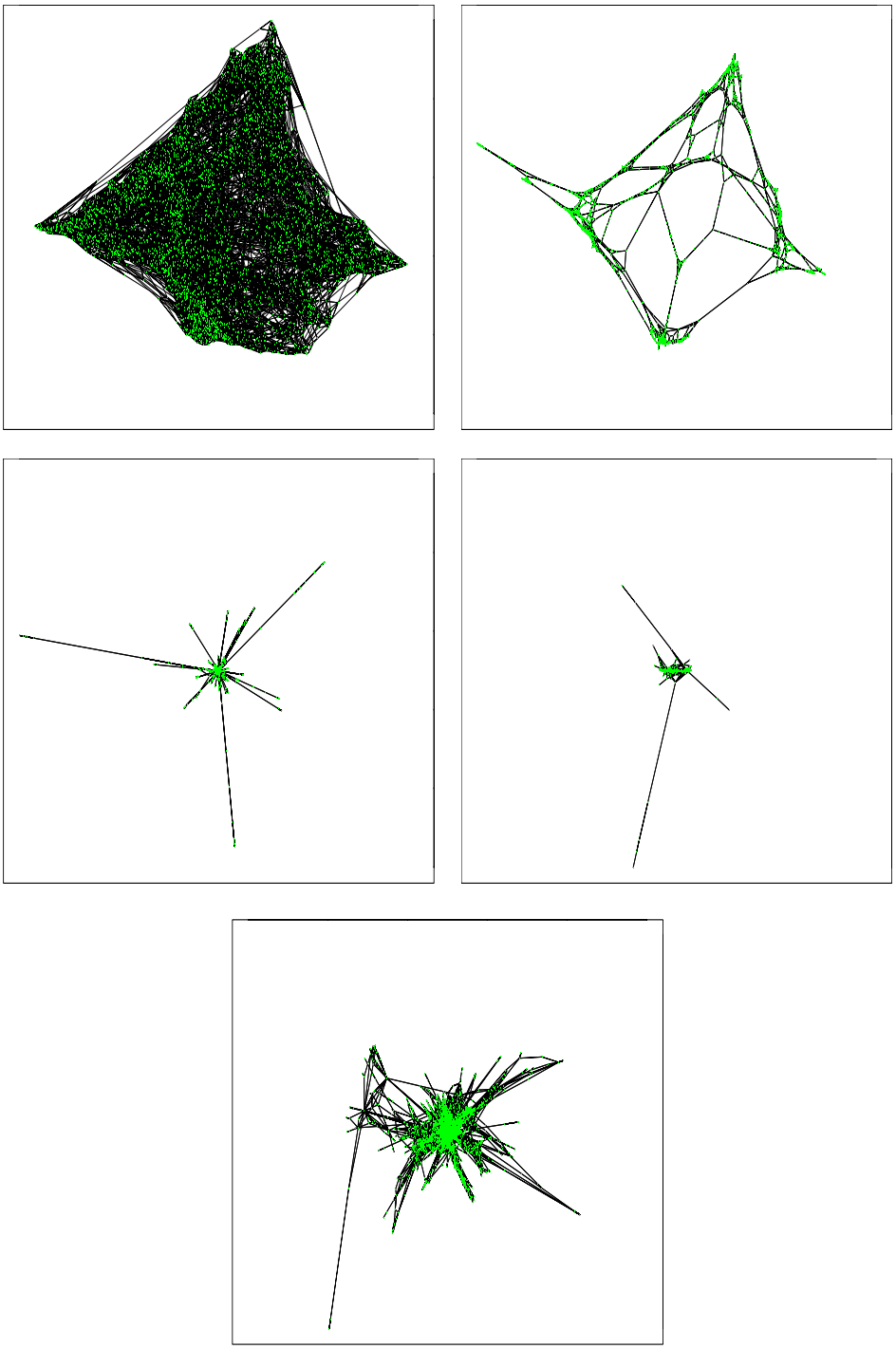

Figure D.1: Embedded networks. From the left to the right. Upper row: GB, PG. Middle row: CM, PB. Lower row: AS.

Appendix E

DETAILS ON THE DYNAMIC PARTITIONING

Regions that participate in a merge or a split belong to the direction of the same node. Let $k = \lfloor \log_2(2\sqrt{n}) \rfloor$ be the number of regions in a single (out of four natural possibilities) direction, and let D_{space} be the diameter of the metric space (the spectral graph-drawing algorithm assigns coordinates within a unit square). The boundaries between the regions of the same directions are circular arcs. Region $j = 2, 3, \cdots, k$ will cover area of the corresponding direction at distance $D_{space}/2^{k-j+1}$ to $D_{space}/2^{k-j}$ from the corresponding node, for $j = 1$ it is the area at distance up to $D_{space}/2^{k-1}$.

In order to allow the dynamic partitioning a node will support an additional counter per region (recall that regions are specific to nodes, and so each node maintains the necessary data). This counter keeps track of the number of times the region was successfully used (the message was eventually delivered to the target) since the latest initialization of the counter. When the sum of all k counters is larger than k^2 and divisible by 10, the algorithm checks if the regions have been used in the expected frequency. The algorithm expects that the regions are roughly used in the linearly biased way, that is, region j (where $j \geq 2$) is expected to be used j times more often than region 1 (initially, the region j is $3 \times 4^{j-2}$ times larger than the region 1, if there are no boundary effects). We introduce the notion of *usage rate* (of a region) to denote the ratio of the actual counter value to the expected counter value of the corresponding region. This notion can be extended to a pair of regions by summation of the values of their counters (actual or expected).

We will support *merge and split procedure* for regions of the same direction of the same node. The most overused region (the region of the direction with the highest usage rate) becomes the *candidate for split region*. The pair of two most underused neighboring regions (the pair of regions of the same direction with the lowest usage rate) neither of which is the candidate for split becomes the *candidate for merge pair*. If the usage rate of the candidate for split is at least 4 times higher than the usage rate of the candidate for merge pair, then the region boundaries will be simultaneously realigned by merging the candidate for merge pair in to one new region and splitting the candidate for split region into two new regions.

When two regions are merged, the weight of an expert in the new region is initialized with the weighted average of the corresponding weights from the merged regions. An expert's counter of number of stages the expert has been used is initialized with the sum of the corresponding counters from the merged regions.

When a region is split, the intermediate boundary between the new regions is drawn at the distance (from the node) which is an arithmetic mean of the distances to the arc boundaries of the old region. An expert's weight of a new region is initialized with the corresponding weight from the split region. An expert's counter of number of stages the expert has been used is initialized as the half of the corresponding counter from the split region.

All k lately introduced counters of the direction (one counter per region) are reset to zero after each merge and split procedure.