UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dottorato di Ricerca in Informatica
XXIV Ciclo

Diffusive Processes on Social Graphs

Relatore: Prof. Sebastiano Vigna

Tesi di Dottorato di:
Marco Rosa

Contents

1 Introduction **1**
1.1 Diffusive Processes 2
1.2 Compression 3
1.3 Distance distribution 6
1.4 Robustness 7
1.5 Arc clustering 10

2 Layered Label Propagation for graph compression **12**
2.1 Introduction 13
2.2 Problem Definition and Related Works 14
2.3 Our Contribution 17
2.4 Recovering Host information from a Random Permutation 18
2.5 Label Propagation Algorithms 20
2.6 Layered Label Propagation 23
2.7 Parallel Implementation 25
2.8 Experiments 26
2.9 Results 30
2.10 Conclusions 32

3 Approximating the neighbourhood function for large graphs **35**
3.1 Introduction 36
3.2 Related work 37
3.3 HyperANF 37
3.3.1 HyperLogLog counters 38
3.3.2 The HyperANF algorithm 40
3.3.3 HyperANF at hyper speed 40
3.3.4 Correctness, errors and memory usage 44

3.4 Deriving useful data . . . 46
3.5 SPID . . . 52
3.6 Experiments . . . 54
3.7 Facebook graph . . . 60
3.8 Conclusions . . . 63

4 Robustness of Social Networks 64
4.1 Introduction . . . 64
4.2 Related work . . . 66
4.3 Removal strategies and their analysis . . . 67
4.3.1 Some removal strategies . . . 68
4.3.2 Measures of divergence . . . 69
4.4 Experiments . . . 70
4.5 Discussion . . . 71

5 Arc-community detection via triangular random walks 77
5.1 Introduction . . . 78
5.2 Triangular random walks . . . 79
5.2.1 Triangular walks and line graphs . . . 82
5.3 Arc-clustering via triangular random walks . . . 86
5.4 Related works . . . 87
5.5 Experiments . . . 88
5.6 Conclusions . . . 97

Abstract

Social networks are emerging as one of the most revolutionary innovations of the last decades. Their impact in politics, social behaviour, economics is just at the very beginning and yet a better understanding of their structure is an urgent task.

Despite the importance of social networks is so self-evident, a scientific study of these objects have to face issues that could appear insurmountable. First of all even a definition, in a mathematical sense, of what a social network is does not exist. Secondly, even assuming we agreed on some definition, social networks have grown at an exceptional rate. Nowadays a typical social network have tens to hundreds millions of nodes and billions of arcs, so any algorithm that is more than linear (or linearithmic) in the number of arcs is out of question. Lastly, evaluating the performances of new techniques is not a trivial task since the majority of meta-data about social networks are industrial secrets of great economical value and are treasured as such.

In this thesis we present several results that, far from solving these problems, try to push a little forward our understanding of the very structure of social networks. The main theme of all the presented results is that much can be understood of the structure of a graph when we let some diffusive process evolve on it. Diffusive processes are inherently local and often randomized: each node of the graph chooses how to update its state just looking at its neighbours. Randomness is usually crucial in the initialization phase and in the update order. These kinds of processes have two major advantages: each round is linear in the number of arcs and they do not require that the whole graph is loaded into main memory. Thus they are perfect candidates for the analysis of huge networks.

Chapter 1

Introduction

The main interest of our research has been in understanding the structural properties that characterize social networks. How can we efficiently store and access huge social networks? How well connected they are? Is there any small subset of nodes which function as hub for the network? Are we able to outline a good clustering algorithm for social networks? Answering these questions is very important for both academic research and commercial applications.

Here we will give a brief outline of the problems addressed in this thesis.

- **Compression** We study how much social networks can be compressed while still being able to answer queries, seeking the neighbours of a node, in few hundreds of nanosecond. Besides the importance of compression for practical applications there is a wealth of evidence (e.g., [46]) that social networks are not random graphs in the usual sense. Studying the compressibility of a social network is akin to studying the degree of "randomness" in the social network.

- **Distance distribution** Since the early 1950s sociologists conjectured that everyone is on average approximately six steps away from any other person on earth, so that a chain of "a friend of a friend" statements can be made. Studying modern social networks we are able for the first time to answer this conjecture. However since an exhaustive research for all-pair shortest paths is out of question probabilistic techniques are the only tool available.

- **Robustness** In communication networks it is crucial to understand if the failure of some node can disrupt the connectivity. In social networks we can do a similar analysis under the hypothesis that influential nodes are the ones with the greatest

impact on the distance distribution. To this aim we need fast tools to evaluate the distance distribution and a good strategy to identify these influential nodes.

- **Arc Clustering** The presence of overlapped communities in social networks has recently given rise to new interest in arc clustering. The underlying idea is that if is true that an individual belongs to many communities the relation between two individuals is usually motivated by one specific reason (much in the same way as, albeit infinite lines pass through a single point, only a single line passes through two distinct points). Thus the shift from *communities of nodes* to *communities of arcs* can be thought of as trying to find the reasons behind relations rather than trying to find the reason behind individuals.

1.1 Diffusive Processes

In this section we will outline the main theme of this thesis: diffusive processes. To understand the importance of this concept we have to start describing the computational issue that arise in analysing social networks.

First of all we always think at social networks as directed graphs $G = (V, E)$ where V is a set of n nodes and $E \subseteq V \times V$ is a set of m arcs. We often assume that the graphs we are dealing with are very sparse, i.e. $m = O(n \cdot \log n)$, thus an algorithm with a computational complexity linear in the number of arcs is still feasible even for very large n (but you can not hide any "nasty" multiplicative constant within the asymptotic). Since we are dealing with sparse graphs we will represent them using adjacency lists like this:

Node	**Outdegree**	**Successors**
...	...	...
15	11	13, 15, 16, 17, 18, 19, 23, 24, 203, 315, 1034
16	10	15, 16, 17, 22, 23, 24, 315, 316, 317, 3041
17	0	
18	5	13, 15, 16, 17, 50
...	...	...

This naive representation will use $m + 2n$ integers instead of n^2 booleans of a dense adjacency matrix or $2m$ integer for an arc list representation. We will see later that we can do better than this, however we can ignore for the moment technical details for optimal compression and assume that an efficient framework for handling huge social

network will be able to answer in few hundreds of nanoseconds a query asking for the neighbourhood of a node.

Summing up, social networks are graphs, that can not be loaded into main memory, but we can recover efficiently the out-neighbours of a single node. Given these constraints diffusive processes arise very naturally. Let us define a diffusive process in the following way. Each node of the graph has some state that is initialised randomly. Then we can have asynchronous or synchronous update phases. In an asynchronous update at each step one node, chosen according to some probability function, "wakes up", looks at the current state of its neighbourhood, and then updates its state according to some rule. In a synchronous update instead each node wakes up in the same moment, each of them looks at the state of its neighbourhood, which is the state at the previous step, and then updates once again according to some rule. The update rule is usually a deterministic or probabilistic function which depends only on the current state of the node and of its neighbourhood. Thus an algorithm simulating a diffusive process must not load the whole graph into main memory, and it needs low memory requirements also for the global state (linear in the number of nodes).

Simple as it is, the idea to apply randomized diffusive processes has proven very effective in our applications. In the following sections we will describe for each problem we addressed how to exploit them.

1.2 Compression

The road to understand how diffusive processes can help in compression is quite tortuous. First of all remember that for typical social networks a good representation is given by an adjacency list. How can we improve upon this? The very first idea is to guarantee that successors are stored in increasing order; then we can store the gaps between two consecutive successors instead of store explicitly all the successors in the list. Following the example of the previous section we will obtain:

Node	**Outdegree**	**Gaps**
...	...	...
15	11	13, 2, 1, 1, 1, 1, 4, 1, 179, 112, 719
16	10	15, 1, 1, 5, 1, 1, 291, 1, 1, 2724
17	0	
18	5	13, 1, 1, 1, 33
...	...	...

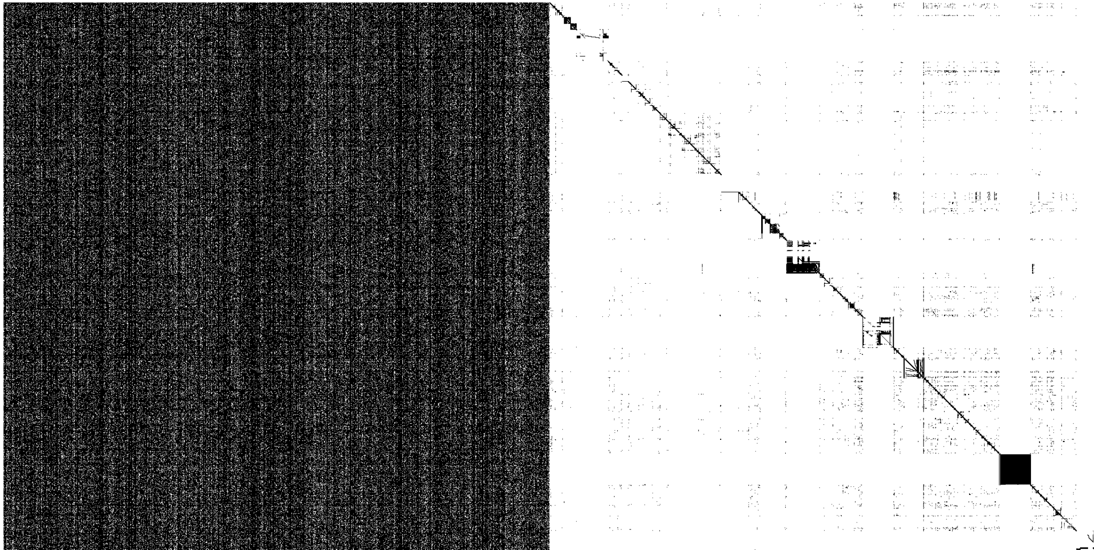

Figure 1.1: Adjacency matrix of the same web-graphs. On the left randomly permuted, on the right ordered lexicographically by URL.

That means that the node 15 has 11 successors, the first one 13 is represented explicitly, then to obtain the others we have to keep summing ($13 + 2 = 15, 15 + 1 = 16, 16 + 1 = 17, \ldots$). The gain of this operation is that we have to store integers that are surely smaller that the original ones. We can exploit this using prefix-free codes which assign small codewords to small integers, like nibble codes, δ-codes and ζ-codes introduced by Boldi and Vigna in [10]. Now, if social graphs were random graphs, we will need

$$m \cdot \log\left(\frac{n^2}{m}\right) + O(m) = m \cdot \log\left(\frac{n}{d}\right) + O(m)$$

bits to represent the graph (where d is the average degree). In term of bits per arc this means we would need $\log(n/d) + O(1)$. Luckily social graphs are not random graphs and we can exploit their structure to improve upon this theoretical lower bound. What we need is an ordering that keeps gaps small. For web graphs ordering nodes lexicographically by URL is sufficient to highlight the inner structure and achieve an impressive compression ratio of less then 3-bits per arc (see Figure 1.1).

Thus if we can find an ordering that highlights a block-diagonal structure on the adjacency matrix we will obtain a very good compression ratio. This task easily translates into a clustering problem, if nodes in the same cluster are consecutive in the ordering

we will obtain a block-structure of the adjacency matrix and thus very small gaps. Here is where diffusive processes come into play.

To cluster the graph we used the following diffusive process. At the beginning of each round every node has a label representing the cluster that the node currently belongs (at the beginning, every node has a different label). At each round, every node takes the label that occurs more frequently in its neighbourhood, the update order being chosen at random at the beginning of the round; the algorithm terminates as soon as no more updates take place. Metaphorically, every node in the network chooses to join the largest neighbouring community (i.e., the one to which the maximum number of its neighbours belongs). As labels propagate, densely connected groups of nodes quickly reach a consensus on a unique label. When many such dense consensus groups are created throughout the network, they continue to expand outwards until it is possible to do so. At the end of the propagation process, nodes having the same labels are grouped together as one community.

It has been proved [70] that this kind of label propagation is formally equivalent to finding the local minima of the Hamiltonian for a kinetic Potts model. This problem has a trivial globally optimal solution when all the nodes have the same label; nonetheless, since the label-propagation optimisation procedure produces only local changes, the search for minima in the Hamiltonian is prone to becoming trapped at a local optimum instead of reaching the global optimum. While normally a drawback of local search algorithms, this characteristic is essential to clustering: the trivial optimal solution is avoided by the dynamics of the local search algorithm, rather than through formal exclusion.

We will see in Chapter 2 details about how this idea can be improved to obtain better compression ratios. However we have already introduced the two fundamental components of this thesis: *randomness* and *locality*.

- **Randomness**: to cite [55] “For many applications a randomized algorithm is either the simplest algorithm available, or the fastest, or both.”. Randomized algorithms are often able to solve hard problems in linear time, which is of course crucial in a setting where even a quadratic deterministic algorithm is out of question.

- **Locality**: By means of locality we almost totally circumvent problems due to the huge size of networks we are analysing. No matter how huge the graph is we can always efficiently load the neighbourhood of a single node, apply our local update rule and let the system evolve toward a stable state.

1.3 Distance distribution

The *neighbourhood function* $N_G(t)$ of a graph G gives, for each $t \in \mathbf{N}$, the number of pairs of nodes $\langle x, y \rangle$ such that y is reachable from x in less that t hops. Normalizing by the total number of reachable pairs we obtain the cumulative function of the distance distribution $H_G(t)$ (the distance function is the shortest path distance).

The distance distribution contains a wealth of information about the connectivity of the graph, but requires a breadth-first for each node to be computed exactly which is clearly unfeasible. As usual we will see how randomization and locality can efficiently solve this problem with precise error bounds.

The main tools we need are the *HyperLogLog counters* (see [30]) to count approximately the number of distinct elements in a stream. Essentially, these probabilistic counters are a sort of *approximate set representations* to which, however, we are only allowed to pose questions about the (approximate) size of the set. To accomplish this we need a very good hash function (we will see in Chapter 3 the precise requirements) from V to 2^∞, and, instead to keep adding elements to our approximate set representation, we only keep track of M the maximum number of trailing zeros in the hash values of elements seen so far. The estimated number of distinct elements seen so far is $\propto 2^M$. To keep track of the maximum for a stream with at most k different items we have to use only $\log \log k$ bits of memory. A crucial observation is that the counter of a stream AB is simply the maximum between the counters of A and B.

Now the basic idea is to put one of these counters on each node, initialize it with the number of trailing zeros in the hashed value of the node itself, and then propagate it by maximization with its neighbourhood at each step with a synchronous update. Thus exploiting locality we never have to load the whole graph into main memory, we just have to load two arrays of counters and the neighbourhood of a single node at a time.

We will see in Chapter 3 how to give precise errors bounds on the standard deviation of the resulting neighbourhood function, moreover there are a lot of technical difficulties and optimizations that can be done. However once again we can see how an intractable problem can be (at least approximately) solved very efficiently by means of a probabilistic diffusive process.

One thing to be aware of is that these kinds of processes are very subtle and often much-neglected little details can have a great impact on the final results. One example of these phenomena is the termination condition of a process like the one we have just described. Making experiments on real graphs shows that after few steps almost no counter changes anymore, and the algorithm spend an incredible amount of time for a ridiculously small number of counters. This can suggest to put as stopping criterion a certain threshold of relative increment in the neighbourhood function. This solution

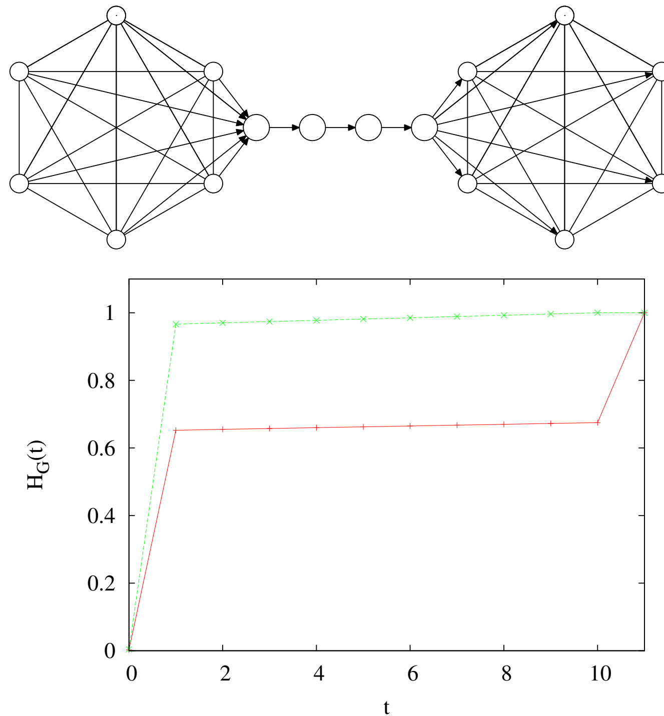

Figure 1.2: Two k-cliques joined by a unidirectional path of ℓ nodes: terminating even one step earlier than stabilisation completely miscalculates the distance cdf; the effective diameter is $\ell + 1$, but terminating even just one step earlier than stabilisation yields an estimated effective diameter of 1.

can lead to an arbitrary high error in the computation of the neighbourhood function as shown by Figure 1.2.

The solution adopted in our work is to run until stabilization exploiting systolic computation: each node *signals* back to predecessors that they may need to update in the next round only when its counter has changed. To do this we need also the transposed version of the graph, but gains in speed and accuracy are striking.

1.4 Robustness

In Chapter 4 we will exploit our tool (that efficiently approximate the distance distribution) to understand which node-removal strategy has the greatest impact on the network structure. More formally, consider a given total order $\prec$ on the nodes of G; we

think of $\prec$ as a removal strategy in the following sense: when we want to remove θm arcs, we start removing the $\prec$-largest node (and its incident arcs), go on removing the second-$\prec$-largest node etc. and stop as soon as $\geq \theta m$ arcs have been removed. The resulting graph will be denoted by $G(\prec, \theta)$. Of course, $G(\prec, 0) = G$ whereas $G(\prec, 1)$ is the empty graph. We are interested in applying some measure of *divergence* between the distribution H_G and the distribution $H_{G(\prec,\theta)}$. By looking at the divergence when θ varies, we can judge the ability of $\prec$ to identify nodes that will disrupt the network.

As in the case of compression we have a different situation between web graphs and social networks. In web graphs removing roots of web-sites has a huge impact on the distance distribution, while in social networks we haven't any extrinsic strategy that achieves similar results. As always we started searching for an intrinsic strategy that works in web graphs as good as the root-removal.

Our first attempt exploits a technique for clustering graphs using PageRank vectors presented in [3]. This technique is a perfect example of how diffusive processes can be exploited to achieve good approximation efficiently so we will describe it in details even if in the end it is not the best technique to find relevant nodes. Notice also that here we will describe the algorithm for the undirected case for sake of simplicity.

First of all we introduce a lazy variation of PageRank, which we define to be the unique solution $\mathrm{pr}(\alpha, s)$ of the equation

$$\mathrm{pr}(\alpha, s) = \alpha s + (1 - \alpha)\mathrm{pr}(\alpha, s)W \tag{1.1}$$

where α is a constant in $(0, 1]$ called the teleportation constant, s is a distribution called the preference vector, and W is the lazy random walk transition matrix $W = \frac{1}{2}(I + D^{-1}A)$ (as usual A is the adjacency matrix of the graph and D is the degree matrix). In the case of local PageRank for a node $v \in V$ we have $s = \chi_v$. We maintain a pair of distributions: an approximate PageRank vector p and its associated residual vector r. Initially, we set $p = 0$ and $r = \chi_v$. We then apply a series of push operations, based on equation (1.1), which alter p and r. Each push operation takes a single vertex u, moves an α fraction of the probability from $r(u)$ onto $p(u)$, and then spreads the remaining $(1 - \alpha)$ fraction within r, as if a single step of the lazy random walk were applied only to the vertex u. Each push operation maintains the invariant

$$p + pr(\alpha, r) = pr(\alpha, \chi_v),$$

which ensures that p is an approximate PageRank vector for $pr(\alpha, \chi_v)$ after any sequence of push operations. We now formally define $\mathtt{push}_u$, which performs this push operation on the distributions p and r at a chosen vertex u.

Algorithm 1 $\texttt{push}_u$

1: Let $p' = p$ and $r' = r$, except for the following changes:

- $p'(u) = p(u) + \alpha r(u)$
- $r'(u) = (1 - \alpha)r(u)/2$
- For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha)r(u)/(2d(u))$.

2: Return (p', r').

Lemma 1. *Let p' and r' be the results of the $\texttt{push}_u$ on p and r. Then*

$$p' + pr(\alpha, r') = p + pr(\alpha, r)$$

During each push, some probability is moved from r to p, where it remains, and after sufficiently many pushes r can be made small. We can bound the number of pushes required by the following algorithm:

Algorithm 2 $\texttt{ApproxPR}(v, \alpha, \varepsilon)$

1: Let $p = \vec{0}$ and $r = \chi_v$
2: **while** $\max_{u \in V} \frac{r(u)}{d(u)} > \varepsilon$ **do**
3: Choose any vertex u such that $\frac{r(u)}{d(u)} > \varepsilon$
4: Apply $\texttt{push}_u$ at vertex u, updating p and r
5: **end while**
6: **return** p

Lemma 2. *Let T be the total number of push operations performed by **ApproxPR**, and let d_i be the degree of the vertex u used in the i-th push. Then*

$$\sum_{i=1}^{T} d_i \leq \frac{1}{\varepsilon\alpha}$$

Proof. The amount of probability $r(u)$ on the vertex pushed at time i is at least εd_i, therefore $|r|_1$ decreases by at least $\alpha\varepsilon d_i$ during the i-th push. Since $|r|_1 = 1$ initially, we have

$$\alpha\varepsilon \sum_{i=1}^{T} d_i \leq 1$$

and the result follows. □

This is an interesting diffusive process because it is one of the few examples in which randomness plays no role, we just exploit the structure of the network. The results are twofold. We can compute personalized PageRank vector very fast; actually what we are computing is the ratio between local PageRank and usual PageRank, however since the stable distribution for PageRank in a symmetric graph is trivial we can easily deduce the local PageRank vector. Moreover, and perhaps more importantly, we can use this approximation to find a provably good cut (in term of conductance) for a set containing the initial vertex v.

We will see in Chapter 4 that the best technique, even more effective then the root-removal in web graphs, is to cluster the graph with a label propagation algorithm and then remove nodes which have more inter-cluster connections. Once again however social networks prove to be completely different from web graphs and exhibit a more connected structure.

1.5 Arc clustering

Our approach to arc clustering is not based on a diffusive process. The main concept presented in Chapter 5 is the triangular random walk. We will define a triangular random walk $X_0, X_1, \dots$ using two parameters, $\alpha, \beta \in [0,1]$: α is a damping factor (it is used to decide whether to follow a link or to teleport); β will instead be used to determine whether triangles or non-triangles should be privileged. In a triangular random walk with parameters α and β, the next node (x_{t+1}) is chosen depending on the current node x_t and on the previous node x_{t-1}, as follows:

- with probability $1-\alpha$, we teleport: x_{t+1} is a randomly chosen node;
- otherwise, we choose among the successors $N(x_t)$ of the current node, but treating differently the *triangular successors* (the set $N(x_t) \cap N(x_{t-1})$) and the *non-triangular successors* (the set $N(x_t) \setminus N(x_{t-1})$)[1]. We first decide whether we shall select a non-triangular successor (with probability β) or a triangular one (with probability $1-\beta$); then, the specific non-triangular or triangular successor is chosen uniformly at random

A triangular random walk is a Markov chain of order 2 [69], because the next state depends on the current state *and* on the previous one. To study the long-term behaviour of higher order chains, it is customary to change the state space and reduce the stochastic

[1] If either set is empty (or if $t = 1$) we choose uniformly in $N(x_t)$ (or in V, if the latter is empty), as in a standard random walk.

process to an equivalent one that is memoryless; this is easily solved by using the notion of *line graph*. Given a graph G, its line graph $L = L(G)$ has the arcs of G as vertices (i.e., $V_L = E_G$), with arcs of the form (xy, yz) (where xy and yz are two arcs of G).

Now, it is easy to see that a triangular random walk with parameters α, β on the (unweighed) graph G is equivalent to a random walk with damping factor α on the weighted line graph $L(G)$, where

$$w_T(xy, yz) = \begin{cases} \frac{\beta}{|N(y)\setminus N(x)|} & \text{if } z \in N(y) \setminus N(x) \\ \frac{1-\beta}{|N(y)\cap N(x)|} & \text{if } z \in N(y) \cap N(x). \end{cases}$$

In other words, every arc in $L(G)$ (that is to say, every two-step walk $x \to y \to z$ in the original graph) has a different weight depending on whether it can be closed by a triangle (i.e., if $x \to z$ was also an arc of G) or not.

Along the same line as [29], instead of clustering directly the arcs of G (as done, for example, by [42]), we turn to some suitably weighted version of the line graph $L(G)$, where we can make good use of all the paraphernalia for node-clustering of a directed graph. In other words, we shall use an off-the-shelf node-clustering algorithm feeding it with the weighted graph $L(G)$. As weighting function (on the arcs of $L(G)$), we exploit its arc-stationary distribution

$$v_T(xy, yz) = v_T(xy)w_T(xy, yz)$$

where $v_T(xy)$ is the stationary distribution of the triangular random surfer on the node xy.

So we need three steps in order to obtain our arc-clustering. First we need to compute $L(G)$ with the weighting function w_T. Next we need to compute the stable distribution of a weighted random walk on $L(G)$ and, in the end, we need a clustering algorithm for directed weighted graphs.

The clustering technique used is the one presented in [9] which is a typical diffusive process (very similar to label propagation). Unfortunately for the first step we have not found any ways to solve the problem efficiently and thus we can only handle networks with few millions of nodes.

Chapter 2

Layered Label Propagation for graph compression

In this chapter we will present highly scalable techniques that improve compressed data structures for representing web graphs and social networks significantly beyond the current state-of-art. These improvements make it possible to analyse in main memory significantly larger graphs, and shed some lights on the internal structure of social networks.

Our starting point is the incredible compression ratio that can be achieved on web graphs. Our first finding is that most of the existing technique heavily rely on the (often implicit) assumption that web graphs are given sorted by URL. We call these compression techniques non coordinate-free since their performances are not independent from the starting order of the nodes of the graph. Since in a social network we do not have anything like the URL ordering it comes to no surprise that these algorithms perform poorly on them.

After a more focused study on web graphs we discover that the crucial property that URL ordering is able to capture is that web pages are clustered on a host basis. Thus the problem induces (although it is not equivalent to) a clustering problem on the graph. So we devise a clustering technique which is scalable to billions of nodes, as it just requires few linear passes over the graph involved. Thanks to this clustering technique we have been able, for the first time, to beat the compressions ratios of the URL ordering starting from a randomly permuted web graph.

Finally, we apply our new method to social networks hoping to achieve similar compressions ratios we observe in web graphs. Unfortunately, even if we strongly improve upon existing techniques, we are very far from the $2 \sim 3$ bits per link of web graphs. The simplest explanation is that social networks are intrinsically less compressible because

of their internal structure, however no one knows how much space for improvements is left.

2.1 Introduction

The acquaintance structure underlying a social network contains a wealth of information about the network itself, and many data mining tasks can be accomplished from this information alone (e.g., detecting outlier nodes, identifying interest groups, estimating measures of centrality etc. [75, 43]). Many of these tasks translate into graph mining problems and can be solved through suitable (sometimes, variants of standard) graph algorithms that often assume that the graph is stored into the main memory. However, this assumption is far from trivial when large graphs are dealt with, and this is actually the case when social networks are considered; for instance, current estimates say that the indexable web contains at least 23.59 billion pages[1], and in 2008 Google announced to have crawled 1 trillion unique URLs: the successor lists for such a graph would require hundreds of terabytes of memory! The situation is somewhat similar in other social networks; for example, as of October 2012[2] , Facebook has more than 700 millions users and 65 billions friendship relations.

In this chapter we will describe a novel technique to store and access large graphs that can be applied fruitfully not only to web graphs but also to social networks of other kinds. The considerations above explain why this problem is lately emerging as one of the central algorithmic issues in the field of information retrieval [35, 23]; it should also be noted that improving the compression performance on a class of networks, apart for its obvious practical consequences, implies (and requires) a better understanding of the regularities and of the very structure of such networks.

It is evident that the relation between social-graph compression and data mining is twofold: on one hand, almost no complex graph-mining task can be performed without relying on a framework that is able to store and access efficiently the graph under consideration; on the other hand, every approach that improves on our ability to compress a class of networks suggests structural properties of those networks, thus becoming itself a data mining challenge. As an example the impressive compression results obtained in [17] put light on a structural property of the web, that is, the majority of links in the web are intra-host and automatically generated.

Here and in the following, we are thinking of *compressed data structures*. A compressed data structure for a graph must provide very fast amortised random access to

[1] `http://www.worldwidewebsize.com/`
[2] `http://www.facebook.com/press/info.php?statistics`

an edge (link), say in the order of few hundreds of nanoseconds, as opposed to a "compression scheme", whose only evaluation criterion is the number of bits per link. While this definition is not formal, it excludes methods in which the successors of a node are not accessible unless, for instance, a large part of the graph is scanned. In a sense, compressed data structures are the empirical counterpart of *succinct* data structures (introduced by Jacobson [39]), which store data using a number of bits equal to the information-theoretical lower bound, providing access asymptotically equivalent to a standard data structure.

The idea of using a compressed data structure to store social networks was already successfully exploited with application to web graphs [17], showing that such graphs may be stored using less than 3 bits/link; this impressive compression ratio is mostly obtained by making good use of two simple properties that can be experimentally observed when nodes are ordered lexicographically by URL [63]:

- *similarity*: nodes that are close to each other in the order tend to have similar sets of neighbours;
- *locality*: most links are between nodes that are close to each other in the order.

The fact that most compression algorithms exploit these (or analogous) properties explains why such algorithms are so sensible to the way nodes are ordered; the solution of ordering nodes lexicographically by URL is usually considered good enough for all practical purposes, and has the extra advantage that even the URL list can be compressed very efficiently via prefix omission. Analogous techniques, which use additional information besides the graph itself, are called *extrinsic*. One natural and important question is whether there exist any *intrinsic* order of the nodes (i.e., one that does not rely on any external data) that produces comparable, or maybe even better, compression ratios. This is particularly urgent for general social networks, where the very notion of URL does no longer apply and finding a natural extrinsic order is problematic [23, 15].

2.2 Problem Definition and Related Works

The general problem we consider may be stated as follows: a graph-compression algorithm $\mathscr{A}$ takes (the adjacency matrix of) a graph as input and stores it in a compressed data structure; the algorithm output depends on the specific numbering chosen for the nodes. We let $\rho_{\mathscr{A}}(G, \pi)$ be the number of bits per link needed by $\mathscr{A}$ to store the graph G under the given node numbering[3] $\pi : V_G \to |V_G|$. The overall objective is to find

[3] We use von Neumann's notation $n = \{ 0, 1, \dots, n-1 \}$.

a numbering $\hat{\pi}$ minimising $\rho_{\mathscr{A}}(G, \hat{\pi})$. In the following, we shall always assume that a graph G with n nodes has $V_G = n$, so a node numbering is actually a permutation $\pi : n \to n$.

Of course, the problem has different solutions depending on the specific compression algorithm $\mathscr{A}$ that is taken into consideration. In the following, we shall focus on the so-called BV compression scheme [17] used within the WebGraph framework, which incorporates the main ideas adopted in earlier systems and is a *de facto* standard for handling large web-like graphs. In particular, the framework strongly relies on similarity and locality to achieve its good compression results; for this reason, we believe that most compressed structures that are based on the same properties will probably display a similar behaviour.

As noted in [23], even a very mild version of the above-stated optimisation problem turns out to be NP-hard, so we can only expect to devise heuristics that work well in most practical cases. Such heuristics may be intrinsic or extrinsic, depending on whether they only use the information contained in the graph itself or they also depend on some external knowledge.

In the class of intrinsic order heuristics, [63] proposes to choose the permutation π that would sort the rows of the adjacency matrix A_G in lexicographic order. This is an example of a more general kind of solution: fix some total ordering $\prec$ on the set of n-bit vectors (e.g., the lexicographic ordering), and let π be the permutation that would sort the rows of the adjacency matrix A_G according to[4] $\prec$.

Another possible solution in the same class, already mentioned in [63] and studied more deeply in [15], consists in letting $\prec$ be a Gray ordering. Recall that [44] an *n-bit Gray ordering* is a total order on the set of the 2^n binary n-bit vectors such that any two successive vectors differ in exactly one position. Although many n-bit Gray ordering exist, a very effective one (i.e., one that is manageable in practice because it is easy to decide which of two vectors come first in the order) is the so-called *reflective n-bit Gray ordering*, which was used in [15].

Chierichetti *et al.* [23] propose a completely different intrinsic approach based on shingles that adopts ideas used for document similarity derived from min-wise independence. The compression results they get are comparable to those achieved through Gray ordering [15]. In [23], the authors also discuss an alternative compression technique (called BL) that provides better ratios; however, while interesting as a compression scheme, BL does not provide a compressed data structure—recovering the successors of

[4] Here we are disregarding the problem that π is not unique if the adjacency matrix contains duplicated rows. This issue turns out to have a negligible impact on compression and will be ignored in the following.

a node requires, in principle, decompressing the whole graph.

Recently, Safro and Temkin [67] presented a multiscale approach for the network minimum logarithmic arrangement problem: their method searches for an intrinsic ordering that optimises directly the sum of the logarithms of the gaps (numerical difference between two successive neighbours). Although their work is not aimed at compression, their ordering is potentially useful for this task if combined with a compression scheme like BV. Indeed, some preliminary tests show that these orderings are promising especially on social networks; however, their implementation does not scale well to datasets with more that a few millions of nodes and so it is impractical for our purpose.

As far as extrinsic orderings are concerned, a central rôle is played by the URL-based ordering in a web graph. If G is a web graph, we can assume to have a permutation π_U of its nodes that sorts them according to the lexicographic URL ordering: this extrinsic heuristic dates back to [8] and, as explained above, turns out to give very good compression, but it is clearly of no use in non-web social networks. Another effective way to exploit the host information is presented in [15], where URLs from the same host are kept adjacent (within the same host, Gray ordering is used instead).

It is worth remarking that all the intrinsic techniques mentioned above produce different results (and, in particular, attain different compression ratios) depending on the *initial* numbering of the nodes, because they work on the adjacency matrix A_G. This fact was overlooked in almost all previous literature, but it turns out to be very relevant: applying one of these intrinsic re-ordering to a randomly numbered graph (see Table 2.7) produces worse compression ratios than starting from a URL-ordered web graph (see Table 2.6).

This problem arises because even if the intrinsic techniques described above do not explicitly use any external information, the initial order of a graph is often obtained by means of some external information, so the compression performances cannot be really considered intrinsic. To make this point clear, we will always speak of *coordinate-free* algorithms for those algorithms that achieve almost the same compression performances starting from any initial ordering; this adjective can be applied both to compression algorithms and to orderings+compression algorithm pairs. From an experimental viewpoint, this means that, unlike in the previous literature, we run all our tests starting from a *random permutation* of the original graph. We suggest this approach as a baseline for future research, as it avoids any dependency on the way in which the graph is presented initially.

The only coordinate-free compression algorithm we are aware of[5] is that proposed

[5]The quite extensive survey in [16] shows that many other approaches to web-graph compression, not quoted here, either fail to compress social networks, or are strongly dependent on the initial ordering

by Apostolico and Drovandi in [4];[6] they exploit a breadth-first search (BFS) to obtain an ordering of the graph and they devise a new compression scheme that takes full advantage of it. Their algorithm has a parameter, the *level*, which can be tuned to obtain different trade-offs between compression performance and time to retrieve the adjacency list of a node: at level 8 they attain better compression performances than those obtained by BV with Gray orderings and have a similar speed in retrieving the adjacency list. Even in this optimal setting, though, their approach is outperformed by the one we are going to present (see Table 2.5).

Finally, Maserrat and Pei [52] propose a completely different approach that does not rely on a specific permutation of the graph. Their method compresses social networks by exploiting Eulerian data structures and multi-position linearisations of directed graphs. Notably, their technique is able to answer both successor and predecessor queries: however, while querying for adjacency of two nodes is a fast operation, the cost per link of enumerating the successors and predecessors of a node is between one and two orders of magnitude larger than what we allowed. In other words, by the standards defined here, their algorithm does not qualify as a compressed data structure.

We must also remark that the comparison given in [52] of the compression ratio w.r.t. WebGraph's BV scheme is quite unfair: indeed, the authors argue that since their algorithm provides both predecessors and successors, the right comparison with the BV scheme requires roughly doubling the number of bits per link (as the BV scheme just returns successors). However, this bound is quite naïve: consider a simple strategy that uses the set E_{sym} of all symmetric edges, and let $G_{\mathrm{sym}} = (V, E_{\mathrm{sym}})$ and $G_{\mathrm{res}} = (V, E \setminus E_{\mathrm{sym}})$. To be able to answer both successor and predecessor queries one can just store G_{sym}, G_{res} and G_{res} transposed. Using this simple strategy and applying the ordering proposed in this chapter to the datasets used in [23] we obtain better compression ratios.

2.3 Our Contribution

In this chapter we give a number of algorithmic and experimental results:

- We identify two measures of fitness for algorithms that try to recover the host structure of the web, and report experiments on large web graphs that suggest

of the graph.

[6] Our experiments show in fact a very limited variation in compression (10–15%) when starting from URL ordering or from a random permutation, except for the `altavista-nd` dataset, which however is quite pathological.

that the success of the best coordinate-free orderings is probably due to their capability of guessing the host structure.

- Since the existing coordinate-free orderings do not work well on social networks, we propose a new algorithm, called *Layered Label Propagation*, that builds on previous work on scalable clustering by label propagation [62, 64]; the algorithm can reorder very large graphs (billions of nodes), and unlike previous proposals, is free from parameters.
- We report experiments on the compression of a wide array of web graphs and social networks using WebGraph after a reordering by Layered Label Propagation; the experiments show that our combination of techniques provides a major increase in compression with respect to all currently known approaches. This is particularly surprising in view of the fact that we obtain the best results *both* on web graphs *and* on social networks. Our largest graph contains more than 600 millions nodes—one order of magnitude more than any published result in this area.

Almost all the datasets can be downloaded from `http://law.dsi.unimi.it/` (or from other public or free sources) and have been widely used in the previous literature to benchmark compression algorithms. The Java code for our new algorithm is distributed at the same URL under the GNU General Public License.

We remark that our new algorithm has also been applied with excellent results to the Minimum Logarithmic Arrangement Problem [67] [7].

2.4 Recovering Host information from a Random Permutation

As a warm-up towards our new algorithm, we propose an empirical analysis that aims at determining objectively why existing approaches compress well web graphs.

The results presented in [15] suggest that what is really important in order to achieve good compression performances on web graphs is not the URL ordering *per se*, but rather an ordering that keeps nodes from the same host close to one another. For this reason, we will be naturally interested to measure how much a given ordering π respects the partition induced by the hosts, $\mathscr{H}$.

[7] `http://www.mcs.anl.gov/~safro/mloga.html`. The authors had been provided a preliminary version of our code to perform their tests.

The *first measure* we propose is the probability to have a *host transition* (HT):

$$\mathrm{HT}(\mathscr{H},\pi) = \frac{\sum_{i=1}^{|V_G|-1} \delta\left(\mathscr{H}[\pi^{-1}(i)], \mathscr{H}[\pi^{-1}(i-1)]\right)}{|V_G|-1}$$

where δ denotes the usual Kronecker's delta and $\mathscr{H}[x]$ is the equivalence class of node x (i.e., the set of all nodes that have the same host as x): this is simply the fraction of nodes that are followed, in the order π, by another node with a different host.

Alternatively, we can reason as follows: the ordering induces a refinement of the original host partition, and the appropriateness of a given ordering can be measured by comparing the original partition with the refined one. More formally, let us denote with $\mathscr{H}_{|\pi}$ the partition induced by the reflexive and transitive closure of the relation ρ defined by

$$x \,\rho\, y \iff |\pi(x) - \pi(y)| = 1 \text{ and } \mathscr{H}[x] = \mathscr{H}[y].$$

Intuitively, the classes of $\mathscr{H}_{|\pi}$ are made of nodes belonging to the same host and that are separated in the order only by nodes of the same host. Notice that this is always a refinement of the partition $\mathscr{H}$.

The *second measure* that we have decided to employ to compare partitions is the Variation of Information (VI) proposed in [53]. Define the entropy associated with the partition $\mathscr{S}$ as:

$$H(\mathscr{S}) = -\sum_{S\in\mathscr{S}} P(S)\log(P(S)) \quad \text{where } P(S) = \frac{|S|}{|V_G|}$$

and the mutual information between two partitions as:

$$I(\mathscr{S},\mathscr{T}) = \sum_{S\in\mathscr{S}}\sum_{T\in\mathscr{T}} P(S,T)\log\frac{P(S,T)}{P(S)P(T)}$$

where $P(S,T) = \frac{|S\cap T|}{|V_G|}$. The Variation of information is then defined as

$$VI(\mathscr{S},\mathscr{T}) = H(\mathscr{S}) + H(\mathscr{T}) - 2\,I(\mathscr{S},\mathscr{T});$$

notice that, in our setting, since $\mathscr{H}_{|\pi}$ is always a refinement of $\mathscr{H}$, we have $I(\mathscr{H},\mathscr{H}_{|\pi}) = H(\mathscr{H})$ and so VI simplifies into

$$VI(\mathscr{H},\mathscr{H}_{|\pi}) = H(\mathscr{H}_{|\pi}) - H(\mathscr{H}).$$

Armed with these definitions, we can determine how much different intrinsic orderings are able to identify the original host structure. We computed the two measures defined above on a number of web graphs (see Section 2.8) and using some different orderings described in the literature; more precisely, we considered:

Name	LLP		BFS		Shingle		Gray		Natural		Random	
	HT	**VI**	**HT**	**VI**	**HT**	**VI**	**HT**	**VI**	**HT**	**VI**	**HT**	**VI**
eu	**1.58%**	**4.60**	2.04%	**4.60**	20.12%	7.33	20.09%	7.55	0.05%	0.00	97.11%	13.80
in	**1.83%**	**1.92**	2.53%	2.32	15.83%	4.51	37.11%	6.76	0.32%	0.00	99.62%	11.37
indochina	**1.37%**	**1.61**	1.99%	2.63	32.05%	6.03	30.96%	5.93	0.26%	0.00	99.93%	11.71
it	3.05%	**2.63**	**2.93%**	2.83	27.04%	5.32	26.18%	5.27	0.34%	0.00	99.99%	11.45
uk	2.52%	2.88	**1.29%**	**2.65**	20.64%	5.52	19.93%	5.46	0.11%	0.00	99.98%	13.76

Table 2.1: Various measures to evaluate the ability of different orderings to recover host information. Smaller values indicate a better recovery.

- *Random*: a random node order;
- *Natural*: for web graphs, this is the URL-based ordering; for the other non-web social networks, it is the order in which nodes are presented, which is essentially arbitrary (and indeed produces compression ratios not very different from random);
- *Gray*: the Gray order explained in [15];
- *Shingle*: the compression-friendly order described in [23];
- *BFS*: the breadth-first search traversal order, exploited in [4];
- *LLP*: the Layered Label Propagation algorithm described in this thesis (see Section 2.6 for details).

The results of this experiment are shown in Table 2.1; comparing them with the compression results of Table 2.7 (that shows the compression performances starting from a truly random order), it is clear that recovering the host structure from random is the key property that is needed for obtaining a real coordinate-free algorithm. However, the only ordering proposed so far that is able to do this is breadth-first search, and its capability to identify hosts seems actually a side effect of the very structure of the web. We use BFS as a strong baseline against which our new results should be compared.[8].

2.5 Label Propagation Algorithms

Most of the intrinsic orderings proposed so far in the literature are unable to produce satisfactory compression ratios when applied to a randomly permuted graph, mainly

[8]It is unlikely that, in presence of the more complicated structure that we expect in social networks, an algorithm as simple as a breadth-first search can identify meaningful clusters (see again the BFS column of Table 2.7), and this leaves room for improvement.

because they mostly fail in reconstructing host information as we discussed in the last section. To overcome their limitations, we can try to approach this issue as a clustering problem. However, this attempt presents a number of difficulties that are rather peculiar. First of all, the size of the graphs we are dealing with imposes to use algorithms that scale linearly with the number of arcs (and there are very few of them; see [32]). Moreover, we do not possess any prior information on the number of clusters we should expect and their sizes are going to be highly unbalanced.

These difficulties strongly restrict the choice of the clustering algorithm. In the last years, a new family of clustering algorithms were developed starting from the label propagation algorithm presented in [62], that use the network structure alone as their guide and require neither optimisation of a predefined objective function nor prior information about the communities. These algorithms are inherently local, linear in the number of edges, and require just few passes on the graph.

The main idea of label propagation algorithms is the following: the algorithms execute in rounds, and at the beginning of each round every node has a label representing the cluster that the node currently belongs (at the beginning, every node has a different label). At each round, every node will update its label according to some rule, the update order being chosen at random at the beginning of the round; the algorithm terminates as soon as no more updates take place. Label propagation algorithms differ from each other on the basis of the update rule.

The algorithm described in [62] (hereafter referred to as *standard label propagation* or just *label propagation*) works on a purely local basis: every node takes the label that occurs more frequently in its neighbourhood[9]. Metaphorically, every node in the network chooses to join the largest neighbouring community (i.e., the one to which the maximum number of its neighbours belongs). As labels propagate, densely connected groups of nodes quickly reach a consensus on a unique label. When many such dense consensus groups are created throughout the network, they continue to expand outwards until it is possible to do so. At the end of the propagation process, nodes having the same labels are grouped together as one community.

It has been proved [70] that this kind of label propagation algorithm is formally equivalent to finding the local minima of the Hamiltonian for a kinetic Potts model. This problem has a trivial globally optimal solution when all the nodes have the same label; nonetheless, since the label-propagation optimisation procedure produces only local changes, the search for maxima in the Hamiltonian is prone to becoming trapped at a local optimum instead of reaching the global optimum. While normally a drawback

[9]In the case of ties, a random choice is performed, unless the current label of the node is one of the most frequent in its neighbourhood, in which case the label is simply not changed.

of local search algorithms, this characteristic is essential to clustering: the trivial optimal solution is avoided by the dynamics of the local search algorithm, rather than through formal exclusion.

Despite its efficiency, it was observed that the algorithm just described tends to produce one giant cluster containing the majority of nodes. The presence of this giant component is due to the very topology of social networks; to try to overcome this problem we have tested variants of the label propagation that introduce further constraints. One of the most interesting is the algorithm developed in [6], where the update rule is modified in such a way that the objective function being optimised becomes the *modularity* [57] of the resulting clustering. Unfortunately, modularity is not a good measure in very large graphs as pointed out by several authors (e.g., [33]) due to its resolution limit that makes it hardly usable on large networks.

Another variant, called Absolute Pott Model (APM) [64], introduces a nonlocal discount based on a resolution parameter γ. For a given node x, let $\lambda_1, \ldots, \lambda_k$ be the labels currently appearing on the neighbours of x, k_i be the number of neighbours of x having label λ_i and v_i be the overall number of nodes in the graph with label λ_i; when x is updated, instead of choosing the label λ_i maximizing k_i (as we would do in standard label propagation), by choosing it as to maximise (see Algorithm 3)

$$k_i - \gamma(v_i - k_i).$$

Observe that when $\gamma = 0$ the algorithm degenerates to label propagation; the reason behind the discount term is that when we decide to join a given community, we are increasing its density because of the k_i new edges joining x to existing members of the community, but we are at the same time decreasing it because of $v_i - k_i$ non-existing edges. Indeed, it can be shown that the density of the sparsest community at the end of the algorithm is never below $\gamma/(\gamma + 1)$.

This algorithm demonstrated to be the best candidate for our needs. However it has two major drawbacks. The first is that there are no theoretical results that can be used to determine *a priori* the optimal value of γ (on the contrary, experiments show that such an optimal value is extremely changeable and does not depend on some obvious parameters like the network size or density). The second is that it tends to produce clusters with sizes that follow a heavy-tailed decreasing distribution, yielding both a huge number of clusters and clusters with a huge number of nodes (see Figure 2.1). Thus to obtain good compression performances we have to decide both the order between clusters and the order of the nodes that belong to the same cluster.

Algorithm 3 The APM algorithm. λ is a function that will provide, at the end, the cluster labels. For the sake of readability, we omitted the resolution of ties.

Require: G a graph, γ a density parameter
1: $\pi \leftarrow$ a random permutation of G's nodes
2: for all x: $\lambda(x) \leftarrow x$, $v(x) \leftarrow 1$
3: **while** (some stopping criterion) **do**
4:     **for** $i = 0, 1, \ldots, n-1$ **do**
5:         for every label ℓ, $k_\ell \leftarrow |\lambda^{-1}(\ell) \cap N_G(\pi(i))|$
6:         $\hat{\ell} \leftarrow \mathrm{argmax}_\ell[k_\ell - \gamma(v(\ell) - k_\ell)]$
7:         decrement $v(\lambda(\pi(i)))$
8:         $\lambda(\pi(i)) \leftarrow \hat{\ell}$
9:         increment $v(\lambda(\pi(i)))$
10:     **end for**
11: **end while**

2.6 Layered Label Propagation

In this section we present a new algorithm based on label propagation that yields a compression-friendly ordering.

A run of the APM algorithm (discussed in the previous section) over a given graph and with a given value of the parameter γ produces as output a clustering, that may be represented as a labelling (mapping each node to the label of the cluster it belongs to). An important observation is that, intuitively, there is no notion of optimality for the tuning of γ: every value of this parameter describes a different resolution of the given graph. Values of γ close to 0 highlight a coarse structure with few, big and sparse clusters, while, as γ grows, the clusters get smaller and denser, unveiling a fine-grained structure. Ideally, we would like to find a way to compose clusterings obtained at different resolution levels.[10]

This intuition leads to the definition of *Layered Label Propagation (LLP)*; this algorithm is iterative and produces a sequence of node orderings; at each iteration, the APM algorithm is run with a suitable value of γ and the resulting labelling is then turned into an ordering of the graph that keeps nodes with the same label close to one another; nodes within the *same cluster* are left in the same order they had before.

To determine the relative order among *different clusters*, it is worth observing that

[10] Of course, such a compositional approach could be applied also to other scalable clustering techniques: we have experimented with several alternatives [32], and APM is by far the most interesting candidate.

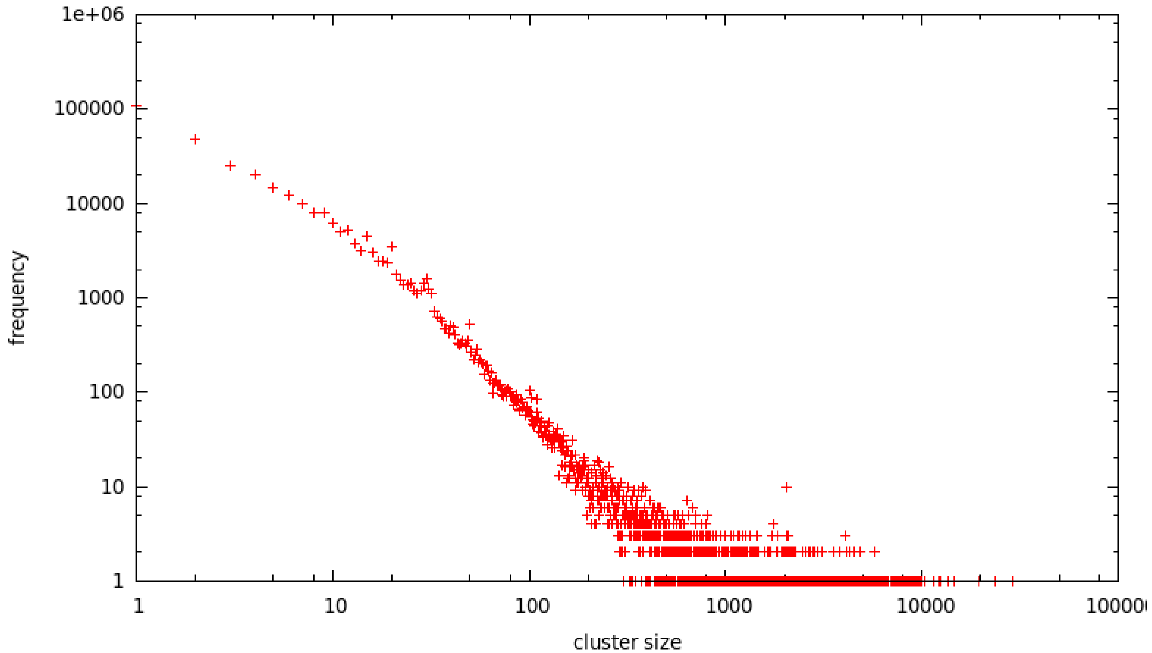

Figure 2.1: An example of the distribution of cluster sizes computed by APM.

the *actual label* produced by the label propagation algorithm suggests a natural choice: since every cluster will be characterised by the initial label of the leader node (the node which flooded that portion of graph; see Algorithm 3), we can sort the clusters according to the order that the leader nodes had.

More formally, let a sequence of non-negative real numbers $\gamma_0, \gamma_1, \gamma_2, \ldots$ and an initial ordering $\pi_0 : V_G \to |V_G|$ of the nodes of G be fixed; we define a sequence of orderings $\pi_1, \pi_2, \ldots : V_G \to |V_G|$ and a sequence of labelling functions $\lambda_0, \lambda_1, \ldots : V_G \to |V_G|$ as follows: λ_k is obtained by running the APM algorithm on the graph G with parameter γ_k; then we let π_{k+1} be the ordering defined by

$$x \leq_{k+1} y \text{ iff } \begin{cases} \pi_k(\lambda_k(x)) < \pi_k(\lambda_k(y)) & \text{or} \\ \lambda_k(x) = \lambda_k(y) \bigwedge \pi_k(x) \leq \pi_k(y). \end{cases}$$

The rationale behind this way of composing the newly obtained clustering with the previous ordering is explained above: elements in the same cluster (i.e., with the same label) are ordered as before; for elements with a *different* label, we use the order that the corresponding labels (i.e., leader nodes) had before.

The output of LLP actually depends on two elements: the initial ordering π_0 and the choice of the parameters γ_k at each iteration.

Regarding the choice of the γ_k's, instead of trying to find at each iteration an optimal value for the parameter we exploit the diverse resolution obtained through different choices of the parameter, thus finding a proper order between clusters that suitably mixes the clusterings obtained at all resolution levels. To obtain this effect, we choose every γ_k uniformly at random in the set[11] $\{0\} \cup \{2^{-i}, i = 0, \ldots, K\}$. Since the APM algorithm is run at every step on the same graph G, it turns out that it is easier (and more efficient) to precompute the labelling function output by the APM algorithm for each γ in the above set, and then to re-use such labellings.

The surprising result is that the final ordering obtained by this mutilresolution strategy is better than the ordering obtained by applying the same strategy with K different clusterings generated with the same value of γ chosen after a grid search for the optimal value (as shown in Table 2.2), and *a fortiori* on the ordering induced by one single clustering generated with the optimal γ. Moreover the final order obtained is essentially independent on the initial permutation π_0 of the graph (as one can see comparing Table 2.6 with Table 2.7).

One may wonder if this iterative strategy can be applied also to improve the performances of other intrinsic orderings. Our experiments rule out this hypothesis. Iterating Gray, lex, or BFS orderings does not produce a significant improvement.

2.7 Parallel Implementation

Layered label propagation lends itself naturally to the *task decomposition* parallel-programming paradigm, which may dramatically improve performances on modern multicore architectures: since the update order is randomised, there is no obstacle in updating several nodes in parallel. Our implementation breaks the set of nodes into a very small number of tasks (in the order of thousands). A large number of threads picks up the first available task and solves it: as a result, we obtain a performance improvement that is linear in the number of cores. In doing this, we are helped by WebGraph's facilities, which allows us to provide each thread with a lightweight copy of the graph that shares the bitstream and associated information with all other threads.

[11] Although in theory γ could be larger than 1, such a choice would be of no practical use on large networks, because it would only yield a complete fragmentation of the graph.

Name	LLP	Fixed LLP	
Amazon	9.12	9.43	(+3%)
DBLP	6.87	7.13	(+3%)
Enron	6.45	6.90	(+6%)
Hollywood	5.17	5.55	(+7%)
LiveJournal	10.95	11.40	(+4%)
Flickr	8.9	9.27	(+4%)
indochina (hosts)	5.57	6.25	(+12%)
uk (hosts)	6.35	6.79	(+6%)
eu	3.88	4.46	(+14%)
in	2.44	2.99	(+22%)
indochina	1.68	1.92	(+14%)
it	2.05	2.59	(+26%)
uk	1.8	2.27	(+26%)

Table 2.2: Comparison between LLP with different values of γ and LLP with the best value of γ only. Values are bits per link.

2.8 Experiments

For our experiments, we considered a number of graphs with various sizes and characteristics; most of them are (directed or undirected) social graphs of some kind, but we also considered some web graphs for comparison (because for web graphs we can rely on the URLs as external source of information). More precisely, we used the following datasets (see also Table 2.3 and 2.4):

- *Hollywood*: One of the most popular *undirected* social graphs, the graph of movie actors: vertices are actors, and two actors are joined by an edge whenever they appeared in a movie together.
- *DBLP*: DBLP[12] is a bibliography service from which an *undirected* scientific collaboration network can be extracted: each vertex of this undirected graph represents a scientist and two vertices are connected if they have worked together on an article.

[12] `http://www.informatik.uni-trier.de/~ley/db/`

- *LiveJournal*: LiveJournal[13] is a virtual community social site started in 1999: nodes are users and there is an arc from x to y if x registered y among his friends (it is not necessary to ask y permission, so the graph is *directed*). We considered the same 2008 snapshot of *LiveJournal* used in [23] for their experiments[14].
- *Amazon*: This dataset describes similarity among books as reported by the Amazon store; more precisely the data was obtained[15] in 2008 using the Amazon E-Commerce Service APIs using `SimilarityLookup` queries.
- *Enron*: This dataset was made public by the Federal Energy Regulatory Commission during its investigations: it is a partially anonymised corpus of e-mail messages exchanged by some Enron employees (mostly part of the senior management). We turned this dataset into a *directed* graph, whose nodes represent people and with an arc from x to y whenever y was the recipient of (at least) a message sent by x.
- *Flickr*: Flickr[16] is an online community where users can share photographs and videos. In Flickr the notion of acquaintance is modelled through *contacts*; we used an *undirected* version of this network, where vertices correspond to users and there is an edge connecting x and y whenever either vertex is recorded as a contact of the other one.
- For comparison, we considered five web graphs of various sizes (ranging from about 800 thousand nodes to more than 650 million nodes), available at the LAW web site `http://law.dsi.unimi.it/`.
- Finally, the *altavista-nd* graph was obtained from the Altavista dataset distributed by Yahoo! within the Webscope program (AltaVista webpage connectivity dataset, version 1.0[17]). With respect to the original dataset, we pruned all dangling nodes ("nd" stands for "no dangling"). The original graph, indeed, contains 53.74% dangling nodes (a preposterous percentage [71]), probably because it also considers the *frontier* of the crawl—the nodes that have been discovered but not visited. We eliminated (one level of) dangling nodes to approximate the set of visited nodes, and also because dangling nodes are of little importance in compression.[18]

[13] `http://www.livejournal.com/`

[14] The dataset was kindly provided by the authors of [23].

[15] `http://www.archive.org/details/amazon_similarity_isbn/`

[16] `http://www.flickr.com/`; we thank Yahoo! for the experimental results on the Flickr graph.

[17] `http://research.yahoo.com/Academic_Relations`

[18] It should be remarked by this graph, albeit widely used in the literature, is not a good dataset. As we already noted, most likely all nodes in the frontier of the crawler (and not only visited nodes) were added to the graph; moreover, the giant component is less than 4% of the whole graph.

Name	Nodes	Edges
Amazon	735 323	5 158 388
DBLP	326 186	1 615 400
Enron	69 244	276 143
Hollywood	1 139 905	113 891 327
LiveJournal	5 363 260	79 023 142
Flickr	526 606	47 097 454

Table 2.3: Social graph description.

Name	Year	Nodes	Edges
eu	2005	862 664	19 235 140
in	2004	1 382 908	16 917 053
indochina	2004	7 414 866	194 109 311
indochina (hosts)	2004	19 123	233 380
it	2004	41 291 594	1 150 725 436
uk (hosts)	2005	587 205	12 825 465
uk	2007	105 896 555	3 738 733 648
altavista-nd	2002	653 912 338	4 226 882 364

Table 2.4: Web graph description.

Each graph was compressed in the BV format using WebGraph [17][19] and we measured the compression performance using the number of bits/link actually occupied by the graph file.

We also compared LLP+BV with the compression obtained using the algorithm proposed by Apostolico and Drovandi [4] at level 8 starting from a randomly permuted graph; the results, shown in Table 2.5, provide evidence that LLP+BV outperforms AD in all cases, and in a significant way on social networks and large web graphs. This is particularly relevant, since the compression algorithm of AD is designed to take full advantage of a specific ordering (the breadth-first search) and is the only known

[19]We adopted the default window size ($W = 7$), disabled intervalisation and put a limit of 3 to the length of the possible reference chains (see [15] for details on the rôle of this parameter). Observe that the latter two settings tend to deteriorate the compression results, but make decompression extremely efficient even when random access is required.

Name	LLP+BV	AD	
Amazon	9.13	12.39	(+36%)
DBLP	6.82	7.47	(+10%)
Enron	6.07	7.74	(+28%)
Hollywood	4.99	7.64	(+53%)
LiveJournal	10.91	14.97	(+37%)
Flickr	8.9	11.19	(+26%)
indochina (hosts)	5.42	6.83	(+26%)
uk (hosts)	6.19	7.85	(+27%)
eu	3.78	4.01	(+6%)
in	2.24	2.39	(+7%)
indochina	1.53	1.70	(+11%)
it	1.91	2.31	(+21%)
uk	1.72	2.32	(+36%)
altavista-nd	5.16	11.04	(+114%)

Table 2.5: Comparison between LLP+BV compression (for this particular table, the full set of compression tecniques available in WebGraph has been used, including intervalisation) and the algorithm proposed by Apostolico and Drovandi (AD) at level 8. Values are bits per link.

coordinate-free alternative we are aware of. In our comparison, contrarily to all other tables, we used the full compression power of the BV format, as our intent is to motivate LLP+BV as a very competitive coordinate-free compression algorithm. We have turned off intervalisation, as our purpose is to study the effect of different permutations on locality and similarity: this explains why the bits per link found in Table 2.5 are smaller than elsewhere.

A comment is needed about the bad performance the Apostolico–Drovandi method on the altavista-nd dataset. Apparently, the size of the dataset is such that scrambling it by a random permutation causes the method to use a bad naming for the nodes, in spite of the initial breadth-first visit. In our previous experiments, the Apostolico–Drovandi method did not show variations of more than 20% in compression due to random permutations, but clearly the issue needs to be investigated more thoroughly.

2.9 Results

Tables 2.6 and 2.7 present the number of bits per link required by our datasets under the different orderings discussed above and produced starting from the natural order and from a random order (the percentages shown in parenthesis give the gain w.r.t. breadth-first search ordering). Here are some observations that the experimental results suggest:

- LLP provides always the best compression, with an average gain of 25% with respect to BFS, and largely outperforms both simple Gray [15] and shingle orderings [23]. Some simple experiments not reported here shows that the same happen for transposed graphs: for instance, `uk` is compressed at 1.06 bits per link. This makes LLP+BV encoding by far the best compressed data structure available today.

- LLP is extremely robust with respect to the initial ordering of nodes and its combination with BV provides actually a coordinate-free compressed data structure. Other orderings (in particular, Gray and shingle) are much more sensitive to the initial numbering, especially on web graphs. We urge researchers in this field *to always generate permutations starting from a randomised copy of the graph*, as "useful" ordering information in the original dataset can percolate as an artifact in the final results.

- As already remarked elsewhere [23], social networks seem to be harder to compress than web graphs: this fact would suggest that there should be some yet unexplained topological difference between the two kinds of graphs that accounts for the different compression ratio.

Despite the great improvement in terms of compression results our technique remains highly scalable. All experiments are performed on a Linux server equipped with Intel Xeon X5660 CPUs (2.80 GHz, 12 MB cache size) for overall 24 cores and 128 GB of RAM; the server cost about 8 900 EUR in 2010. Our Java implementation of LLP sports a linear scaling in the number of arcs with an average speed of $\approx$ 80 000 000 arcs/s per iteration. The overall time cost of the algorithm depends on the number γ's and on the stopping criterion. With our typical setting the overall speed of the algorithm is $\approx$ 800 000 arcs/s.

The algorithm is also very memory efficient (it uses $3n$ integers plus the space required by the graph[20]) and it is easy to distribute, making it a good candidate for huge

[20]It is possible in principle to avoid keeping the graph in main memory, but the cost becomes $O(n \log n)$.

networks. Indeed, most of the time is spent on sampling values of γ to produce base clusterings,[21] and this operation can be performed for each γ in a fully parallel way. Applying LLP to a web graph with 1 billion nodes and 50 billions arcs would require few hours in this setting.

For comparison, we also tried to compress our dataset using the alternative versions of LLP described in Section 2.5: in particular, we considered APM (with the optimal choice of γ) and the combination APM+Gray (that sorts each APM cluster using Gray). Besides the number of bits per link, we also analysed two measures that quantify two different structural properties:

- the *average gap cost* (i.e., the average of the base-2 logarithms of the gaps between the successors of a node: this is an approximate measure of the number of bits required to write the gaps using a universal variable-length code); this measure is intended to account for locality: the average gap cost is small if the ordering tends to keep well-connected nodes close to one another;[22]

- the *percentage of copied arcs* (i.e., the number of arcs that are not written explicitly but rather obtained by reference from a previous successor list); this is intended to account for similarity: this percentage is small if the ordering tends to keep nodes with similar successor lists close to one another.

The results obtained are presented in Table 2.8. In most cases APM copies a smaller percentage of arcs than APM+Gray, because Gray precisely aims at optimising similarity rather than locality; this phenomenon is less pronounced on web graphs, where anyway the overall number of copied arcs is larger; looking at the average gap cost, all clustering methods turn out to do a better job than Gray in improving locality (data not shown in the table). LLP usually copies less arcs than APM+Gray, but the difference is often negligible and definitely balanced by the gain in locality.

We would like to point out that, at least when using the best compression currently available (LLP+BV), the average gap cost is definitely more correlated with compression rates than the *average distance cost*, that is, the average of the logarithms of the (absolute) difference between the source and target of each arc (see Figure 2.2). Indeed, the correlation coefficient is 0.9681 between bits per link and average gap cost and 0.1742 between bits per link and average distance cost. In [23] the problems MLogA and MLogGapA consist exactly in minimising the average distance and the average

[21]The combination of clusterings is extremely fast, as it is linear in the number of nodes, rather than in the number of arcs, and has little impact on the overall run time.

[22]We remark that the average gap cost is essentially an amortised version of the standard *gap measure* used in the context of data-aware compressed data structures [36].

gap cost, respectively: that authors claim that both problems capture the essence of a good ordering, but our extensive experimentation suggests otherwise.

As a final remark, it is worth noticing that similarity and locality have a different impact in social networks than in web graphs: in web graphs the percentage of copied arcs is much larger (a clue of the presence of a better-defined structure) and in fact it completely determines the number of bits per link, whilst in social networks the compression ratio is entirely established by the gain of locality (measured, as usual, by the average gap cost).

2.10 Conclusions

We have presented highly scalable techniques that improve compressed data structures for representing web graphs and social networks significantly beyond the current state-of-art. More importantly, we have shown that coordinate-free methods can outperform state-of-art extrinsic techniques on a large range of networks. The clustering techniques we have devised are scalable to billions of nodes, as they just require few linear passes over the graphs involved. In some cases (e.g., the `uk` dataset) we bring down the cost of a link to almost 1.8 bits. We remark again that our improvements are measured w.r.t. the BFS baseline, which is itself often an improvement when compared to the existing literature.

Finally, we leave for future work a full investigation of the compression ratio that can be obtained when fast access is not required. For instance, `uk` compressed by LLP+BV at maximum compression requires only 1.21 bits per link—better, for instance, than the Apostolico–Drovandi method with maximum compression (1.44). Some partial experimental data suggests that we would obtain by far the highest compression ratio currently available.

The experiments that we report required several thousands of hours of computation: we plan to make available the results both under the form of WebGraph *property files* (which contain a wealth of statistical data) and under the form of comprehensive graphical representations. This information will be most useful to researchers studying the structure of social networks. A more thorough attempt to explain the reasons behind the greater compressibility of web graphs is essential to try to enhance social-network compression further.

Another interesting issue is the impact of increased of locality on cache usage. In principle, assuming that nodes with nearby identifiers have their successors stored in nearby memory (as it happens in WebGraph) increasing locality could also speed up all graph traversals and algorithms based on them.

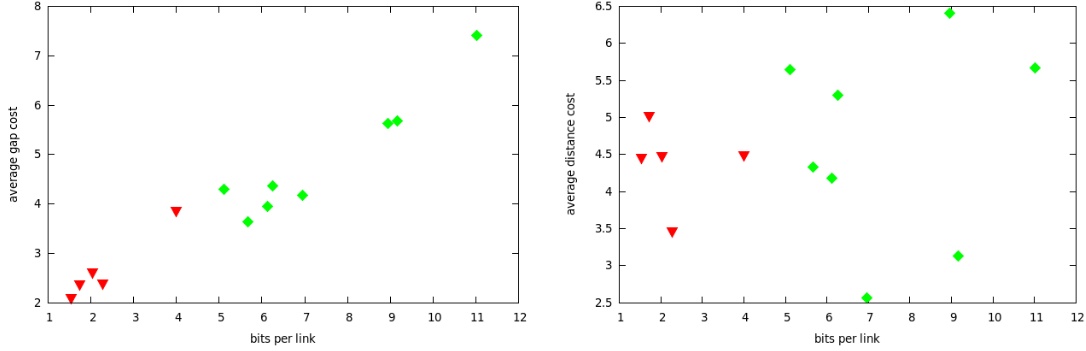

Figure 2.2: Bits per link against average gap (left) and distance (right) cost. ▽ points indicates web graphs while ◇ points indicates social graph.

Name	LLP		BFS	Shingle		Gray		Natural		Random	
Amazon	9.12	(-30%)	13.01	14.36	(+10%)	13.11	(+0%)	16.92	(+30%)	23.62	(+81%)
DBLP	6.87	(-24%)	8.98	11.39	(+26%)	8.50	(-6%)	11.36	(+26%)	22.07	(+145%)
Enron	6.45	(-26%)	8.68	9.80	(+12%)	9.78	(+12%)	13.43	(+54%)	14.02	(+61%)
Hollywood	5.17	(-33%)	7.64	6.68	(-13%)	6.35	(-17%)	15.20	(+98%)	16.23	(+112%)
LiveJournal	10.95	(-28%)	15.05	15.66	(+4%)	14.19	(-6%)	14.35	(-5%)	23.50	(+56%)
Flickr	8.90	(-19%)	10.92	10.22	(-7%)	10.82	(-1%)	13.87	(+27%)	14.49	(+32%)
indochina (hosts)	5.57	(-15%)	6.55	7.15	(+9%)	7.49	(+14%)	9.26	(+41%)	10.59	(+61%)
uk (hosts)	6.35	(-17%)	7.59	8.07	(+6%)	8.13	(+7%)	10.81	(+42%)	15.58	(+105%)
eu	3.88	(-21%)	4.87	6.09	(+25%)	4.98	(+2%)	5.24	(+7%)	19.89	(+308%)
in	2.44	(-26%)	3.29	4.19	(+27%)	2.90	(-12%)	2.99	(-10%)	21.15	(+542%)
indochina	1.68	(-24%)	2.21	2.91	(+31%)	2.12	(-5%)	2.19	(-1%)	21.46	(+871%)
it	2.05	(-26%)	2.76	3.61	(+30%)	2.67	(-4%)	2.83	(+2%)	26.40	(+856%)
uk	1.80	(-26%)	2.43	3.26	(+34%)	2.47	(+1%)	2.75	(+13%)	27.55	(+1033%)
altavista-nd	5.25	(-10%)	5.78	8.12	(+40%)	6.40	(+10%)	8.37	(+44%)	34.76	(+501%)

Table 2.6: Compression results starting from natural order (percentages are relative to BFS). Values are bits per link.

Name	LLP		BFS	Shingle		Gray		Natural		Random	
Amazon	9.16	(-30%)	12.96	14.43	(+11%)	14.60	(+12%)	16.92	(+30%)	23.62	(+82%)
DBLP	6.88	(-23%)	8.91	11.42	(+28%)	11.50	(+29%)	11.36	(+27%)	22.07	(+147%)
Enron	6.51	(-24%)	8.54	9.87	(+15%)	9.94	(+16%)	13.43	(+57%)	14.02	(+64%)
Hollywood	5.14	(-35%)	7.81	6.72	(-14%)	6.40	(-19%)	15.20	(+94%)	16.23	(+107%)
LiveJournal	10.90	(-28%)	15.1	15.77	(+4%)	15.73	(+4%)	14.35	(-5%)	23.50	(+55%)
Flickr	8.89	(-22%)	11.26	10.22	(-10%)	10.23	(-10%)	13.87	(+23%)	14.49	(+28%)
indochina (hosts)	5.53	(-17%)	6.63	7.16	(+7%)	7.49	(+12%)	9.26	(+39%)	10.59	(+59%)
uk (hosts)	6.26	(-18%)	7.62	8.12	(+6%)	8.13	(+6%)	10.81	(+41%)	15.58	(+104%)
eu	3.90	(-21%)	4.93	6.86	(+39%)	6.27	(+27%)	5.24	(+6%)	19.89	(+303%)
in	2.46	(-30%)	3.51	4.79	(+36%)	4.40	(+25%)	2.99	(-15%)	21.15	(+502%)
indochina	1.71	(-26%)	2.31	3.59	(+55%)	3.09	(+33%)	2.19	(-6%)	21.46	(+829%)
it	2.10	(-28%)	2.89	4.39	(+51%)	3.79	(+31%)	2.83	(-3%)	26.40	(+813%)
uk	1.91	(-33%)	2.84	4.09	(+44%)	3.36	(+18%)	2.75	(-4%)	27.55	(+870%)
altavista-nd	5.22	(-11%)	5.85	8.12	(+38%)	7.52	(+28%)	8.37	(+43%)	34.76	(+494%)

Table 2.7: Compression results starting from a random order (percentages are relative to BFS). Values are bits per link.

Name	Bits/link			Copied arcs			Avg. gap cost		
	LLP	APM + Gray	APM	LLP	APM + Gray	APM	LLP	APM + Gray	APM
Amazon	9.14	10.45	10.67	31.22	32.32	28.87	5.64	6.87	6.97
DBLP	6.87	8.38	8.48	36.55	37.66	36.42	4.04	5.73	5.80
Enron	6.48	7.15	7.97	24.07	25.45	10.86	3.92	4.58	4.76
Hollywood	5.13	5.38	6.10	44.22	42.49	38.68	4.14	4.38	4.92
LiveJournal	10.90	12.00	12.79	23.57	23.66	17.48	7.34	8.29	8.69
Flickr	8.89	9.22	9.69	13.65	11.77	8.88	5.59	5.84	6.17
eu	3.90	4.86	5.76	65.84	66.33	59.57	3.62	5.16	5.78
in	2.46	3.11	4.05	72.45	73.04	65.11	2.31	4.02	4.60
indochina	1.71	2.17	3.00	80.36	80.78	75.53	2.06	3.59	4.09
indochina (hosts)	5.54	6.04	6.16	33.51	34.69	26.94	3.46	4.02	3.90
it	2.10	2.56	3.94	77.18	79.76	69.53	2.43	4.36	5.16
uk (hosts)	6.26	6.68	6.90	33.94	37.34	30.48	4.24	4.76	4.79
uk	1.91	2.39	3.73	79.16	81.92	71.73	2.31	4.71	5.35

Table 2.8: Comparison between LLP and the ordering produced by other clustering algorithms (APM and the combination APM+Gray) when compressing with the BV algorithm. We consider the value of γ that minimises the number of bits/link.

Chapter 3

Approximating the neighbourhood function for large graphs

A small-world network [25] is a graph where the average distance between nodes is logarithmic in the size of the network, whereas the clustering coefficient is large (that is, neighbourhoods tend to be denser) than in a random Erdős-Rényi graph with the same size and average distance.[1] Here, and in the following, by "distance" we mean the length of the shortest path between two nodes. The fact that social networks (either electronically mediated or not) exhibit the small-world property is known at least since Milgram's famous experiment [54] [2], and is arguably the most popular of all features of complex networks.

In this chaper we turn to the study of the *neighbourhood function*. The neighbourhood function $N_G(t)$ of a graph G gives, for each $t \in \mathbf{N}$, the number of pairs of nodes $\langle x, y \rangle$ such that y is reachable from x in less than t hops. The neighbourhood function provides a wealth of information about the graph [61] (e.g., it easily allows one to compute its diameter), but it is very expensive to compute it exactly. Recently, the ANF algorithm [61] (approximate neighbourhood function) has been proposed with the purpose of approximating $N_G(t)$ on large graphs. We describe a breakthrough improve-

[1] The reader might find this definition a bit vague, and some variants are often spotted in the literature: this is a general problem, also highlighted recently in [49].

[2] It should be noticed that Milgram's experiment tried to prove two properties at the same time. First, that the average distance between individuals is much smaller than expected; second, that individuals are able to exploit such a feature to route messages along short paths, albeit they only possess local information about the network they live in. This second property is, in a sense, much more interesting than the former, but also much more difficult to describe and study, because it has to do with some information that the nodes possess about the environment they inhabit.

ment over ANF in terms of speed and scalability. Our algorithm, called HyperANF, uses the new HyperLogLog counters [30] and combines them efficiently through *broadword programming* [45]; our implementation uses *task decomposition* to exploit multi-core parallelism. With HyperANF, for the first time we can compute in a few hours the neighbourhood function of graphs with billions of nodes with a small error and good confidence using a standard workstation.

Then, we turn to the study of the distribution of *distances* between reachable nodes (that can be efficiently approximated by means of HyperANF), and discover the surprising fact that its *index of dispersion* provides a clear-cut characterisation of proper social networks vs. web graphs. We thus propose the *spid* (Shortest-Paths Index of Dispersion) of a graph as a new, informative statistics that is able to discriminate between the above two types of graphs. We believe this is the first proposal of a significant new non-local structural index for complex networks whose computation is highly scalable.

3.1 Introduction

The *neighbourhood function* $N_G(t)$ of a graph returns for each $t \in \mathbf{N}$ the number of pairs of nodes $\langle x, y \rangle$ such that y is reachable from x in less than t steps. It provides data about how fast the "average ball" around each node expands. From the neighbourhood function, several interesting features of a graph can be estimated, such as, for example, the *effective diameter*, a measure of the "typical" distance between nodes.

Palmer, Gibbons and Faloutsos [61] proposed an algorithm to *approximate* the neighbourhood function (see their paper for a review of previous attempts at approximate evaluation); the authors distribute an associated tool, `snap`, which can approximate the neighbourhood function of medium-sized graphs. The algorithm keeps track of the number of nodes reachable from each node using *Flajolet–Martin counters*, a kind of *sketch* that makes it possible to compute the number of distinct elements of a stream in very little space. A key observation was that counters associated to different streams can be quickly combined into a single counter associated to the concatenation of the original streams.

In this chapter, we describe HyperANF—a breakthrough improvement over ANF in terms of speed and scalability. HyperANF uses the new HyperLogLog counters [30], and combines them efficiently by means of *broadword programming* [45]. Each counter is made by a number of *registers*, and the number of registers depends only on the required precision. The size of each register is *doubly logarithmic* in the number of nodes of the graph, so HyperANF, for a fixed precision, scales almost linearly in memory (i.e., $O(n \log \log n)$). By contrast, ANF memory requirement is $O(n \log n)$.

Using HyperANF, for the first time we can compute in a few hours the neighbourhood function of graphs with more than one billion nodes with a small error and good confidence using a standard workstation with 128 GB of RAM. Our algorithms are implement in a tool distributed as free software within the WebGraph framework.[3]

Armed with our tool, we study several datasets, spanning from small social networks to very large web graphs. We isolate a statistically defined feature, the *index of dispersion of the distance distribution*, and show that it is able to tell "proper" social networks from web graphs in a natural way.

3.2 Related work

HyperANF is an evolution of ANF [61], which is implemented by the tool `snap`. We will give some timing comparison with `snap`, but we can only do it for relatively small networks, as the large memory footprint of `snap` precludes application to large graphs.

Recently, a MapReduce-based distributed implementation of ANF called HADI [41] has been presented. HADI runs on one of the fifty largest supercomputers—the Hadoop cluster M45. The only published data about HADI's performance is the computation of the neighbourhood function of a Kronecker graph with 2 billion links, which required half an hour using 90 machines. HyperANF can compute the same function in less than fifteen minutes on a laptop.

The rather complete survey of related literature in [41] shows that essentially no data mining tool was able before ANF to approximate the neighbourhood function of very large graphs reliably. A remarkable exception is Cohen's work [24], which provides strong theoretical guarantees but experimentally turns out to be not as scalable as the ANF approach; it is worth noting, though, that one of the proposed applications of [24] (*On-line estimation of weights of growing sets*) is structurally identical to ANF.

All other results published before ANF relied on a small number of breadth-first visits on uniformly sampled nodes—a process that has no provable statistical accuracy or precision. Thus, in the rest of the chapter we will compare experimental data with `snap` and with the published data about HADI.

3.3 HyperANF

In this section, we present the HyperANF algorithm for computing an approximation of the neighbourhood function of a graph; we start by recalling from [30] the notion of

[3] See [17]. `http://webgraph.dsi.unimi.it/`.

HyperLogLog counter upon which our algorithm relies. We then describe the algorithm, discuss how it can be implemented to be run quickly using broadword programming and task decomposition, and give results about its memory requirements and precision.

3.3.1 HyperLogLog counters

HyperLogLog counters, as described in [30] (which is based on [28]), are used to count approximately the number of distinct elements in a stream. For our purposes, we need to recall briefly their behaviour. Essentially, these probabilistic counters are a sort of *approximate set representation* to which, however, we are only allowed to pose questions about the (approximate) size of the set.[4]

Let $\mathscr{D}$ be a fixed domain and $h : \mathscr{D} \to 2^\infty$ be a hash function mapping each element of $\mathscr{D}$ into an infinite binary sequence. The function is fixed with the only assumption that "bits of hashed values are assumed to be independent and to have each probability $\frac{1}{2}$ of occurring" [30].

For a given $x \in 2^\infty$, let $h_t(x)$ denote the sequence made by the leftmost t bits of $h(x)$, and $h^t(x)$ be the sequence of remaining bits of x; each point in the codomain of h_t is identified with its corresponding integer value in the range $\{ 0, 1, \ldots, 2^t - 1 \}$. Moreover, given a binary sequence w, we let $\rho^+(w)$ be the number of leading zeroes in w plus one[5] (e.g., $\rho^+(00101) = 3$). Unless otherwise specified, all logarithms are in base 2.

The value E printed by Algorithm 4 is [30][Theorem 1] an asymptotically almost unbiased estimator for the number n of distinct elements in the stream; for $n \to \infty$, the *relative standard deviation* (that is, the ratio between the standard deviation of E and n) is at most $\beta_m/\sqrt{m} \leq 1.06/\sqrt{m}$, where β_m is a suitable constant (given in [30]). Moreover [28] even if the size of the registers (and of the hash function) used by the algorithm is unbounded, one can limit it to $\log\log(n/m) + \omega(n)$ bits obtaining almost certainly the same output ($\omega(n)$ is a function going to infinity arbitrarily slowly); overall, the algorithm requires $(1 + o(1)) \cdot m \log\log(n/m)$ bits of space (this is the reason why these counters are called HyperLogLog). Here and in the followings we tacitly assume that $m \geq 64$ and that registers are made of $\lceil \log\log n \rceil$ bits.

---

[4] We remark that in principle $O(\log n)$ bits are necessary to estimate the number of unique elements in a stream [2]. HyperLogLog is a practical counter that starts from the assumption that a hash function can be used to turn a stream into an *idealised multiset* (see [30]).

[5] We remark that in the original HyperLogLog papers ρ is used to denote ρ^+, but ρ is a somewhat standard notation for the ruler function [45].

Algorithm 4 The Hyperloglog counter as described in [30]: it allows one to count (approximately) the number of distinct elements in a stream. α_m is a constant whose value depends on m and is provided in [30]. Some technical details have been simplified.

```
0   h : 𝒟 → 2^∞, a hash function from the domain of items
1   M[−] the counter, an array of m = 2^b registers
2      (indexed from 0) and set to −∞
3
4   function add(M: counter, x: item)
5   begin
6      i ← h_b(x);
7      M[i] ← max{ M[i], ρ⁺( h^b(x) ) }
8   end; // function add
9
10  function size(M: counter)
11  begin
12     Z ← ( Σ_{j=0}^{m−1} 2^{−M[j]} )^{−1};
13     return E = α_m m² Z
14  end; // function size
15
16  foreach item x seen in the stream begin
17     add(M,x)
18  end;
19  print size(M)
```

3.3.2 The HyperANF algorithm

The approximate neighbourhood function algorithm described in [61] is based on the observation that $B(x, r)$, the ball of radius r around node x, satisfies

$$B(x, r) = \bigcup_{x \to y} B(y, r-1).$$

Since $B(x, 0) = \{ x \}$, we can compute each $B(x, r)$ incrementally using sequential scans of the graph (i.e., scans in which we go in turn through the successor list of each node). The obvious problem is that during the scan we need to access randomly the sets $B(x, r-1)$ (the sets $B(x, r)$ can be just saved on disk on a *update file* and reloaded later). Here probabilistic counters come into play; to be able to use them, though, we need to endow counters with a primitive for the union. Union can be implemented provided that the counter associated to the stream of data AB can be computed from the counters associated to A and B; in the case of HyperLogLog counters, this is easily seen to correspond to maximising the two counters, register by register.

The observations above result in Algorithm 5: the algorithm keeps one HyperLogLog counter for each node; at the t-th iteration of the main loop, the counter $c[v]$ is in the same state as if it would have been fed with $B(v, t)$, and so its expected value is $|B(v, t)|$. As a result, the sum of all $c[v]$'s is an (almost) unbiased estimator of $N_G(t)$ (for a precise statement, see Theorem 1).

We remark that the only sound way of running HyperANF (or ANF) is to wait for all counters to stabilise (e.g., the last iteration must leave all counters unchanged). As we will see, any alternative termination condition may lead to arbitrarily large mistakes on pathological graphs.[6]

3.3.3 HyperANF at hyper speed

Up to now, HyperANF has been described just as ANF with HyperLogLog counters. The effect of this change is an exponential reduction in the memory footprint and, consequently, in memory access time. We now describe the the algorithmic and engineering ideas that made HyperANF much faster, actually so fast that it is possible to run it up to stabilisation.

Union via broadword programming. Given two HyperLogLog counters that have been set by streams A and B, the counter associated to the stream AB can be build by

[6] We remark that `snap` uses a threshold over the relative increment in the number of reachable pairs as a termination condition, but this trick makes the tail of the function unreliable.

Algorithm 5 The basic HyperANF algorithm in pseudocode. The algorithm uses, for each node $i \in n$, an initially empty HyperLogLog counter c_i. The function union$(-, -)$ maximises two counters register by register.

```
0   c[−], an array of n HyperLogLog counters
1
2   function union(M: counter, N: counter)
3     foreach i < m begin
4       M[i] ← max(M[i], N[i])
5     end
6   end; // function union
7
8   foreach v ∈ n begin
9     add v to c[v]
10  end;
11  t ← 0;
12  do begin
13    s ← ∑_v size(c[v]);
14    Print s (the neighbourhood function N_G(t))
15    foreach v ∈ n begin
16      m ← c[v];
17      foreach v → w begin
18        m ← union(c[w], m)
19      end;
20      write ⟨v, m⟩ to disk
21    end;
22    Read the pairs ⟨v, m⟩ and update the array c[−]
23    t ← t + 1
24  until no counter changes its value.
```

maximising in parallel the registers of each counter. That is, the register i of the new counter is given by the maximum between the i-th register of the first counter and the i-th register of the second counter.

Each time we scan a successor list, we need to maximise a large number of registers and store the resulting counter. The immediate way of obtaining this result requires extracting the value of each register, maximise it with the other corresponding registers, and writing down the result in a temporary counter. This process is extremely slow, as registers are packed in 64-bit memory words. In the case of Flajolet–Martin counters, the problem is easily solved by computing the logical OR of the words containing the registers. In our case, we resort to *broadword programming* techniques. If the machine word is w, we assume that at least w registers are allocated to each counter, so each set of registers is word-aligned.

Let $\gg$ and $\ll$ denote right and left (zero-filled) shifting, &, | and $\oplus$ denote bit-by-bit not, and, or, and xor; $\overline{x}$ denotes the bit-by-bit complement of x. We use L_k to denote the constant whose ones are in position 0, k, $2k$, ... that is, the constant with the *lowest* bit of each k-bit subword set (e.g, $L_8 = 0x0101010101010101$). We use H_k to denote $L_k \ll k-1$, that is, the constant with the *highest* bit of each k-bit subword set (e.g, $H_8 = 0x8080808080808080$).

It is known (see [45], or [72] for an elementary proof), that the following expression

$$x <_k^u y := \Big(\big(((x \mid H_k) - (y \,\&\, \overline{H_k})) \mid x \oplus y\big) \oplus (x \mid \overline{y})\Big) \,\&\, H_k.$$

performs a parallel unsigned comparison k-by-k-bit-wise. At the end of the computation, the highest bit of each block of k bits will be set iff the corresponding comparison is true (i.e., the value of the block in x is strictly smaller than the value of the block in y).

Once we have computed $x <_k^u$, we generate a mask that is made entirely of 1s, or of 0s, for each k-bit block, depending on whether we should select the value of x or y for that block:

$$m = \Bigg(\Big(\big((x <_k^u y) \gg k-1 \mid H_k\big) - L_k\Big) \mid H_k\Bigg) \oplus (x <_k^u y)$$

This formula works by moving the high bit denoting the result of the comparison to the least significant bit (of each k-bit block). Then, we or with H_k and subtract 1 from each block, obtaining either a mask with just the high bit set (if we were starting from 1) or a mask with all bits sets except for the high bit (if we were starting from 0). The last two operation fix those values so that they become $00\cdots0$ or $11\cdots1$. The result of the maximisation process is now just $x \,\&\, m \mid y \,\&\, \overline{m}$.

This discussion assumed that the set of registers of a counter is stored in a single machine word. In a realistic setting, the registers are spread among several consecutive words, and we use multiple precision subtractions and shifts to apply the expressions above on a sequence of words. All other (logical) operations have just to be applied to each word in sequence.

All in all, by using the techniques above we can improve the speed of maximisation by a factor of w/k, which in our case is about 13 (for graphs of up to 2^{32} nodes). This actually results in a sixfold speed improvement of the overall application in typical cases (e.g., web graphs and $b = 8$), as about 90% of the computation time is spent in maximisation.

Parallelisation via task decomposition. Although HyperANF is written as a sequential algorithm, the outer loop lends itself to be executed in parallel, which can be extremely fruitful on a modern multicore architecture; in particular, we approach this idea using *task decomposition*. We divide the iteration on the whole set of nodes into a set of small tasks (in the order of the thousands), where each task consists in iterating on a contiguous segment of nodes. A pool of threads picks up the first available task and solves it: as a result, we obtain a performance improvement that is linear in the number of cores. Threads can be designed to be extremely agile, helped by WebGraph's facilities which allow us to provide each thread with a lightweight copy of the graph that shares the bitstream and associated information with all other threads.

Tracking modified counters. It is an easy observation that a counter c that does not change its value is not useful for the next step of the computation: all counters using c during their update would not change their value when maximising with c (and we do not even need to write c on disk). We thus keep track of modified counters and skip altogether the maximisation step with unmodified ones. Since, as we already remarked, 90% of computation time is spent in maximisation, this approach leads to a large speedup after the first phases of the computation, when most counters are stabilised.

For the same reason, we keep track of the harmonic partial sums of small blocks (e.g., 64) of counters. The amount of memory required is negligible, but if no counter in the block has been modified, we can avoid a costly computation.

Systolic computation. HyperANF can be run in *systolic* mode. In this case, we use also the transposed graph: whenever a counter changes, it *signals* back to its predecessors that at the next round they could change their values. Now, at each iteration nodes that have not been signalled are entirely skipped during the computation. Systolic computations are fundamental to get high-precision runs, as they reduce the cost of an iteration to scanning only the arcs of the graph that are actually moving information around. We switch to systolic computation when less than one quarter of the counters

change their values.

3.3.4 Correctness, errors and memory usage

Very little has been published about the statistical behaviour of ANF. The statistical properties of approximate counters are well known, but the values of such counters for each node are *highly dependent*, and adding them in a large amount can in principle lead to an arbitrarily large variance. Thus, making precise statistical statements about the outcome of a computation of ANF or HyperANF requires some care. The discussion in the following sections is based on HyperANF, but its results can be applied *mutatis mutandis* to ANF as well.

Consider the output $\hat{N}_G(t)$ of algorithm 5 at a fixed iteration t. We can see it as a random variable

$$\hat{N}_G(t) = \sum_{i \in n} X_{i,t}$$

where[7] each $X_{i,t}$ is the HyperLogLog counter that counts nodes reached by node i in t steps; what we want to prove in this section is a bound on the relative standard deviation of $\hat{N}_G(t)$ (such a proof, albeit not difficult, is not provided in the papers about ANF). First observe that [30], for a fixed a number of registers m per counter, the standard deviation of $X_{i,t}$ satisfies

$$\frac{\sqrt{\mathrm{Var}[X_{i,t}]}}{|B(i,t)|} \leq \eta_m,$$

where η_m is the guaranteed relative standard deviation of a HyperLogLog counter. Using the subadditivity of standard deviation (i.e., if A and B have finite variance, $\sqrt{\mathrm{Var}[A+B]} \leq \sqrt{\mathrm{Var}[A]} + \sqrt{\mathrm{Var}[B]}$), we prove the following

Theorem 1. *The output $\hat{N}_G(t)$ of Algorithm 5 at the t-th iteration is an asymptotically almost unbiased estimator[8] of $N_G(t)$, that is*

$$\frac{E[\hat{N}_G(t)]}{N_G(t)} = 1 + \delta_1(n) + o(1) \text{ for } n \to \infty,$$

where δ_1 is the same as in [30][Theorem 1] (and $|\delta_1(x)| < 5 \cdot 10^{-5}$ as soon as $m \geq 16$).

[7] we use von Neumann's notation $n = \{0, 1, \dots, n-1\}$, so $i \in n$ means that $0 \leq i < n$.

[8] From now on, for the sake of readability we shall ignore the negligible bias on $\hat{N}_G(t)$ as an estimator for $N_G(t)$: the other estimators that will appear later on will be qualified as "(almost) unbiased", where "almost" refers precisely to the above mentioned negligible bias.

Moreover, $\hat{N}_G(t)$ has the same relative standard deviation of the X_i's, that is

$$\frac{\sqrt{\mathrm{Var}[\hat{N}_G(t)]}}{N_G(t)} \le \eta_m.$$

Proof. We have that $E[\hat{N}_G(t)] = E\left[\sum_{i\in n} X_{i,t}\right]$. By Theorem 1 of [30], $E[X_{i,t}] = |B(i,t)|\,(1+\delta_1(n)+o(1))$, hence the first statement. For the second result, we have:

$$\frac{\sqrt{\mathrm{Var}[\hat{N}_G(t)]}}{N_G(t)} \le \frac{\sum_{i\in n}\sqrt{\mathrm{Var}[X_i]}}{N_G(t)} \le \frac{\eta_m \sum_{i\in n}|B(i,t)|}{N_G(t)} = \eta_m.$$

□

Since, as we recalled in Section 3.3.1, the relative standard deviation η_m satisfies $\eta_m \le 1.06/\sqrt{m}$, to get a specific value η it is sufficient to choose $m \approx 1.12/\eta^2$; this assumption yields an overall space requirement of about

$$\frac{1.12}{\eta^2} n \log\log n \qquad \text{bits}$$

(here, we used the obvious upper bound $|B(i,t)| \le n$). For instance, to obtain a relative standard deviation of 9.37% (in every iteration) on a graph of one billion nodes one needs 74.5 GB of main memory for the registers (for a comparison, `snap` would require 550 GB). Note that since we write to disk the new values of the registers, this is actually the only significant memory requirement (the graph can be kept on disk and mapped in memory, as it is scanned almost sequentially).

Applying Chebyshev's inequality, we obtain the following:

Corollary 1. *For every ε,*

$$\Pr\left[\frac{\hat{N}_G(t)}{N_G(t)} \in (1-\varepsilon, 1+\varepsilon)\right] \ge 1 - \frac{\eta_m^2}{\varepsilon^2}.$$

In [30] it is argued that the HyperLogLog error is approximately Gaussian; the counters, however, are *not* statistically independent and it is an empirically established fact that the overall error does not appear to be normally distributed. Nonetheless, for every fixed t, the random variable $\hat{N}_G(t)$ seems to be unimodal (for example, the average

p-value of the Dip unimodality test [37] for the `cnr-2000` dataset is 0.011), so we can apply the Vysochanskiĭ-Petunin inequality [74], obtaining the bound

$$\Pr\left[\frac{\hat{N}_G(t)}{N_G(t)} \in (1-\varepsilon, 1+\varepsilon)\right] \geq 1 - \frac{4\eta_m^2}{9\varepsilon^2}.$$

In the rest of this chapter, to state clearly our theorems we will always assume error ε with confidence $1-\delta$. It is useful, as a practical reminder, to note that because of the above inequality for each point of the neighbourhood function we can assume a relative error of $k\eta_m$ with confidence $1 - 4/(9k^2)$ (e.g., $2\eta_m$ with 90% confidence, or $3\eta_m$ with 95% confidence).

As an empirical counterpart to the previous results, we considered a relatively small graph of about 325 000 nodes (`cnr-2000`, see Section 3.6 for a full description) for which we can compute the exact neighbourhood function $N_G(-)$; we ran HyperANF 500 times with $m = 256$. At least 96% of the samples (for all t) has a relative error smaller than twice the theoretical relative standard deviation 6.62%. The percentage jumps up to 100% for three times the relative standard deviation, showing that the distribution of the values behaves better than what the theory would guarantee.

3.4 Deriving useful data

As advocated in [61], being able to estimate the neighbourhood function on real-world networks has several interesting applications. Unfortunately, all published results we are aware of lack statistical satellite data (such as confidence intervals, or distribution of the computed values) that make it possible to compare results from different research groups. Thus, in this section we try to discuss in detail how to derive useful data from an approximation of the neighbourhood function.

The distance cdf. We start from the apparently easy task of computing the *cumulative distribution function of distances* of the graph G (in short, *distance cdf*), which is the function $H_G(t)$ that gives the fraction of reachable pairs at distance at most t, that is,

$$H_G(t) = \frac{N_G(t)}{\max_t N_G(t)}.$$

In other words, given an exact computation of the neighbourhood function, the distance cdf can be easily obtained by dividing all values by the largest one. Being able to estimate $N_G(t)$ allows one to produce a reliable approximation of the distance cdf:

Theorem 2. *Assume $N_G(t)$ is known for each t with error ε and confidence $1-\delta$, that is*

$$\Pr\left[\frac{\hat{N}_G(t)}{N_G(t)} \in (1-\varepsilon, 1+\varepsilon)\right] \geq 1-\delta.$$

Let $\hat{H}_G(t) = \hat{N}_G(t)/\max_t \hat{N}_G(t)$. Then $\hat{H}_G(t)$ is an (almost) unbiased estimator for $H_G(t)$; moreover, for a fixed sequence $t_0, t_1, \ldots, t_{k-1}$, for every ε and all $0 \leq i < k$ we have that $\hat{H}_G(t_k)$ is known with error 2ε and confidence $1-(k+1)\delta$, that is,

$$\Pr\left[\bigwedge_{i\in k} \frac{\hat{H}_G(t_i)}{H_G(t_i)} \in (1-2\varepsilon, 1+2\varepsilon)\right] \geq 1-(k+1)\delta.$$

Proof. Note that if

$$1-\varepsilon \leq \hat{N}_G(t)/N_G(t) \leq 1+\varepsilon$$

holds for every t, then *a fortiori*

$$1-\varepsilon \leq \max_t \hat{N}_G(t)/\max_t N_G(t) \leq 1+\varepsilon$$

(because, although the maxima might be first attained at different values of t, the same holds for any larger values). As a consequence,

$$1-2\varepsilon \leq \frac{1-\varepsilon}{1+\varepsilon} \leq \frac{\hat{H}_G(t)}{H_G(t)} \leq \frac{1+\varepsilon}{1-\varepsilon} \leq 1+2\varepsilon.$$

The probability $1-(k+1)\delta$ is immediate from the union bound, as we are considering $k+1$ events at the same time. □

Note two significant limitations: first of all, making precise statements (i.e., with confidence) about *all* points of $H_G(t)$ requires a very high initial precision and confidence. Second, the theorem holds if HyperANF has been run up to stabilisation, so that the probabilistic guarantees of HyperLogLog hold for all t.

The first limitation makes in practice impossible to get directly sensible confidence intervals, for instance, for the average distance or higher moments of the distribution (we will elaborate further on this point later). Thus, only statements about a small, finite number of points can be approached directly.

The second limitation is somewhat more serious in theory, albeit in practice it can be circumvented making suitable assumptions about the graph under examination (which however should be clearly stated along the data). Consider the graph G made by two

k-cliques joined by a unidirectional path of ℓ nodes (see Figure 3.2). Even neglecting the effect of approximation, G can "fool" HyperANF (or ANF) so that the distance cdf is completely wrong (see Figure 3.1) when using *any* stopping criterion that is not stabilisation.

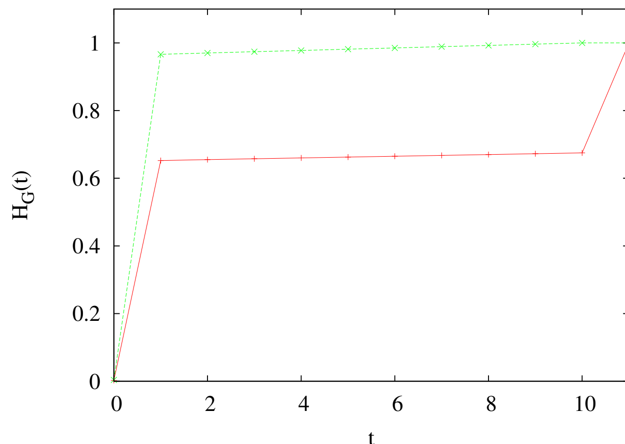

Figure 3.1: The real cdf of the graph in Figure 3.2 (+), and the one that would be computed using *any* termination condition that is not stabilisation (*); here $\ell = 10$ and $k = 260$.

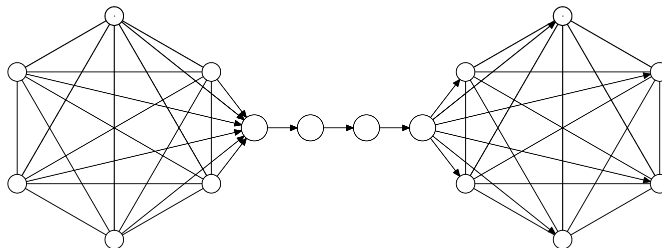

Figure 3.2: Two k-cliques joined by a unidirectional path of ℓ nodes: terminating even one step earlier than stabilisation completely miscalculates the distance cdf (see Figure 3.1); the effective diameter is $\ell + 1$, but terminating even just one step earlier than stabilisation yields an estimated effective diameter of 1.

Indeed, the exact neighbourhood function of G is given by:

$$N_G(t) = \begin{cases} 2k + \ell & \text{if } t = 0 \\ (t+1)\left(2k + \ell - \frac{t}{2}\right) - 2k + 2k^2 & \text{if } 1 \le t \le \ell \\ (\ell+1)\left(2k + \frac{\ell}{2}\right) - 2k + 3k^2 & \text{if } \ell < t. \end{cases}$$

The key observation is that the very last value is significantly larger than all previous values, as at the last step the nodes of the right clique become reachable from the nodes of the first clique. Thus, if iteration stops before stabilisation,[9] the normalisation factor used to compute the cdf will be smaller by $\approx k^2$ than the actual value, causing a completely wrong estimation of the cdf, as shown in Figure 3.1.

Although this counterexample (which can be easily adapted to be symmetric) is definitely pathological, it suggests that a particular care should be taken when handling graphs that present narrow "tubes" connecting large connected components: in such scenarios, the function $N_G(t)$ exhibits relatively long plateaux (preceded and followed by sharp bumps) that may fool the computation of the cdf.

The effective diameter. The first application of ANF was the computation of the *effective diameter*. The effective diameter of G at α is the smallest t_0 such that $H_G(t_0) \ge \alpha$; when α is omitted, it is assumed to be $\alpha = .9$.[10] The *interpolated* effective diameter is obtained in the same way on the *linear interpolation* of the points of the neighbourhood function.

Since that the function $\hat{H}_G(t)$ is necessarily monotone in t (independently of the approximation error), from Theorem 2 we obtain:

Corollary 2. *Assume $\hat{N}_G(t)$ is known for each t with error ε and confidence $1-\delta$, and there are points s and t such that*

$$\frac{\hat{H}_G(s)}{1-2\varepsilon} \le \alpha \le \frac{\hat{H}_G(t)}{1+2\varepsilon}.$$

Then, with probability $1-3\delta$ the effective diameter at α lies in $[s\,..\,t]$.

Unfortunately, since the effective diameter depends sensitively on the distance cdf, again termination conditions can produce arbitrarily large errors. Getting back to the example of Figure 3.2, with a sufficiently large k, for example $k = 2\ell^2 + 5\ell + 2$, the

[9] We remark that stabilisation can occur, in principle, even before the last step because of hash collisions in HyperLogLog counters, but this will happen with a controlled probability.

[10] The actual diameter of G is its effective diameter at 1, albeit the latter is defined for all graphs whereas the former makes sense only in the strongly connected case.

effective diameter is $\ell+1$, which would be correctly output after $\ell+1$ iterations, whereas even stopping one step earlier (i.e., with $t = \ell$) would produce 1 as output, yielding an arbitrarily large error. `snap`, indeed, fails to produce the correct result on this graph, because it stops iterating whenever the ratio between two successive iterates of N_G is sufficiently close to 1.

Algorithm 6 Computing the effective diameter at α of a graph G; Algorithm 5 is used to compute $\hat{N}_G$.

0 **foreach** $t = 0, 1, \ldots$ **begin**
1 compute $\hat{N}_G(t)$ (error ε, confidence $1-\delta$)
2 **if** (some termination condition holds) **break**
3 **end**;
4 $M \leftarrow \max \hat{N}_G(t)$
5 find the largest D^- such that $\hat{N}_G(D^-)/M \leq \alpha(1-2\varepsilon)$
6 find the smallest D^+ such that $\hat{N}_G(D^+)/M \geq \alpha(1+2\varepsilon)$
7 output $[D^- \mathinner{\ldotp\ldotp} D^+]$ with confidence $1-3\delta$
8 **end**;

Algorithm 6 is used to estimate the effective diameter of a graph; albeit this approach is reasonable (and actually it is similar to that adopted by `snap`, although the latter does not provide any confidence interval), unless the neighbourhood function is known with very high precision it is almost impossible to obtain good upper bounds, because of the typical flatness of the distance cdf after the 90th percentile. Moreover, results computed using a termination condition different from stabilisation should always be taken with a grain of salt because of the discussion above.

The distance density function. The situation, from a theoretical viewpoint, is somehow even worse when we consider the density function $h_G(-)$ associated to the cdf $H_G(-)$. Controlling the error on $h_G(-)$ is not easy:

Lemma 3. *Assume that, for a given t, $\hat{H}_G(t)$ is an estimator of $H_G(t)$ with error ε and confidence $1-\delta$. Then $\hat{h}_G(t) = h_G(t) \pm 2\varepsilon$ with confidence $1-2\delta$.*

Proof. With confidence $1-2\delta$,

$$\begin{aligned}\hat{h}_G(t) &= \hat{H}_G(t) - \hat{H}_G(t-1)\\ &\leq (1+\varepsilon)H_G(t) - (1-\varepsilon)H_G(t-1) \leq h_G(t) + 2\varepsilon,\end{aligned}$$

and similarly $\hat{h}_G(t) \geq h_G(t) - 2\varepsilon$. $\square$

Note that the bound is very weak: since our best generic lower bound is $h_G(t) \geq 1/n^2$, the relative error with which we known a point $h_G(t)$ is $2\varepsilon n^2$ (which, of course, is pretty useless).

Moments. Evaluation of the moments of $h_G(-)$ poses further problems. Actually, by Lemma 3 we can deduce that

$$\sum_t t h_G(t) - 2\varepsilon D_G \leq \sum_t t\hat{h}_G(t) \leq \sum_t t h_G(t) + 2\varepsilon D_G$$

with confidence $1 - 2D_G\delta$, where D_G is the diameter of G, which implies that the expected value of $\hat{h}_G(-)$ is an (almost) unbiased estimator of the expected value of $h_G(-)$. Nonetheless, the bounds we obtain are horrible (and actually unusable).

The situation for the variance is even worse, as we have to *prove* that we can use $\mathrm{Var}[\hat{h}_G]$ as an estimator to $\mathrm{Var}[h_G]$. Note that for a fixed graph G, H_G is a precise distribution and $\mathrm{Var}[h_G]$ is an *actual number*. Conversely, $\hat{h}_G$ (and hence $\mathrm{Var}[\hat{h}_G]$) is a random variable[11]. By Theorem 2, we know that $\hat{H}_G$ is an (almost) unbiased pointwise estimator for H_G, and that we can control its concentration by suitably choosing the number m of counters. We are going to derive bounds on the approximation of $\mathrm{Var}[h_G]$ using the values of $\hat{H}_G(t)$ up to $\hat{D}_G$ (i.e., the iteration at which HyperANF stabilises):

Lemma 4. *Assume that, for every $0 \leq t \leq \hat{D}_G$, $\hat{H}_G(t)$ is an estimator of $H_G(t)$ with error ε and confidence $1 - \delta$; then, $\mathrm{Var}[\hat{h}_G]$ is an estimator of $\mathrm{Var}[h_G]$ with error*

$$\varepsilon \leq 8\varepsilon \frac{D_G^3}{\mathrm{Var}[h_G]} + 4\varepsilon^2 \frac{D_G^4}{\mathrm{Var}[h_G]}$$

and confidence $1 - (D_G + 1)\delta$.

Proof. Assuming error ε on the values of $\hat{H}$ in $[0\,..\,D_G]$ implies confidence $1-(D_G+1)\delta$. Since $\hat{D}_G \leq D_G < \infty$, and by definition $\hat{h}_G(t) = 0$ for $t > \hat{D}_G$ we have (t ranges in

[11] More precisely, $\hat{h}_G$ is a sequence of (stochastically dependent) random variables $\hat{h}_G(0)$, $\hat{h}_G(1)$, ...

$[0\,..\,D_G]$):

$$\begin{aligned}
\mathrm{Var}[\hat{h}_G] &= \sum_t t^2 \hat{h}_G(t) - \Bigl(\sum_t t\hat{h}_G(t)\Bigr)^2 \\
&\le \sum_t t^2\bigl(h_G(t) + 2\varepsilon\bigr) - \Bigl(\sum_t th_G(t) - 2\varepsilon\sum_t t\Bigr)^2 \\
&\le \mathrm{Var}[h_G] + 2\varepsilon\sum_t t^2 + 4\varepsilon E[h_G]\sum_t t \\
&\le \mathrm{Var}[h_G] + 4\varepsilon D_G^2\bigl(D_G + E[h_G]\bigr) \\
&\le \mathrm{Var}[h_G] + 8\varepsilon D_G^3,
\end{aligned}$$

where $E[h_G]$ is the average path length. Similarly

$$\mathrm{Var}[\hat{h}_G] \ge \mathrm{Var}[h_G] - 8\varepsilon D_G^3 - 4\varepsilon^2 D_G^4.$$

Hence the statement. □

The error and confidence we obtain are again unusable, but the lemma proves that with enough precision and confidence on $\hat{H}_G(-)$ we can get precision and confidence on $\mathrm{Var}[h_G]$.

The results in this section suggests that if computations involve the moments the only realistic possibility is to resort to parametric statistics to study the behaviour of the value of interest on a large number of samples. That is, it is better to compute a large number of relatively low-precision approximate neighbourhood functions than a small number of high-precision ones, as from the former the latter are easily computable by averaging, whereas it is impossible to obtain a large number of samples of derived values from the latter. As we will see, this approach works surprisingly well.

3.5 SPID

The main purpose of computing aggregated data such as the distance distribution is that we can try to define indices that express some structural property of the graph we study, an obvious example being the average distance, or the effective diameter.

One of the main goal of our recent research has been finding a simple property that clearly distinguishes between social networks deriving from human interaction (what is usually called a social network in the strong or proper sense: DBLP, Facebook, etc.) and web-based graphs, which share several properties of social networks, and as the latter arise from human activity, but present a visibly different structure.

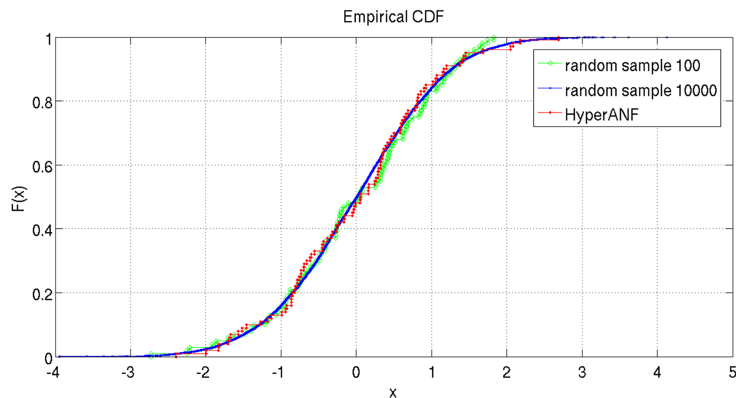

Figure 3.3: Cumulative density function of 100 values of the spid computed using HyperANF on `cnr-2000`. For comparison, we also plot random samples of size 100 and 10 000 drawn from a normal distribution.

We propose for the first time to use the *index of dispersion* σ^2/μ (a.k.a. *variance-to-mean ratio*) of the distance distribution as a measure of the "webbiness" of a social network. We call such an index the *spid (shortest-paths index of dispersion)*[12] *of* G. In particular, networks with a spid larger than one are to be considered "web-like", whereas networks with a spid smaller than one are to be considered "properly social". We recall that a distribution is called under- or over-dispersed depending on whether its index of dispersion is smaller or larger than 1, so a network is properly social or not depending on whether its distance distribution is under- or over-dispersed.

The intuition behind the spid is that "properly social" networks strongly favour short connections, whereas in the web long connection are not uncommon: this intuition will be confirmed in Section 3.6.

As discussed in the previous section, in theory estimating the spid is an impossible task, due to the inherent difficulty of evaluating the moments of $h_G(-)$. In practice, however, the estimate of the spid computed directly on runs of HyperANF are quite precise. From the actual neighbourhood function computed for `cnr-2000` we deduce that the graph spid is 2.49. We then ran 100 iteration of HyperANF with a relative standard deviation of 9.37%, computing for each of them an estimation of the spid; these values approximately follow a normal distribution of mean 2.489 and standard deviation 0.9 (see Figure 3.3). We obtained analogous concentration results for the average distance. In some pathological cases, the distribution is not Gaussian, albeit

[12]If we were to follow strictly the terminology used here, this would be the index of dispersion of the distance distribution, but we guessed that the acronym IDDD would not have been as as successful.

it always turns out to be unimodal (in some cases, discarding few outliers), so we can apply the Vysochanskiĭ-Petunin inequality. We will report some relevant observations on the spid of a number of graphs after describing our experiments.

3.6 Experiments

We ran our experiments on the datasets described in Table 3.2:

- the web graphs are almost all available at `http://law.dsi.unimi.it/`, except for the `altavista` dataset that was provided by Yahoo! within the Webscope program (AltaVista webpage connectivity dataset, version 1.0, `http://research.yahoo.com/Academic_Relations`);[13]

- for the social networks: `hollywood` (`http://www.imdb.com/`) is a co-actorship graph where vertices represent actors; `dblp` (`http://www.informatik.uni-trier.de/~ley/db/`) is a scientific collaboration network where each vertex represents a scientist and two vertices are connected if they have worked together on an article; in `ljournal` (`http://www.livejournal.com/`) nodes are users and there is an arc from x to y if x registered y among his friends (it is not necessary to ask y permission, so the graph is *directed*); `amazon` (`http://www.archive.org/details/amazon_similarity_isbn/`) describes similarity among books as reported by the Amazon store; `enron` is a partially anonymised corpus of e-mail messages exchanged by some Enron employees (nodes represent people and there is an arc from x to y whenever y was the recipient of a message sent by x); finally in `flickr` (`http://www.flickr.com/`[14]) vertices correspond to Flickr users and there is an edge connecting x and y whenever either vertex is recorded as a contact of the other one.

[13] It should be remarked by this graph, albeit widely used in the literature, is not a good dataset. The dangling nodes are 53.74%—an impossibly high value [71], and an almost sure indication that all nodes in the frontier of the crawler (and not only visited nodes) were added to the graph, and the giant component is less than 4% of the whole graph.

[14] We thank Yahoo! for the experimental results on the Flickr graph.

Graph	`snap`	HyperANF
`amazon`	9.5 m	5 s
`indochina-2004`	4.62 h	1.83 m
`altavista`	-	1.2 h
	HADI (90 machines)	HyperANF
Kronecker (177 K nodes, 2 B arcs)	30 m	2.25 m

Table 3.1: A comparison of the speed of `snap`/HADI vs. HyperANF. The tests on `snap` were performed on our hardware. Both algorithms were stopped at a relative increment of 0.001. The timings of HADI on the M45 cluster are the best reported in [41], and both algorithms ran three iterations. We remark that a run of HyperANF on the Kronecker graph takes *less than fifteen minutes on a laptop.*

Name	Type	Nodes	Arcs	spid ($\pm\sigma$)	ad ($\pm\sigma$)	ied ($\pm\sigma$)	ed (2)
`amazon`	social (u)	735 323	5 158 388	0.76 (±0.060)	12.05 (±0.206)	15.50 (±0.433)	[14 . . 18]
`dblp`	social (u)	326 186	1 615 400	0.36 (±0.034)	7.34 (±0.114)	8.96 (±0.215)	[8 . . 10]
`enron`	social (d)	69 244	276 143	0.21 (±0.020)	4.24 (±0.065)	4.94 (±0.103)	[4 . . 6]
`ljournal`	social (d)	5 363 260	79 023 142	0.21 (±0.023)	5.99 (±0.078)	6.92 (±0.143)	[6 . . 8]
`flickr`	social (u)	526 606	47 097 454	0.14 (±0.009)	3.50 (±0.047)	3.92 (±0.049)	[3 . . 5]
`hollywood`	social (u)	1 139 905	113 891 327	0.14 (±0.012)	3.87 (±0.045)	4.42 (±0.109)	[4 . . 5]
`indochina-2004-hosts`	host (d)	19 123	233 380	0.35 (±0.021)	4.26 (±0.079)	5.44 (±0.164)	[5 . . 7]
`uk-2005-hosts`	host (d)	587 205	12 825 465	0.30 (±0.018)	5.93 (±0.081)	7.06 (±0.151)	[6 . . 8]
`cnr-2000`	web (d)	325 557	3 216 152	2.50 (±0.086)	17.35 (±0.313)	25.45 (±0.357)	[23 . . 29]
`eu-2005`	web (d)	862 664	19 235 140	1.25 (±0.209)	10.17 (±0.363)	14.31 (±0.988)	[13 . . 16]
`in-2004`	web (d)	1 382 908	16 917 053	1.30 (±0.173)	15.40 (±0.374)	20.74 (±0.792)	[20 . . 24]
`indochina-2004`	web (d)	7 414 866	194 109 311	1.64 (±0.134)	15.63 (±0.338)	21.68 (±0.658)	[20 . . 26]
`uk@10E6`	web (d)	100 000	3 050 615	1.64 (±0.111)	5.97 (±0.172)	10.36 (±0.251)	[8 . . 12]
`uk@10E7`	web (d)	1 000 000	41 247 159	1.76 (±0.043)	8.96 (±0.172)	14.31 (±0.341)	[12 . . 17]
`it-2004`	web (d)	41 291 594	1 150 725 436	2.14 (±0.149)	15.02 (±0.300)	19.65 (±0.698)	[18 . . 22]
`uk-2007-05`	web (d)	105 896 555	3 738 733 648	1.10 (±0.234)	15.39 (±0.418)	19.93 (±1.030)	[18 . . 23]
`altavista`	web (d)	1 413 511 390	6 636 600 779	4.24 (±0.764)	16.69 (±0.779)	23.04 (±2.517)	[19 . . 31]

Table 3.2: Our main data table. “Type” describes whether the given graph is a web-graph, a proper social network, or the host quotient of a web graph (u=undirected, d=directed). The graphs `uk@10E6` and `uk@10E7` are obtained by visiting in a breadth-first fashion `uk-2007-05` starting from a random node. They simulate smaller crawls of a larger network. We show spid, average distance and interpolated effective diameter *a posteriori* with their empirical standard deviation, and intervals for the effective diameter with 85% confidence for a comparison.

All experiments are performed on a Linux server equipped with Intel Xeon X5660 CPUs (2.80 GHz, 12 MB cache size) for overall 24 cores and 128 GB of RAM; the server cost about 8 900 EUR in 2010.

A brief comparison with `snap` and HADI timings is shown in Table 3.1. Essentially, on our hardware HyperANF is two orders of magnitudes faster than `snap`. Our run on the Kronecker graph is one order of magnitude faster than HADI's (or three orders of magnitude faster, if you take into consideration the number of machines involved), but this comparison is unfair, as in principle HADI can scale to arbitrarily large graphs, whereas we are limited by the amount of memory available. Nonetheless, the speedup is clearly a breakthrough in the analysis of large graphs. It would be interesting to compare our timings for the `altavista` dataset with HADI's, but none have been published.

It is this speed that makes it possible, for the first time, to compute data associated with the distance distribution with high precision and for a large number of graphs. We have 100 runs with relative standard deviation of 9.37% for all graphs, except for those on the `altavista` dataset (13.25%). All graphs are run to stabilisation. Our computations are necessarily much longer (usually, an order of magnitude longer in iterations) than those used to compute the effective diameter or similar measures. This is due to the necessity of computing with high precision second-order statistics that are used to compute the spid.

Previous publications used few graphs, mainly because of the large computational effort that was necessary, and no data was available about the number of runs. Moreover, we give precise confidence intervals based on parametric statistics for data depending on the second moment, such as the spid—something that has never done before. We gather here our findings.

***A posteriori* parameters are highly concentrated.** According to our experiments, computing the effective diameter, average distance and spid on a large number of low-precision runs generates highly concentrated distributions (see the empirical standard deviation in Table 3.2). Thus, we suggest this approach for computing such values, *provided that termination is by stabilisation.*

Effective diameter and average distance are essentially linearly correlated. Figure 3.4 shows a scatter plot of the two values, and the line $2x/3 + 1$. The correlation between the two values has always been folklore in the study of social networks, but we can confirm that on both social and web networks the connection can be exactly expressed in linear terms (it would be of course interesting to prove such a correlation formally, under suitable restrictions on the structure of the graph). This fact suggests that the average distance (which is more principled from a statistic viewpoint, and parameter-free) should be used as the reference parameter to express the closeness

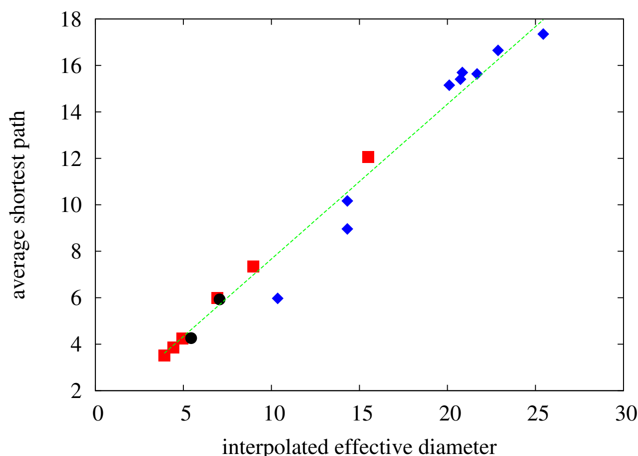

Figure 3.4: A plot showing the strong linear correlation between the average distance and the effective diameter.

between nodes. Moreover, experimentally the standard deviation of the effective diameter in a posteriori computations is usually significantly larger than that of the average distance.

Incidentally, the average distance of the `altavista` dataset is 16.5—slightly more than what reported in [41] (possibly because of termination conditions artifacts).

It is difficult to give *a priori* confidence intervals for the effective diameter with a small number of runs. Unless a large number of runs is available, so that the precision of the approximation of the neighbourhood function can be significantly lowered, it is impossible to provide interesting upper bounds for the effective diameter.

The spid can tell social networks from web graphs. As shown in Table 3.2, even taking the standard deviation into account spids are pretty much below 1 for social networks and above 1 for web graphs; host graphs (not surprisingly) behave like social networks. Note that this works *both for directed and undirected graphs.* Figure 3.5 shows the spid values obtained for our datasets plotted against the graph size, and also witnesses that there is no correlation (a similar graph, not shown here, testifies that spid is also independent from density). Figure 3.6 shows that there is some slight correlation between the spid and the average distance: nonetheless, there is no way to tell networks from our dataset apart using the latter value, whereas the under- or over-dispersion of the distance distribution, as defined by the spid, never makes a mistake. Of course, we expect to enrich this graph in time with more datasets: we are particularly interested in gathering very large social networks to test the spid at large sizes.

We remark that, as a sanity check, we have also computed on several web-graph datasets the spid of the *giant component*, which turned out to be very similar to the spid of the whole graph. We see this as a clear sign that the spid *is largely independent of the artifacts of the crawling process*.

Direction should not be destroyed when analysing a graph. We confirm that symmetrising graphs destroys the combinatorial structure of the network: the average distance drops to very low values in all cases, as well as the spid. This suggests that there is important structural information that is being ignored. We also note that since all web snapshot we have at hand are gathered by some kind of breadth-first visit, they represent balls of small diameter centred at the seed: symmetrising the graph we cannot expect to get an average distance that is larger than twice the radius of the ball. All in all, the only advantage of symmetrising a graph is a significant reduction in the number of iterations that are needed to complete a computation of the neighbourhood function.[15]

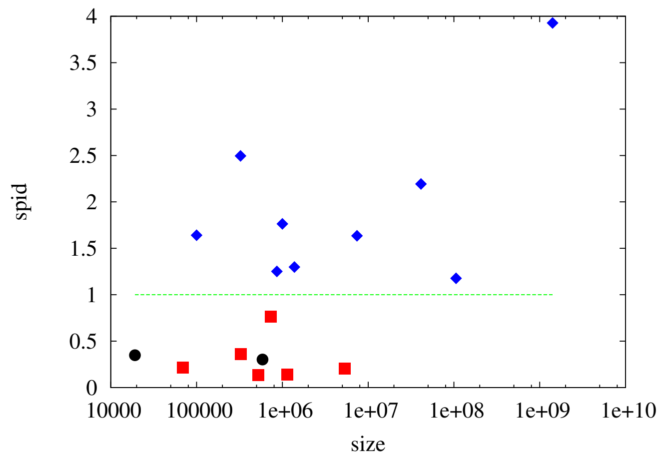

Figure 3.5: A plot showing the spid values (vertical) for our datasets compared with their size (i.e., number of nodes, horizontal): red squares correspond to social networks, blue diamonds to web graphs and black circles to host graphs.

To give a more direct idea of the level of precision of our diameter estimation, we computed the actual diameter at α for the `cnr-2000` dataset, and plotted it against the interval estimation obtained by HyperANF

[15]We remark that the "diameter $7 \sim 8$" claim in [41] about the `altavista` dataset refers to the *effective* diameter for the *symmetrised* version of the graph.

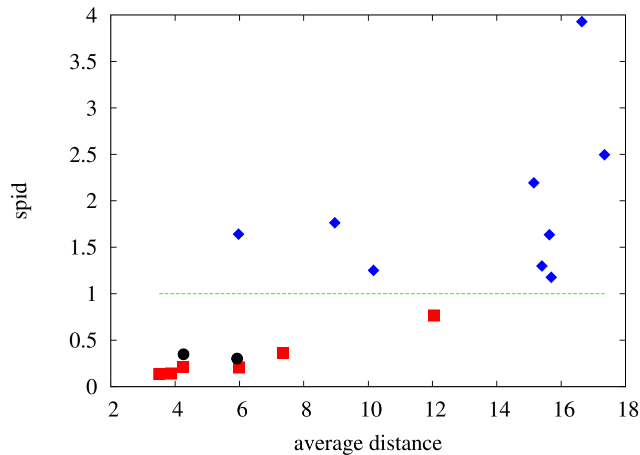

Figure 3.6: A plot showing the spid against the average distance using the same conventions of Figure 3.5.

3.7 Facebook graph

At the 20th World–Wide Web Conference, in Hyerabad, during the presentation of HyperANF one of the main open questions was "What is the spid of Facebook?". Lars Backstrom happened to listen to the talk, and made himself available to run our tools at Facebook. This was of course an extremely intriguing possibility: beside testing our "spid hypothesis", computing the distance distribution of the Facebook graph would have been the largest Milgram-like [54] experiment ever been performed, several orders of magnitued larger than previous attemps (during our experiments Facebook has ≈750 million *active* users).

This section reports our findings in studying the distance distribution of the largest electronic social network ever created. That world is smaller than we thought: the current Facebook graph has an average distance of 4.79 hops. Moreover, the spid of the graph is just 0.21, corroborating our conjecture that proper social networks have a spid well below one.

The obvious precursor of our experiment is Milgram's celebrated "small world" experiment [54]. There is of course a fundamental difference in the two experiments: Milgram was measuring the average length of a *greedy routing path* on a social network, which is just an upper bound on the average distance (as the people involved in the experiment were not sending necessarily the postcard to an acquaintance on a shortest path to the destination).

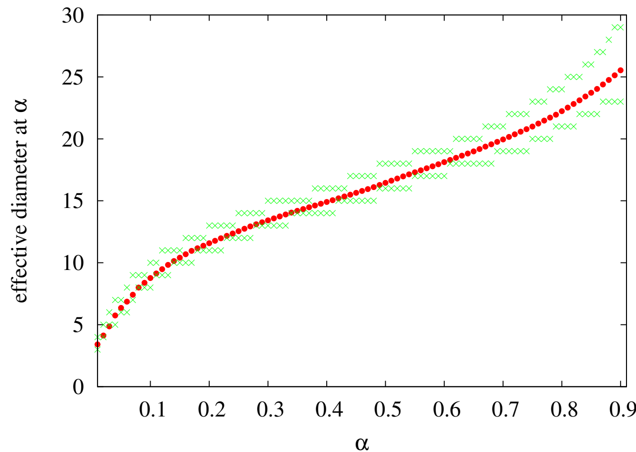

Figure 3.7: Effective diameters at α for the `cnr-2000` dataset; red bullets show the real effective diameter, whereas green crosses show the upper and lower extreme of the confidence interval obtained running 100 HyperANF with $m = 128$.

The graph we analysed is the graph of *active* (i.e., logged in within the last 28 days) Facebook users at the start of each year from 2007 onwards. The "current" graph is the graph of active users at the time when the experiments were performed (circa April 2011). The graph does not include commercial accounts that people may "like", such as famous people's accounts, and there is a limit of 5000 friends on standard accounts.

We decided to extend our experiments in two direction: regional and temporal. We thus analyse the entire Facebook graph, the US subgraph, the Swedish subgraph, and the Italian subgraph. We also analysed a combination of the Swedish and Italian graph to check whether combining regional but far networks would significantly change the average distance. For each graph we compute the distance distribution from 2007 up to today by running 10 times HyperANF. We can thus claim that the values of the neighbourhood function we compute are within 8.4% with 90% probability for all graphs except for the entire Facebook graph, where they are within 11.9%.

We note that both on the Italian and Swedish graph we have a significantly lower average distance, showing that the average distance is actually dependent on the size of the graph. During the fastest growing years of Facebook our graphs show a quick decrease in the average distance, which however appears to be stabilizing. This is not surprising, as "shrinking diameter" phenomena are always observed when a large network is "uncovered" (in the sense that we look at larger and larger induced subgraphs of the original network).

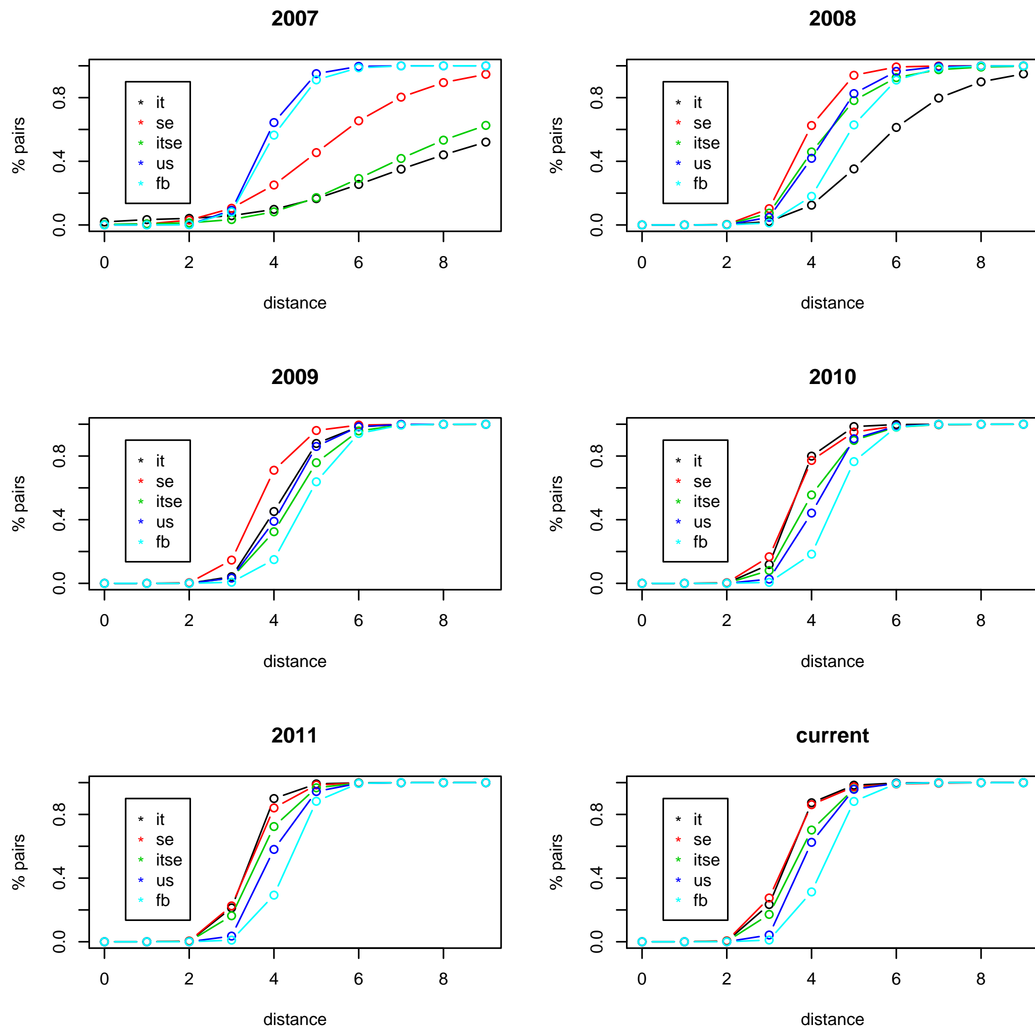

Figure 3.8: Comparison between regional graphs from 2007 to current.

	it	se	itse	us	fb
2007	10.43 (± 0.81)	5.89 (± 0.32)	9.20 (± 0.42)	4.31 (± 0.06)	4.44 (± 0.10)
2008	6.30 (± 0.15)	4.34 (± 0.03)	4.79 (± 0.06)	4.74 (± 0.07)	5.28 (± 0.08)
2009	4.65 (± 0.09)	4.19 (± 0.02)	4.93 (± 0.09)	4.73 (± 0.06)	5.27 (± 0.07)
2010	4.10 (± 0.04)	4.12 (± 0.04)	4.48 (± 0.10)	4.64 (± 0.05)	5.05 (± 0.07)
2011	3.89 (± 0.04)	3.94 (± 0.06)	4.14 (± 0.06)	4.44 (± 0.09)	4.81 (± 0.09)
current	3.90 (± 0.04)	3.89 (± 0.11)	4.16 (± 0.07)	4.38 (± 0.04)	4.79 (± 0.08)

Table 3.3: The average distance (± sample standard deviation)

	it	se	itse	us	fb
2007	3.07 (± 0.28)	0.71 (± 0.13)	2.05 (± 0.36)	0.12 (± 0.01)	0.14 (± 0.01)
2008	0.53 (± 0.05)	0.14 (± 0.01)	0.27 (± 0.02)	0.17 (± 0.01)	0.16 (± 0.02)
2009	0.14 (± 0.03)	0.14 (± 0.01)	0.17 (± 0.02)	0.13 (± 0.01)	0.13 (± 0.01)
2010	0.09 (± 0.01)	0.15 (± 0.02)	0.15 (± 0.02)	0.11 (± 0.01)	0.10 (± 0.00)
2011	0.09 (± 0.01)	0.12 (± 0.00)	0.13 (± 0.01)	0.10 (± 0.01)	0.09 (± 0.01)
current	0.11 (± 0.01)	0.14 (± 0.04)	0.14 (± 0.01)	0.10 (± 0.01)	0.09 (± 0.01)

Table 3.4: The index of dispersion of distances, a.k.a. spid (± sample standard deviation)

3.8 Conclusions

HyperANF is a breakthrough improvement over the original ANF techniques, mainly because of the usage of the more powerful HyperLogLog counters combined with their fast broadword combination and systolic computation. HyperANF can run to stabilisation very large graphs, computing data with statistical guarantees.

The most interesting features of such counters is that the *precision* depends *exclusively* on the number of registers; for instance, if a relative standard deviation of 6.50% is acceptable, one can stick with 256 registers per counter. At that point, the only dependence on the graph size is the size of each register, which however is $\lceil \log \log n \rceil$, so it is unlikely we will ever need more than 6 bits per register. In practice, this feature makes HyperANF scale linearly with the number of nodes.

We consider, however, the introduction of the spid of a graph the main conceptual contribution of this chapter. HyperLogLog is instrumental in making the computation of the spid possible, as the latter requires a number of iterations that is an order of magnitude larger than those required for an estimate of the effective diameter.

Chapter 4

Robustness of Social Networks

Given a social network, which of its nodes have a stronger impact in determining its structure? More formally: which node-removal order has the greatest impact on the network structure? In this chapter we will take advantage of the results presented in previous chapters in order to study what we call the robustness of a graph. Mainly, we exploit HyperANF to approximate accurately the number of reachable pairs and the distribution of distances in a graph. But also the clustering technique presented in Chapter 2 turns out to be the best tool we are aware of to locate nodes that are important from a structural viewpoint.

As always, we also look for differences and similarities between social networks and web graphs. Our experiments show for the first time that also under this respect there is a clear-cut structural difference between social networks and web graphs. Probably the most important conclusion is that "scale-free" models, which are currently proposed for both web graphs and social networks, do not to capture this important difference: for this reason, they can only make sense as long as they are adopted as baselines.

4.1 Introduction

One of the most important notions that researchers have been trying to capture is "node centrality": ideally, every node (often representing an individual) has some degree of influence or importance within the social domain under consideration, and one expects such importance to be reflected in the structure of the social network; centrality is a quantitative measure that aims at revealing the importance of a node.

Among the types of centrality that have been considered in the literature (see [18] for a good survey), many have to do with shortest paths between nodes; for example,

the *betweenness centrality* of a node v is the sum, over all pairs of nodes x and y, of the fraction of shortest paths from x to y passing through v. The role played by shortest paths is justified by one of the most well known features of complex networks, the so-called small-world phenomenon.

Based on the above observation that the small-world property is by far the most crucial of all the features that social networks exhibit, it is quite natural to consider centrality measures that are based on node distance, like betweenness. On the other hand, albeit interesting and profound, such measures are often computationally too expensive to be actually computed on real-world graphs; for example, the best known algorithm to compute betweenness centrality [21] takes time $O(nm)$ and requires space for $O(n + m)$ integers (where n is the number of nodes and m is the number of arcs): both bounds are infeasible for large networks, where typically $n \approx 10^9$ and $m \approx 10^{11}$. For this reason, in most cases other strictly local measures of centrality are usually preferred (e.g., degree centrality).

One of the ideas that have emerged in the literature is that node centrality can be evaluated based on how much the removal of the node "disrupts" the graph structure [1]. This idea provides also a notion of robustness of the network: if removing few nodes has no noticeable impact, then the network structure is clearly robust in a very strong sense. On the other hand, a node-removal strategy that quickly affects the distribution of distances probably reflects an importance order of the nodes.

Previous literature has used mainly the diameter or some analogous measure to establish whether the network structure changed. However, as we have seen in Chapter 3, we are now able to produce reliable estimates of the *neighbourhood function* of very large graphs; an immediate application of these approximate algorithms is the computation of the number of *reachable pairs* of the graph (the number of pairs $\langle x, y\rangle$ such there is a directed path from x to y) and its *distance distribution*. From this data, a number of existing measures can be computed quickly and accurately, and new one can be conceived.

We thus consider a certain ordering of the nodes of a graph (that is supposed to represent their "importance" or "centrality"). We remove nodes (and of course their incident arcs) following this order, until a certain percentage θ of the arcs have been deleted[1]; finally, we compare the number of reachable pairs and distance distribution of the new graph with the original one. The chosen ordering is considered to be a reliable measure of centrality if the measured difference increases rapidly with θ (i.e., it

[1]Observe that we delete nodes but count the percentage of arcs removed, and not of nodes: this choice is justified by the fact that otherwise node orderings that put large-degree nodes first would certainly be considered (unfairly) more disruptive.

is sufficient to delete a small fraction of important nodes to change the structure of the graph).

In this work, we applied the described approach to a number of complex networks, considering different orderings, and obtained the following results:

- In all complex networks we considered, the removal of a limited fraction of randomly chosen nodes does not change the distance distribution significantly, confirming previous results.
- We test strategies based on PageRank and on clustering (see Section 4.3.1 for more information about this), and show that they (in particular, the latter) disrupt quickly the structure of a web graph.
- Maybe surprisingly, none of the above strategies seem to have an impact when applied to social networks other than web graphs. This is yet another example of a profound structural difference between web graphs and social networks,[2] on the same line as those discussed in [13] and [23]. This observation, in particular, suggests that social networks tend to be much more robust and cohesive than web graphs, at least as far as distances are concerned, and that "scale-free" models, which are currently proposed for both type of networks, do not to capture this important difference.

4.2 Related work

The idea of grasping information about the structure of a network by repeatedly removing nodes out of it is not new: Albert, Jeong and Barabási [1] study experimentally the variation of the diameter on two different models of *undirected* random graphs when nodes are removed either randomly or in "connectedness order" and report different behaviours. They also perform tests on some small real data set, and we will compare their results with ours in Section 4.5.

More recently, node-centrality measures that look at how some graph invariant changes when some vertices or edges are deleted (sometimes called "vitality" [22] or "induced" measures) have been studied for example in [19] (identifying nodes that maximally disconnect the network) or in [20] (related to the uncertainty of data).

[2] We remark that several proposals have been made to find features that highlight such structural differences in a computationwise-feasible way (e.g., assortative mixing [58]), but all instances we are aware of have been questioned by the subsequent literature, so no clear-cut results are known as yet.

Donato, Leonard, Millozzi and Tsaparas [27] study how the size of the giant component changes when nodes of high indegree or outdegree are removed from the graph. While this is an interesting measure, it does not provide information about what happens outside the component. They develop a library for semi-external visits that make it possible to compute in an exact way the strongly connected components on large graphs.

Finally, Fogaras [31] considers how the *harmonic diameter*[3] (the harmonic mean of the distances) changes as nodes are deleted from a small (less than one million node) snapshot of the `.ie` domain, reporting a large increase (100%) when as little as 1000 nodes with high PageRank are removed. The harmonic diameter is estimated by a small number of visits, however, which gives no statistical guarantee on the accuracy of the results.

Our study is very different. First of all, we use graphs that are two orders of magnitude larger than those considered in [1] or [31]; moreover, we study the impact of node removal on the whole spectrum of distances. Second, we apply removal procedures to large social networks (previous literature used only web or Internet graphs), and the striking difference in behaviour shows that "scale-free" models fail to capture essential differences between these kind of networks and web graphs. Third, we document in a reproducible way all our experiments, which have provable statistical accuracy.

4.3 Removal strategies and their analysis

In the previous chapter, we discussed how we can effectively approximate the distance distribution of a given graph G; we shall use such a distribution as the graph structural property of interest.

Consider now a given total order $\prec$ on the nodes of G; we think of $\prec$ as a removal strategy in the following sense: when we want to remove θm arcs, we start removing the $\prec$-largest node (and its incident arcs), go on removing the second-$\prec$-largest node etc. and stop as soon as $\geq \theta m$ arcs have been removed. The resulting graph will be denoted by $G(\prec, \theta)$. Of course, $G(\prec, 0) = G$ whereas $G(\prec, 1)$ is the empty graph. We are interested in applying some measure of *divergence*[4] between the distribution H_G and the distribution $H_{G(\prec,\theta)}$. By looking at the divergence when θ varies, we can judge the ability of $\prec$ to identify nodes that will disrupt the network.

[3]Actually, the notion had been introduced before by Marchiori and Latora and named *connectivity length* [51], but we find the name "harmonic diameter" much more insightful.

[4]We purposely use the word "divergence" between distributions, instead of "distance", to avoid confusion with the notion of distance in a graph.

4.3.1 Some removal strategies

We considered several different strategies for removing nodes from a graph. Some of them embody actually significant knowledge about the structure of the graph, whereas others are very simple (or even independent of the graph) and will be used as baseline. Some of them have been used in the previous literature, and will be useful to compare our results.

As a first observation, some strategies requires a symmetric graph (a.k.a., undirected). In this case, we symmetrise the graph by adding the missing arcs[5].

The second obvious observation is that some strategies might depend on available metadata (e.g., URLs for web graphs) and might not make sense for all graphs.

Random. No strategy: we pick random nodes and remove them from the graph. It is important to test against this "nonstrategy" as we can show that the phenomena we observe are due to the peculiar choice of nodes involved, and not to some generic property of the graph.

Largest-degree first. We remove nodes in decreasing (out)degree order. This strategy is an obvious baseline, as *degree centrality* is the first shot at centrality in a network.

Near-Root. In web graphs, we can assume that nodes that are roots of web sites and their (quasi-)immediate successors (e.g., pages linked by the root) are most important in establishing the distance distribution, as people tend to link higher levels of web sites. This strategy removes essentially first root nodes, then the nodes that are children of a root on, and so on.

PageRank. PageRank [59] is an well-known algorithm that assigns ranks to nodes using a Markov chain based on the structure of the graph. It has been designed as an improvement over degree centrality, because nodes with high degree which however are connected to nodes of low rank will have a rather low rank, too (the definition is indeed recursive). There is a vast body of literature on the subject: see [14, 47] and the references therein.

Label propagation. We try the clustering technique presented in Chapter 2 always with γ fixed to zero (see Algorithm 3). Our removal strategy picks first, for each cluster in decreasing size order, the node with the highest number of neighbours in

[5]It is mostly a matter of taste whether to use directed symmetric graphs or simple undirected graphs. In our case, since we have to cope with both directed and undirected graph, we prefer to speak of directed graphs that are symmetric, that is, for every arc $x \to y$ there is a symmetric arc $y \to x$.

other clusters: intuitively, it is a representative of a set of tightly connected nodes (the cluster) which however has a very significant connection with the outside world (the other clusters) and thus we expect that its removal should seriously disrupt the distance distribution. Once we have removed all such nodes, we proceed again, cluster by cluster, using the same criterion (thus picking the second node of each cluster that has more connection towards other clusters), and so on.

4.3.2 Measures of divergence

Once we changed the structure of a graph by deleting some of its nodes (and arcs), there are several ways to measure whether the structure of the graph has significantly changed. The first, basic raw datum we consider is the *fraction of pairs of nodes that are still reachable* (w.r.t. the number of pairs initially reachable). Then, to estimate the change of the distance distribution we considered the following possibilities (here P denotes the original distance distribution, and Q the distribution after node removal):

Relative average-distance change. This is somehow the simplest and most natural measure: how much has the average distance changed? We use the measure

$$\delta(P, Q) = \frac{\mu_Q}{\mu_P} - 1$$

where μ denotes the average; in other words, we measure how much the average value changed. This measure is non-symmetric, but it is of course easy to obtain $\delta(P, Q)$ from $\delta(Q, P)$.

Relative harmonic-diameter change. This measure is analogous to the relative average distance change, but the average on distances is *harmonic* and *computed on all pairs*, that is:

$$\frac{n(n-1)}{\sum_{x \neq y} \frac{1}{d(x,y)}} = n(n-1) \Big/ \sum_{t>0} \frac{1}{t} (N_G(t) - N_G(t-1)),$$

where n is the number of nodes of the graph. This measure, used in [31], combines reachability information, as unreachable pairs contribute zero to the sum. It is easily computable from the neighbourhood function, as shown above.

Kullback-Leibler divergence. This is a measure of *information gain*, in the sense that it gives the number of additional bits that are necessary to code samples drawn from P when using an optimal code for Q. Also this measure is non-symmetric, but there is no way obtain the divergence from P to Q given that from Q to P.

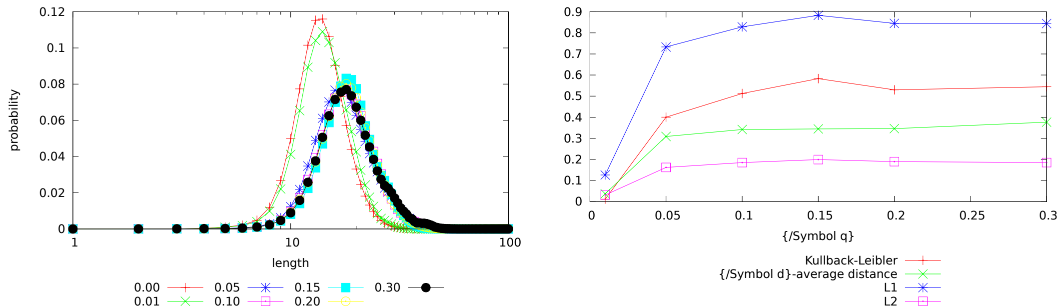

Figure 4.1: Testing various divergence measures on a web graph (a snapshot of the `.it` domain of 2004) and the near-root removal strategy. You can see how the distance distribution changes for different values of θ and the behaviour of divergence measures. We omitted to show the harmonic-diameter change to make the plot easier to read.

ℓ **norms.** A further alternative is given by viewing distance distributions as functions $\mathbf{N} \to [0\,..\,1]$ and measure their distance using some ℓ-norm, most notably ℓ_1 or ℓ_2. Such distances are of course symmetric.

We tested, with various graphs and removal strategies, how the choice of distribution divergence influences the interpretation of the results obtained. In Figure 4.1 we show this for a single web graph and a single strategy, but the outcomes agree on all the graphs and strategies tested: the interpretation is that all divergences agree, and for this reason we shall use the (simple) measure δ applied to the average distance in the experimental section. The advantage of δ over the other measures is that it is very easy to interpret; for example, if δ has value, say, 0.3 it means that node removal has increased the average distance by 30%. We also discuss δ applied to the harmonic diameter.

4.4 Experiments

For our experiments, we considered a number of networks with various sizes and characteristics; most of them are either web graphs or (directed or undirected) social graphs of some kind (note that for web graphs we can rely on the URLs as external source of information). More precisely, we used the following datasets:

- *Hollywood*: One of the most popular *undirected* social graphs, the graph of movie actors: vertices are actors, and two actors are joined by an edge whenever they appeared in a movie together.

- *LiveJournal*: LiveJournal is a virtual community social site started in 1999: nodes are users and there is an arc from x to y if x registered y among his friends (it is not necessary to ask y permission, so the graph is *directed*). We considered the same 2008 snapshot of *LiveJournal* used in [23] for their experiments

- *Amazon*: This dataset describes similarity among books as reported by the Amazon store; more precisely the data was obtained in 2008 using the Amazon E-Commerce Service APIs using `SimilarityLookup` queries.

- *Enron*: This dataset was made public by the Federal Energy Regulatory Commission during its investigations: it is a partially anonymised corpus of e-mail messages exchanged by some Enron employees (mostly part of the senior management). We turned this dataset into a *directed* graph, whose nodes represent people and with an arc from x to y whenever y was the recipient of (at least) a message sent by x.

- For comparison, we considered two web graphs of different size: a 2004 snapshot of the `.it` domain ($\approx$ 40 million nodes), and a snapshot taken in May 2007 of the `.uk` domain ($\approx$ 100 million nodes).

4.5 Discussion

Table 4.1 shows that social networks suffer spectacularly less disconnection than web graphs when their nodes are removed using our strategies. Our most efficient removal strategy, label propagation, can disconnect almost all pairs of a web graph by removing 30% of the arcs, whereas it disconnects only half (or less) of the pairs on social networks. This entirely different behaviour shows that web graphs have a path structure that passes through fundamental hubs.

Moreover, the average distance of the web graphs we consider increases by 50−80% upon removal of 30% of the arcs, whereas in most social networks there is just a 5% increase, the only exception being Amazon (15%).[6]

Note that random removal can separate a good number of reachable pairs, but the increase in average distance is very marginal. This shows that considering both measures is important in evaluating removal strategies.

[6] We remark that in some cases the measure is negative or does not decrease monotonically. This is an artifact of the probabilistic technique used to estimate the number of pairs—small relative errors are unavoidable.

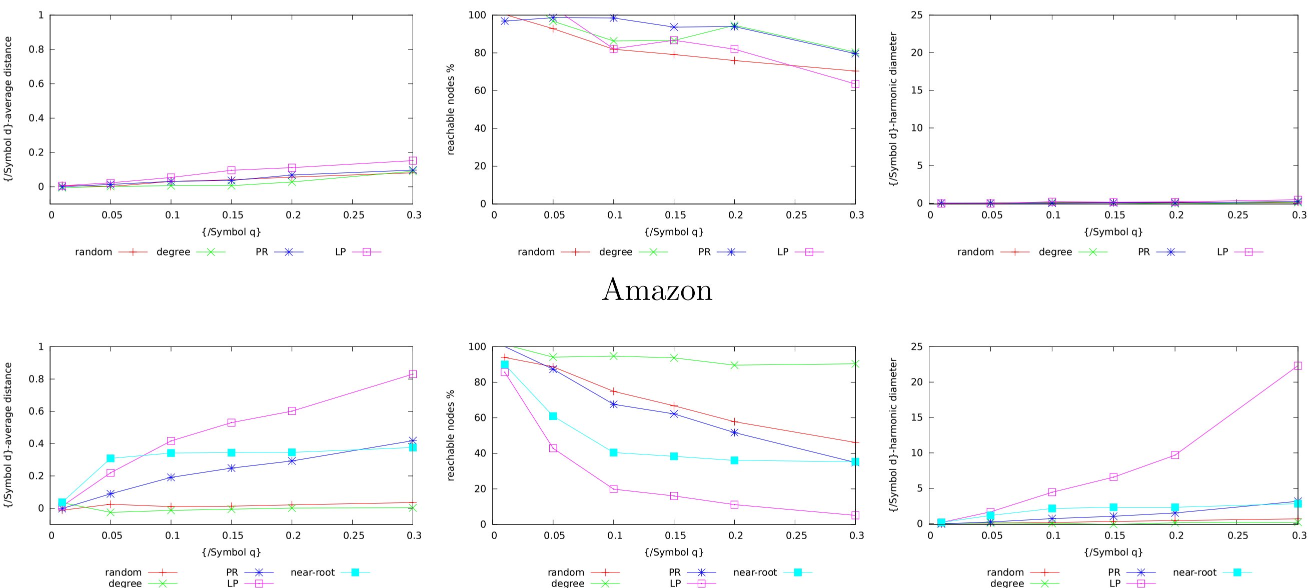

Figure 4.2: Typical behaviour of social networks (Amazon, upper) and web graphs (`.it`, lower) when a θ fraction of arcs is removed using various strategies. None of the proposed strategies completely disrupts the structure of social networks, but the effect of the label-propagation removal strategy on web graphs is very visible.

Graph	Strategy	0.01		0.05		0.1		0.15		0.2		0.3	
Amazon	RND	0.008	(100%)	0.002	(93%)	0.031	(82%)	0.041	(79%)	0.056	(76%)	0.082	(70%)
	DEG	−0.005	(118%)	0.002	(97%)	0.006	(86%)	0.006	(87%)	0.028	(95%)	0.091	(80%)
	PR	0.001	(97%)	0.014	(99%)	0.032	(98%)	0.037	(94%)	0.069	(94%)	0.097	(80%)
	LP	0.006	(104%)	0.023	(104%)	0.054	(82%)	0.096	(87%)	0.112	(82%)	0.153	(64%)
Enron	Random	0.013	(99%)	0.014	(93%)	0.006	(83%)	0.003	(80%)	0.007	(76%)	0.022	(88%)
	Degree	0.006	(97%)	0.017	(86%)	0.056	(75%)	0.061	(72%)	0.064	(67%)	0.13	(52%)
	PR	0.007	(99%)	0.033	(81%)	0.055	(63%)	0.067	(53%)	0.093	(45%)	0.135	(34%)
	LP	0.005	(99%)	0.029	(80%)	0.04	(72%)	−0.048	(59%)	0.061	(57%)	0.05	(52%)
Hollywood	Random	−0.003	(101%)	0.018	(104%)	0.009	(92%)	0.017	(87%)	−0.004	(74%)	0.021	(77%)
	Degree	0.005	(87%)	0.015	(105%)	0.001	(98%)	0.006	(92%)	0.022	(112%)	0.02	(93%)
	PR	0.001	(102%)	0.004	(94%)	0.023	(100%)	0.025	(100%)	0.03	(94%)	0.036	(90%)
	LP	0.018	(90%)	0.038	(78%)	0.052	(65%)	0.066	(57%)	0.061	(54%)	0.058	(52%)
LiveJournal	Random	0.007	(97%)	0.006	(94%)	0.009	(89%)	0.014	(92%)	0.02	(84%)	0.032	(78%)
	Degree	0.003	(95%)	0.02	(91%)	0.053	(105%)	0.065	(108%)	0.064	(92%)	0.101	(91%)
	PR	0.002	(97%)	0.018	(102%)	0.042	(99%)	0.063	(112%)	0.07	(96%)	0.104	(99%)
	LP	0.006	(102%)	0.013	(103%)	0.02	(90%)	0.024	(89%)	0.043	(98%)	0.058	(93%)
`.it`	Random	−0.012	(94%)	0.025	(89%)	0.01	(75%)	0.013	(67%)	0.021	(58%)	0.035	(46%)
	Degree	0.035	(101%)	−0.025	(94%)	−0.013	(95%)	−0.005	(93%)	0.001	(90%)	0.002	(90%)
	PR	−0.002	(100%)	0.089	(87%)	0.191	(68%)	0.249	(62%)	0.293	(52%)	0.418	(35%)
	Near-Root	0.037	(90%)	0.309	(61%)	0.342	(40%)	0.344	(38%)	0.346	(36%)	0.376	(35%)
	LP	0.013	(86%)	0.219	(43%)	0.417	(20%)	0.53	(16%)	0.601	(11%)	0.83	(5%)
`.uk`	Random	0.002	(100%)	0.023	(85%)	0.044	(85%)	0.089	(93%)	0.054	(68%)	0.035	(49%)
	Degree	0.015	(98%)	0.013	(96%)	−0.043	(75%)	−0.031	(78%)	−0.019	(80%)	0.001	(74%)
	PR	0.032	(89%)	0.076	(80%)	0.125	(66%)	0.149	(59%)	0.173	(52%)	0.267	(39%)
	Near-Root	0.054	(80%)	0.261	(54%)	0.286	(48%)	0.297	(45%)	0.311	(44%)	0.387	(41%)
	LP	0.059	(87%)	0.235	(38%)	0.303	(22%)	0.394	(19%)	0.445	(14%)	0.505	(6%)

Table 4.1: For each graph and a sample of fractions of removed arcs we show the change in average distance (by the measure δ defined in Section 4.3.2) and the percentage of reachable pairs. PR stands for PageRank, and LP for label propagation.

aolpictures.aol.co.uk/	www.direct.gov.uk/en/index.htm
www.epsrc.ac.uk/	www.direct.gov.uk/
allgirltogaparty.co.uk/resources.html	www.redhotchilli.co.uk/
booth.lse.ac.uk/	www.escortmatch.co.uk/
www.cornwall.ac.uk/	www.247partypeople.co.uk/login.asp
www.nwleicsdc.gov.uk/home/	www.names.co.uk/
www.axcis.co.uk/	www.kelkoo.co.uk/
www.access-programmers.co.uk/forums/...	www.rottweiler.co.uk/forums/lofiversion/index.php
www.indiesoc.co.uk/	www.kelkoo.co.uk/b/a/sm_site-map.html...
www.aspandjavascript.co.uk/	www.kelkoo.co.uk/b/a/kc_top_searches_charts.html

Table 4.2: A comparison of the first ten URLs of the `.uk` snapshot by label-propagation rank (left) and PageRank (right). The two rankings are completely uncorrelated (Kendall's τ is ≈ -0.002).

Of course, we cannot state that there is no strategy able to disrupt social networks as much as a web graph (simply because this strategy may be different from the ones that we considered), but the fact all strategies work very similarly in both cases (e.g., label propagation is by far the most disruptive strategy) suggests that the phenomenon is intrinsic.

There is of course a candidate easy explanation: shortest paths in web graphs pass frequently through home pages, which are linked more than other pages. But this explanation does not take into account the fact that clustering by label propagation is significantly more effective than the near-root removal strategy. Rather, it appears that there are fundamental hubs (not necessarily home pages) which act as shortcuts and through which a large number of shortest paths pass. Label propagation is able to identify such hubs, and their removal results in an almost disconnected graph and in a very significant increase in average distance.

These hubs are not necessarily of high outdegree: quite the opposite, rather, is true. The behaviour of web graphs under the largest-degree strategy is illuminating: we obtain the smallest reduction in reachable pairs and an almost unnoticeable change of the average distance, which means that nodes of high outdegree are not actually relevant for the global structure of the network.

Social networks are much more resistant to node removal. There is not strict clustering, nor definite hubs, that can be used to eliminate or elongate shortest paths. This is not surprising, as networks emerging from social interaction are much less engineered (there is no notion of "site" or "page hierarchy", for example) than web graphs.

The second important observation is that the removal strategies based on PageRank and label propagation are always the best (with the exception of the near-root strategy for `.uk`, which is better than PageRank). This suggests that label propagation is actually

able to identify structurally important nodes in the graph—in fact, significantly better than any other method we tested.

Is the ranking provided by label propagation correlated to other rankings? Certainly not to the other rankings described here, due to the different level of disruption it produces on the network. The closest ranking with similar behaviour is PageRank, but, for instance, Kendall's τ between PageRank and ranking by label propagation on the `.uk` dataset is ≈ -0.002 (complete uncorrelation).

It is interesting to compare our results against those in the previous literature. With respect to [1], we test much larger networks. We can confirm that random removal is less effective that -based removal, but clearly the variation in diameter measured in [1] has been made on a *symmetrised* version of the web graph. Symmetrisation destroys much of the structure of the network, and it is difficult to justify (you cannot navigate links backwards). We have evaluated our experiment using the variation in diameter instead of the variation in average distance (not shown here), but the results are definitely inconclusive. The behaviour is wildly different even between graphs of the same type, and shows no clear trend. This was expected, as the diameter is defined by a maximisation property, so it is very unstable.

We also evaluated the variation in harmonic diameter (see Table 4.3), to compare our results with those of [31]. The harmonic diameter is very interesting, as it combines reachability and distance. The data confirm what we already stated: web graphs react to removal of 30% of their arcs by label propagation by increasing their harmonic diameter by an order of magnitude—something that does not happen with social networks. Table 4.3 is even more striking than Table 4.1 in showing that label propagation selects highly disruptive nodes in web graphs.

Our criterion for node elimination is a threshold on the number of *arcs* removed, rather than nodes, so it is not possible to compare our results with [31] directly. However, for `.uk` PageRank at $\theta = 0.01$ removes 648 nodes, which produced in the `.ie` graph a relative increment of 100%, whereas we find 14%. This is to be expected, due to the very small size of the dataset used in [31]: experience shows that connectedness phenomena in web graphs are very different in the "below ten million nodes" region. Nonetheless, the growth trend is visibile in both cases. However, the experiments in [31] fail to detect both the disruptive behaviour at $\theta = .3$ and the striking difference in behaviour between largest-degree and PageRank strategy.

Graph	Strategy	0.01		0.05		0.1		0.15		0.2		0.3	
Amazon	RND	−0.01	(100%)	0.03	(93%)	0.13	(82%)	0.13	(79%)	0.13	(76%)	0.14	(70%)
	DEG	−0.15	(118%)	0	(97%)	0.09	(86%)	0.05	(87%)	−0.05	(95%)	0.1	(80%)
	PR	0.03	(97%)	0.02	(99%)	0.02	(98%)	0.06	(94%)	0.06	(94%)	0.23	(80%)
	LP	−0.04	(104%)	−0.04	(104%)	0.2	(82%)	0.15	(87%)	0.19	(82%)	0.47	(64%)
Enron	RND	0.01	(99%)	0.04	(93%)	0.11	(83%)	0.12	(80%)	0.16	(76%)	0.05	(88%)
	DEG	0.03	(97%)	0.19	(86%)	0.41	(75%)	0.47	(72%)	0.59	(67%)	1.17	(52%)
	PR	0.01	(99%)	0.27	(81%)	0.67	(63%)	0.99	(53%)	1.38	(45%)	2.27	(34%)
	LP	0.01	(99%)	0.18	(80%)	0.29	(72%)	0.53	(59%)	0.55	(57%)	0.62	(52%)
Hollywood	RND	−0.02	(101%)	−0.07	(104%)	−0	(92%)	0.01	(87%)	0.11	(74%)	−0.02	(77%)
	DEG	0.15	(87%)	−0.04	(105%)	0.02	(98%)	0.1	(92%)	−0.09	(112%)	0.09	(93%)
	PR	−0.02	(102%)	0.06	(94%)	0.02	(100%)	0.02	(100%)	0.09	(94%)	0.14	(90%)
	LP	0.02	(90%)	−0.12	(78%)	−0.11	(65%)	−0.11	(57%)	−0.12	(54%)	−0.15	(52%)
LiveJournal	RND	0.05	(97%)	−0.01	(94%)	0.05	(89%)	−0.02	(92%)	0.06	(84%)	0.13	(78%)
	DEG	−0.03	(95%)	0.12	(91%)	0.08	(105%)	0.01	(108%)	−0.07	(92%)	0.21	(91%)
	PR	0.04	(97%)	0	(102%)	0.11	(99%)	0.18	(112%)	0.12	(96%)	0.23	(99%)
	LP	−0.06	(102%)	0.04	(103%)	0.04	(90%)	0.03	(89%)	−0.02	(98%)	0.15	(93%)
`.it`	RND	0.04	(94%)	0.1	(89%)	0.17	(75%)	0.32	(67%)	0.45	(58%)	0.69	(46%)
	DEG	0.03	(101%)	0.12	(94%)	0.05	(95%)	−0.1	(93%)	0.13	(90%)	0.21	(90%)
	PR	−0.02	(100%)	0.25	(87%)	0.72	(68%)	1.05	(62%)	1.52	(52%)	3.17	(35%)
	NR	0.18	(90%)	1.17	(61%)	2.15	(40%)	2.32	(38%)	2.32	(36%)	2.83	(35%)
	LP	0.18	(86%)	1.68	(43%)	4.44	(20%)	6.58	(16%)	9.68	(11%)	22.32	(5%)
`.uk`	RND	−0	(100%)	0.13	(85%)	0.12	(85%)	0	(93%)	0.28	(68%)	0.58	(49%)
	DEG	−0.02	(98%)	−0.01	(96%)	0.04	(75%)	0.28	(78%)	0.26	(80%)	0.1	(74%)
	PR	0.14	(89%)	0.33	(80%)	0.79	(66%)	0.98	(59%)	1.16	(52%)	2.19	(39%)
	NR	0.31	(80%)	1.27	(54%)	1.45	(48%)	1.37	(45%)	1.37	(44%)	1.84	(41%)
	LP	0.2	(87%)	2.02	(38%)	3.71	(22%)	5.13	(19%)	7.33	(14%)	16.61	(6%)

Table 4.3: For each graph and a sample of fractions of removed arcs we show the change in harmonic diameter (by the measure δ defined in Section 4.3.2) and the percentage of reachable pairs. PR stands for PageRank, and LP for label propagation.

Chapter 5

Arc-community detection via triangular random walks

In this chapter we will turn our attention to a completely different problem. Let us resume the findings presented in previous chapters about social networks. Clustering techniques works poorly on them, they are not able to highlight a clear cluster structure, there are not clear cut-point to separate small subsets of nodes from the graph, the average distance is very small and the distance distribution is very concentrated around its mean. All these characteristics are typical of random graphs.

Thus we can conjecture that social networks are simply more similar to random graphs than web graphs. However there exists another hypothesis which is not ruled out by our findings. Social networks spot a strongly overlapped community structure. This hypothesis is very sound, think about a typical social network like Facebook. Each node of the graph belongs to different clusters which reflects different areas of its life, this assumption is so sound that Google+ has explicitly introduced the notion of "circle". However even in these social networks we, as user, find odd that two friends of us that we know for different reasons, know each other. This can lead to a reasonable assumption, that triangles are mostly inside one specific community.

In this chapter, we propose the notion of triangular random walk as a way to unveil arc-community structure in social graphs: a triangular walk is a random process that insists differently on arcs that close a triangle. We prove that triangular walks can be used effectively, by translating them into a standard weighted random walk on the line graph, and experiment our idea to show that triangular walks are in fact very effective in determining the similarity between arcs and yield high-quality clustering.

5.1 Introduction

Complex networks and, especially, social networks often exhibit a finer internal structure where individuals interact in small subgroups (called communities or modules), based on the individuals' common interests, geographic location, political opinions etc. Understanding how such subgroups are structured and evolve in time is essential for applications like targeted advertising, viral marketing, friend suggestion etc. Social-network mining traditionally understands a community as a densely connected set of nodes that is in turn only loosely attached to the rest of the network [34]; in this view, community detection translates into finding a partition of the nodes that optimizes some quality function. Most of the literature on this topic focused on the discussion of the mutual merits of various quality functions and on the comparison of algorithms that try to optimize (in an exact or approximate way) some of those functions. It is worth noticing that we are here thinking of the clustering problem in a situation where the only available information is the (directed or undirected) graph underlying the social network, possibly with some weight on its arcs denoting the strength of that bound.

The main limit of the approach discussed above is that rarely a node is part of a single community: more often than not, communities overlap giving rise to a complex intertwining that can hardly be reflected into a node partition. For this reason, recent research (see, for example, [7, 60]) has turned its attention to the problem of finding overlapping communities, where each node can be a member of more than one module.

This idea is well motivated and neat for those (frequent) situations in which membership to multiple communities is an exception more than a rule, and most nodes belong clearly to one single communities, with a number of borderline individuals for whom membership is less straightforward. In a large number of scenarios, however, belonging to more communities is a rule, and actually the notion of community hardly applies to each single node. In those cases, it is often more sensible and interesting to individuate *communities of arcs* rather than *communities of nodes*: this shift of interest (witnessed in the most recent literature [77]) can be thought of as trying to find the reasons behind relations rather than trying to find the reason behind individuals.

This idea is clear if one thinks of social networks such as Facebook: every Facebook user has probably many interests and belongs to a multiplicity of communities; however, every friendship is probably due to one main reason (working together, being parents, having the same hobby etc.). This thought is so natural that Google+ has explicitly introduced the notion of "circle".

In this work, we propose to continue along this line of research trying to exploit the following simple observation: if xy and yz are two relations that have the same

motivation (e.g., working together), then probably xz will also be present: in other words, triangles tend to live inside communities. Based on this intuition, we propose the notion of triangular random walk, a stochastic process that treats differently triangular and non-triangular arcs; although this process is not memoryless, we can reduce it to a standard Markov chain on the line graph (using a tool similar to [29], but in a different way). With our approach, we obtain a weighted graph whose nodes correspond to the arcs of the original network, and that can in turn be clustered using standard tools. Experiments on real-world networks of different sizes and types show that triangular walks can be extremely helpful in finding meaningful communities, outperforming other approaches, usually with a negligible loss in computation time.

5.2 Triangular random walks

Given a (directed loopless) graph $G = (V_G, A_G)$, we let $n_G = |V_G|$ and $m_G = |A_G|$ be the number of nodes and arcs of G, respectively; for every node x we let $N_G(x)$ be the set of *successors* of x (that is, $\{y \mid (x, y) \in A\}$) and $d_G(x) = |N_G(x)|$ (the *(out)degree* of x). If G is symmetric (i.e., if $(x, y) \in A_G$ implies $(y, x) \in A_G$), we treat G as if it was undirected; in this case, we use the term *edge* to refer to an unordered pair of nodes that are connected by an arc. We sometimes write xy to denote the arc (x, y) (or the edge $\{(x, y), (y, x)\}$, if the graph is undirected). The subscript G will be omitted when it is clear from the context.

A *random walk* on a directed graph G is a stochastic process $X_0, X_1, \ldots$ where $X_0, \cdots \in V$, and for each $x, y \in V$, $P[X_0 = x] = 1/n$ and $P[X_{t+1} = y \mid X_t = x]$ is $1/d(x)$ if $y \in N(x)$, 0 otherwise[1]; this definition can be easily extended to weighted graphs (making $P[X_{t+1} = y \mid X_t = x]$ proportional to the weight of (x, y)). Intuitively, a random walk describes the behavior of a surfer walking on the graph, who starts from a random node and at each step chooses uniformly at random among the successors of the current node (jumping to a random node if the current one has no successors).

The random walk is a Markov chain and if G is undirected, connected and not bipartite, then the random walk has a unique stationary distribution $\mathbf{v}$ with $v_x = d(x)/2m$ [69]. For a general graph, however, the random walk is not ergodic, hence the stationary distribution may not be unique; to circumvent this problem, one can introduce [11, 47, 73] the notion of restart.

For a fixed $\alpha \in [0, 1]$, a *random walk with restart with damping factor* α on G is a stochastic process $X_0, X_1, \ldots$ as before, but where the surfer chooses the next node as follows: with probability α she picks a node uniformly at random among the successors

[1]For the sake of completeness, when $d(x) = 0$ we let $P[X_{t+1} = y \mid X_t = x] = 1/n$ for all y.

of the current node; with probability $1 - \alpha$, instead, she jumps to a random node in the graph[2]. The latter event is called *teleportation* or "restart". It can be shown [11] that for all $\alpha < 1$ the random walk with restart has a unique stationary distribution (actually, the PageRank of G with damping factor α); when $\alpha = 1$ we get back to the standard random walk.

One suggestive way to think of this random process is the following: a random surfer is trying to collect some knowledge and every node represents an expert that may provide some piece of information. After the surfer has finished visiting expert x she receives a list of other possible people that x trusts; the surfer may decide (with probability α) to accept x's suggestion and to visit one of them, or may rather decide to do it her way and to teleport to a random expert instead.

It is interesting to observe that one may also actually consider the stationary distribution *on the arcs of* G: the probability $P[X_t = x, X_{t+1} = y]$ that the random surfer goes along the arc (x, y) is $P[X_{t+1} = y \mid X_t = x]P[X_t = x] \propto v_x w(x, y)$, where $\mathbf{v}$ is the stationary distribution on the nodes and $w(x, y)$ is the weight on the arc (x, y) (that is, $1/d(x)$ in the unweighted case); the proportionality constant serves to take teleporting into account. We will refer to this distribution as the *arc-stationary distribution*.

The main idea of this chapter is that we want to introduce a bias in the behavior of the random surfer, by allowing her some amount of short-term memory; in particular, the choice of the next node will not depend only on the current node but *also on the previous one*. The bias is finalized to privilege (or punish) triangles, i.e., suggestions of the current node that were also suggested by the previous node. Whether we decide to privilege triangles or to punish them depends on our interpretation of triangles: if we think that the double suggestion reinforces the idea that the suggested node is reliable, we will privilege triangles; if otherwise we suspect that the double suggestion is rather a form of lobbying, we will tend to avoid triangles.

Thus, we will define a triangular random walk $X_0, X_1, \ldots$ on the graph using two parameters, $\alpha, \beta \in [0, 1]$: α is a damping factor and will have the same meaning as before (it is used to decide whether to follow a link or to teleport); β will instead be used to determine whether triangles or non-triangles should be privileged.

Two subtly different definitions of triangular random walks can be given, depending on the specific meaning of β: we will call them mass-triangular and ratio-triangular, respectively. In a triangular random walk with parameters α and β, the next node (x_{t+1}) is chosen depending on the current node x_t and on the previous node x_{t-1}, as follows:

- with probability $1 - \alpha$, we teleport: x_{t+1} is a randomly chosen node;

[2] As before, if the current node has no successors then the next node is chosen at random.

- otherwise, we choose among the successors $N(x_t)$ of the current node, but treating differently the *triangular successors* (the set $N(x_t) \cap N(x_{t-1})$) and the *non-triangular successors* (the set $N(x_t) \setminus N(x_{t-1})$)[3]; here, the two definitions differ:
  - in the *(mass-)triangular random walk*, we first decide whether we shall select a non-triangular successor (with probability β) or a triangular one (with probability $1 - \beta$); then, the specific non-triangular or triangular successor is chosen uniformly at random;
  - in the *ratio-triangular random walk*, all triangular successors are selected with the same probability, say p, and all non-triangular successors with probability βp (p should be chosen so that the sum of such probabilities is 1).

The names we adopted for the two kinds of random walks should be evocative of the meaning of β: in the mass-triangular random walk, β is the overall amount of probability of choosing a non-triangular successor; in the ratio-triangular random walk, it is the ratio between the probability of choosing a(ny) non-triangular successor over the probability of choosing a(ny) triangular one.

The two kinds of processes coincide when $\beta = 0$ (in that case, they both only choose triangular successors, except when teleporting). Moreover, ratio-triangular random walks reduce to standard random walks with restart when $\beta = 1$ (because, in that case, the probability of choosing triangles and non-triangles is the same), whereas there is no choice of β that makes a mass-triangular random walk the same as a standard random walk.

The latter observation may suggest that ratio-triangular walks should be preferred, but the mathematical treatment of mass-triangular walks is simpler, and for this reason we shall actually treat the latter as our "default" type of triangular walk (and omit "mass" in the following). We will get back to the similarities and differences between the two definitions in Section 5.5. Triangular walks can have a number of potential applications; for example, they may be used fruitfully in bibliometry to moderate the problem of nepotistic citations in scientific works (in this case, triangles should be punished rather than promoted). In this thesis, however, we wish to speculate on the possible usage of triangular walks to single out arc-communities in social networks.

To start playing with our idea, let us consider Zachary's famous karate club network [78]: this is an undirected graph whose nodes represent the members of a karate club and with an edge between two individuals if they happened to have seen each other

[3]If either set is empty (or if $t = 1$) we choose uniformly in $N(x_t)$ (or in V, if the latter is empty), as in a standard random walk.

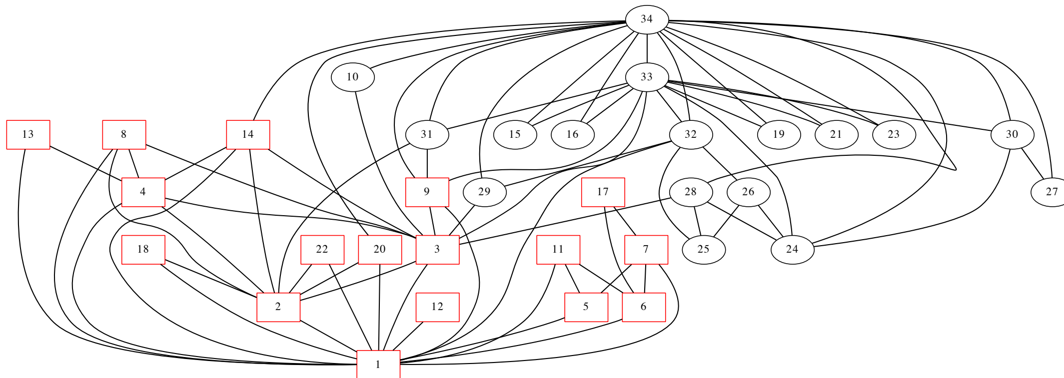

Figure 5.1: Standard random walk on the karate club dataset; edge width is proportional to the frequency with which that edge was run through in either direction.

outside of the club for some reason; the club ended up splitting in two (in our drawings, the nodes are depicted differently according to the group they will end up in), and one can hope to find information about how the members will decide to group based solely on their friendship relations.

We first tried a standard random walk on this dataset to see how frequently each edge was run through in either direction (Figure 5.1): no pattern is evident. But if we do the same with a triangular walk some edge gets more emphasis, witnessing that some bounds are stronger than others (Figure 5.2, with $\beta = 0.2$): those edges are usually between members that will end up in the same group (with an exception concerning node 9 that indeed seems to be more strictly bound to the group of circles than to the group of squares). If we decrease β to 0.01, some clans become almost grotesquely evident (Figure 5.3).

5.2.1 Triangular walks and line graphs

A triangular random walk is a Markov chain of order 2 [69], because the next state depends on the current state *and* on the previous one. To study the long-term behavior of higher order chains, it is customary to change the state space and reduce the stochastic process to an equivalent one that is memoryless; this is easily solved by using the notion of *line graph*.

Given a graph G, its line graph $L = L(G)$ has the arcs of G as vertices (i.e., $V_L = A_G$), with arcs of the form (xy, yz) (where xy and yz are two arcs of G). Note that even when G is symmetric, $L(G)$ is not; for example, if G is the undirected graph in Figure 5.4, its corresponding line graph $L(G)$ is represented in Figure 5.5 (for the time being, ignore

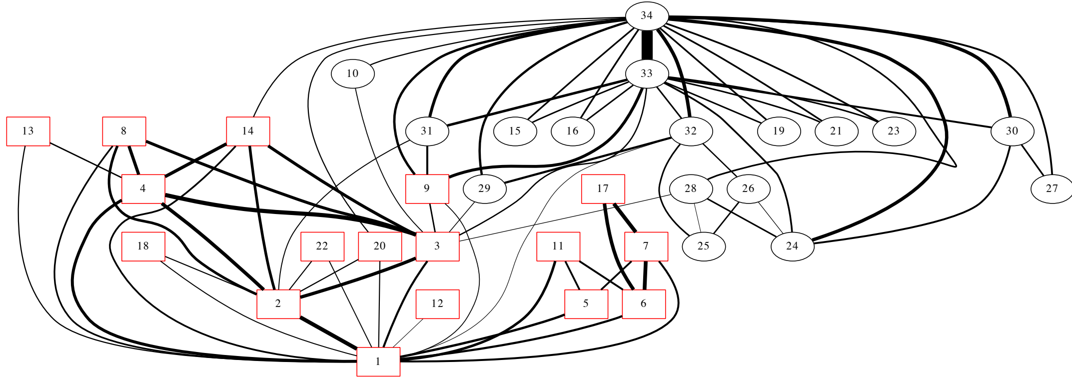

Figure 5.2: Triangular random walk on the karate club dataset, with $\beta = 0.2$ (see also Figure 5.1).

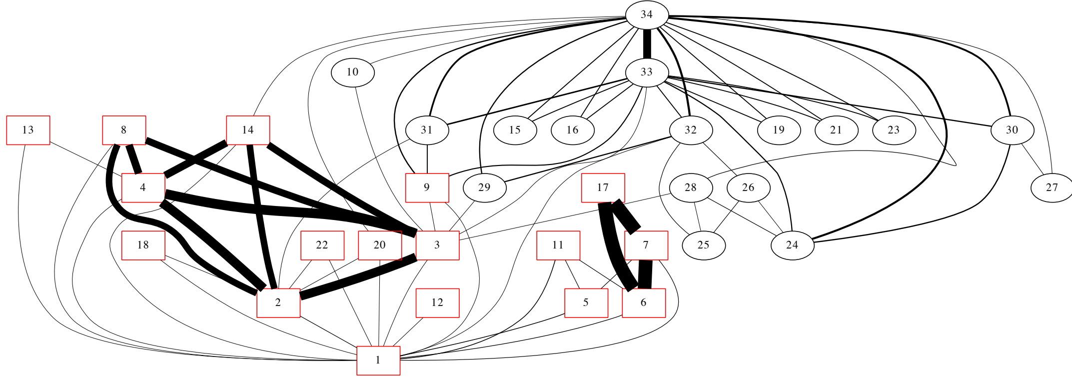

Figure 5.3: Triangular random walk on the karate club dataset, with $\beta = 0.01$. (see also Figure 5.1).

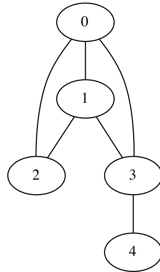

Figure 5.4: A small undirected graph G.

the colors on its arcs).

The idea of using line graphs to study the behavior of an arc-aware random surfer was already proposed in [29], but they adopt a subtly different notion of line graph that is undirected; for our purposes, instead, the directed definition is much more well-suited (also because it adapts readily to the case when the original graph is itself directed).

Now, it is easy to see that a triangular random walk with parameters α, β on the (unweighted) graph G is equivalent to a random walk with damping factor α on the weighted line graph $L(G)$, where

$$w_T(xy, yz) = \begin{cases} \frac{\beta}{|N(y)\setminus N(x)|} & \text{if } z \in N(y) \setminus N(x) \\ \frac{1-\beta}{|N(y)\cap N(x)|} & \text{if } z \in N(y) \cap N(x). \end{cases}$$

In other words, every arc in $L(G)$ (that is to say, every two-step walk $x \to y \to z$ in the original graph) has a different weight depending on whether it can be closed by a triangle (i.e., if $x \to z$ was also an arc of G) or not. If you look again at Figure 5.5, continuous (red) arcs correspond to the first case (e.g., $10 \to 03$ is one such arc, because 13 is also an arc of G), whereas dashed (black) arcs correspond to the second case (e.g., $31 \to 12$); note, in particular, that all arcs of the form $xy \to yx$ fall in the second class[4]. Some nodes of $L(G)$ (i.e., arcs of G) require some attention, because their outgoing arcs are all non-triangular; those outgoing arcs are hence not weighted using the formula above (it does not make sense since one of the denominators is zero), but they have a constant weight instead (such arcs are drawn as dotted (blue) arrows in Figure 5.5).

For $\alpha < 1$ the random walk with restart on $L(G)$ weighted by w_T has a stationary distribution $\mathbf{v}_T$: note that, since the nodes of $L(G)$ are arcs of G, $\mathbf{v}_T$ assigns a probability

[4]Differently from [29], we do not reserve stuttering walks (walks of the form $x \to y \to x$) a special treatment.

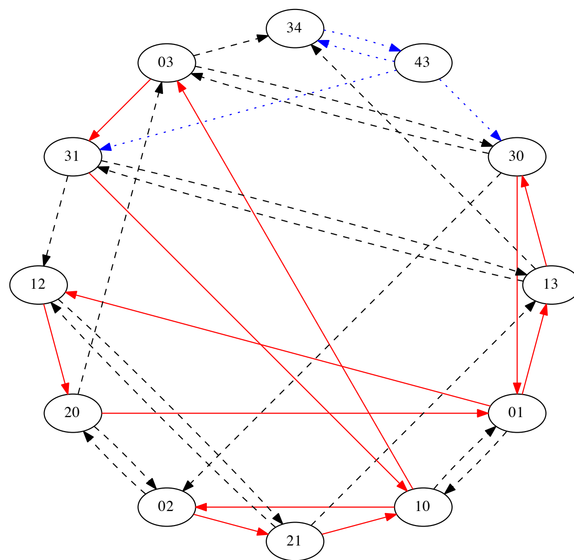

Figure 5.5: The line graph $L(G)$, where G is depicted in Figure 5.4. Continuous (red) arcs correspond to choosing triangular successors; dashed (black) arcs correspond to the choice of non-triangular successors; dotted (blue) arcs are used for the cases where either set is empty.

$v_T(xy)$ with each arc xy of the original graph. Note also that, as explained in the previous section, the stationary distribution on the nodes of $L(G)$ induces a stationary distribution on its arcs: $v_T(xy, yz) \propto v_T(xy)w_T(xy, yz)$: this is the probability that the random surfer runs through the path $x \to y \to z$.

Computing the stationary distribution $\mathbf{v}_T$ is a well-understood task (it amounts to a weighted version of PageRank) for which efficient and computationally sound algorithms exist [47, 76]; of course, $L(G)$ is larger than G (it has m_G nodes and $\sum_x d_G(x)^2$ arcs), but not much larger actually because of the sparsity of G and of the way its degrees are distributed. In particular, if G is undirected and has $\approx Ck^{-\alpha}$ nodes of degree k, then $L(G)$ will have $\approx C^2k^{-2\alpha}$ nodes of outdegree k.

5.3 Arc-clustering via triangular random walks

As outlined in the previous sections, along the same line as [29], instead of clustering directly the arcs of G (as done, for example, by [42]), we turn to some suitably weighted version of the line graph $L(G)$, where we can make good use of all the paraphernalia for node-clustering of a directed graph. In other words, we shall use an off-the-shelf node-clustering algorithm feeding it with the weighted graph $L(G)$. As weighting function (on the arcs of $L(G)$), we can either use the weights $w_T(xy, yz)$ that define the transition probabilities of a triangular surfer or, alternatively, we can exploit its arc-stationary distribution $v_T(xy, yz) = v_T(xy)w_T(xy, yz)$ (where, as explained above, $v_T(xy)$ is the stationary distribution of the triangular random surfer on the node xy).

For comparison, we may consider the weights of a standard random walk $w_S(xy, yz) = 1/d(y)$ or the corresponding arc stationary distribution $v_S(xy, yz) = v_S(xy)w_S(xy, yz)$ (as before, $v_S(xy)$ is the stationary distribution of the standard random surfer on the node xy); here, the subscript "S" stands for "standard". Another baseline is to feed the clustering algorithm with the unweighted graph $L(G)$ itself.

The proposed method is tailored around directed graphs and parallel opposite arcs may end up in two different communities; the main limit of this approach is that it cannot be directly applied to truly undirected graphs. In cases when this is really necessary, one has to decide what to do if two opposite arcs happen to be clustered differently — one possible solution is to place the corresponding edge in either community, or to use a special community that corresponds to the given pair.

Computational issues Computing the line graph $L(G)$ and its weights w_T is straightforward and can be performed in time $O(m_{L(G)})$ (i.e., linear in the output size), provided that one has direct access to G; moreover, although their size is obviously larger than

	n_G	$m_G = n_{L(G)}$	$m_{L(G)}$
free word association	10 225	71 679	955 552
DBLP	986 324	6 707 236	211 808 396
Hollywood	2 180 759	228 985 632	242 026 293 162

Table 5.1: Size of line graphs for some of the datasets we shall use in Section 5.5; observe that Hollywood is comparatively denser than the other graphs (with an average degree of about 105), which is why the number of arcs in $L(G)$ is so large (the average degree is in this case 1 057).

the original graph (see Table 5.1), line graphs turn out to be easily compressible (about 2 to 3 bits/link in their natural order, much less if suitably permuted [12]). After $L(G)$ has been produced, weighted PageRank can be computed very quickly (using for example the techniques of [26]), and always converges in less than 100 iterations even for $\alpha = 1 - 10^{-2}$ (the value used in most of our clustering experiments). The final node-clustering phase clearly depends on the algorithm used, but our method of choice [9] turns out to be extremely fast — actually, the line graph construction is by far the most expensive step. In fact, the explicit construction of the line graph is the main limit of our approach, especially for networks that are comparatively denser (such as Hollywood); we will get back to this problem later.

5.4 Related works

Although node-clustering is traditionally much more developed and better understood (see [68] for an up-to-date survey), recently many authors advocated the adoption of link-clustering [77, 29, 42] as a way to overcome the problem of overlapping communities in complex networks. The advantage of this approach over the solution of soft or hierarchical node-clustering [50, 40] is that the latter is better suited for situations where the presence of a node in many communities is an exception rather than a rule; on the contrary, using link-clustering allows one to give multiple membership a more understandable meaning in the common situations when every single node is likely to belong to more than one cluster but each node-to-node relation can be explained as co-affiliation to some community (like in the well-known model of affiliation networks [48]). Of course, even in the latter situation co-affiliation can be due to many reasons (co-affiliation to many communities), but also in that situation in many cases one reason prevails.

The usage of line graphs to model link-clustering is especially promoted by Evans and

Lambiotte [29] (who also take into consideration notions of weighting that deal with the problem of over-representing high-degree nodes), but they exploit the undirected version of line graphs instead of the directed one [38], and they do not distinguish between triangular and non-triangular arcs.

As explained, our technique relies on some external node-clustering algorithm that uses a weighted version of $L(G)$, with the hope that triangular random walks highlight clear cuts between communities as they should. To test our hypothesis, we obviously need a clustering algorithm that can handle large weighted directed graphs; we tried three different clustering algorithms which satisfy our requirements and are considered the state of the art for massive complex networks: clustering via Potts' model as proposed in [65], the hierarchical Infomap algorithm presented in [66] and the Louvain method [9]. In our tests the latter proved to be the fastest among these candidates and produces also the best results in term of accuracy, so we will adopt it in our experiments. Actually, however, all the tested methods improve their performance on the versions of $L(G)$ that were weighted according to our criterion.

5.5 Experiments

The experiments that we are going to describe have been run using public datasets and relying heavily on the WebGraph [17] framework (in particular, the line-graph transformation was implemented as a part of it). The remaining tools will be made available as "Satellite Software" in the `law.dsi.unimi.it` website.

Triangular walks on DBLP For this set of experiments, we worked on the DBLP graph[5]; DBLP is a scientific collaboration network where each vertex represents a scientist and two vertices are connected if they have worked together on an article. The current version (July 2011) of the DBLP dataset contains 986 324 authors and 2 684 847 publications, giving rise to 3 353 618 co-authorship edges. This network corresponds to the typical situation in which every author can belong to more than one scientific community (because typically, during their life, scientists work on many different and often scarcely related topics), but collaborations usually correspond to a specific topic. Based on this interpretation, we labelled each edge of DBLP with the concatenation of all titles of the co-authored papers, and the similarity between two edges is computed as the cosine distance between the corresponding term vectors (we normalized the words though a Porter's stemmer and used TF-IDF [5] for term weighting).

[5] `http://www.informatik.uni-trier.de/~ley/db/`

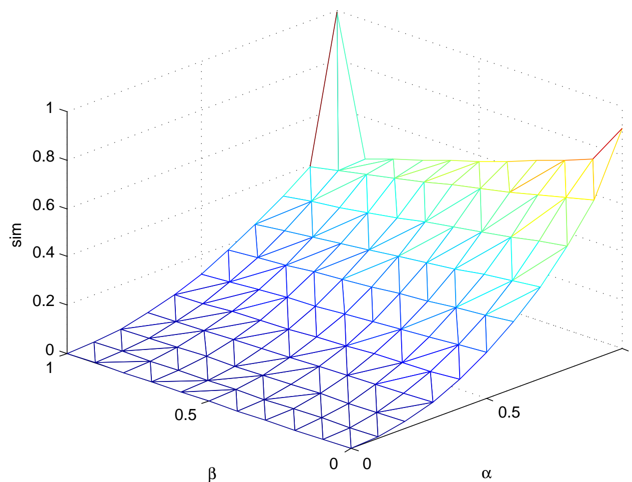

Figure 5.6: The average similarity in a triangular random walk of length 1 000 000 on the DBLP graph, as a function of α and β.

We then performed some triangular random walks on the resulting graph (for different values of the parameters α and β) and every time the walk passed over two consecutive arcs we computed the similarity between those arcs. The average similarity is plotted in Figure 5.6.

As the reader can see, for all α the average similarity usually increases when β is decreased, witnessing the (expected) phenomenon that following triangles more often leads to better average similarity. The peak at $\alpha = \beta = 1$ is explained as follows: in that situation, we never teleport (because $\alpha = 1$) and we never follow triangles, which more often than not makes the surfer perform a stuttering walk (after passing on the arc xy the surfer gets back along yx, and the two arcs correspond to the same edge and of course have similarity 1). More in general, when α is small (i.e., when we teleport often) or when β is too close to 0 or 1 (i.e., when we follow only triangles or only non-triangles), the percentage of arcs visited in the random walk will be small (as one can see from Figure 5.7).

To keep this phenomenon into account, we prefer to use a different measure, that we call *discounted average similarity*: we multiply the average similarity by the percentage of arcs discovered during the random walk, obtaining what is shown in Figure 5.8.

This picture suggests that α close to 1 (almost no teleportation) can be adopted giving a reasonable tradeoff between coverage and similarity, provided that β is not too close to 0 or 1. For this reason, the remaining experiments have been run only with $\alpha = 1$. The results are shown in Figure 5.9, 5.10 and 5.11: we once again computed

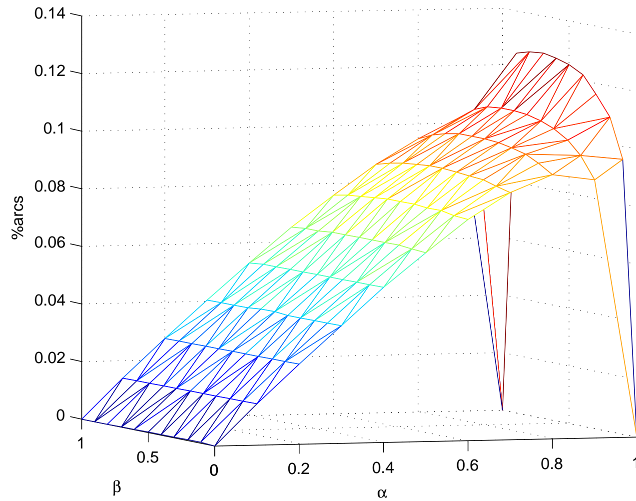

Figure 5.7: The percentage of arcs that have been visited in a triangular random walk of length 1 000 000 on the DBLP graph, as a function of α and β.

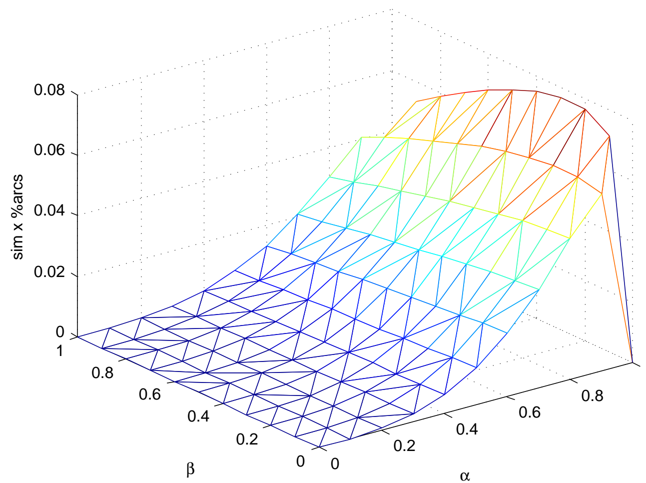

Figure 5.8: The discounted average similarity in a triangular random walk of length 1 000 000 on the DBLP graph, as a function of α and β.

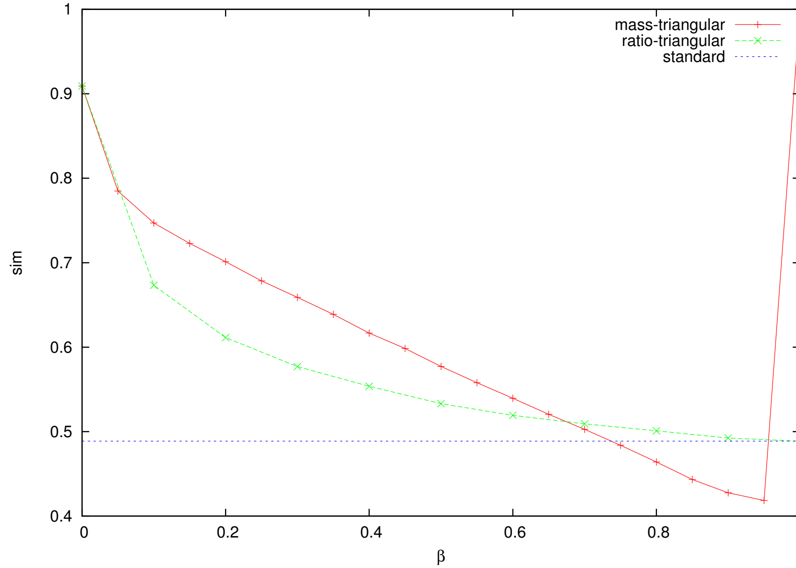

Figure 5.9: The average similarity in a mass-triangular and in a ratio-triangular random walk of length 1 000 000 on the DBLP graph (with $\alpha = 1$) as a function of β. For comparison, also the results obtained with a standard random walk (horizontal line) is shown.

the average similarity, percentage of visited arcs and discounted average similarity as β ranges between 0 and 1, but this time we also show the behavior of a ratio-triangular walk and of a standard random walk (the same as a ratio-triangular walk with $\beta = 1$). Some remarks are in order:

- triangular walks obviously discover less arcs, because they tend to be trapped in specific areas of the graph; when $\beta = 0$, both types of triangular walks remain in the small cliques where they started from, whereas the mass-triangular version tends to stutter when $\beta = 1$, as explained above;
- similarity is largely improved by reducing β (i.e., when triangles are privileged): a standard walk accumulates an average similarity of about 0.488, whereas a mass-triangular walk with $\beta = 0.25$ reaches 0.679, an improvement of about 39%;
- discounted similarity behavior is characteristic, with a maximum around $\beta = 0.25$.

Triangular walks on Hollywood Performing the same experiments with other walk lengths and other networks give the same qualitative behavior, with an optimal dis-

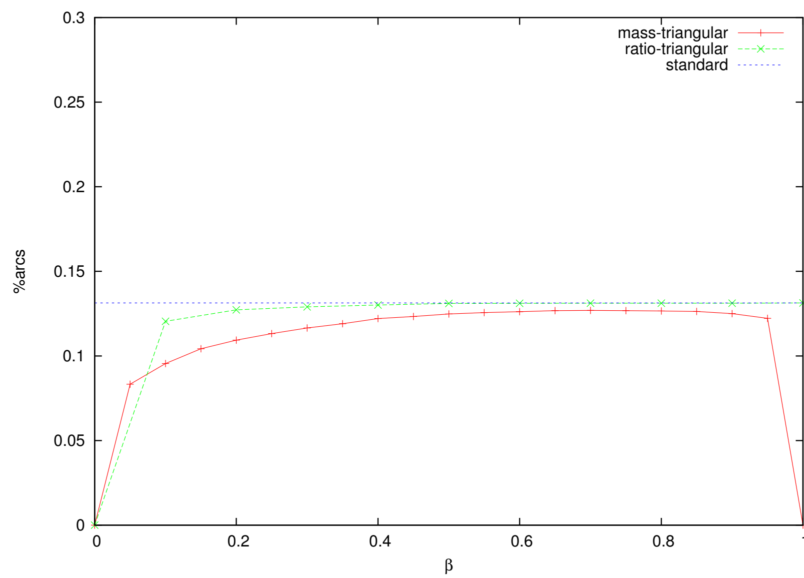

Figure 5.10: The fraction of arcs discovered in a mass-triangular and in a ratio-triangular random walk of length 1 000 000 on the DBLP graph (with $\alpha = 1$) as a function of β. For comparison, also the results obtained with a standard random walk (horizontal line) is shown.

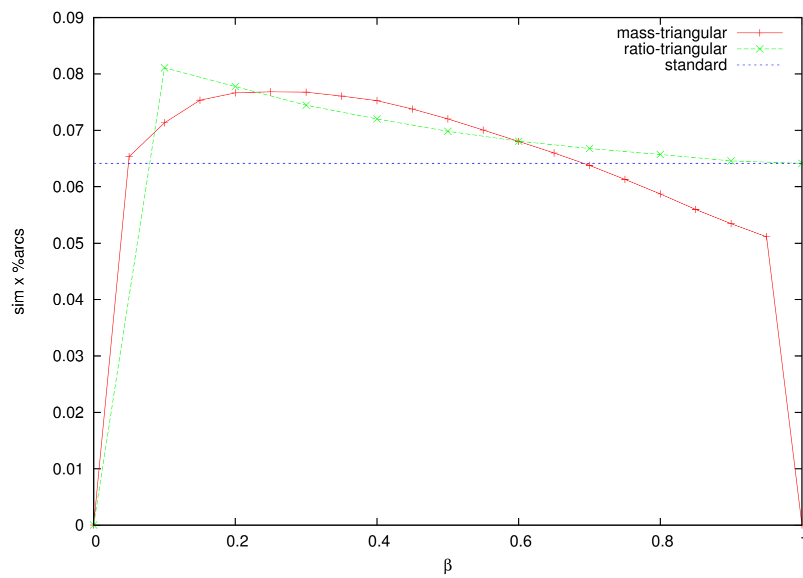

Figure 5.11: The discounted average similarity in a in a mass-triangular and in a ratio-triangular random walk of length 1 000 000 on the DBLP graph (with $\alpha = 1$) as a function of β. For comparison, also the results obtained with a standard random walk (horizontal line) is shown.

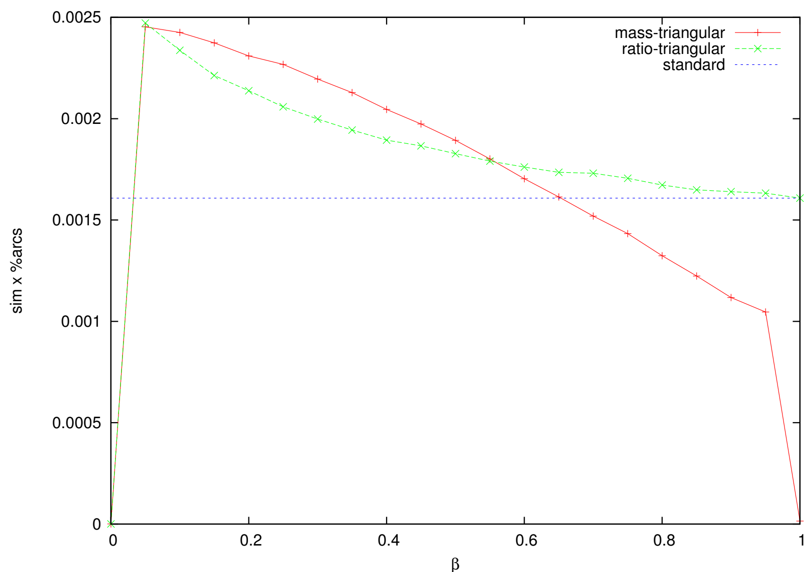

Figure 5.12: The discounted average similarity in a in a mass-triangular and in a ratio-triangular random walk of length 1 000 000 on the Hollywood graph (with $\alpha = 1$) as a function of β. For comparison, also the results obtained with a standard random walk (horizontal line) is shown.

counted similarity that is typically between 0.1 and 0.3. In particular, we performed the same experiments on the Hollywood graph obtained from the Internet Movie Database[6]; this undirected graph has, in its current version (July 2011), 2 180 759 nodes (actors and actresses) and 114 492 816 edges corresponding to having acted together in some movie. Here the edge xy is labelled with the multiset of directors that directed the movies co-acted by x and y, with the interpretation that a specific actor may have worked in many different movies, but directors tend often to collaborate with the same set of "trusted" actors. Similarity between arcs is once again computed using TF-IDF (here, the vocabulary is made by director IDs).

The results obtained for this dataset are absolutely similar to the ones presented for DBLP, so we limit ourselves to show, in Figure 5.12, the discounted average similarity when $\alpha = 1$ and with β ranging in $[0, 1]$.

Clustering on DBLP For this set of experiments, we considered again the DBLP graph G and clustered its arcs in various ways (see below). To determine the quality

[6] `http://www.imdb.com/`.

of the clusterings obtained, we used again the ground truth at our disposal (i.e., the fact that we know how similar two given arcs are supposed to be) and proceeded as follows: we randomly sampled a large number of pairs of arcs, and computed separately the average similarity of the pairs that happened to belong to the same cluster and the average similarity of the pairs that did not belong to the same cluster; the ratio between these two quantities is used as a measure of quality — a good clustering should provide a ratio larger than 1. Three sampling techniques were considered: *uniform arc sampling* (the two arcs are sampled uniformly at random among all arcs); *uniform node sampling* (a node of degree larger than one is sampled uniformly at random, and two of its incident arcs are chosen again at random); *degree node sampling* (a node is sampled with probability proportional to its degree, and two of its incident arcs are chosen).

We compare the results obtained by applying the Louvain method to various weighted versions of $L(G)$; for comparison, we also tried to cluster the arcs using the system proposed by [29] (that uses the undirected version of the link graph) and *LINK*, a link clustering technique proposed in [77][7]. We also tried to cluster the arcs indirectly, through some of the best node clustering techniques; we transform a node clustering into an arc clustering with the following strategy: since a node clustering algorithm produces a labeling function $f : V_G \to \mathbb{N}$, we map each arc xy to the pair $(f(x), f(y)) \in \mathbb{N}^2$, and use the latter as arc label. If the original graph is symmetric, we can forget about the order of labels and assign an unique identifier to each unordered pair of labels.

In Table 5.2 you can see the results of this experiment; the Louvain method on $L(G)$ invariably produces less communities than all other methods, although triangular weights tend to create more clusters. Since our community-detection algorithm aims at capturing local communities more than global ones, our performance deteriorates on uniformly sampled arcs (but it is still much better than all the other approaches); locally, however, we perform better (in some cases, much better) than the other techniques. The second best method is certainly [29]. As far as the difference between the two types of weights, the gain in using the arc-stationary state instead of the simple triangular weights is small, but PageRank computation is so fast that the effort is anyway worth.

Clustering of the word association network For these experiments, we considered the Free Word Association network [56]; this is a directed graph describing the results of an experiment of free word association performed by more than 6 000 participants in the United States: its nodes correspond to words and arcs represent a cue-target pair (the arc xy means that the word y was output by some of the participants based

[7] We also experimented with the software described in [42], but could not have it work on networks of more than about 100 nodes.

		no. of clusters	Arc sampling method		
			Unif. node	Deg. node	Unif. arc
Louvain [9]	$\mathbf{v_T}$	529	**17.19**	**15.40**	**2.14**
	$\mathbf{w_T}$	569	**16.66**	**14.84**	**2.15**
	v_S	231	1.01	1.25	1.42
	w_S	229	2.69	4.16	1.42
	-	247	1.32	1.42	1.54
Evans et al. [29]	-	231	5.70	5.94	1.44
LINK [77]	-	630	1.04	2.61	1.41
Infomap [66]	-	62680	1.80	0.74	1.10
Louvain (on G) [9]	-	6414	1.67	2.98	1.45

Table 5.2: Clustering quality obtained using different techniques on the DBLP graph (in boldface, the two triangular weights suggested in this chapter, using $\alpha = 0.99$ and $\beta = 0.2$; see Section 5.3). The quality is measured as the ratio between the average similarity of pairs of arcs in the same cluster vs. arcs in different clusters, using different arc sampling methods. The upper group refers to the application of the Louvain algorithm to various (weighted or unweighted) versions of $L(G)$; the middle group consists of algorithms that produce an arc-clustering on G; the bottom group, instead, produce a node-clustering on G, that we interpret as an arc-clustering.

on the stimulus x). This graph contains 10 225 words and 71 679 associations (arcs). In Figure 5.13 you can see how the Louvain method [9] on $L(G)$ produces different clusterings (here, we show the vicinity of the words "gum" and "gums"), depending on whether it is fed with no weights or with the arc-stationary distribution of triangular walks. Observe that correctly the latter clearly distinguishes between the associations related to edible chewing gum (yellow), those that have to do with teeth and mouth (blue), those that are related to chewing (purple), with other small communities (like the one about "stickiness"). Also notice that the arcs connecting "chew" with "spit" are in different communities (the arc going towards "chew" stays together with the "chewing" community, whereas "spit" is more generic).

5.6 Conclusions

We introduced a new kind of random process that helps in singling out arc communities in social networks; this can be seen as a Markov chain on the line graph whose arc-stationary state contains a big deal of information on the communities, and can be fruitfully used to gain a more accurate and fine-grained resolution, at least at a local level. In our experiments, using this information ended up in producing more reasonable and significant clusters, with a limited computational cost.

These results are preliminary but very encouraging; we also believe that the weights proposed here can be beneficial for other types of mining tasks. Such tasks can be made reasonably scalable by exploiting the possibility (here explored with ALP) of writing implicit versions of mining algorithms that work on the weighted line graph without having to build it explicitly.

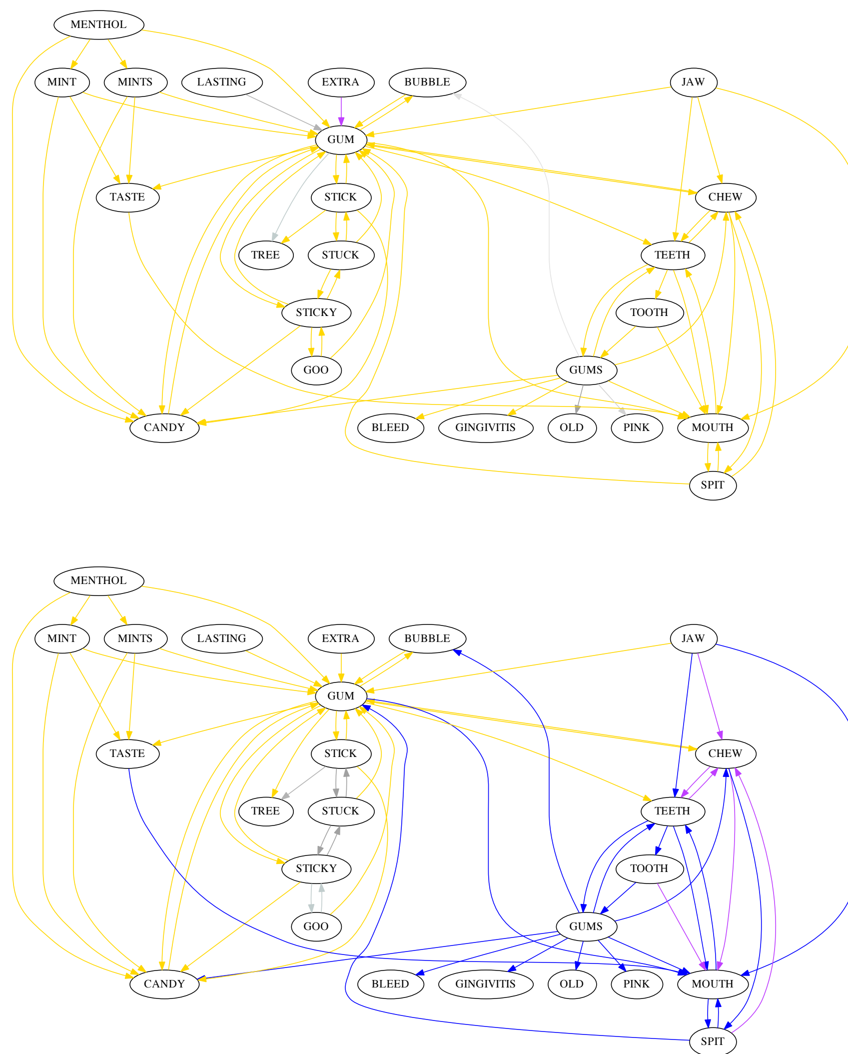

Figure 5.13: Different clusterings of the word association network, in the vicinity of the words "gum" and "gums". Both are obtained using the Louvain method on the line graph; on the left, the results with no weights (802 clusters); on the right, with the arc-stationary distribution of triangular random walks, $\beta = 0.2$ and $\alpha = 0.99$ (373 clusters)).

Bibliography

[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

[2] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci*, 58(1):137–147, 1999.

[3] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Local partitioning for directed graphs using PageRank. *Internet Math.*, 5(1):3–22, 2008.

[4] Alberto Apostolico and Guido Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[5] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[6] Michael J. Barber and John W. Clark. Detecting network communities by propagating labels under constraints. *Phys. Rev. E*, 80(2):026129, Aug 2009.

[7] Jeffrey Baumes, Mark K. Goldberg, Mukkai S. Krishnamoorthy, Malik Magdon-Ismail, and Nathan Preston. Finding communities by clustering a graph into overlapping subgraphs. In *IADIS AC'05*, pages 97–104, 2005.

[8] Krishna Bharat, Andrei Broder, Monika Henzinger, Puneet Kumar, and Suresh Venkatasubramanian. The Connectivity Server: fast access to linkage information on the Web. *Computer Networks and ISDN Systems*, 30(1-7):469–477, 1998.

[9] V.D. Blondel, J.L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, 2008.

[10] P. Boldi and S. Vigna. Codes for the world wide web. *Internet mathematics*, 2(4):407–429, 2005.

[11] Paolo Boldi, Violetta Lonati, Massimo Santini, and Sebastiano Vigna. Graph fibrations, graph isomorphism, and PageRank. *RAIRO Inform. Théor.*, 40:227–253, 2006.

[12] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM, 2011.

[13] Paolo Boldi, Marco Rosa, and Sebastiano Vigna. HyperANF: Approximating the neighbourhood function of very large graphs on a budget. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 625–634. ACM, 2011.

[14] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. PageRank: Functional dependencies. *ACM Trans. Inf. Sys.*, 27(4):1–23, 2009.

[15] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Permuting web graphs. In *WAW '09: Proceedings of the 6th International Workshop on Algorithms and Models for the Web-Graph*, pages 116–126, Berlin, Heidelberg, 2009. Springer-Verlag.

[16] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. Permuting web and social graphs. *Internet Math.*, 6(3):257–283, 2010.

[17] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference*, pages 595–601. ACM Press, 2004.

[18] S.P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55–71, 2005.

[19] S.P. Borgatti. Identifying sets of key players in a social network. *Computational & Mathematical Organization Theory*, 12(1):21–34, 2006.

[20] S.P. Borgatti, K.M. Carley, and D. Krackhardt. On the robustness of centrality measures under conditions of imperfect data. *Social Networks*, 28(2):124–136, 2006.

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

[22] U. Brandes and T. Erlebach. *Network analysis: methodological foundations*, volume 3418. Springer Verlag, 2005.

[23] Flavio Chierichetti, Ravi Kumar, Silvio Lattanzi, Michael Mitzenmacher, Alessandro Panconesi, and Prabhakar Raghavan. On compressing social networks. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 219–228, New York, NY, USA, 2009. ACM.

[24] Edith Cohen. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55:441–453, 1997.

[25] R. Cohen and S. Havlin. *Complex Networks: Structure, Robustness and Function*. Cambridge Univ Pr, 2010.

[26] Gianna M. Del Corso, Antonio Gullì, and Francesco Romani. Fast pagerank computation via a sparse linear system. *Internet Mathematics*, 2:118–130, 2004.

[27] Debora Donato, Stefano Leonardi, Stefano Millozzi, and Panayiotis Tsaparas. Mining the inner structure of the web graph. *Journal of Physics A: Mathematical and Theoretical*, 41(22):224017, 2008.

[28] Marianne Durand and Philippe Flajolet. Loglog counting of large cardinalities (extended abstract). In Giuseppe Di Battista and Uri Zwick, editors, *Algorithms - ESA 2003, 11th Annual European Symposium, Budapest, Hungary, September 16-19, 2003, Proceedings*, volume 2832 of *Lecture Notes in Computer Science*, pages 605–617. Springer, 2003.

[29] T. S. Evans and R. Lambiotte. Line graphs, link partitions, and overlapping communities. *Phys. Rev. E*, 80(1):016105, Jul 2009.

[30] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm. In *Proceedings of the 13th conference on analysis of algorithm (AofA 07)*, pages 127–146, 2007.

[31] Dániel Fogaras. Where to start browsing the web? In *Innovative Internet Community Systems, Third International Workshop, IICS 2003*, volume 2877 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2003.

[32] Santo Fortunato. Community detection in graphs. *Physics Report*, 486:75–174, February 2010.

[33] Santo Fortunato and Marc Barthelemy. Resolution limit in community detection. *Proceedings of the National Academy of Science*, 104:36–41, January 2007.

[34] Santo Fortunato and Claudio Castellano. Community structure in graphs. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 1141–1163. Springer, 2009.

[35] Anna C. Gilbert and Kirill Levchenko. Compressing network graphs. In *Proceedings of the LinkKDD workshop at the 10th ACM Conference on KDD*, August 2004.

[36] Ankur Gupta, Wing-Kai Hon, Rahul Shah, and Jeffrey Scott Vitter. Compressed data structures: Dictionaries and data-aware measures. *Theoret. Comput. Sci.*, 387(3):313–331, 2007.

[37] J. A. Hartigan and P. M. Hartigan. The dip test of unimodality. *Ann. Statist.*, 13(1):70–84, 1985.

[38] R. L. Hemminger and L. W. Beineke. Line graphs and line digraphs. In L. W. Beineke and R. J. Wilson, editors, *Selected Topics in Graph Theory*, pages 271–305. Academic Press Inc., 1978.

[39] Guy Jacobson. Space-efficient static trees and graphs. In *30th Annual Symposium on Foundations of Computer Science*, pages 549–554, Research Triangle Park, North Carolina, 1989. IEEE.

[40] Chen Jianbin, Fang Deying, and Shi Tong. A graph partition-based soft clustering algorithm. In *Proceedings of the 2008 Second International Symposium on Intelligent Information Technology Application - Volume 02*, pages 572–577, Washington, DC, USA, 2008. IEEE Computer Society.

[41] U Kang, Charalampos E. Tsourakakis, Ana Paula Appel, Christos Faloutsos, , and Jure Leskovec. HADI: Mining radii of large graphs. *ACM Transactions on Knowledge Discovery from Data*, 2010.

[42] Youngdo Kim and Hawoong Jeong. The map equation for link community. *CoRR*, abs/1105.0257, 2011.

[43] David Knoke and Song Yang. *Social Network Analysis*. Sage Publications, Inc, second edition edition, 2008.

[44] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations (Art of Computer Programming)*. Addison-Wesley Professional, 2005.

[45] Donald E. Knuth. The Art of Computer Programming. Pre-Fascicle 1A. Draft of Section 7.1.3: Bitwise Tricks and Techniques, 2007.

[46] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. *Link Mining: Models, Algorithms, and Applications*, pages 337–357, 2010.

[47] Amy N. Langville and Carl D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 1(3):355–400, 2004.

[48] Silvio Lattanzi and D. Sivakumar. Affiliation networks. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 427–434, New York, NY, USA, 2009. ACM.

[49] Lun Li, David L. Alderson, John Doyle, and Walter Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications. *Internet Math.*, 2(4), 2005.

[50] Bo Long, Mark Zhang, Philip S. Yu, and Tianbing Xu. Clustering on complex graphs. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*, pages 659–664. AAAI Press, 2008.

[51] Massimo Marchiori and Vito Latora. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(3-4):539 – 546, 2000.

[52] Hossein Maserrat and Jian Pei. Neighbor query friendly compression of social networks. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 533–542. ACM, 2010.

[53] Marina Meilă. Comparing clusterings: an axiomatic view. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 577–584, New York, NY, USA, 2005. ACM.

[54] S. Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[55] R. Motwani and P. Raghavan. Randomized algorithms. In *Algorithms and theory of computation handbook*, pages 12–12. Chapman & Hall/CRC, 2010.

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

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

[58] Mark E. J. Newman and Juyong Park. Why social networks are different from other types of networks. *Phys. Rev. E*, 68(3):036122, 2003.

[59] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, Stanford University, Stanford, CA, USA, 1998.

[60] Gergely Palla, Illes J. Farkas, Peter Pollner, Imre Derenyi, and Tamas Vicsek. Directed network modules. *New J.Phys.*, 9:186, 2007.

[61] Christopher R. Palmer, Phillip B. Gibbons, and Christos Faloutsos. Anf: a fast and scalable tool for data mining in massive graphs. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 81–90, New York, NY, USA, 2002. ACM.

[62] Usha N. Raghavan, Réka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in large-scale networks. *Physical Review E (Statistical, Nonlinear, and Soft Matter Physics)*, 76(3), 2007.

[63] Keith H. Randall, Raymie Stata, Janet L. Wiener, and Rajiv G. Wickremesinghe. The Link Database: Fast access to graphs of the web. In *Proceedings of the Data Compression Conference*, pages 122–131, Washington, DC, USA, 2002. IEEE Computer Society.

[64] Peter Ronhovde and Zohar Nussinov. Local resolution-limit-free Potts model for community detection. *Phys. Rev. E*, 81(4):046114, Apr 2010.

[65] Peter Ronhovde and Zohar Nussinov. Local resolution-limit-free potts model for community detection. *Phys. Rev. E*, 81(4):046114, Apr 2010.

[66] Martin Rosvall and Carl T. Bergstrom. Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems. *PLoS ONE*, 6(4):e18209, 04 2011.

[67] Ilya Safro and Boris Temkin. Multiscale approach for the network compression-friendly ordering. *Journal of Discrete Algorithms*, 2010.

[68] Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[69] E. Seneta. *Non-negative matrices and Markov chains*. Springer–Verlag, New York, 1981.

[70] Gergely Tibély and János Kertész. On the equivalence of the label propagation method of community detection and a Potts model approach. *Physica A Statistical Mechanics and its Applications*, 387:4982–4984, August 2008.

[71] Sebastiano Vigna. Stanford matrix considered harmful. In Andreas Frommer, Michael W. Mahoney, and Daniel B. Szyld, editors, *Web Information Retrieval and Linear Algebra Algorithms*, number 07071 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[72] Sebastiano Vigna. Broadword implementation of rank/select queries. In *WEA 2008: Proc. of the 7th International Workshop on Experimental Algorithms*, number 5038 in Lecture Notes in Computer Science, pages 154–168. Springer–Verlag, 2008.

[73] Sebastiano Vigna. Spectral ranking, 2009.

[74] D. F. Vysochanskiĭ and Yu. Ī. Petunīn. Remark: "Proof of the 3σ rule for unimodal distributions" [Teor. Veroyatnost. i Mat. Statist. **21** (1979), 23–35]. *Teor. Veroyatnost. i Mat. Statist.*, 27:26–27, 157, 1982.

[75] Stanley Wasserman, Katherine Faust, and Dawn Iacobucci. *Social Network Analysis : Methods and Applications (Structural Analysis in the Social Sciences)*. Cambridge University Press, 1994.

[76] Wenpu Xing and Ali Ghorbani. Weighted pagerank algorithm. *Communication Networks and Services Research, Annual Conference on*, 0:305–314, 2004.

[77] Sune Lehmann Yong-Yeol Ahn, James P. Bagrow. Link communities reveal multi-scale complexity in networks. *Nature*, 466(7307):761–764, August 2010.

[78] W. W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.