Network Structural Analysis via Core-Tree-Decomposition

Takuya Akiba*
The University of Tokyo
JST, ERATO,
Kawarabayashi Project
t.akiba@is.s.u-tokyo.ac.jp

Takanori Maehara
National Institute of Informatics
JST, ERATO,
Kawarabayashi Project
maehara@nii.ac.jp

Ken-ichi Kawarabayashi
National Institute of Informatics
JST, ERATO,
Kawarabayashi Project
k_keniti@nii.ac.jp

ABSTRACT

The study of network analysis is still a new science; currently, the structure of real-world networks is described only in terms of the coarsest and basic details such as diameter and degree distribution. To efficiently and effectively implement network analysis techniques, a more comprehensive grasp of their finer mathematical structure is required. The seminal work by Leskovec et al. tackled this issue by decomposing networks into two parts; namely "whiskers" and "cores."

In this study, we progress toward this issue by obtaining a novel "core-tree-decomposition," which is a variant of the well-known tree-decomposition. Specifically, we perform experiments to construct core-tree-decompositions for as many as 40 publicly available datasets. Our decomposition provides more structural information than the previous method in the following ways:

1. By comparing the eigenvalue distribution of the cores obtained by our decomposition method with that of the Erdős-Rényi random graphs, we confirm that, unlike the previously defined core, our "core" behaves like a random graph, i.e., an expander graph. Thus, the intuition that the cores should be "expander-like" is confirmed by the eigenvalue distribution of our cores.

2. By deleting the core, we obtain a tree-decomposition of small width, which behaves like a tree. Therefore, we can say that whiskers are "tree-like," and can be explained in terms of "tree-width."

3. We show that the cores of real networks widely range in size, which implies that their tree-widths are also quite varied. Furthermore, we show theoretical and empirical evidence that tree-width plays a significant role in the efficiency of certain types of algorithms.

*Supported by a Grant-in-Aid for JSPS Fellows (256563).

KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623753.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Application—*Data Mining*

General Terms

Measurement, Theory, Algorithms

Keywords

Tree-decomposition; tree-width; core-tree-decomposition; expander; social network; web graph

1. INTRODUCTION

In recent years, considerable interest in graph structures has been arising in social and information networks, i.e., graphs in which the vertices represent underlying social entities and the edges represent interactions between pairs of vertices. However, the study of network analysis is still in its infancy; currently, the structure of real-world networks is described only in terms of the coarsest and basic details such as diameter and degree distribution. To efficiently and effectively implement network analysis techniques, a more comprehensive grasp of their finer mathematical structure is required. In fact, in most cases, we need to extract such a structural property.

The seminal work by Leskovec et al. [24] tackled this issue by decomposing web graphs and social networks into two parts; namely the "whiskers" and the "cores". We closely examine this approach first.

1.1 "Whiskers" and "Core"

In social networks and web graphs, there is a substantial fraction of vertices that are barely connected to the main part of the network, i.e., they are part of a small subgraph between 10 and 100 vertices that are attached to the remainder of the network via one or a small number of edges. In particular, a large fraction of the network comprise vertices that are not in the "core," i.e., they are in components attached to the core of the network via a single edge. In fact, according to [24], their empirical results show that a substantial fraction of vertices are connected to the main network by a single cut edge, and the largest 2-edge-connected component of this network contains only around 60% of the vertices and 80% of the edges of the original network.

This is a motivation to define a "core" and a "whisker" more precisely. The "core" of a network is the largest 2-edge-connected component. "Whiskers" are maximal subgraphs that can be detached from the rest of the network by removing a single edge. To find whiskers, we only need to detect bridges (i.e., an edge e such that $G - e$ is disconnected) and the largest 2-edge-connected component (which is the core), which can be easily achieved in linear time.

1.2 Structural Properties of Real Networks

In [24], it is stated that web graphs and social networks should have both numerous small, well-separated, whisker-like clusters and an expander-like core[1]. Specifically, the following should be considered for the structural property of web graphs and social networks.

1. They should have a relatively large number of comparatively small (except when compared with random graphs), well-connected, and distinct whisker-like communities.
2. They should have a large expander-like core graph, which may be considered as intermingled communities, possibly emerging from whisker-like communities. The boundaries of these communities become less well-defined as they grow larger, and as they gradually blend in with the rest of the network.

1.3 Our Contributions

The above two points are certainly realistic network structures; however, several important questions remain unanswered. Our main motivation is the following.

> In reality, decomposing web graphs and social networks into "whiskers" and "cores" using 2-edge-connected components does not seem to be sufficient to resolve these points.

Let us make our motivations clearer and more specific:

1. The previous "cores" of web graphs and social networks seem to have several cuts of size two. This suggests that the "core" defined by a maximal 2-edge-connected subgraph is not really an expander, because if it were, it would be globally highly connected. To construct expander-like cores, what types of operations are required for web graphs and social networks?
2. In response to the above question, whiskers do not seem to capture enough structural information. What type of structure should we define instead of the previous whiskers?

It turns out that we can answer these questions using our "core-tree-decomposition" method. This decomposition is a type of tree-decomposition, which has been extensively studied in the algorithm research community (for examples, see [3, 4, 13, 17, 18, 28, 29]). However, in this core-tree-decomposition, we shall detect a "core" of a given graph. Specifically, our method allows us to decompose any graph into two parts, "core" and "almost tree," such that

> the "almost tree" part induces a tree-decomposition of small tree-width. Specifically, it behaves like a "tree" (Subsection 2.3).

[1]For the exact definition of expander graphs, see Subsection 2.2. For the time being, we can assume that expander graphs behave like random graphs.

Moreover, such decomposition can be computed very quickly (Section 3).

We constructed core-tree-decompositions for as many as 40 publicly available datasets. Results from these experiments show that our cores provide more structural network information in the following sense:

> By comparing the eigenvalue distribution of the cores obtained by our decomposition method with that of the Erdős-Rényi random graphs, we confirm that our "core" behaves like an expander graph, whereas the "core" defined by 2-edge-connected components [24] does not. Details are given in Subsection 6.2.

The intuition behind this fact will be explained in Subsection 2.4.

In summary, our core-tree-decomposition can resolve the above two issues in such a way that whiskers are "tree-like," which can be explained in the context of tree-width, and the intuition that the cores should be "expander-like" can be explained via their eigenvalue distribution. Moreover, our core-tree-decomposition yields the following features, which are of independent interest:

1. Our analysis based on core-tree-decomposition scales to large networks with billions of edges.
2. We observe that many widely used network datasets have a small core, which implies that they are of small tree-width. For these datasets, we perform experiments to show that tree-width plays a significant role in small index space for certain types of algorithms.
3. Social networks tend to have larger cores. Indeed, for social networks, the size of cores is up to 23% of the vertices; however, for web graphs, it is at most 13% of the vertices. These facts contrast with the core defined by 2-edge-connected components [24] that contains 60% of the vertices.

This paper is organized as follows. In the next section, we introduce tree-decompositions and core-tree-decompositions that are key concepts in our study. Our decomposition algorithm is explained in Section 3, and the datasets for our experiments are explained in Section 4. We classify the networks into two classes: small and large datasets, where the former are mainly discussed in Section 5 to deal with full tree-decompositions, and the latter are only used in Section 6 to analyze the structural properties of cores. First, however, we introduce several additional related works.

Related Work on Tree-Decomposition.

A vast amount of theoretical work has been conducted using tree-decomposition and tree-width, such as [4, 13, 17, 21, 22, 28–30]. In the area of machine learning, tree decomposition is further employed to solve the probabilistic inference problem on a small tree-width graph [23, 32, 35]. Tree-width is used to characterize a polynomially solvable class of the constraint satisfaction problem (CSP); under some conditions, a CSP is solvable in polynomial time if and only if an associated graph is of small tree-width [16], and some algorithms achieve this bound [11, 19].

There are a few recent papers that propose the construction of a core-tree-decomposition in preprocessing to helping computation [25, 3, 33]. We use the algorithm presented in [25] to compute a core-tree-decomposition.

2. KEY CONCEPTS

In this section, we formally define our key concepts: core-tree-decompositions and expander graphs. Before giving their definitions, we introduce basic notations from graph theory and matrix schemes that are required in our paper.

2.1 Basic Notation

Let $G = (V, E)$ be a graph. We use symbols n and m to denote the number of vertices and edges of a chosen graph. For a vertex set $S \subseteq V$, let $e(S)$ be the set of edges between S and $V - S$. We define $d(S) := |e(S)|$. For a vertex $v \in V$, $d(\{v\})$ is simply written as $d(v)$ and is called the *degree* of v. If G is a digraph, then $d^{-}(v)$ is the number of edges whose tail is v.

In this study, we also require a matrix A that arises from a graph, because this matrix and its eigenvalues help us decide whether a given graph is close to an expander (Subsection 2.2). Formally, given a directed graph G, we are interested in the *adjacency* matrix. Set the vertex set $V(G) = \{1, \dots, n\}$. Then $A(i, j) = 1$ if there is an edge from i to j. If G is undirected, then we assume that G is a bidirectional graph (i.e., each edge corresponds to both directions). The *transition* matrix of G is $P = AD^{-1}$, where D is a diagonal matrix and $D(i, i) = 1/d^{-}(i)$.

Let B be an $n \times n$ matrix. An eigenvalue decomposition of B is a decomposition of the form

$$B = U^{\top} \Lambda U$$

where U is an orthogonal matrix and Λ is a diagonal matrix. Each element of Λ is called *eigenvalue* and the corresponding row of U is called *eigenvector*.

2.2 Expander Graph and Tools from Spectral Graph Theory

Roughly, an *expander graph* is a sparse graph in which every subset of the vertices that is not "too large" has a "large" boundary. (i.e., "globally connected"). One of the key properties given in our paper is that our core is close to an expander graph (Subsection 6.2). In this subsection, we formally define an expander graph. Whether or not a given graph G is close to an "expander" is important in our study. We explain our method of solving this at the end of this subsection.

To formally define an expander graph, we need further definitions. For a vertex set $S \subseteq V$,

$$\mu(S) := \sum_{v \in S} d(v),$$

and the *conductance* is defined as $\phi(S) := d(S)/\mu(S)$. The *conductance* of an undirected graph G is

$$\phi(G) := \min_{S \subseteq V : \mu(S) \leq \mu(V)/2} \phi(S).$$

We say that a graph G is an *η-expander* if $\phi(G) \geq \eta$. Intuitively, if the conductance η is large, then we say that G is "globally" connected. In this case, we say that G is an *expander*. It is well-known that expander graphs behave like random graphs.

The quantity $\phi(G)$ is NP-complete to compute. However, the following result from Jerrum and Sinclair [20] says that we can approximately calculate η by considering the second eigenvalue λ_2 of the transition matrix of a graph.

$$\eta^2/16 \leq 1 - \lambda_2 \leq \eta.$$

Indeed, from spectral graph theory [10], it has been known that expander graphs and eigenvalue distribution are closely related. Therefore, to decide whether or not a given graph G is close to an expander, the easiest way seems to be the following:

> looking at the eigenvalues (and their distribution) of the transition matrix of G.

We perform this in our experiments in Section 6 (Subsections 6.1 and 6.2).

2.3 Tree-Decomposition

The main tool in this study is a core-tree-decomposition. This is based on a tree-decomposition. Let us first define tree-decomposition and tree-width. Let G be an undirected graph, T a tree and let $\mathcal{V} = \{V_t \subseteq V(G) \mid t \in V(T)\}$ be a family of vertex sets $V_t \subseteq V(G)$ indexed by the vertices t of T. Following [4, 29], the pair $(T, \mathcal{V})$ (or $(V_t)_{t \in T}$) is called a *tree-decomposition* of G if it satisfies the following three conditions:

- $V(G) = \bigcup_{t \in T} V_t$,
- for every edge $e \in E(G)$ there exists a $t \in T$ such that both ends of e lie in V_t,
- if $t, t', t'' \in V(T)$ and t' lies on the path of T between t and t'', then $V_t \cap V_{t''} \subseteq V_{t'}$.

Each V_t is sometimes called a *bag*. The width of $(T, \mathcal{V})$ (or $(V_t)_{t \in T}$) is the number $\max\{|V_t| - 1 \mid t \in T\}$ and the tree-width of G is the minimum width of any tree-decomposition of G. In this way, tree-width is a measure of how close a given graph is to a tree. Specifically, if the tree-width is small, then it behaves like a tree.

Furthermore, tree-width is a measure of "global" connectivity. To support this claim, let us first observe that in the framework of tree-width, it is well-known that an expander graph cannot be of small tree-width. Therefore,

> expander graphs are a counterpart of graphs of small tree-width.

The converse is not quite true, but it is true that,

> if the tree-width of a given graph is large, it contains a "highly" connected set that behaves like an expander graph (see Section 12.4 of Diestel [12] for more details).

For more details on tree-width, we refer the reader to a survey by Reed [28]. It turns out that this intuition is very important for our core-tree-decomposition, which will be defined in the next subsection.

2.4 Core-Tree-Decomposition

We now introduce the most important concept in this paper. We say that G has a "core-tree-decomposition of width d" if a tree-decomposition $(T, \mathcal{V})$ satisfies the following:

1. r is a root of the tree T, and
2. for all $t \in T - r$, $|V_t| \leq d$.

Thus, only the bag V_r has more than d vertices, and indeed, V_r corresponds to the "core". Sometimes we call $V - V_r$ the *bounded tree-width part* or the *small tree-width part*.

The purpose of our core-tree-decomposition is the following:

Algorithm 1 Core-tree-decomposition algorithm

```
 1: procedure DecomposeGraph($G$, $d$)
 2:     $\mathscr{X} \leftarrow$ Empty list of bags
 3:     ▽ Reduce vertices from those with lower degree
 4:     while $G$ is not empty do
 5:         $v \leftarrow$ A vertex with minimum degree
 6:         break if $d(v) > d$
 7:         $V_v \leftarrow$ ReduceVertex($G$, $v$)
 8:         Append $V_v$ to list $\mathscr{X}$
 9:     ▽ Construct the root bag $V_r$
10:     $V_r \leftarrow V(G)$
11:     Append $V_r$ to list $\mathscr{X}$
12:     ▽ Compute the tree
13:     return ConstructTree($\mathscr{X}$)
```

1. We want to extract as many vertices in the core as possible so that the extracted vertices induce a graph W of small tree-width. If the remaining core is also small (i.e., it has less than d vertices), then the tree-width is at most d.
2. Once we are left with extracting vertices from the core and the remaining core contains more than d vertices, the core no longer induces a graph of tree-width d. This suggests that it contains a highly connected subgraph that behaves like an expander graph. We want this core to be close to an expander.

Therefore, we need to choose d. In Section 3, we propose, given d, an efficient algorithm to obtain our core-tree decomposition of width d. In Subsection 6.3, we discuss how to choose d.

3. DECOMPOSITION ALGORITHM

In this section, we explain the algorithm that we use to construct a core-tree-decomposition. The algorithm is based on the *min-degree heuristic* [5, 34], which was originally designed for computing standard tree-decompositions. We use a modified version that receives a parameter d and produces a core-tree-decomposition of width d. Note that it can also compute a standard tree-decomposition by setting d as a sufficiently large value.

This algorithm is different from Bodlaender's algorithm [6] and its successor [27]. The time complexity of these algorithms is $\Omega(2^d(n+m))$ time, i.e., the exponential of tree-width. Therefore, while the algorithms run in linear time when the tree-width is very small, in practice, tree-width is not that small and thus they are impractical.

3.1 Algorithm Overview

Note that, in this section, we are only interested in undirected graphs, even if the input graphs are directed. To handle a directed graph, we apply the following algorithm to the undirected graph produced by ignoring the direction of each edge.

The total algorithm is presented as Algorithm 1. It first generates a list of bags, and then constructs a tree of these bags. To generate a list of bags, the algorithm repeatedly *reduces* a vertex with minimum degree while minimum degree is at most d.

Reducing a vertex v involves three steps, as described in Algorithm 2. First, we create a new bag V_v that includes v and all its neighbors. Then, we remove vertex v from graph G. Finally, we add edges between the neighbors to make

Algorithm 2 Reduction of vertex

```
1: procedure ReduceVertex($G$, $v$)
2:     $V_v \leftarrow \{v\} \cup N_G(v)$
3:     for all $x, y \in N_G(v)$ such that $x \neq y$ do
4:         ▽ Make v's neighbors a clique
5:         Add edge $(x, y)$ to $E(G)$ if it is not in $E(G)$
6:     Remove $v$ from $G$
7:     return $V_v$
```

a clique among those vertices in $V_v - v$. When no vertex with degree less than or equal to d remains, we finalize this process by creating a new bag V_r that consists of all the remaining vertices. The last bag V_r corresponds to the "core" and can be arbitrarily large.

After generating the list of bags, the tree of these bags is constructed. It can be guaranteed that we obtain a valid core-tree-decomposition by setting the parent of bag V_v as bag V_p where $(V_v - v) \subseteq V_p$.

3.2 Efficient Implementation

As reducing a vertex with degree δ adds $\Theta(\delta^2)$ edges, naive implementation of the above heuristic consumes considerable space and time to compute core-tree-decompositions for today's large networks. Therefore, the following speed-up technique is employed to address this issue.

To reduce a vertex v, instead of adding a clique around v, we add a vertex with a special label, named a *hub vertex*. We add edges to make the new hub vertex incident to the neighbors of v. Note that here we add only $O(d(v))$ edges. Then, we handle the resulting graph as if there were cliques around hub vertices, i.e., we consider that there is an edge between two vertices that have a hub vertex as a common neighbor.

4. EXPERIMENTAL SETUP

All experiments are conducted on a Linux server with Intel Xeon E5-2690 2.90GHz CPU with 256GB memory. Our algorithm is implemented in C++ and compiled with g++v4.6 with -O3 option.

We conducted experiments on a number of real-world networks that are publicly available from the following sources:

- Datasets `youtube`, `flickr`, `livejournal`, and `orkut` are from the social computing group at the Max Planck Institute [26][2].
- Datasets `in-2004`, `indochina-2004`, `it-2004`, `twitter-2010`, and `uk-2007-05` are from the Laboratory for Web Algorithmics at the Università degli studi di Milano [9, 8, 7][3].
- The other datasets are from the Stanford Network Analysis Project (SNAP)[4].

We classified the networks into two classes: small and large datasets. Small datasets are mainly discussed in Section 5, and large ones are only used in Section 6.

[2]`http://socialnetworks.mpi-sws.org/datasets.html`
[3]`http://law.di.unimi.it/datasets.php`
[4]`http://snap.stanford.edu/data/`

Table 1: Information of small datasets, results of our full tree-decomposition, and sizes of 2-hop indices constructed by state-of-the-art indexing methods for shortest-path distance queries.

Dataset Information				Tree-decomposition			Distance Indices (MB)	
Name	$\lvert V\rvert$	$\lvert E\rvert$	Type	Time (s)	Width d	$d/\lvert V\rvert$	PLL [1]	ISL [14]
`ca-grqc`	5,242	28,980	social (u)	0.02	253	0.048	1.4	3.9
`ca-hepth`	9,877	51,971	social (u)	0.16	798	0.081	4.4	24.7
`wiki-vote`	7,115	103,689	social (d)	0.59	1,332	0.187	2.4	23.8
`ca-condmat`	23,133	186,936	social (u)	1.5	2,160	0.093	13.4	156.8
`ca-hepph`	12,008	237,010	social (u)	0.52	1,406	0.117	8.8	65.5
`email-enron`	36,692	367,662	social (d)	1.43	2,178	0.059	8.4	136.1
`ca-astroph`	18,772	396,160	social (u)	3.34	3,497	0.186	19.5	233.5
`email-euall`	265,214	420,045	social (d)	1.67	1,033	0.004	84.1	453.0
`soc-epinions1`	75,879	508,837	social (d)	11.43	5,504	0.073	45.5	1,033.3
`soc-slashdot0811`	77,360	905,468	social (d)	28.31	8,555	0.111	77.1	1,772.3
`soc-slashdot0902`	82,168	948,464	social (d)	29.41	9,181	0.112	85.3	2,028.5
`web-notredame`	325,729	1,497,134	web (d)	0.99	2,938	0.009	95.4	2,835.4
`web-stanford`	281,903	2,312,497	web (d)	2.82	1,611	0.006	64.3	2,086.7
`web-google`	875,713	5,105,039	web (d)	80.28	18,229	0.021	712.3	64,540.1
`web-berkstan`	685,230	7,600,595	web (d)	10.81	3,272	0.005	195.5	10,482.0
`p2p-gnutella08`	6,301	20,777	p2p (u)	0.12	1,263	0.200	5.0	26.3
`p2p-gnutella09`	8,114	26,013	p2p (u)	0.19	1,618	0.199	8.0	43.5
`p2p-gnutella06`	8,717	31,525	p2p (u)	0.66	2,199	0.252	10.2	65.2
`p2p-gnutella05`	8,846	31,839	p2p (u)	0.35	2,215	0.250	10.5	65.5
`p2p-gnutella04`	10,876	39,994	p2p (u)	0.61	2,789	0.256	15.8	102.6
`p2p-gnutella25`	22,687	54,705	p2p (u)	0.9	3,618	0.159	45.8	284.1
`p2p-gnutella24`	26,518	65,369	p2p (u)	1.59	4,320	0.163	50.0	389.6
`p2p-gnutella30`	36,682	88,328	p2p (u)	2.04	5,596	0.153	100.9	707.6
`p2p-gnutella31`	62,586	147,892	p2p (u)	5.78	9,385	0.150	233.6	2,050.2
`cit-hepth`	27,770	352,807	citation (d)	11.61	8,515	0.307	158.4	687.0
`cit-hepph`	34,546	421,578	citation (d)	10.21	10,718	0.310	178.4	1,155.1

5. TREE-DECOMPOSITIONS FOR SMALL DATASETS

First, we study tree-decompositions on smaller datasets. In this section we focus on standard tree-decompositions, and then we move to core-tree-decompositions in the next section. As computing tree-decompositions is considerably more expensive than core-tree-decompositions with moderate width parameter d, in this section, we only use small datasets with up to 7.5 million edges, for which we successfully obtained a tree-decomposition in 100 seconds. In Table 1, the information of small datasets and results of our tree-decomposition are given.

Note that our tree-decompositions are not necessarily optimal, and hence the width given in Table 1 is just an upper bound of tree-width. However, the aim of this section is to show how informative the width of a tree-decomposition obtained by our algorithm is. For this purpose, we take *shortest-path distance indexing methods* [1, 14] as an example and tackle the long-standing question in this field: what is the key factor in addition to network size that has a large effect on the size of constructed shortest-path distance indices? We believe that similar results also hold for other problems where state-of-the-art methods are designed to exploit the structures of real networks.

5.1 Shortest-Path Distance Indices

There has been great interest in the database research community in indexing methods that efficiently find the distance between two arbitrary points on graphs [33, 3, 1, 14, 2]. This remains a highly challenging problem with a wide range of applications such as network-aware search and network analysis.

In the following section, we focus on two state-of-the-art indexing methods: *pruned landmark labeling* [1, 2] and *IS-label* [14]. Both are based on the *2-hop cover framework*. The general framework of *2-hop cover* is as follows. For simplicity, we assume that the input graph G is undirected.

For each vertex v, we precompute a *label* denoted as $L(v)$, which is a set of pairs (u, δ_{uv}), where u is a vertex and $\delta_{uv} = d_G(u, v)$. We sometimes call the set of labels $\{L(v)\}_{v\in V}$ an *index*. To answer a distance query between vertices s and t, we compute and answer $\textsc{Query}(s, t, L)$ defined as follows,

$$\textsc{Query}(s,t,L) = \min\{\delta_{vs} + \delta_{vt} \mid (v,\delta_{vs}) \in L(s), (v,\delta_{vt}) \in L(t)\}.$$

We call L a *2-hop cover index* of G if $\textsc{Query}(s, t, L)$ is the correct distance for any pair of vertices s and t.

While both pruned landmark labeling and IS-label use further sophisticated frameworks based on 2-hop cover, they can be used to construct standard 2-hop indices. Therefore, for simplicity, in our experiments we constructed standard 2-hop indices by these methods, i.e., pruned landmark labeling was not combined with the bit-parallel labeling technique and IS-label constructed complete vertex hierarchy.

5.2 Non-Trivial Factors for Index Size

Table 1 lists the sizes of 2-hop indices constructed by the two methods, pruned landmark labeling [1] and IS-label [14] for our datasets. The results indicate that, even if two graphs are of similar size, indices constructed from these graphs by the same algorithm may be of quite different sizes.

For example, datasets `email-enron`, `ca-astroph`, `cit-hepth` and `cit-hepph` have similar sizes in terms of the number of vertices and edges. However, sizes of indices for these

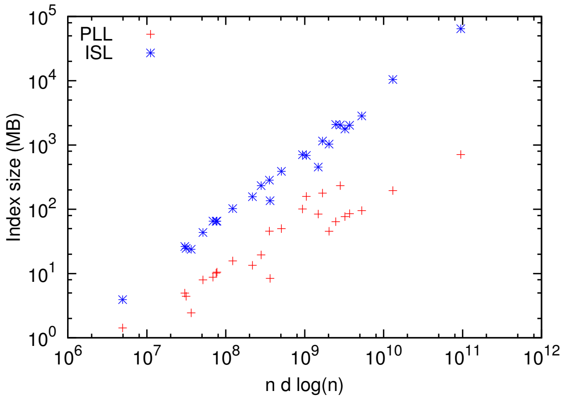


Figure 1: Actual index sizes and estimation using the widths of tree-decompositions.

datasets vary largely and, surprisingly, indices for `cit-hepph` are approximately ten times larger than those for `email-enron` for both indexing algorithms. Similar differences can be observed from datasets `ca-condmat` and `p2p-gnutella30`.

This is because these state-of-the-art indexing methods heuristically exploit the structures of real networks explicitly or implicitly, and thus they depend on the properties of each network. Indeed, it is proved that 2-hop indices may have $\Theta(n^2)$ space for general graphs [15]. Therefore, it is impossible to construct small 2-hop indices without exploiting the network structures.

However, the long-standing question among researchers in this field is that, having understood this, what is the key factor besides network size that has a large effect on the size of constructed shortest-path distance indices? In the following section, we show that obtaining the width of a tree-decomposition can take us closer to the answer.

5.3 Qualitative Empirical Analysis

Widths of tree-decompositions obtained using our algorithm are listed in Table 1. We can observe that the difference in index sizes seems to be highly related to widths. For example, widths of the tree-decompositions for datasets `email-enron` and `cit-hepph` are 10,718 and 2,178, respectively. Similarly, tree-decompositions for `ca-condmat` and `p2p-gnutella30` have widths of 1,406 and 5,596, respectively. Interestingly, from our results, index sizes for graphs with larger widths are almost always larger than those for graphs with smaller widths. Therefore, tree-width could be the key factor that has a large effect on the sizes of constructed shortest-path distance indices.

5.4 Theoretical Bound with Tree-width

To further analyze the relation between widths of tree-decompositions and sizes of distance indices, we first present theoretical analysis result. While there are no non-trivial theoretical bounds on the size of 2-hop indices that are better than $O(n^2)$ for general graphs [15], we can prove that there are small 2-hop indices for graphs of small tree-width.

In what follows, a *centroid* of a tree T denotes a node $v \in V(T)$ such that any connected component after deleting v has maximum size $V(T)/2$. It is easy to prove that any tree has at least one centroid.

Table 2: Spearman's correlation between actual index sizes and estimation with and without width d.

Methods	n	m	$n+m$	d	$nd \log n$
PLL [1, 2]	0.819	0.774	0.798	0.795	**0.899**
ISL [14]	0.940	0.792	0.875	0.719	**0.983**

THEOREM 1. *Let G be a graph of tree-width d. There is a distance-aware 2-hop index for G with total size $O(nd \log n)$.*

PROOF SKETCH. We start from an empty index L. Let bag V_t be a centroid of the tree-decomposition. We add the distance $d(u, v)$ to L for all pairs (u, v) where $u \in V$ and $v \in V_t$. We then recurse to each part that can be obtained by deleting V_t. We add $O(nd)$ pairs in each depth of recursion, where the maximum depth is $O(\log n)$. $\square$

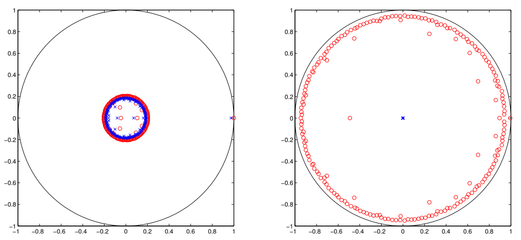
5.5 Quantitative Empirical Analysis

Although the aforementioned methods do not necessarily yield 2-hop indices of that size (i.e., $O(nd \log n)$), we show that this theoretical bound works quite well as an estimation. Figure 1 illustrates the relation between the estimated value $nd \log n$ and actual sizes of constructed indices showing that these values correlate well.

Moreover, in Table 2, we see that Spearman's correlation coefficient between estimation $nd \log n$ and actual index sizes is significantly higher than other estimations such as n and m. This indicates that the width d of our tree-decomposition is indeed informative. Note that Spearman's correlation coefficient uses only ranks and thus, for example, the score for estimation of n^2 would be exactly the same as that of n.

6. CORE-TREE-DECOMPOSITIONS FOR LARGE DATASETS

In this section, we look at datasets of large size (between 5M and 3.6B edges). Because of the time and space constraints for our algorithm in Section 3, we cannot fully obtain a tree-decomposition for these datasets; however, we can apply our proposed algorithm to construct a core-tree-decomposition for these datasets up to $d = 1000$. The experimental results are shown in Table 3. It takes at most 45 minutes to construct a core-tree-decomposition of width



(a) Erdős-Rényi random graph ($n = 10000$, $m = 99759$) (b) Binary tree with random edges added ($n = 100000$, $m = 115041$)

Figure 2: Extreme eigenvalues of an Erdős-Rényi random graph and a binary tree with random edges added; x axis is for the real part and y axis is for the imaginary part. Red point is for the whole network and blue point is for the core.

Table 3: **Information of large datasets, size of the largest biconnected components (BC), and results of our core-tree-decomposition.**

Dataset Information				BC Size	CT-Decomp. Time (s)			Core Size $\|V_r\|$		
Name	$\|V\|$	$\|E\|$	Type		$d=10$	$d=100$	$d=1000$	$d=10$	$d=100$	$d=1000$
`wiki-talk`	2,394,385	5,021,410	social (d)	65,778	2.21	2.75	6.45	63,366	19,047	15,966
`youtube`	3,238,848	18,524,095	social (u)	438,730	3.70	8.72	30.33	352,714	129,389	111,556
`flickr`	1,861,233	22,613,981	social (d)	504,983	2.17	5.88	27.28	318,788	156,117	131,477
`soc-pokec`	1,632,804	30,622,564	social (d)	1,339,202	1.10	16.17	107.42	1,023,232	648,417	574,188
`livejournal`	5,284,458	77,402,652	social (d)	3,046,940	6.19	46.91	220.28	2,266,055	1,221,374	1,054,805
`orkut`	3,072,627	223,534,301	social (u)	2,937,152	0.84	41.37	329.5	2,744,856	1,986,681	1,797,098
`twitter-2010`	41,652,230	1,468,365,182	social (d)	38,213,230	37.17	383.01	2514.82	24,992,877	7,767,763	5,571,211
`web-notredame`	325,729	1,497,134	web (d)	72,904	0.28	0.94	1.11	52,927	8,919	3,237
`web-google`	916,428	5,105,039	web (d)	564,705	1.27	4.81	7.75	322,633	49,645	23,450
`in-2004`	1,382,908	16,917,053	web (d)	670,103	0.84	3.93	5.14	547,537	47,036	5,583
`indochina-2004`	7,414,865	194,109,311	web (d)	4,160,949	2.97	32.45	63.02	4,109,397	470,944	53,069
`it-2004`	41,291,318	1,150,725,436	web (d)	30,683,202	18.00	246.96	741.07	25,771,538	4,711,353	1,366,281
`uk-2007-05`	105,896,435	3,738,733,648	web (d)	80,399,275	30.05	678.55	2253.75	73,394,948	13,990,744	3,995,511

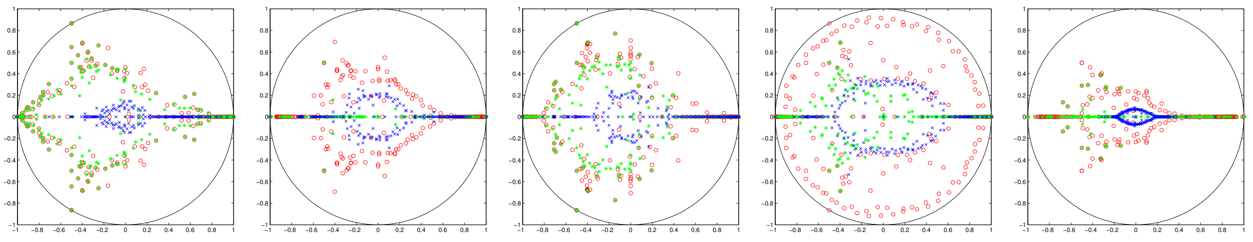

(a) flickr (b) soc-pokec (c) livejournal (d) twitter-2010 (e) wiki-talk

Figure 3: Social Networks; red points are the distribution of eigenvalues of the whole network, green points are those of the largest 2-edge-connected components, and blue points are those of our cores with $d = 100$.

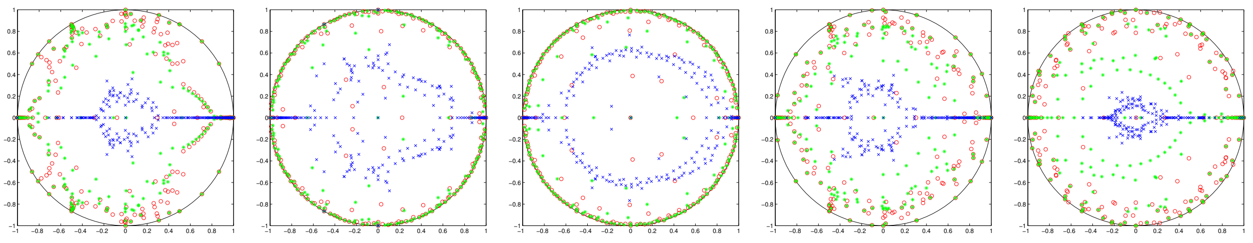

(a) in-2004 (b) it-2004 (c) uk-2007-05 (d) web-google (e) web-notredame

Figure 4: Web Graphs; red, green and blues points are the same as in Figure 3.

$d = 1000$ even for the largest dataset. Furthermore, we can confirm that there is still a large core even for the case $d = 1000$ for these datasets (Table 3). This suggests that our proposed algorithm will decompose graphs into the "core" part and the "small tree-width" part.

The main object of this section is to investigate structural properties of cores. More specifically, we look at the following points.

1. How close is the core to an expander?
2. What happens if we change d?
3. What is the density of the core?

We propose a way to answer the first question in the next subsection.

6.1 Typical Eigenvalues

In general, given a graph G, it is difficult to determine how close it is to an expander. The best way is by looking at the eigenvalues distribution, as discussed in Subsection 2.2.

For a typical expander graph (Erdős-Rényi random graph with $n = 10000$ vertices and $m = 99759$ edges) and a typical non-expander graph (binary tree with random edges added. $n = 100000$ vertices and $m = 115041$ edges), we compute 200 extreme eigenvalues of their cores and their whole networks, respectively, by the Arnoldi method [31].

We actually conduct experiments only for directed graphs because there are many more large directed graphs than large undirected graphs in the datasets, and we want to look

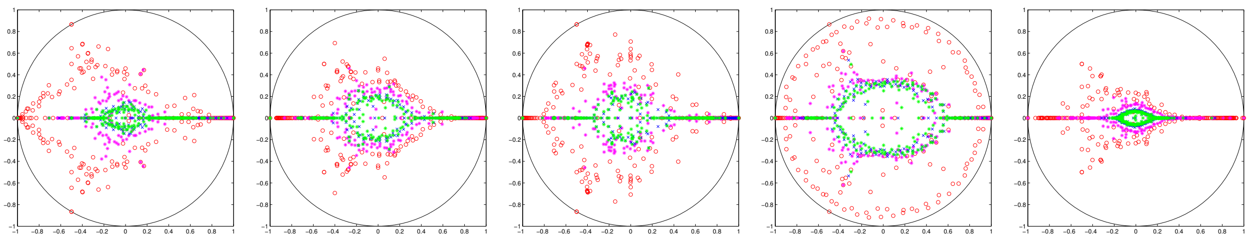

(a) flickr (b) soc-pokec (c) livejournal (d) twitter-2010 (e) wiki-talk

Figure 5: Social Networks; red points are the distribution of eigenvalues of the whole network, purple points are those of the core with $d = 10$, blue points are those of the core with $d = 100$, and green points are that of the core with $d = 1000$.

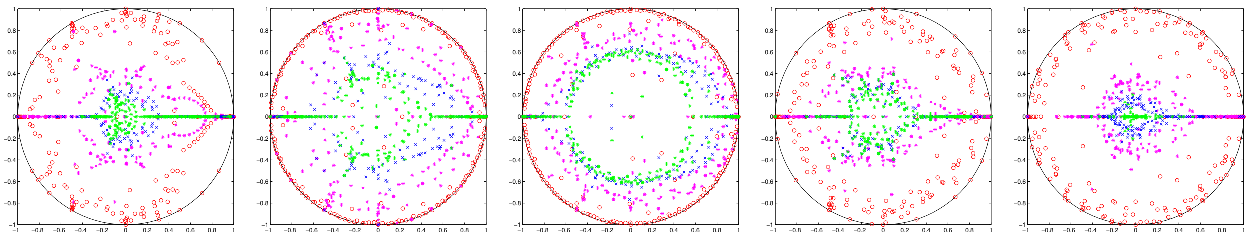

(a) in-2004 (b) it-2004 (c) uk-2007-05 (d) web-google (e) web-notredame

Figure 6: Web Graphs; red, green, purple and blue points are the same as in Figure 5.

at the eigenvalues distribution for as many "large" datasets as possible[5]; thus, it is fair to look only at digraphs.

The results are shown in Figure 2. Figure 2a is for the directed version of an Erdös-Rényi random graph, which is supposed to be a typical expander graph, and Figure 2b is for a directed complete binary tree with few random edges added, which is a typical non-expander graph (i.e., a small tree-width graph). As the transition matrices of directed graphs are not necessarily symmetric, the eigenvalues may not be real. Indeed, they contain both the real and imaginary parts, as shown in Figures 2a and 2b.

From Figures 2a and 2b, we can observe that a given network is close to an expander graph if the eigenvalues are clustered near the origin, and a given network would contain a subgraph that behaves like a tree (i.e., a small tree-width graph), if the eigenvalues were scattered. We now look at real networks.

6.2 How Close the Core is to an Expander

In this subsection, we compare our core with the core defined by Leskovec et al. [24] (i.e., the maximum 2-edge-connected component) and the original networks. More precisely, for ten real (directed) networks that contain five social networks and five web graphs, we compute 200 "extreme" eigenvalues of their cores (from our core-tree-decompositions), their maximum 2-edge-connected components, and the whole networks, by the Arnoldi method [31]. For simplicity, we fix $d = 100$, and, in the next subsection, we examine how the distribution of eigenvalues changes if we change d. The results are shown in Figure 3 for social networks and in Figure 4 for web graphs.

[5]Moreover, the eigenvalue distribution for directed graphs is considerably easier to visualize than that for undirected graphs.

We conclude that the eigenvalues of whole networks (red points) are scattered, like the distribution of a non-expander graph (Figure 2b). The eigenvalues of the maximum 2-edge-connected components (green points) are also scattered, although the green points are slightly closer to the origin than the red points. However, the eigenvalues of the cores (blue points) are clustered relatively near the origin, like the distribution of an expander graph (Figure 2a). Therefore, we can conclude that,

> neither real networks nor the maximum 2-edge-connected components are expander, but our cores of the networks are closer to expander.

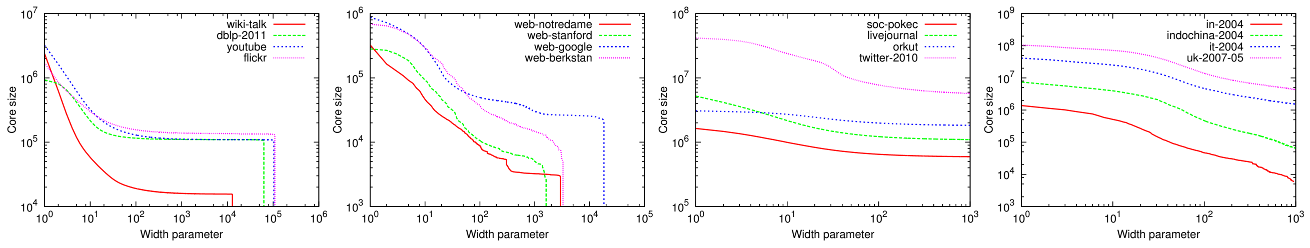
There is one more observation we can make from Table 3. For social networks, except for orkut and soc-pokec (which seem to have considerably bigger cores than other social networks), the size of cores is a maximum of 23% of the vertices of the original network (when $d = 100$), but for web graphs, it is at most 13% of the vertices. These facts are in contrast with the maximum 2-edge-connected component (i.e., the core defined by Leskovec et al. [24]) which contains 60% of the vertices, as previously mentioned. Therefore, we can conclude that

> social networks tend to have a larger "core" than web graphs. Moreover, our core contains a considerably smaller number of vertices than the core defined by 2-edge-connected components [24].

6.3 Changing Width Parameter *d*

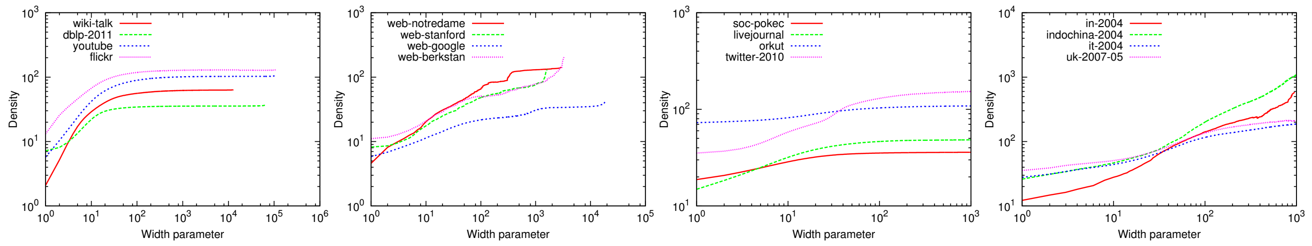
We now look at cores with respect to the width d of our core-tree-decomposition for social networks and web graphs, respectively.

Let us first look at Figure 5 for social networks and Figure 6 for web graphs, respectively. Our process is as follows.



(a) Small social networks (b) Small web graphs (c) Large social networks (d) Large web graphs

Figure 7: Size of the cores of core-tree-decompositions for different width parameter d.



(a) Small social networks (b) Small web graphs (c) Large social networks (d) Large web graphs

Figure 8: Density of the cores of core-tree-decompositions for different width parameter d.

For ten real (directed) networks that contain five social networks and five web graphs, we compute 200 extreme eigenvalues of the whole networks and of their cores for the cases $d = 10, 100, 1000$, respectively, by the Arnoldi method [31]. The results are shown in Figure 5 for social networks and in Figure 6 for web graphs, respectively.

We can conclude that as d increases, the eigenvalues of web graph cores are clustered closer to the origin. This indicates that as d increases, the cores of web graphs gets closer to expander (like the distribution of an expander graph (Figure 2a)). On the other hand, as d increases, the eigenvalues of the social network cores are first clustered close to the origin, but then stay for a while. This indicates that for social networks, we can stop at some value of d, but for web graphs, we can increase d. Indeed, $d = 100$ seems sufficient for social networks.

This can be further explained from Figure 7 which shows the size of cores for social networks and web graphs, with respect to the width d. As the width d increases, the size of web graph cores decreases. On the other hand, the size of social network cores first decreases, but then remains constant a while until the cores of the networks form one bag of core-tree-decompositions. Indeed, for large social networks, if we consider $d \geq 100$, then the core size does not change considerably, but this is not the case for web graphs. Namely, the core size still decreases, even if d increases beyond 100. This indicates that,

> for social networks, $d = 100$ is sufficient in the sense that the core behaves like an expander, because increasing d does not affect the expander property of the core. On the other hand, for web graphs, we need to increase d (i.e., the core for the case where $d = 1000$ gets closer to an expander than that for the case where $d = 100$).

6.4 Density of Cores

Let us look at the density of a core with respect to the width d of our core-tree-decomposition for social networks and web graphs. Surprisingly, there is a large (and interesting) difference between social networks and web graphs. For web graphs, as the width d increases, the density continues to increase. On the other hand, for social networks, as the width d increases, the density first increases, but then remains constant for a while (or increase very gradually), as shown in Figure 8.

7. CONCLUSION

In this paper, using the notion of *core-tree-decomposition*, we obtained many structural properties for social networks and web graphs. Specifically, we performed experiments to construct a core-tree-decomposition for as many as 40 publicly available datasets, and we conclude that whiskers are "tree-like," which can be explained in the framework of tree-with; moreover, the intuition that the cores should be "expander-like" can be explained via their eigenvalue distribution. We have also shown that social networks tend to have larger cores. Indeed, for social networks, the size of cores is a maximum of 23% of the vertices, but for web graphs, it is at most 13% of the vertices. Additionally, we have also demonstrated the theoretical and empirical evidence that tree-width plays a significant role in the efficiency of certain types of algorithms.

8. REFERENCES

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Fast exact shortest-path distance queries on large networks by pruned landmark labeling. In *SIGMOD*, pages 349–360, 2013.

[2] T. Akiba, Y. Iwata, and Y. Yoshida. Dynamic and historical shortest-path distance queries on large

evolving networks by pruned landmark labeling. In *WWW*, pages 237–248, 2014.

[3] T. Akiba, C. Sommer, and K. Kawarabayashi. Shortest-path queries for complex networks: exploiting low tree-width outside the core. In *EDBT*, pages 144–155, 2012.

[4] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 2:11–24, 1989.

[5] A. Berry, P. Heggernes, and G. Simonet. The minimum degree heuristic and the minimal triangulation process. In *WG*, volume 2880 of *LNCS*, pages 58–70. 2003.

[6] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[7] P. Boldi, B. Codenotti, M. Santini, and S. Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software Pract. Ex.*, 34(8):711–726, 2004.

[8] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, pages 587–596, 2011.

[9] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, pages 595–601, 2004.

[10] F. Chung. *Spectral Graph Theory*. Cbms Regional Conference Series in Mathematics, 1997.

[11] R. Dechter and J. Pearl. Tree clustering for constraint networks. *Artif. Intell.*, 38(3):353–366, 1989.

[12] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[13] M. Frick and M. Grohe. Deciding first-order properties of locally tree-decomposable structures. *J. ACM*, 48(6):1184–1206, 2001.

[14] A. W.-C. Fu, H. Wu, J. Cheng, and R. C.-W. Wong. Is-label: an independent-set based labeling scheme for point-to-point distance querying. *PVLDB*, 6(6):457–468, 2013.

[15] C. Gavoille, D. Peleg, S. Pérennes, and R. Raz. Distance labeling in graphs. *J. Algorithms*, 53(1):85 – 112, 2004.

[16] M. Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), 2007.

[17] M. Grohe, K. Kawarabayashi, and B. A. Reed. A simple algorithm for the graph minor decomposition - logic meets structural graph theory. In *SODA*, pages 414–431, 2013.

[18] R. Halin. s-function for graphs. *J. Geometry*, 8:171–186, 1976.

[19] P. Jégou and C. Terrioux. Hybrid backtracking bounded by tree-decomposition of constraint networks. *Artif. Intell.*, 146(1):43–75, 2003.

[20] M. R. Jerrum and A. Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18:1149–1178, 1989.

[21] K. Kawarabayashi and P. Wollan. A shorter proof of the graph minor algorithm: the unique linkage theorem. In *STOC*, pages 687–694, 2010.

[22] K. Kawarabayashi and P. Wollan. A simpler algorithm and shorter proof for the graph minor decomposition. In *STOC*, pages 451–458, 2011.

[23] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. R. Stat. Soc. Series B, Stat. Methodol.*, pages 157–224, 1988.

[24] J. Leskovec, K. Lang, A. Dasgupta, and M. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Math.*, 6(1):29–123, 2009.

[25] T. Maehara, T. Akiba, Y. Iwata, and K. Kawarabayashi. Computing personalized pagerank quickly by exploiting graph structures. *PVLDB*, 7, 2014. to appear.

[26] A. Mislove, M. Marcon, K. P. Gummadi, P. Druschel, and B. Bhattacharjee. Measurement and analysis of online social networks. In *IMC*, pages 29–42, 2007.

[27] L. Perković and B. Reed. An improved algorithm for finding tree decompositions of small width. *Int. J. Found. Comput. Sci.*, 11(03):365–371, 2000.

[28] B. Reed. Tree width and tangles: a new connectivity measure and some applications. *Surveys in Combinatorics*, 241:87–162, 1997.

[29] N. Robertson and P. D. Seymour. Graph minors. iii. planar tree-width. *J. Comb. Theory, Ser. B*, 36:49–63, 1984.

[30] N. Robertson and P. D. Seymour. Graph minors .xiii. the disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[31] Y. Saad. *Numerical methods for large eigenvalue problems*, volume 158. SIAM, 1992.

[32] D. Sheldon, T. Sun, A. Kumar, and T. G. Dietterich. Approximate inference in collective graphical models. In *ICML*, pages 1004–1012, 2013.

[33] F. Wei. Tedi: efficient shortest path query answering on graphs. In *SIGMOD*, pages 99–110, 2010.

[34] J. Xu, F. Jiao, and B. Berger. A tree-decomposition approach to protein structure prediction. In *CSB*, pages 247–256, 2005.

[35] Y. Zheng, P. Chen, and J.-Z. Cao. Map-mrf inference based on extended junction tree representation. In *CVPR*, pages 1696–1703, 2012.