

# Beyond ‘Caveman Communities’: Hubs and Spokes for Graph Compression and Mining

U Kang and Christos Faloutsos  
Carnegie Mellon University  
{ukang, christos}@cs.cmu.edu

*Abstract*—Given a real world graph, how should we lay-out its edges? How can we compress it? These questions are closely related, and the typical approach so far is to find clique-like communities, like the ‘cavemen graph’, and compress them. We show that the block-diagonal mental image of the ‘cavemen graph’ is the wrong paradigm, in full agreement with earlier results that real world graphs have no good cuts. Instead, we propose to envision graphs as a collection of hubs connecting spokes, with super-hubs connecting the hubs, and so on, recursively.

Based on the idea, we propose the SLASHBURN method (burn the hubs, and slash the remaining graph into smaller connected components). Our view point has several advantages: (a) it avoids the ‘no good cuts’ problem, (b) it gives better compression, and (c) it leads to faster execution times for matrix-vector operations, which are the back-bone of most graph processing tools.

Experimental results show that our SLASHBURN method consistently outperforms other methods on all datasets, giving good compression and faster running time.

*Keywords*-Graph Compression, Graph Mining, Hubs and Spokes

## I. INTRODUCTION

How can we compress graphs efficiently? How can we find communities in graphs? The two questions are closely related: if we find good communities, then we can compress the graph well since the nodes in the same community have redundancies (e.g. similar neighborhood) which help us shrink the size of the data (and thus, also shrink the I/O and communication costs for graph processing). Similarly, good compression implies good communities. The traditional research focus was on finding homogeneous regions in the graph so that nodes inside a region are tightly connected to each other than to nodes in other regions. In other words, the focus was to search for ‘caveman communities’ where a person in a cave knows others in the same cave very well, while knows very little about persons in different caves as shown in Figure 1(a). In terms of the adjacency matrix, the goal was to find an ordering of nodes so that the adjacency matrix is close to block-diagonal, containing more ‘square’ blocks as in Figure 1(b). Spectral clustering [1], [2], co-clustering [3], cross-associations [4], and shingle-ordering [5] are typical examples for such approaches.

However, real world graphs are much more complicated and inter-connected than caveman graphs. It is well known

that most real world graphs follow power-law degree distributions with few ‘hub’ nodes having very high degrees and majority of the nodes having low degrees [6]. These hub nodes break the assumption of caveman-like communities since the hubs are well connected to most of the nodes in graphs, effectively combining all the caves into a huge cave. Thus, it is not surprising that well defined communities in real world networks are hard to find [7].

In this paper, we propose a novel approach to finding communities and compressions in graphs. Our approach, called SLASHBURN, is to exploit the hubs and the neighbors (‘spokes’) of the hubs to define an alternative community different from the traditional community. SLASHBURN is based on the observation that real world graphs are easily disconnected by hubs, or high degree nodes: removing hubs from a graph creates many small disconnected components, and the remaining giant connected component is substantially smaller than the original graph. The communities defined using hubs and spokes correspond to skinny blocks in an adjacency matrix as shown in Figure 1(d), in contrast to the square blocks in caveman communities as shown in Figure 1(b). We show that these hubs and spokes can be carefully ordered to get a compact representation of the adjacency matrix, which in turn leads to good compression. Our contributions are the following:

- 1) **Paradigm shift.** Instead of looking for near-cliques (‘caves’), we look for hubs and spokes for a good graph compression. Our approach is much more suitable for real world, power-law graphs like social networks.
- 2) **Compression.** We show that our method gives good compression results when applied on real world graphs, consistently outperforming other methods on all datasets.
- 3) **Speed.** Our method boosts the performance of matrix-vector multiplication of graph adjacency matrices, which is the building block for various algorithms like PageRank, connected components, etc.

The rest of the paper is organized as follows. Section II precisely describes the problem and our proposed method for laying out edges for better compressing graphs. We give experimental results in Section III, showing the compression

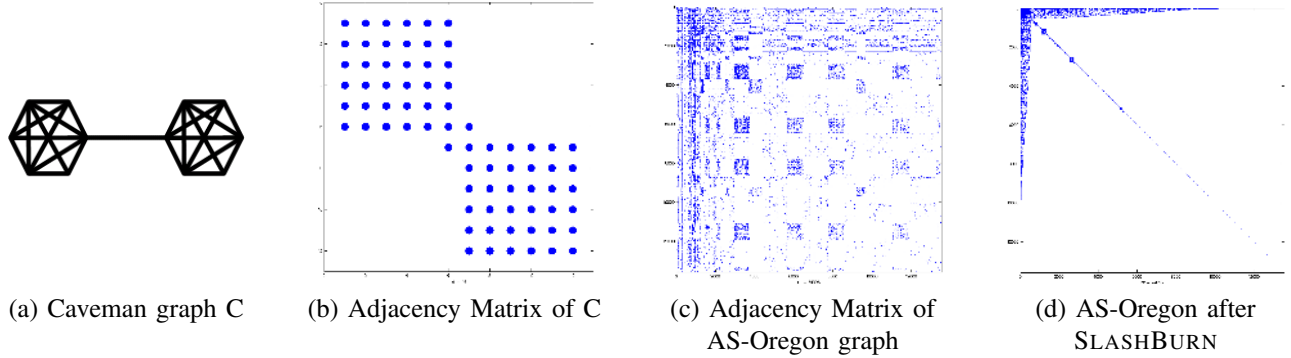


Figure 1. Caveman graph, real-world graph, and the result from our proposed SLASHBURN ordering. Real world graphs are much more complicated and inter-connected than caveman graph, with few ‘hub’ nodes having high degrees and majority of nodes having low degrees. Finding a good ‘cut’ on real world graphs to extract homogeneous regions (like the square diagonal blocks in the caveman adjacency matrix (b)) is difficult due to the hub nodes. Instead, our proposed SLASHBURN finds novel ‘skinny’ communities which lead to good compression: in (d), the edges are concentrated to the left, top, and diagonal lines while making empty spaces in most of the areas.

Symbol	Definition
$G$	A graph.
$V$	Set of nodes in a graph.
$E$	Set of edges in a graph.
$A$	Adjacency matrix of a graph.
$n$	Number of nodes in a graph.
$GCC$	Giant connected component of a graph.
$k$	Number of hub nodes to slash per iteration in SLASHBURN.
$w(G)$	Wing width ratio of a graph $G$ , meaning the ratio of the number of total hub nodes to $n$ .
$b$	Block width used for block based matrix-vector multiplication.

Table I  
TABLE OF SYMBOLS

and running time enhancements. After discussing related works on Section IV, we conclude in Section V.

To enhance the readability of this paper, we listed the symbols frequently used in this paper in Table I.

## II. PROPOSED METHOD

In this section, we give a formal definition of the problem, describe our proposed method, and analyze its complexity.

### A. Problem Definition

Given a large graph, we want to layout its edges so that the graph can be compressed well, and graph mining queries can be answered quickly. Specifically, we consider the application of large scale matrix-vector multiplication which is the building block of many graph mining algorithms including PageRank, diameter estimation, and connected components [8], [9]. The state-of-the-art method for the large scale matrix-vector multiplication is the block multiplication method [8], where the original matrix is divided into  $b$  by  $b$  square matrix blocks, the original vector is divided into

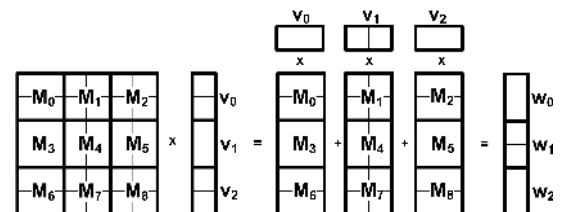


Figure 2. Block method for large scale matrix-vector multiplication. The original 6 by 6 matrix is divided into 2 by 2 square matrix blocks ( $M_0$  to  $M_8$ ), the original length 6 vector is divided into length 2 vector blocks ( $v_0$  to  $v_2$ ), and the blocks are multiplied to get the resulting vector ( $w_0$  to  $w_2$ ).

length  $b$  vector blocks, and the matrix-vector blocks are multiplied.

For example, see Figure 2 for the block multiplication method where a 6 by 6 matrix is multiplied with a length 6 vector using 2 by 2 matrix blocks and length 2 vector blocks. We assume that each block is stored independently from each other, without requiring neighbor or reciprocal blocks to decode its edges, since such independency among blocks allows more scalable processing in large scale, distributed platforms like MAPREDUCE [10].

In this scenario, it is desired that the adjacency matrix has clustered edges: *smaller* number of *denser* blocks is better than *larger* number of *sparser* blocks. There are two reasons for this. First, smaller number of denser blocks reduces the number of disk accesses. Second, it provides better opportunity for compression. For example, see Figure 3. The left matrix is the adjacency matrix of Figure 1(a) with a random ordering of nodes, while the right matrix is the adjacency matrix of the same graph with a compression-friendly ordering where nodes 1 to 6 are assigned to the left clique, and nodes 7 to 12 are assigned to the right clique. Assume we use 2 by 2 blocks to cover all the nonzero elements inside the matrix. Then the right matrix requires

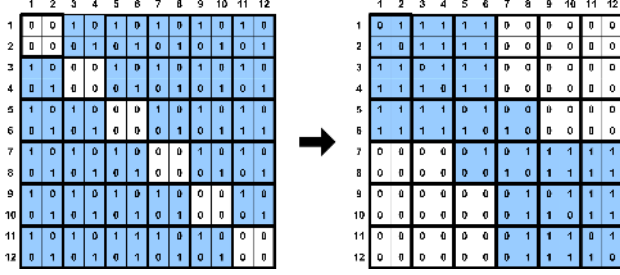


Figure 3. Importance of ordering. Left: adjacency matrix of Figure 1(a) with a random ordering of nodes. Right: adjacency matrix of the same graph, but with a compression-friendly ordering, where nodes 1 to 6 are assigned to the left clique, and nodes 7 to 12 are assigned to the right clique. If we use 2 by 2 blocks to cover all the nonzero elements inside the matrix, the right matrix requires smaller number of denser blocks which lead to better compression.

smaller number of blocks than the left matrix. Furthermore, each block in the right matrix is denser than the one in the left matrix, which could lead to better compression of graphs.

Formally, our main problem is as follows.

*Problem 1:* Given a graph with the adjacency matrix  $A$ , find a permutation  $\pi : V \rightarrow [n]$  such that the storage cost function  $cost(A)$  is minimized.

The notation  $[n]$  means the ordering of  $n$  nodes. Following the motivation that *smaller* number of *denser* blocks is better for compression than *larger* number of *sparser* blocks, the first cost function we consider is the number of nonempty,  $b$  by  $b$  square blocks in the adjacency matrix:

$$cost_{nz}(A, b) = \text{number of nonempty blocks}, \quad (1)$$

where  $b$  is the block width. The second, and more precise cost function uses the required number of bits to encode the adjacency matrix using a block-wise encoding (divide the matrix into blocks, and encode each block using standard compression algorithms like gzip). The required bits are decomposed into two parts: one for the nonzero elements inside blocks, the other for storing the meta information about the blocks.

- *Nonzeros inside blocks.* Bits to compress nonzero elements inside blocks.
- *Meta information on blocks.* Bits to store the row and column ids of blocks.

Using the decomposition, we define a cost function  $cost_{it}(A, b)$  assuming a compression method achieving the information theoretic lower bound [11], [4]:

$$cost_{it}(A, b) = |T| \cdot 2 \log \frac{n}{b} + \sum_{\tau \in T} b^2 \cdot H\left(\frac{z(\tau)}{b^2}\right), \quad (2)$$

where  $n$  is the number of nodes,  $T$  is the set of nonempty blocks of size  $b$  by  $b$ ,  $z(\tau)$  is the number of nonzero elements

within a block  $\tau$ , and  $H(p) = p \log \frac{1}{p} + (1-p) \log \frac{1}{1-p}$  is the binary Shannon entropy function. The first term  $|T| \cdot 2 \log \frac{n}{b}$  in Equation (2) represents the bits to encode the meta information on blocks. Since each block requires two  $\log \frac{n}{b}$  bits to encode the block row and the block column ids, the total required bits are  $|T| \cdot 2 \log \frac{n}{b}$ . The second term in Equation (2) is the bits to store nonzeros inside blocks: we use information theoretic lower bound for encoding the bits, since it gives the minimum number of bits achievable by any coding methods. Note  $b^2$  is the maximum possible edge counts in a  $b$  by  $b$  block, and  $\frac{z(\tau)}{b^2}$  is the density of the block.

The two cost functions defined in Equation (1) and (2) will be evaluated and compared on different ordering methods in Section III.

### B. Why Not Classic Partitioning?

In general, directly minimizing the cost functions is a difficult combinatorial problem which could require  $n!$  trials in the worst case. Traditional approach is to use graph partitioning algorithms to find good ‘cuts’ and homogeneous regions so that nodes inside a region form a dense community, and thereby leading to better compressions. Examples include spectral clustering [1], [2], co-clustering [3], cross-associations [4], and shingle-ordering [5]. However, such approaches don’t work well for real world, power law graphs since there exists no good cuts in such graphs [7], which we also experimentally show in Section III.

The reason of the ‘no good cut’ in most real world graphs is explained by their power-law degree distributions and the existence of ‘hub’ nodes. Such hub nodes combine the communities to blend into each other, making the cut-based algorithms fail. Rather than resorting to the cut-based algorithms that are not designed to work on power-law graphs, we take a novel approach to finding communities and compressions, which we explain next.

### C. Graph Shattering

As described in the previous section, finding homogeneous regions in real world graphs is infeasible due to hub nodes. Our main idea to solve the problem is to exploit the hubs to define an alternative community different from the traditional community. Remember that most real-world graphs have a power law in its degree distribution: there exist few hub nodes with very high degrees, while majority of the nodes having low degrees, as shown in Figure 4(a). In contrast, random graphs have degree distributions whose tails drop exponentially: this means there doesn’t exist hubs with extremely high degrees, as shown in Figure 4(b).

We start with an observation that real-world graphs are easily shattered by removing hub nodes from them: while majority of the nodes still belong to the giant connected component, a nontrivial portion of the nodes belong to small disconnected components by the removal. The nodes belonging to the small disconnected components after the

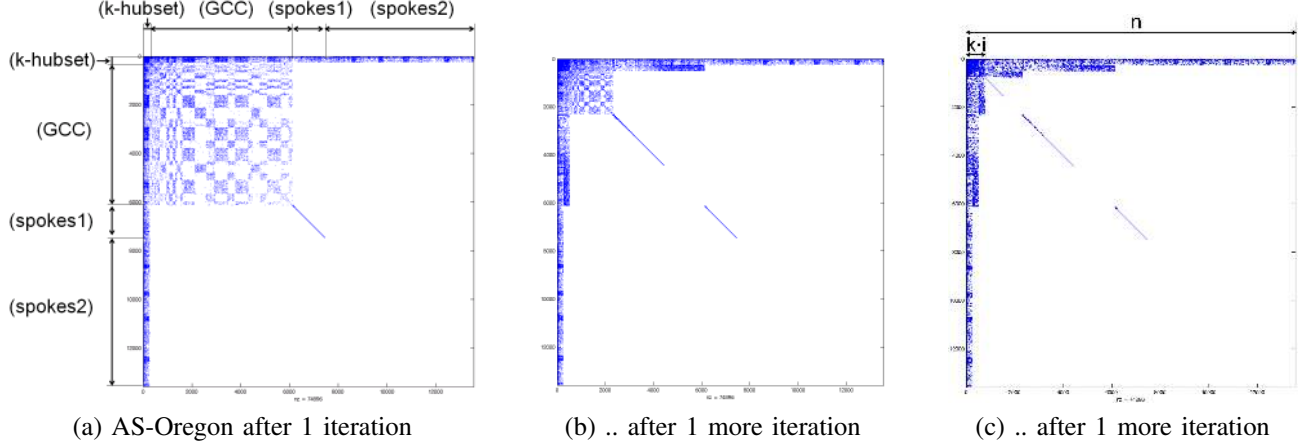


Figure 5. SLASHBURN in action: adjacency matrices of AS-Oregon graph after applying SLASHBURN ordering. After 1 iteration, the nodes are decomposed into  $k$ -hubset, GCC, and the spokes. The spokes are only connected to  $k$ -hubset, while completely disconnected to the GCC, which makes large empty spaces in the bottom-right area of the adjacency matrix. The same process applies to the remaining GCC recursively. Notice that the nonzero elements in the matrix are concentrated to the left, top, and diagonal lines of the matrix, making an arrow-like shape. Compared to the original adjacency matrix in Figure 1(c), the final matrix has much larger empty spaces, enabling better compression.

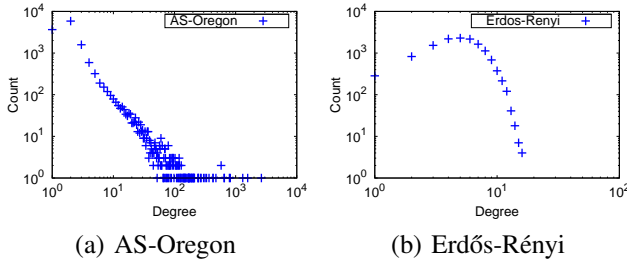


Figure 4. Degree distributions of power-law vs. random graphs. The left graph is a real-world graph showing a power law degree distribution with few hub nodes having very high degrees, and majority of nodes having low degrees. The right graph is a random(Erdős-Rényi) graph having the same number of nodes and edges as the left graph. The random graph has an exponential tail in its degree distribution without distinct hubs.

removal of the hub nodes can be regarded as satellite nodes connected to the hub nodes. In other words, those satellite nodes have links only to the hub nodes, and completely disconnected from the rest of the nodes in the graph. This is the exact property we are utilizing.

To precisely describe our method, we define related terms.

**Definition 1 ( $k$ -hubset):** The  $k$ -hubset of a graph  $G$  is the set of nodes with top  $k$  highest centrality scores.

We use the degree of a node as the centrality score in this paper, but any centrality(e.g., closeness, betweenness [12], PageRank, eigendrop [13], etc.) can be used for the score. Removing  $k$ -hubset from a graph leads to the definition of  $k$ -shattering.

**Definition 2 ( $k$ -shattering):** The  $k$ -shattering of a graph  $G$  is the process of removing the nodes in  $k$ -hubset, as well as edges incident to  $k$ -hubset, from  $G$ .

Let's consider the following shattering process. Given a graph  $G$ , we perform a  $k$ -shattering on  $G$ . Among the re-

maining connected components, choose the giant connected component(GCC). Perform a  $k$ -shattering on the GCC, and do the whole process recursively. Eventually, we stop at a stage where the size of the GCC is less than or equal to  $k$ . A natural question is, how quickly is a graph shattered? To measure the speed of the shattering process, we define the wing width ratio  $w(G)$  of a graph  $G$ .

**Definition 3:** The wing width ratio  $w(G)$  of a graph  $G$  is  $\frac{k \cdot i}{n}$  where  $k$  is the number used for the  $k$ -shattering,  $i$  is the number of iterations until the shattering finishes, and  $n$  is the number of nodes in  $G$ .

Intuitively, the wing width ratio  $w(G)$  corresponds to the width of the blue wing of the typical spyplot (visualization of the adjacency matrix; see Figure 5(c)); notice that for all real world graphs, the corresponding spyplots look like ultra-modern airplanes, with the blue lines being their wings.  $w(G)$  is the ratio of 'wing' width to the number of nodes in the graph. A low  $w(G)$  implies that the graph  $G$  is shattered quickly, while a high  $w(G)$  implies that it takes long to shatter  $G$ . As we will see in Section III-C, real-world, power-law graphs have low  $w(G)$ . Our proposed SLASHBURN method utilizes the low wing width ratio in real world graphs.

#### D. Slash-and-Burn

In this section, we describe SLASHBURN, our proposed ordering method for compressing graphs. Given a graph  $G$ , the SLASHBURN method defines a permutation  $\pi : V \rightarrow [n]$  of a graph so that nonzero elements in the adjacency matrix of  $G$  are grouped together. Algorithm 1 shows the high-level idea of SLASHBURN.

The lines 1 and 2 removes('slash-and-burn') top  $k$  highest centrality scoring nodes, thereby decomposing nodes in  $G$  into the following three groups:

---

**Algorithm 1: SLASHBURN**

---

**Input:** Edge set  $E$  of a graph  $G = (V, E)$ ,  
a constant  $k$ (default = 1).

**Output:** Array  $\Gamma$  containing the ordering  $V \rightarrow [n]$ .

- 1: Remove  $k$ -hubset from  $G$  to make the new graph  $G'$ . Add the removed  $k$ -hubset to the front of  $\Gamma$ .
  - 2: Find connected components in  $G'$ . Add nodes in non-giant connected components to the back of  $\Gamma$ , in the decreasing order of sizes of connected components they belong to.
  - 3: Set  $G$  to be the giant connected component(GCC) of  $G'$ . Go to step 1 and continue, until the number of nodes in the GCC is smaller than  $k$ .
- 

- $k$ -hubset: top  $k$  highest centrality scoring nodes in  $G$ .
- GCC: nodes belonging to the giant connected component of  $G'$ . Colored blue in Figure 6.
- Spokes to the  $k$ -hubset: nodes belonging to the non-giant connected component of  $G'$ . Colored green in Figure 6.

Figure 6 shows a graph before and after 1 iteration of SLASHBURN. After removing the ‘hub’ node at the center, the graph is decomposed into the GCC and the remaining ‘spokes’ which we define to be the non-giant connected component connected to the hubs. The hub node gets the lowest id(1), the nodes in the spokes get the highest ids(9~16)) in the decreasing order of the connected component size they belong to, and the GCC takes the remaining ids(2~8). The same process applies to the nodes in GCC, recursively.

Figure 5(a) shows the AS-Oregon graph after the lines 1 and 2 are executed for the first time with  $k = 256$ . In the figure, we see that a  $k$ -hubset comes first with GCC and spokes following after them. The difference between (spokes1) and (spokes2) is that the nodes in (spokes2) are connected only to some of the nodes in  $k$ -hubset, thereby making large empty spaces in the adjacency matrix. Notice also that nodes in (spokes1) make a thin diagonal line, corresponding to the edges among themselves. A remarkable result is that the remaining GCC takes only 45% of the nodes in the original graph, after removing 256(=1.8 %) high degree nodes. Figure 5(b) and (c) shows the adjacency matrix after doing the same operation on the remaining GCC, recursively. Observe that nonzero elements in the final adjacency matrix are concentrated on the left, top, and diagonal lines of the adjacency matrix, creating an arrow-like shape. Observe also that the final matrix has huge empty spaces which could be utilized for better compression, since the empty spaces need not be stored.

An advantage of our SLASHBURN method is that it works on any power-law graphs without requiring any domain-specific knowledge or a well defined natural ordering on the graph for better permutation. Finally, we note that setting  $k$

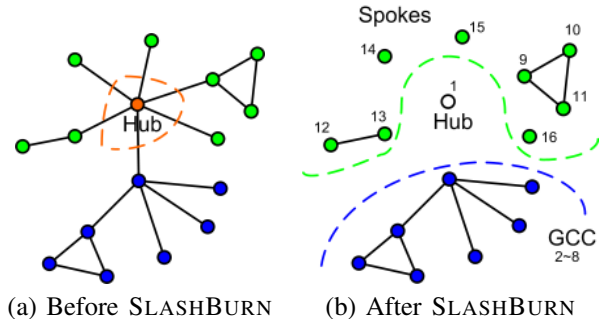


Figure 6. [Best viewed in color.] A graph before and after 1 iteration of SLASHBURN. Removing a hub node creates many smaller ‘spokes’, and the GCC. The hub node gets the lowest id(1), the nodes in the spokes get the highest ids(9~16)) in the decreasing order of the connected component size they belong to, and the GCC takes the remaining ids(2~8). The next iteration starts on the GCC.

to 1 often gives the best compression by making the wing width ratio  $w(G)$  minimum or close to minimum. However, setting  $k$  to 1 requires many iterations and longer running time. We found that setting  $k$  to 0.5% of the number of nodes gives good compression results with small number of iterations on most real world graphs.

### E. Analysis

We analyze the time and the space complexities of the SLASHBURN algorithm.

**Lemma 1 (Time Complexity of SLASHBURN):**

SLASHBURN takes  $O(|E| + |V|\log|V|)i$  time where  $i = \frac{|V| \cdot w(G)}{k}$  is the number of iterations.

*Proof:* In Algorithm 1, step 1 takes  $O(|V| + |E|)$  time to compute the degree of nodes, and to remove  $k$ -hubset. Step 2 requires  $O(|E| + |V|\log|V|)$  time since connected components require  $O(|V| + |E|)$  time, and sorting takes  $|V|\log|V|$  time. Thus, 1 iteration of SLASHBURN takes  $O(|E| + |V|\log|V|)$  time, and the lemma is proved by multiplying the number  $i$  of iterations to it. ■

Lemma 1 implies that smaller wing width ratio  $w(G)$  will result in faster running time. We note that real world, power-law graphs have small wing width ratio, which we show experimentally in Section III-C.

For space complexity, we have the following result.

**Lemma 2 (Space Complexity of SLASHBURN):**

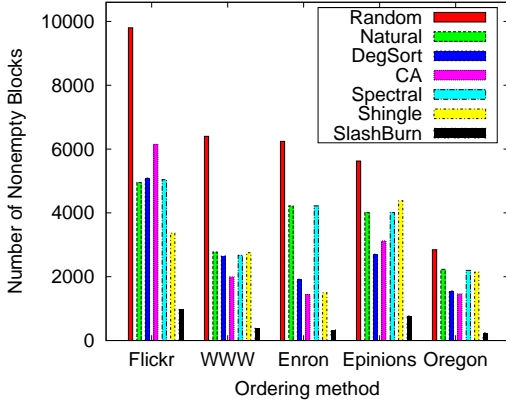
SLASHBURN requires  $O(V)$  space.

*Proof:* In step 1, computing the degree requires  $O(V)$  space. In step 2, connected component requires  $O(V)$  space, and sorting requires at most  $O(V)$  space. The lemma is proved by combining the space requirements for the two steps. ■

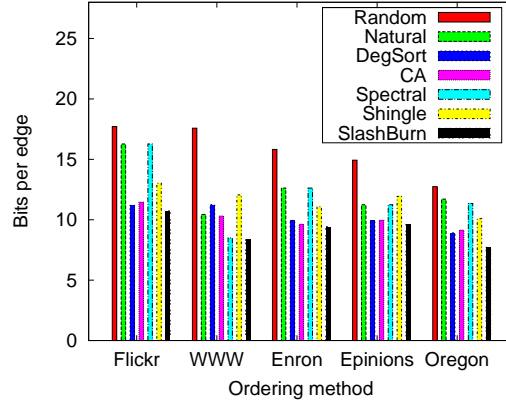
## III. EXPERIMENTS

In this section, we present experimental results to answer the following questions:





(a)  $cost_{nz}(A, b)$ : number of nonempty blocks



(b)  $cost_{it}(A, b)$ : information theoretic cost

Figure 7. Compression comparison of ordering methods. DegSort: degree sort, CA: cross association, and Spectral: spectral clustering. For all the cost functions, SLASHBURN performs the best. (a): SLASHBURN reduces the number of nonempty blocks by up to  $20\times$  compared to the random ordering, and by up to  $6.1\times$  compared to the second best orderings. (b): SLASHBURN reduces the bits per edge by up to  $2.1\times$  compared to the random ordering, and by up to  $1.2\times$  compared to the second best orderings.

Name	Nodes	Edges	Description
LiveJournal	4,847,571	68,993,773	Friendship social network
Flickr	404,733	2,110,078	Social network
WWW-Barabasi	325,729	1,497,134	WWW pages in nd.edu
Enron	80,163	312,725	Enron email
Epinions	75,888	508,960	Who trusts whom
AS-Oregon	13,579	74,896	Router connections

Table II

SUMMARY OF GRAPHS USED. AS-OREGON IS AN UNDIRECTED GRAPH, WHILE ALL OTHERS ARE DIRECTED GRAPHS.

**Q1** How well does SLASHBURN compress graphs compared to other methods?

**Q2** How does SLASHBURN decrease the running time of large scale matrix-vector multiplication?

**Q3** How quickly can we shatter real world graphs? What are the wing width ratio of real world, power-law graphs?

We compare SLASHBURN with the following six methods.

- **Random.** Random ordering of the nodes.
- **Natural.** Natural ordering of the nodes, that is, the original adjacency matrix. For some graphs, the natural ordering provides high locality among consecutive nodes (e.g. lexicographic ordering in Web graphs [14]).
- **Degree Sort(DegSort).** Ordering based on the decreasing degree of the nodes.
- **Cross Association(CA).** Cross-association [4] based ordering so that nodes in a same group are numbered consecutively.
- **Spectral Clustering.** Normalized spectral clustering [1], also known as the normalized cut. Order

nodes by the second smallest eigenvector score of a generalized eigenvector problem.

- **Shingle.** Shingle ordering is the most recent method for compressing social networks [5]. It groups nodes with similar fingerprints(min-wise hashes) obtained from the out-neighbors of nodes.

The graphs used in our experiments along with their descriptions are summarized in Table II.

### A. Compression

We compare the ordering methods based on the cost of compression using the two cost functions defined in Equation (1) and (2) of Section II:

- $cost_{nz}(A, b)$ : number of nonempty blocks.
- $cost_{it}(A, b)$ : required bits using information-theoretic coding methods.

Figure 7 shows the costs of ordering methods. Figure 7(a) shows the number of nonempty blocks( $cost_{nz}(A)$ ), and Figure 7(b) shows the bits per edge computed using  $cost_{it}(A, b)$ . The exact numbers are listed in Table III and IV, respectively. Notice that for all the cost functions, SLASHBURN performs the best. For the number of nonempty blocks, SLASHBURN reduces the counts by up to  $20\times$  compared to the random ordering, and by up to  $6.1\times$  compared to the second best orderings. For the bits per edge, SLASHBURN reduces the bits by up to  $2.1\times$  compared to the random ordering, and by up to  $1.2\times$  compared to the second best orderings.

The amount of compression can be checked visually. Figure 8 show the spyplots, which are nonzero patterns in the adjacency matrices, of real world graphs permuted from the different ordering methods. Random ordering makes the spyplot almost filled; natural ordering provides more

Graph	Random	Natural	Degree Sort	Cross Association	Spectral Clustering	Shingle	SLASHBURN
LiveJournal (bw=4096)	1401856	1060774	885153	*	*	960642	<b>873469</b>
Flickr (bw=4096)	9801	4950	5091	6149	5042	3366	<b>994</b>
WWW-Barabasi (bw=4096)	6400	2774	2647	1997	2671	2751	<b>384</b>
Enron (bw=1024)	6241	4220	1922	1442	4220	1498	<b>339</b>
Epinions (bw=1024)	5624	4010	2703	3124	4010	4381	<b>768</b>
AS-Oregon (bw=256)	2845	2232	1552	1463	2197	2142	<b>239</b>

Table III

NUMBER OF NONEMPTY BLOCKS FOR THE COMPETING ORDERING METHODS. ‘bw’ DENOTES THE BLOCK WIDTH, AND THE WINNERS ARE IN BOLD FONTS. FOR THE LIVEJOURNAL DATA, CROSS ASSOCIATION AND SPECTRAL CLUSTERING(MARKED \*) COULD NOT BE PERFORMED SINCE THE ALGORITHMS ARE TOO HEAVY TO RUN ON SUCH A LARGE GRAPH. NOTICE THAT SLASHBURN, FORMATTED IN BOLD FONTS, OUTPERFORMS ALL OTHERS. THE RESULTS WERE SIMILAR FOR OTHER BLOCK WIDTHS.

Graph	Random	Natural	Degree Sort	Cross Association	Spectral Clustering	Shingle	SLASHBURN
LiveJournal	19.89	16.82	16.87	*	*	18.52	<b>16.67</b>
Flickr	17.71	16.27	11.19	11.45	16.27	13.02	<b>10.73</b>
WWW-Barabasi	17.58	10.43	11.25	10.32	8.5	12.06	<b>8.41</b>
Enron	15.82	12.62	9.94	9.63	12.62	11.08	<b>9.43</b>
Epinions	14.93	11.24	9.93	9.96	11.24	11.93	<b>9.61</b>
AS-Oregon	12.74	11.71	8.92	9.14	11.34	10.09	<b>7.71</b>

Table IV

BITS PER EDGE FOR THE COMPETING ORDERING METHODS, ACCORDING TO THE INFORMATION THEORETIC LOWER BOUND. FOR THE LIVEJOURNAL DATA, CROSS ASSOCIATION AND SPECTRAL CLUSTERING(MARKED \*) COULD NOT BE PERFORMED SINCE THE ALGORITHMS ARE TOO HEAVY TO RUN ON SUCH A LARGE GRAPH. NOTE THAT THE RESULT FROM SLASHBURN, FORMATTED IN BOLD FONTS, OUTPERFORMS ALL OTHERS.

empty space than random ordering, meaning that the natural ordering exploits some form of localities. Degree sort makes the upper-left area of the adjacency matrix more dense. Cross association makes many rectangular regions that are homogeneous. Spectral clustering tries to find good cuts, but obviously can’t find such cuts on the real world graphs. In fact, for all the graphs except AS-Oregon in Figure 8, the spyplot after the spectral clustering looks very similar to that of the natural ordering. Shingle ordering makes empty spaces on the top portion of the adjacency matrix of some graphs: the rows of such empty spaces correspond to nodes without outgoing neighbors. However, the remaining bottom portion is not concentrated well. Our SLASHBURN method collects nonzero elements to the left, top, and the diagonal lines of the adjacency matrix, thereby making an arrow-like shape. Notice that SLASHBURN requires the smallest number of square blocks to cover the edges, leading to the best compression as shown in Table IV.

### B. Running Time

We show the performance implication of SLASHBURN for large scale graph mining on distributed platform, using HADOOP, an open source MAPREDUCE framework. We test the performance of block-based PageRank using HADOOP on graphs created from different ordering methods. For storing blocks, we used the standard gzip algorithm to compress the 0-1 bit sequences. Figure 9 shows file size vs. running time on different ordering methods on LiveJournal

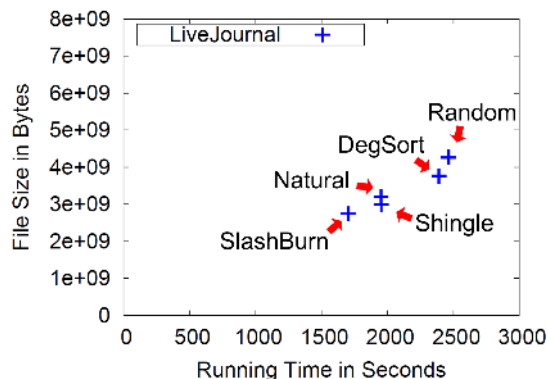


Figure 9. File size vs. running time of different ordering methods on LiveJournal graph. The running time is measured for one iteration of PageRank on HADOOP. Notice that SLASHBURN results in the smallest file size, as well as the smallest running time.

graph. The running time is measured for one iteration of PageRank on HADOOP. Notice that SLASHBURN results in the smallest file size, as well as the smallest running time. We note that LiveJournal is one of the dataset that is very hard to compress. In fact, a similar dataset was analyzed in the paper that proposed the shingle ordering [5]: however, their proposed ‘compression’ method *increased* the bits per edge, compared to the original graph. Our SLASHBURN outperforms all other methods, including the shingle and the natural ordering, even on this ‘hard to compress’ dataset.

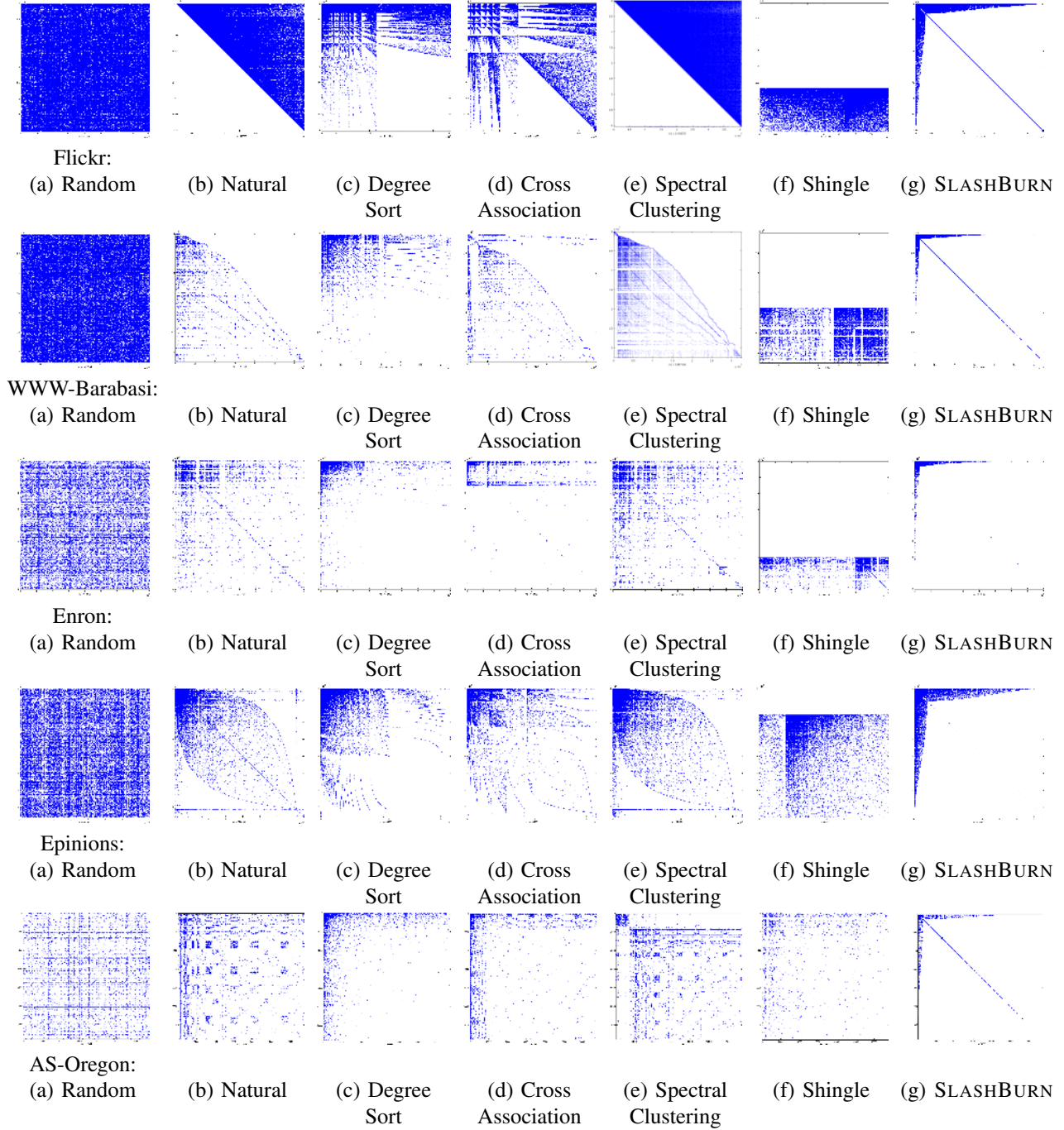


Figure 8. Adjacency matrix of real world graphs on different ordering methods. Random ordering requires the maximum number of square blocks to cover the edges. Natural ordering requires smaller number of blocks, implying that the natural ordering exploits some form of localities. Degree sort makes the upper-left area of the adjacency matrix more dense. Cross association makes homogeneous square regions. Spectral clustering tries to find good cuts, but obviously can't find such cuts on the real world graphs. Shingle ordering makes empty spaces on the top portion of the adjacency matrix of some graphs. The rows of such empty spaces correspond to nodes without outgoing neighbors. However, the remaining bottom portion is not concentrated well. In fact, for all the graphs except AS-Oregon, the spyplot after the spectral clustering looks very similar to that of the natural ordering. Our SLASHBURN method concentrates edges to the left, top, and the diagonal lines of the adjacency matrix, thereby making an arrow-like shape. Notice that SLASHBURN requires the smallest number of square blocks to cover the edges, leading to the best compression as shown in Table IV.



Graph Type	Graph	$w(G)$
Real world	Flickr	0.078
Real world	WWW-Barabasi	0.037
Real world	Enron	0.044
Real world	Epinions	0.099
Real world	AS-Oregon	0.040
Erdős-Rényi	ER-Epinions	0.611
Erdős-Rényi	ER-AS-Oregon	0.358

Table V  
WING WIDTH RATIO  $w(G)$  OF REAL WORLD AND RANDOM(ERDŐS-RÉNYI) GRAPHS. NOTICE THAT  $w(G)$ 'S ARE SMALL FOR ALL THE REAL WORLD GRAPHS, MEANING THAT SLASHBURN WORKS WELL ON SUCH GRAPHS. IN CONTRAST, RANDOM GRAPHS HAVE HIGH  $w(G)$  (AT LEAST  $6.2\times$  LARGER THAN THEIR REAL WORLD COUNTERPARTS), MEANING THAT IT CAN NOT BE SHATTERED QUICKLY.

### C. Real World Graphs Shatter Quickly

How quickly can a *real world* graph be shattered into tiny components? What are the differences of the wing width ratio between real world, power-law graphs and random(Erdős-Rényi [15]) graphs? Table V shows the wing width ratio  $w(G)$  of real world and random graphs. We see that real world graphs have coefficients between 0.037 and 0.099 which are relatively small. For WWW-Barabasi graph, it means that removing 3.7 % of high degree nodes can shatter the graph.

In contrast, random(Erdős-Rényi) graphs have higher wing width ratio  $w(G)$ . We generated two random graphs, 'ER-Epinions', and 'ER-AS-Oregon', which have the same number of nodes and edges as 'Epinions', and 'AS-Oregon', respectively. The wing width ratios of the two random graphs are 0.611 and 0.358, respectively, at least  $6.2\times$  larger than their real world counterparts.

## IV. RELATED WORKS

The related works form three groups: structure of networks, graph partition and compression, and large graph mining.

**Structure of Networks.** Research on the structure of complex networks has been receiving significant amount of attention. Most real world graphs have power law in its degree distribution [6], a property that distinguishes them from random graphs [15] with exponential tail distribution. The graph shattering has been researched in the viewpoint of attack tolerance [16] and characterizing real world graphs [17]. Chen et al. [18] studied the statistical behavior of a fragmentation measure from the removal of nodes in graphs. None of the previous works relate the shattering and the power law to the problem of node permutation for graph compression.

**Graph Partition and Compression.** There has been a lot of works on network community detection, including METIS and related works [19], [20], edge betweenness [21],

co-clustering [3], [22], cross-associations [4], spectral clustering [2], [23], and shingle-ordering [5]. All of them aimed to find homogeneous regions in the graph so that cross edges between different regions are minimized. A recent result [7] studied real world networks using conductance, and showed that real world graphs don't have good cuts.

Graph compression has also been an active research topic. Boldi [14] studied the compression of web graphs using the lexicographic localities; Chierichetti et al. [5] extended it to the social networks; Apostolico et al. [24] used BFS based method for compression. Maserrat et al. [25] used multi-position linearizations for better serving neighborhood queries. Our SLASHBURN is the first work to take the power-law characteristic of most real world graphs into advantage for addressing the 'no good cut' problem and graph compression. Furthermore, our SLASHBURN is designed for large scale block based matrix vector multiplication where each square block is stored independently from each other for scalable processing in distributed platforms like MAPREDUCE [10]. The previously mentioned works are not designed for this purpose: the information of the outgoing edges of a node is tightly inter-connected to the outgoing edges of its predecessor or successor, making them inappropriate for square block based distributed matrix vector multiplication.

**Large Graph Mining.** Large scale graph mining poses challenges in dealing with massive amount of data: they exceed memory and even disks of a single machine. A promising alternative for large graph mining is MAPREDUCE [10], a parallel programming framework for processing web-scale data, and its open-source version HADOOP. MAPREDUCE has two advantages. First, the data distribution, replication, fault-tolerance, and load balancing are handled automatically. Second, it uses the familiar concept of functional programming: the programmer needs to define only two functions, a *map* and a *reduce*.

There has been several works [22], [8], [9], [26], [27] on large graph mining using MAPREDUCE. Among them, PEGASUS [8] unifies several important graph mining operations(PageRank, diameter, connected components, etc.) into a generalized matrix-vector multiplication. They provided the block method for fast matrix-vector multiplication framework. Our SLASHBURN is an algorithm for reordering nodes in graphs so that the block method performs better.

## V. CONCLUSION

In this paper, we propose SLASHBURN, a novel method for laying out the edges of real world graphs, so that they can be easily compressed, and graph mining algorithms based on block matrix-vector multiplication can run quickly.

The main novelty is the focus on real world graphs, that typically have *no good cuts* [7], and thus can't create good *caveman-like* communities and graph partitions. On

the contrary, our SLASHBURN is tailored towards *jellyfish*-type graphs [28], with spokes connected by hubs, and hubs connected by super-hubs, and so on, recursively. Our realistic view-point pays off: the resulting graph lay-outs enjoy

- faster processing times (e.g., matrix-vector multiplications, that are in the inner loop of most typical graph mining operations, like PageRank, connected components, etc), and
- lower disk space requirements.

Future research directions include extending SLASHBURN for better supporting time evolving graphs.

#### ACKNOWLEDGEMENT

Research was sponsored by the Defense Threat Reduction Agency under contract No. HDTRA1-10-1-0120, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. This work is also partially supported by an IBM Faculty Award, a Google Focused Research Award, and a Yahoo Research Alliance Gift. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory, the U.S. Government, or other funding parties. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

#### REFERENCES

- [1] J. Shi and J. Malik, "Normalized cuts and image segmentation," *CVPR*, 1997.
- [2] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *NIPS*, 2002.
- [3] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *KDD*, 2003, pp. 89–98.
- [4] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, "Fully automatic cross-associations," in *KDD*, 2004, pp. 79–88.
- [5] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *KDD*, 2009, pp. 219–228.
- [6] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," *SIGCOMM*, pp. 251–262, Aug–Sept. 1999.
- [7] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *WWW*, 2008, pp. 695–704.
- [8] U. Kang, C. Tsourakakis, and C. Faloutsos, "Pegasus: A peta-scale graph mining system - implementation and observations," *ICDM*, 2009.
- [9] U. Kang, C. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec., "Radius plots for mining tera-byte scale graphs: Algorithms, patterns, and observations," *SIAM International Conference on Data Mining*, 2010.
- [10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *OSDI*, 2004.
- [11] J. Rissanen and G. G. L. Jr., "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.
- [12] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social Networks*, 2006.
- [13] B. A. Prakash, H. Tong, N. Valler, M. Faloutsos, and C. Faloutsos, "Virus propagation on time-varying networks: Theory and immunization algorithms," in *ECML/PKDD*, 2010.
- [14] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *WWW*, 2004.
- [15] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
- [16] R. Albert, H. Jeong, and A.-L. Barabasi, "Error and attack tolerance of complex networks," *Nature*, 2000.
- [17] A. P. Appel, D. Chakrabarti, C. Faloutsos, R. Kumar, J. Leskovec, and A. Tomkins, "Shatterplots: Fast tools for mining large graphs," in *SDM*, 2009.
- [18] Y. gChen, G. Paul, R. Cohen, S. Havlin, S. P. Borgatti, F. Liljeros, and H. E. Stanley, "Percolation theory and fragmentation measures in social networks," in *Physica A* 378, 2007, pp. 11–19.
- [19] G. Karypis and V. Kumar, "Multilevel -way hypergraph partitioning," in *DAC*, 1999, pp. 343–348.
- [20] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *KDD*, 2009.
- [21] M. Girvan and M. Newman, "Community structure in social and biological networks," *National Academy of Sciences*, vol. 99, pp. 7821–7826, 2002.
- [22] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce," *ICDM*, 2008.
- [23] U. von Luxburg, "A tutorial on spectral clustering," *Technical Report 149, Max Plank Institute for Biological Cybernetics*, 2006.
- [24] A. Apostolico and G. Drovandi, "Graph compression by bfs," *Algorithms*, vol. 2, no. 3, pp. 1031–1044, 2009.
- [25] H. Maserrat and J. Pei, "Neighbor query friendly compression of social networks," in *KDD*, 2010.
- [26] U. Kang, C. E. Tsourakakis, A. P. Appel, C. Faloutsos, and J. Leskovec, "Hadi: Mining radii of large graphs," *ACM Trans. Knowl. Discov. Data*, vol. 5, pp. 8:1–8:24, February 2011.
- [27] C. Liu, H. chih Yang, J. Fan, L.-W. He, and Y.-M. Wang, "Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce," in *WWW*, 2010, pp. 681–690.
- [28] G. Siganos, S. L. Tauro, and M. Faloutsos, "Jellyfish: A conceptual model for the as internet topology," *Journal of Communications and Networks*, 2006.