This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

SlashBurn: Graph Compression and Mining beyond Caveman Communities

Yongsub Lim, U Kang, and Christos Faloutsos

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 to recursively split a graph into hubs and spokes connected only by the hubs. We also propose techniques to select the hubs and give an ordering to the spokes, in addition to the basic SLASHBURN. We give theoretical analysis of the proposed hub selection methods.
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. Through experiments, we show that SLASHBURN consistently outperforms other methods for all datasets, resulting in better compression and faster running time. Moreover, we show that SLASHBURN with the appropriate spokes ordering can further improve compression while hardly sacrificing the running time.

Index Terms—Graph Compression, Graph Mining, Hubs and Spokes

1 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. This compression gives benefits in graph mining. For example, space for storing a graph and time for transmitting it, including I/O and communication costs, can be reduced. Furthermore, recent researches report that good compression is helpful in reducing running time of graph mining algorithms [1], [2].

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 he/she knows very little about persons in different caves as shown in Fig. 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 Fig. 1(b). Spectral clustering [3], [4], co-clustering [5], cross-associations [6], and shingle-ordering [7] 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 [8]. Also it is known that a significant proportion of the hub nodes effectively combines many caves into a huge cave [9], which breaks the assumption of the caveman-like community structure. Thus, it is not surprising that well defined communities in real world networks are hard to find [10].

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 Fig. 1(d), in contrast to the square blocks in caveman communities as shown in Fig. 1(b). Our method is to order these hubs and spokes to get such a compact representation of the adjacency matrix, which in turn leads to good compression.

We also propose improvements of SLASHBURN by choosing alternate options for selecting hub nodes and ordering spokes. We give theoretical and experimental analysis of the alternate options. Our main results include that our advanced spokes ordering gives better compression without seriously degrading speed, regard-

- *Y. Lim and U Kang are with Department of Computer Science, KAIST, South Korea,*
  E-mail: yongsub@kaist.ac.kr, ukang@cs.kaist.ac.kr
- *C. Faloutsos is with Computer Science Department, CMU, U.S.,*
  E-mail: christos@cs.cmu.edu

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

TABLE 1: Table of symbols

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: ratio of the number of total hub nodes to n.
b	Block width used for block based matrix-vector multiplication.

less of the hub selection methods.

Our contributions are summarized as follows:

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 2 precisely describes the problem and our proposed method for laying out edges for better compressing graphs. In Section 3, we analyze our proposed method with respect to its complexity and performance. We give experimental results in Section 4, showing the compression and running time enhancements. After discussing related works on Section 5, we conclude in Section 6.

To enhance the readability of this paper, we list the symbols frequently used in this paper in TABLE 1.

2 PROPOSED METHOD

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

2.1 Problem Definition

Given a large graph, we want to reorder the nodes so that the graph can be compressed well, implying a small number of bits required to store the graph. 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 [1]. The state-of-the-art method for the large scale matrix-vector multiplication is the block multiplication method [1], where the original matrix is divided into b by b square matrix blocks, the original vector is divided into length b vector blocks, and the matrix-vector blocks are multiplied.

For example, see Fig. 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 of 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 [11].

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 Fig. 3. The left matrix is the adjacency matrix of Fig. 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. Assume we use 2 by 2 blocks to cover all the nonzero elements inside the matrix. Then the right matrix requires smaller number of blocks than the left matrix. Furthermore, each nonempty 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:

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

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.

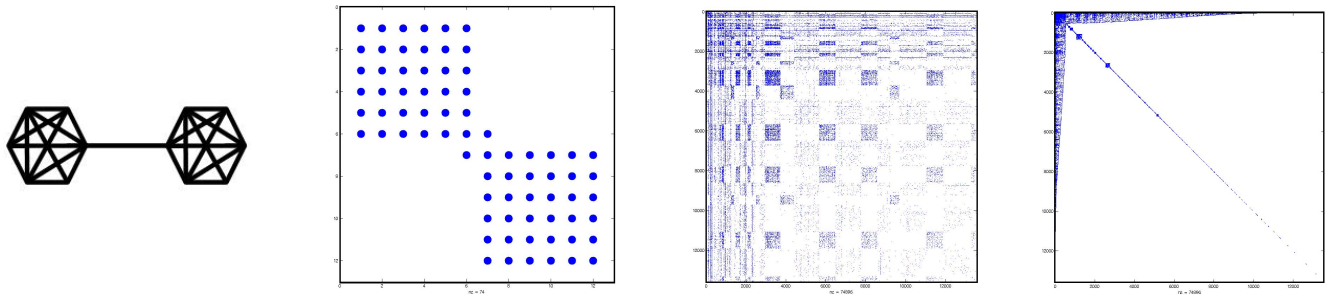
- *Nonzeros inside blocks.* Bits to compress nonzero elements inside blocks.
- *Meta information on blocks.* Bits to store the row and column ids of blocks.

Using the decomposition, we define a cost function $cost_{it}(A, b)$ assuming a compression method achieving the information theoretic lower bound [12], [6]:

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

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering



(a) Caveman graph C (b) Adjacency Matrix of C (c) Adjacency Matrix of AS-Oregon graph (d) AS-Oregon after SLASHBURN

Fig. 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.

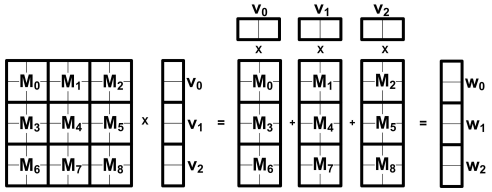


Fig. 2: Block method [1] for large scale matrix-vector multiplication. The original 6 by 6 matrix is divided into 2 by 2 square matrix blocks (M_0 to M_8), the original length 6 vector is divided into length 2 vector blocks (v_0 to v_2), and the blocks are multiplied to get the resulting vector (w_0 to w_2).

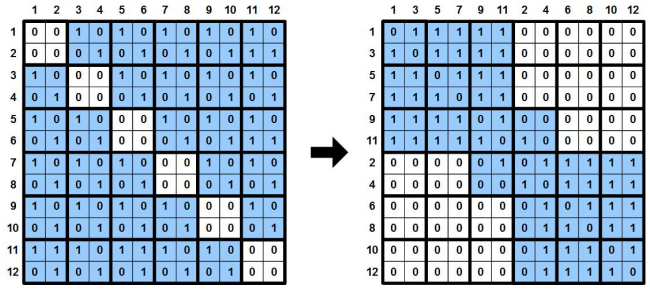


Fig. 3: Importance of ordering. Left: adjacency matrix of Fig. 1(a) with a random ordering of nodes. Right: adjacency matrix of the same graph, but with a compression-friendly ordering. If we use 2 by 2 blocks to cover all the nonzero elements inside the matrix, the right matrix requires smaller number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

elements within a block τ, 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 4.

2.2 Why Not Classic Partitioning?

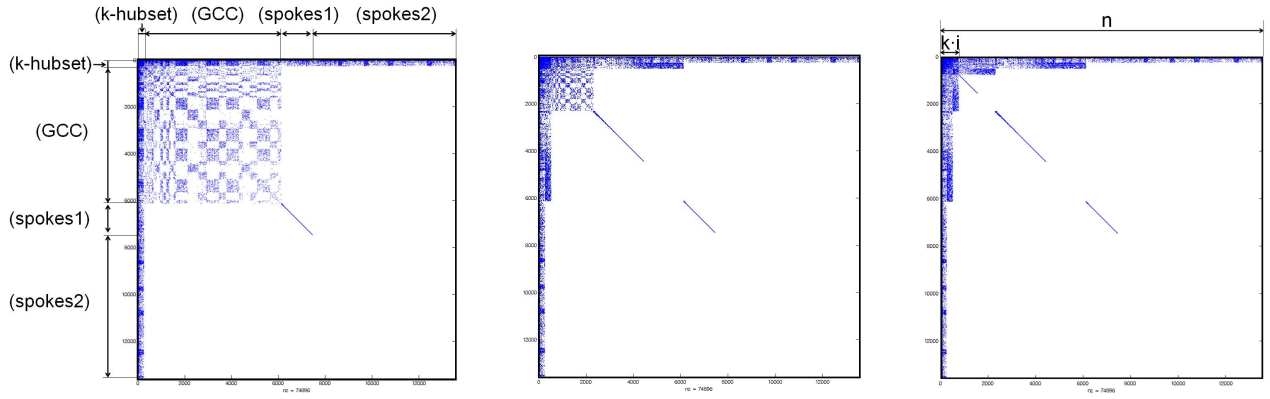
In general, directly minimizing the cost functions is a difficult combinatorial problem which could require $n!$ trials in the worst case. Traditional approach is to use graph partitioning algorithms to find good 'cuts' and homogeneous regions so that nodes inside a region form a dense community, and thereby leading to better compressions. Examples include spectral clustering [3], [4], co-clustering [5], cross-associations [6], and shingle-ordering [7]. However, such approaches do not work well for real world, power-law graphs since there exists no good cuts in such graphs [10], which we also experimentally show in Section 4.

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—especially hub nodes bridging communities [9]. Such hub nodes make 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.

2.3 Graph Shattering

Our main idea to solve the problem is to exploit the hubs to define alternative communities different from the traditional communities.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering



(a) AS-Oregon after 1 iteration (b) .. after 1 more iteration (c) .. after 1 more iteration

Fig. 4: 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 areas of the matrix, making an arrow-like shape. Compared to the original adjacency matrix in Fig. 1(c), the final matrix has much larger empty spaces, enabling better compression.

We start with an observation that real-world graphs are easily shattered by removing hub nodes from them. By the removal, the graph is broken into many connected components, and while the majority of the nodes still belong to the giant connected component, a nontrivial portion of the nodes belong to small disconnected components. The nodes belonging to the small disconnected components after the 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 [13], PageRank, eigendrop [14], 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 us consider the following shattering process. Given a graph G, we perform a k-shattering on G. Among the remaining 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 Fig. 4(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.

Computing the exact wing width ratio of a graph is not easy. It requires to select k-hubset so that the incident edges include many *bridges* or the number of those edges is maximized. Even for the latter which is a more simpler objective function than the former, the problem becomes NP-hard. In fact, it is formulated as submodular function maximization which we will show in Section 3.

As we will see in Section 4.3, real-world, power-law graphs have low $w(G)$. Our proposed SLASHBURN method utilizes the low wing width ratio in real world graphs.

2.4 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 remove top k highest centrality scoring nodes and incident edges, thereby decomposing nodes in G into the following three groups:

- k-hubset: top k highest centrality scoring nodes in G.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1: SLASHBURN

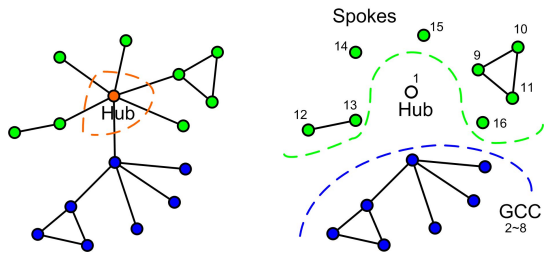
Input: Edge set E of a graph $G = (V, E)$, a constant k (default = 1).
Output: Array Γ containing the ordering $V \rightarrow [n]$.
1: Remove k-hubset from G to make the new graph G'. Add the removed k-hubset to the front of Γ.
2: Find connected components in G'. Add nodes in non-giant connected components to the back of Γ, 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.

- GCC: nodes belonging to the giant connected component of G'. Colored blue in Fig. 5.
- Spokes to the k-hubset: nodes belonging to the non-giant connected component of G'. Colored green in Fig. 5.

Fig. 5 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 components 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 sizes of connected components they belong to, and the GCC takes the remaining ids (2∼8). The same process applies to the nodes in GCC, recursively. If there exist more than one GCCs having the same size, we choose one of them randomly, which is a very rare case in real graphs.

Fig. 4(a) shows the AS-Oregon graph after the lines 1 and 2 of Algorithm 1 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. Fig. 4(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 areas 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 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.



(a) Before SLASHBURN (b) After SLASHBURN

Fig. 5: [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 sizes of connected components they belong to, and the GCC takes the remaining ids (2∼8). The next iteration starts on the GCC.

2.5 Improvements of SLASHBURN

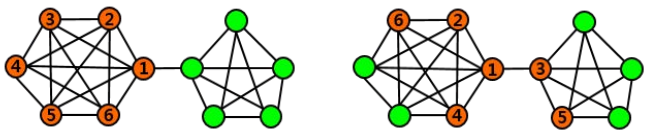
The basic SLASHBURN algorithm results in good compression which we will show in Section 4, but there is still a room for improvement in both compression quality and running time. In this section, we discuss possible improvements for SLASHBURN in efficiency and effectiveness. To this end, we consider SLASHBURN to alternate two main operations as described in Algorithm 1: 1) selecting k-hubset, and 2) putting the remaining connected components (CCs) in appropriate places in ordering. The goal is to refine the two operations with the goal of obtaining compression quality and running time comparable to the basic SLASHBURN with $k = 1$ and $k \gg 1$, respectively.

2.5.1 Selecting k-hubset

Let $v_i(G)$ be the i^{th} largest degree node in G, and G_i be the graph reduced by removing $\{v_j(G) : 1 \leq j \leq i\}$ with $G_0 = G$. For $k > 1$, the problem is that $v_i(G)$ may not be a high degree node any more in G_{i-1}; $v_i(G)$ may become isolated in G_{i-1} if $i \approx k \gg 1$. This is depicted in Fig. 6a. With $k = 6$, if the nodes labeled by 1 to 5 are removed, the node labeled by 6 is no more a high degree node but selected because of its high degree in the initial graph.

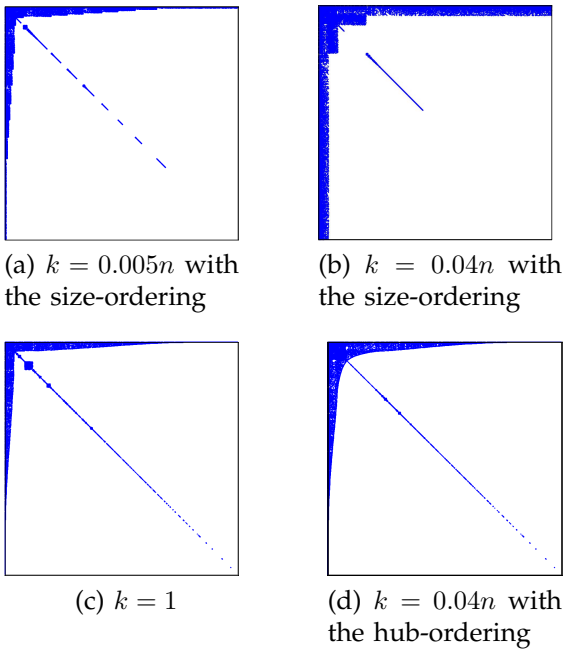
Alternatively, we can select k-hubset in a greedy way: whenever selecting v, we update $\deg(u)$ for $(u, v) \in E$ where $\deg(u)$ denotes the degree of a node u. In other words, we select $v_1(G_{i-1})$ for the i^{th} hub node instead of $v_i(G)$. This can be understood to compromise the basic SLASHBURN with $k = 1$ and $k > 1$ in the following aspects: it puts more effort into selecting k-hubset than SLASHBURN with $k > 1$ while sacrificing accuracy by omitting GCC computation for each iteration compared with SLASHBURN with $k = 1$. Fig. 6 depicts the differ-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering



(a) Basic k-hub Selection (b) Greedy k-hub Selection

Fig. 6: [Best viewed in color.] Comparison between the basic and the greedy k-hubset selections. The orange nodes denote k-hubset selected by each method. For tie nodes in different 'caves', the one in the left cave is selected.



(a) $k = 0.005n$ with the size-ordering (b) $k = 0.04n$ with the size-ordering

(c) $k = 1$ (d) $k = 0.04n$ with the hub-ordering

Fig. 7: Adjacency matrices of AS-Oregon where nodes are ordered by SLASHBURN with the specified settings.

ence between the basic and the greedy k-hub selection methods.

Consequently, we have two selection methods for k-hubset of G:

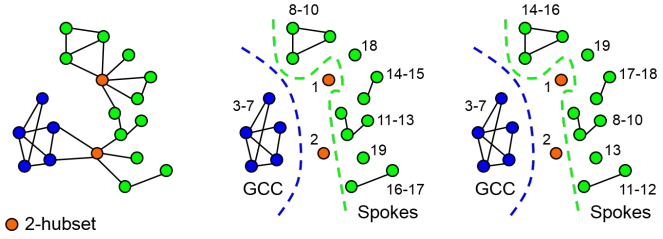
- Basic: $\{v_i(G) : 1 \le i \le k\}$.
- Greedy: $\{v_1(G_{i-1}) : 1 \le i \le k\}$ where $G_0 = G$.

The effect of k-hubset selection is evaluated experimentally in Section 4.4.

2.5.2 Ordering Spokes

As k gets larger, SLASHBURN tends to have a thick wing as shown in Fig. 7a and 7b, which leads to more nonzero blocks. The main reason for the thick wing is due to the ordering of spokes. Originally, SLASHBURN use the 'size-ordering': spokes are sorted according to the sizes of connected components to which they belong. Formally, let H be a size vector of CCs (i.e., $H_c = |\{u \in V : L(u) = c\}|$). Then,

- Size-ordering: descending order by H.



(a) Original graph (b) Size-ordering (c) Hub-ordering

Fig. 8: [Best viewed in color.] Size-ordering and Hub-ordering for spokes after 1 iteration of SLASHBURN with $k = 2$. Note that nodes with the ids 8∼10 in the size-ordering (b) have ids 14∼16 in the hub-ordering (c). The nodes are moved to the back since they are attached to the node with the largest degree.

But, by sorting with respect to hub nodes to which CCs of spokes are attached, we can obtain a sharper wing, implying better compression, which has more similar look to a result by SLASHBURN with $k = 1$ (see Fig. 7c and 7d). Let v_i be the i^{th} hub node selected for $1 \le i \le k$. To order spokes as described above, for each connected component c we need the largest hub node id T_c to which c is connected: i.e., $T_c = \max\{1 \le i \le k : \exists u \in c,\ (v_i, u) \in E\}$. This can be constructed during the GCC computation by enumerating nodes, at which Breadth First Search (BFS) starts, in a certain order. We describe the idea in Algorithm 2 where the Blocked-BFS by $V' \subset V$ is the BFS that does not go further if any of $v' \in V'$ is met, and $L(u)$ is the CC label assigned to u. Note that the Blocked-BFS by V' simulates the usual BFS for the graph reduced by removing V'. The 'hub-ordering' uses the T vector in addition to the H (size) vector as follows.

- Hub-ordering: descending order primarily by T and secondarily by H.

Note that for both size-ordering and hub-ordering, there is no order among nodes belonging to the same CC. Fig. 8 shows the difference of the size and the hub ordering. The effect of the ordering methods is evaluated experimentally in Section 4.4.

3 DISCUSSIONS

In this section, we give theoretical results for SLASHBURN. We first analyze the time and the space complexities of SLASHBURN, and then examine performance bounds of our k-hub selection methods.

3.1 Complexity Analysis

Here we analyze the time and the space complexities of the basic 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.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2: Finding Connected Components for the Hub-ordering in SLASHBURN

Input: Graph G and k-hubset $\{v_1, \ldots, v_k\}$.
Output: Attached top-k information T for all CCs, and CC labels L for all nodes.

1: $\ell \leftarrow 1$.
2: **for** $i = 1$ to k, and $u \in Neighbors(v_i)$ **do**
3: **if** u is not visited yet **then**
4: */* For every visited node $\bar{u}$, $L(\bar{u}) = \ell$. */*
5: Start Blocked-BFS by $\{v_1, \ldots, v_k\}$ at u with label ℓ.
6: $\ell \leftarrow \ell + 1$.
7: **end if**
8: $T_{L(u)} \leftarrow i$.
9: **end for**

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 4.3.

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. □

3.2 Performance Analysis

As described in Section 2, SLASHBURN works over iteratively cutting off k-hubset to shatter the graph. Then, it is natural to regard the following question: how well are graphs shattered by our k-hubset selection methods? To quantify the performance, we use the number of edges removed by cutting of the k-hubset as our objective function:

$$f(S) = |\{(u, v) : u \in S \text{ or } v \in S\}|, \tag{3}$$

where $S \subseteq V$. We want to analyze the performance bound of k-hubset selection methods in terms of maximizing f.

Before beginning our analysis, we note that f is a monotone submodular function[1] [15]. Intuitively, the effect to f of adding a node $u \notin S$ to a set $S \subseteq V$ tends to be marginal as $|S|$ gets larger because incident edges of u are more likely to be counted in advance by nodes in S sharing those edges. Hence, f is submodular. Also since the number of incident edges of a node is non-negative, f is monotone.

1. A submodular function is a set function g satisfying the diminishing marginal return property $g(A \cup \{u\}) - g(A) \geq g(B \cup \{u\}) - g(B)$ if $A \subseteq B$ and $u \notin B$; the monotonicity means $g(A) \leq g(B)$ if $A \subseteq B$.

Algorithm 3: Greedy Algorithm for Monotone Submodular Maximization

Input: A finite set X, a monotone submodular function f defined over 2^X, and solution size k.
Output: A set of nodes S of size k.

1: $S \leftarrow \emptyset$.
2: $S \leftarrow S \cup \{\text{argmax}_{u \in X \setminus S} f(S \cup \{u\}) - f(S)\}$.
3: If $|S| < k$, go to Line 2; otherwise return S.

Despite the NP-hardness of submodular function maximization, it is known that a monotone submodular function can be maximized in a greedy way within approximation factor $1 - (1/e)$ [15], which is described in Algorithm 3. In what follows, based on this general result on monotone submodular function maximization, we examine lower bounds of f for the basic and greedy k-hubset selection methods.

3.2.1 The Greedy k-hubset Selection

In this case, the analysis is simple: the greedy k-hubset selection becomes an implementation of Algorithm 3 to maximize f with the constraint $|S| = k$. Hence, for a set of nodes $\hat{S}$ selected by our greedy k-hubset selection, the following lower bound holds.

$$f(\hat{S}) \geq \left(1 - \frac{1}{e}\right) \max_{|S|=k} f(S).$$

3.2.2 The Basic k-hubset Selection

Now we focus on the basic k-hubset selection which shows good performance in compression quality and running time with the hub-ordering for spokes (see Section 4).

Let $P_k = \{p_1, \ldots, p_k\}$ where p_i is the i^{th} selected node by the greedy k-hub selection, and $Q_k = \{q_1, \ldots, q_k\}$ where q_i is the node having i^{th} largest degree among k nodes selected by the basic k-hubset selection. We want to give a lower bound of $f(Q_k)$, which can be obtained by estimating the number $W(Q_k)$ of edges inside Q_k. Concretely, the lower bound of $f(Q_k)$ becomes

$$\begin{aligned} f(Q_k) &= \Big(\sum_{u \in Q_k} \deg(u)\Big) - W(Q_k) \geq \Big(\sum_{u \in P_k} \deg(u)\Big) - W(Q_k) \\ &\geq f(P_k) - W(Q_k) \geq \left(1 - \frac{1}{e}\right) \max_{|S|=k} f(S) - W(Q_k). \end{aligned}$$

General Graphs. For general graphs, Q_k can form a clique of size k, which means that $W(Q_k) = O(k^2)$:

$$f(Q_k) \geq \left(1 - \frac{1}{e}\right) \max_{|S|=k} f(S) - O(k^2).$$

Unfortunately, the term of $O(k^2)$ is quite large, which makes the bound less meaningful. Below, we give a tighter bound for power-law graphs which are usually observed in the real world.

Power-law Graphs. Let us consider a graph generated as follows: 1) fix a power-law degree probability distribution function (PDF) $d(x) = \beta x^{-\alpha}$ where $\alpha > 2$ and $x \geq 1$, and 2) draw $w_1, \ldots, w_n$ from $d(x)$ and generate

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

TABLE 2: The exponents of $\mathbb{E}[W(Q_k)]$ for various α and p values in power-law graphs.

	p					
	0.25	0.33	0.50	0.67	0.75	1
$\alpha = 2.2$	0.75	0.78	0.83	0.89	0.92	1
$\alpha = 2.5$	0.50	0.56	0.67	0.78	0.83	1
$\alpha = 2.7$	0.38	0.45	0.59	0.73	0.79	1
$\alpha = 3.0$	0.25	0.33	0.50	0.67	0.75	1

a random graph with Chung-Lu model [16]. Then, the expected number of edges inside Q_k is expressed by

$$\mathbb{E}[W(Q_k)] = \mathbb{E}\left[\frac{\sum_{u,v\in Q_k}\deg(u)\deg(v)}{2\sum_{u\in V}\deg(u)}\right]. \quad (4)$$

Lemma 3: Given a power-law graph generated by the process above, if we set $k = n^p$, the following holds for the basic k-hub selection.

$$\mathbb{E}[W(Q_k)] = O\left(n^{\frac{2(\alpha-2)p+(3-\alpha)}{\alpha-1}}\right). \quad (5)$$

The proof is provided at the end of this section. TABLE 2 shows the exponents of (5) for various α and p. In expectation, although we obtain better bounds compared with that for a general graph, it might not be satisfactory because $\mathbb{E}[W(Q_k)]$ is still order-magnitude larger than $k = n^p$. However, as we will show in Section 4, SLASHBURN works well in practice for both k-hubset selection methods.

Proof of Lemma 3: Let D be the negative cumulative distribution function (NCDF)[2] of d, and $r_k = D^{-1}(k/n)$. Note that r_k is the expected minimum degree of k-hubset and can be written as

$$r_k = \left[\frac{k(\alpha-1)}{n\beta}\right]^{\frac{1}{1-\alpha}}, \quad (6)$$

since

$$k = nD(r_k) = \frac{n\beta}{\alpha-1}(r_k)^{1-\alpha}.$$

Also, letting $d_k(x) \propto d(x)$ be a PDF over $[r_k, \infty)$, the sum of degrees of nodes in Q_k is

$$\begin{aligned}\sum_{u\in Q_k}\deg(u) &= k\int_{r_k}^{\infty} x d_k(x)\, dx = k\left(\frac{\alpha-1}{\alpha-2}\right) r_k \\ &= C_1 \cdot k^{\frac{2-\alpha}{1-\alpha}} n^{\frac{1}{\alpha-1}} \qquad \text{(substituted by (6))} \\ &= C_1 \cdot n^{\frac{\alpha p-2p+1}{\alpha-1}} \qquad \text{(let } k = n^p\text{)},\end{aligned}$$

where $C_1 = \frac{\alpha-1}{\alpha-2}(\frac{\alpha-1}{\beta})^{\frac{1}{1-\alpha}}$ is a constant. It remains to compute the denominator of (4), as follows:

$$\sum_{u\in V}\deg(u) = n\int_1^{\infty} x d(x)\, dx = C_2 \cdot n,$$

where $C_2 = \beta/(\alpha-2)$ is a constant. As a consequence, the expected number of edges within Q_k becomes

$$\mathbb{E}[W(Q_k)] = C_3 \cdot n^{\frac{2(\alpha-2)p+(3-\alpha)}{\alpha-1}},$$

where $C_3 = (C_1)^2/2C_2$. □

2. For a probability distribution p, $p^{NCDF}(x) = 1 - p^{CDF}(x)$.

TABLE 3: Summary of graphs used. AS-Oregon is an undirected graph, while all others are directed graphs.

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
Wordnet	144,511	643,863	Word association network
Enron	80,163	312,725	Enron email
Epinions	75,888	508,960	Who trusts whom
Slashdot	51,083	131,175	Reply network
AS-Oregon	13,579	74,896	Router connetions

4 EXPERIMENTS

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

- **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?
- **Q4** How much improvement can we obtain by the greedy k-hubset selection and hub-ordering?

We compare SLASHBURN with the following six competitors.

- **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 [17]).
- **Degree Sort (DegSort).** Ordering based on the decreasing degree of the nodes.
- **Cross Association (CA).** Cross-association [6] based ordering so that nodes in a same group are numbered consecutively.
- **Spectral Clustering.** Normalized spectral clustering [3], 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 [7]. It groups nodes with similar fingerprints (min-wise hashes) obtained from the out-neighbors of nodes.

We note that SLASHBURN, the degree sort and the spectral clustering run on graphs whose edge directions are ignored for ordering nodes.

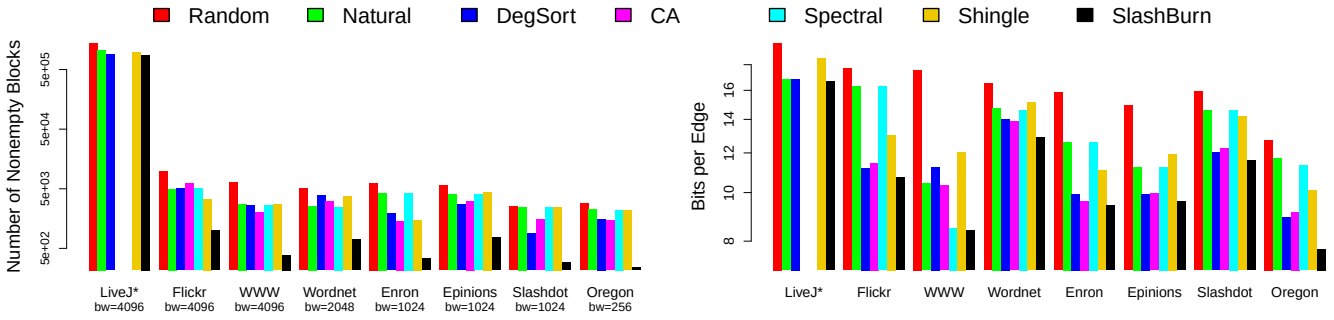
The graphs used in our experiments along with their descriptions are summarized in Table 3.

4.1 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 2:

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

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering



(b) $cost_{nz}(A, b)$: number of nonempty blocks (c) $cost_{it}(A, b)$: information theoretic cost

Fig. 9: Compression comparison of ordering methods. DegSort: degree sort, CA: cross association, and Spectral: spectral clustering. For all the cost functions, SLASHBURN performs the best. For the LiveJournal data, CA and Spectral (colored by magenta and cyan, respectively) could not be performed since the algorithms are not scalable enough to run on such a large graph. (a): SLASHBURN reduces the number of nonempty blocks by up to **20**× compared to the random ordering, and by up to **6.1**× compared to the second best orderings where 'bw' denotes the block width. (b): SLASHBURN reduces the bits per edge by up to **2.1**× compared to the random ordering, and by up to **1.2**× compared to the second best orderings.

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

Fig. 9 shows the costs of ordering methods. Fig. 9(a) shows the number of nonempty blocks ($cost_{nz}(A)$), and Fig. 9(b) shows the bits per edge computed using $cost_{it}(A, b)$. 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× compared to the random ordering, and by up to 6.1× compared to the second best orderings. For the bits per edge, SLASHBURN reduces the bits by up to 2.1× compared to the random ordering, and by up to 1.2× compared to the second best orderings.

The amount of compression can be checked visually. Fig. 10 shows the spyplots, which are nonzero patterns in the adjacency matrices, of real world graphs permuted from different ordering methods. Random ordering makes the spyplot almost filled; natural ordering provides more 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 Fig. 10, 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 Fig. 9.

4.2 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. Fig. 11 shows file size vs. running time on 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. 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 [7]: 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.

4.3 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 Erdős-Rényi random graphs [18]? TABLE 4 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, Erdős-Rényi random graphs have higher wing width ratio $w(G)$. We generated two random graphs, 'ER-Epinions', and 'ER-AS-Oregon', which have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

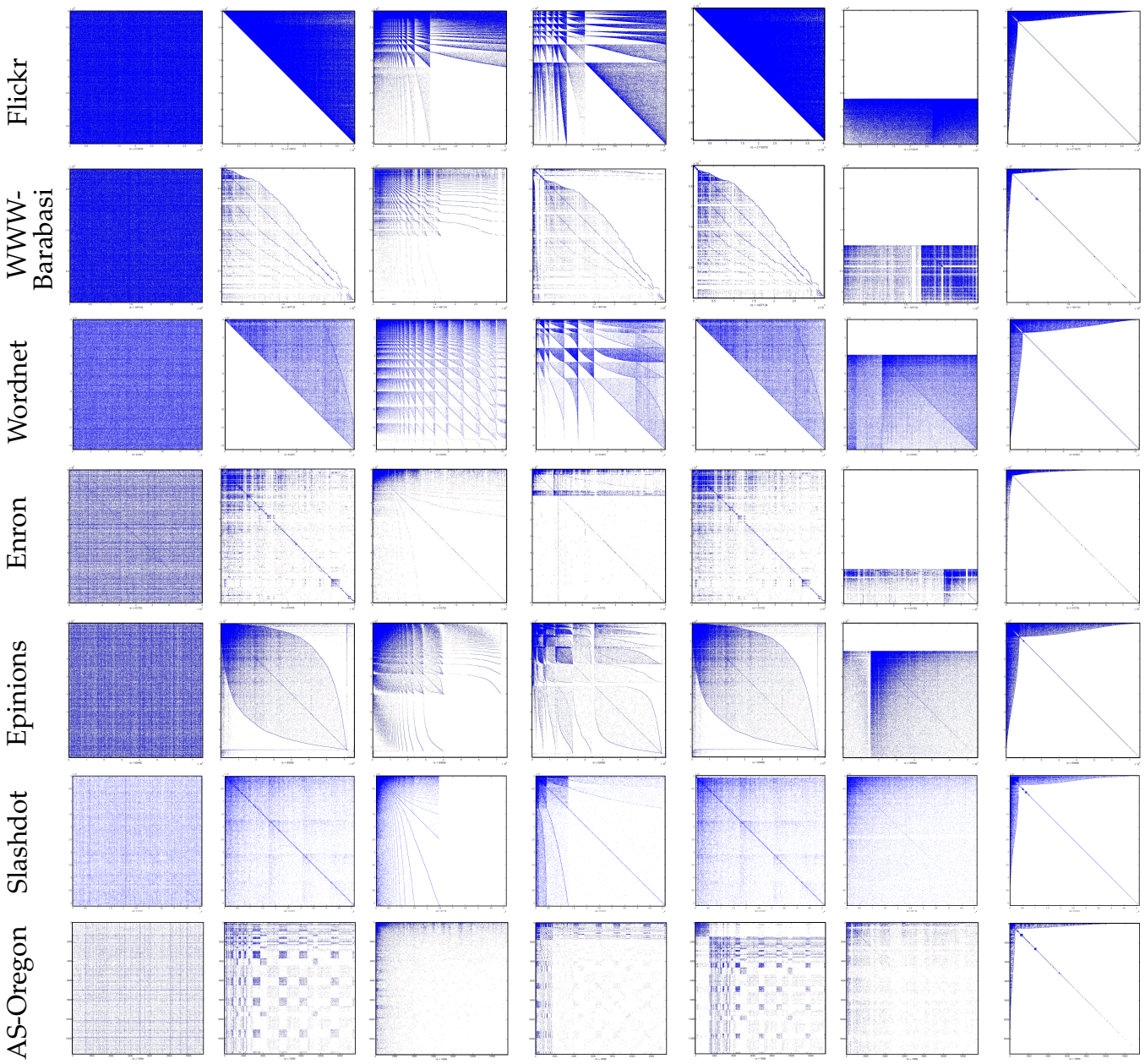


Fig. 10: Adjacency matrix of real world graphs on different ordering methods. Random ordering requires the maximum number of square blocks to cover the edges. Natural ordering requires smaller number of blocks, implying that the natural ordering exploits some form of localities. Degree sort makes the 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 Fig. 9

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

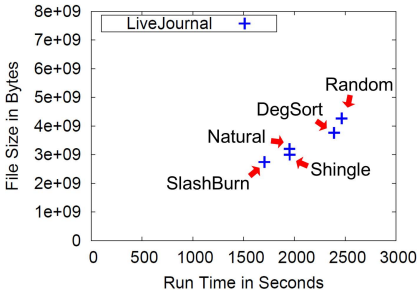


Fig. 11: File size vs. running time of different ordering methods on LiveJournal graph. The running time is measured for one iteration of PageRank on HADOOP. Notice that SLASHBURN results in the smallest file size, as well as the smallest running time.

TABLE 4: 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× larger than their real world counterparts), meaning that they cannot be shattered quickly.

Graph Type	Graph	$w(G)$
Real world	Flickr	0.078
Real world	WWW-Barabasi	0.037
Real world	Wordnet	0.099
Real world	Enron	0.044
Real world	Epinions	0.099
Real world	Slashdot	0.068
Real world	AS-Oregon	0.040
Erdős-Rényi	ER-Epinions	0.611
Erdős-Rényi	ER-AS-Oregon	0.358
Chung-Lu	CL-Epinions	0.099
Chung-Lu	CL-AS-Oregon	0.071

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, which are at least 6.2× larger than their real world counterparts. We also compute wing width ratios for random graphs generated by Chung-Lu model [19] in which the original degree distribution is preserved in expectation. Note that the values are very similar to those of the corresponding real graphs. These results match the previous studies showing that real and random graphs having heavy-tailed degree distributions fragment quickly [20], [21], [22].

Fig. 12 shows comparison of the wing width ratio and the running time between real graphs and random graphs. Note that the random graphs have large wing width ratios, which leads to slow running times.

4.4 Comparison of Variants of SLASHBURN

Now we compare the methods proposed in Section 2: SLASHBURN with two k-hub selections and two spoke orderings. Precisely, the methods we consider here are as follows:

- S-1: The basic SLASHBURN with $k = 1$.

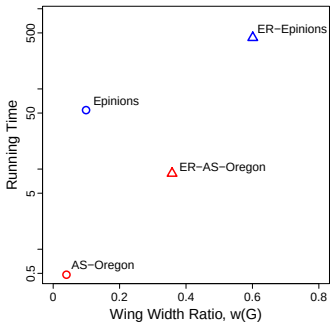
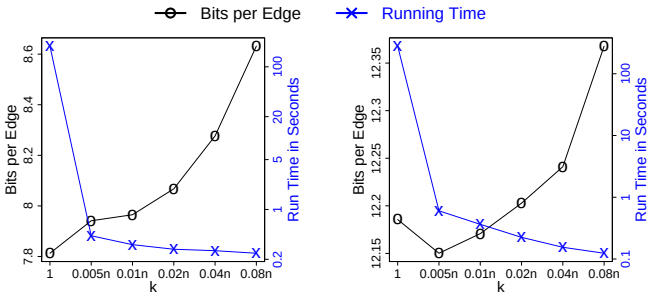


Fig. 12: Wing width ratio vs. running time of real graphs (denoted by ○) and random graphs (denoted by △). Note that random graphs (ER-AS-Oregon and ER-Epinions) have large wing width ratios, leading to slower running times compared with their real world counterparts.



(a) WWW-Barabasi (b) Wordnet

Fig. 13: Bits per edge (black line with linear scale) and running time (blue line with log scale) over increasing k with S-KH which shows the best bits per edge in TABLE 6. For most of the graphs, as k gets larger, the running time greatly decreases while bits per edge slightly increases.

- S-KS: The basic k-hubset selection with the size-ordering.
- S-KH: The basic k-hubset selection with the hub-ordering.
- S-K_GS: The greedy k-hubset selection with the size-ordering.
- S-K_GH: The greedy k-hubset selection with the hub-ordering.

Tables 5, 6, and 7 show the number of nonzero blocks, bits per edge, and running time for those five methods, respectively. Overall, as k gets larger, the performance is degraded but by the greedy hubset selection and the hub ordering, the amount of the degradation is greatly reduced. Especially, the hub ordering reduces bits per edge for some graphs like AS-Oregon though k gets larger.

As expected, the hub-ordering consistently outperforms the size-ordering in terms of compression: S-KH and S-K_GH show the best performance for almost all cases in bits per edge and the number of nonzero blocks, respectively. For the number of nonzero blocks, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

TABLE 5: The number of nonempty blocks for five versions of SLASHBURN. The block width b is the same as that in Fig. 9 for each graph. Note that for a fixed k-hub selection method, the hub-ordering clearly outperforms the size-ordering. Also, in many cases the greedy k-hub selection results in smaller nonzero blocks than the basic one. For each $k > 1$, the best method is in bold.

Graph	$k = 1$	$k = 0.005n$				$k = 0.02n$				$k = 0.08n$			
	S-1	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH
Flickr	991	991	1009	975	995	1130	1064	1072	1028	1782	1235	1758	1190
WWW-Barabasi	375	433	418	420	409	621	500	588	464	1227	698	1185	642
Wordnet	701	735	720	733	717	822	738	817	735	1174	851	1149	845
Enron	332	363	342	355	334	499	358	471	348	1125	430	1090	365
Epinions	728	778	749	770	750	827	767	816	763	1202	853	1157	823
Slashdot	291	289	293	293	289	287	285	283	293	459	298	443	293
AS-Oregon	233	234	231	220	237	298	237	304	233	536	267	538	255
Average	522	546	537	538	**533**	641	564	622	**552**	1072	662	1046	**630**

TABLE 6: Bits per edge for five versions of SLASHBURN, according to the information theoretic lower bound. Note that S-KH shows the best performance for almost all cases. Also, the greedy k-hub selection provides comparable quality compared with the basic k-hub selection. For each $k > 1$, the best method is in bold.

Graph	$k = 1$	$k = 0.005n$				$k = 0.02n$				$k = 0.08n$			
	S-1	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH
Flickr	10.67	10.69	10.65	10.75	10.73	10.70	10.50	11.01	10.80	10.72	10.30	11.46	10.95
WWW-Barabasi	7.81	8.88	7.94	9.36	8.55	9.46	8.07	10.79	9.82	10.54	8.63	12.64	11.64
Wordnet	12.19	12.17	12.15	12.23	12.24	12.41	12.20	12.49	12.34	12.86	12.37	13.06	12.64
Enron	8.73	9.07	8.73	9.08	8.78	9.40	8.72	9.53	8.86	9.56	8.70	10.11	9.08
Epinions	9.69	9.67	9.69	9.69	9.72	9.71	9.65	9.77	9.72	9.81	9.54	10.05	9.77
Slashdot	11.06	10.93	11.06	10.94	11.06	11.26	11.07	11.28	11.10	11.61	11.05	11.69	11.10
AS-Oregon	7.91	7.90	7.93	7.90	7.94	8.01	7.88	8.14	7.96	8.37	7.78	8.72	8.05
Average	9.72	9.90	**9.74**	9.99	9.86	10.14	**9.73**	10.43	10.08	10.50	**9.77**	11.10	10.46

TABLE 7: The running time of five versions of SLASHBURN in seconds. Note that the hub-ordering affects the running time very marginally. Also, the greedy k-hub selection is much faster than S-1, since the time-consuming connected component computation step is omitted. For each $k > 1$, the best method is in bold.

Graph	$k = 1$	$k = 0.005n$				$k = 0.02n$				$k = 0.08n$			
	S-1	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH	S-KS	S-KH	S-K_GS	S-K_GH
Flickr	2012.3	1.58	1.60	12.3	12.3	0.634	0.659	13.0	13.0	0.402	0.443	16.2	16.2
WWW-Barabasi	195.0	0.394	0.428	6.46	6.52	0.252	0.277	19.1	19.1	0.221	0.243	55.1	55.1
Wordnet	280.3	0.617	0.603	5.59	5.58	0.210	0.224	5.34	5.54	0.136	0.124	8.73	8.65
Enron	15.5	0.118	0.110	0.385	0.401	0.066	0.067	0.476	0.471	0.052	0.058	0.956	1.01
Epinions	54.4	0.250	0.251	1.00	1.01	0.094	0.100	0.959	0.966	0.060	0.064	1.18	1.18
Slashdot	13.8	0.121	0.111	0.265	0.259	0.056	0.054	0.268	0.261	0.032	0.034	0.360	0.348
AS-Oregon	0.481	0.017	0.018	0.031	0.032	0.010	0.011	0.031	0.032	0.008	0.010	0.038	0.040
Average	367.4	**0.44**	0.45	3.72	3.73	**0.19**	0.20	5.59	5.62	**0.13**	0.14	11.8	11.8

performance gap between the size and the hub ordering gets much significant as k gets larger. While the hub-ordering works effectively, its running time remains very fast, almost the same as that of the size-ordering.

On the other hand, the greedy selection is remarkable in decreasing the number of nonzero blocks. For a fixed spoke ordering, it almost always produces a smaller nonzero blocks. The reason is that the greedy selection produces a slightly sharper wing since it shatters better than the basic k-hubset selection. The bits per edge from the greedy k-hubset selection is almost the same as that from the basic k-hubset selection. In terms of running time, the greedy selection runs much faster than S-1, since the time-consuming connected component computation step is omitted.

Fig. 13 shows the bits per edge and the running time over increasing k for S-KH whose performance is the best in terms of bits per edge. For most of the graphs, as k gets larger, the running time gets faster while bits per edge slightly increases as shown in Fig. 13.

Fig. 14 shows bits per edge vs. running time for the five methods. For almost all graphs, S-KH is located in the left bottom corner, implying fast running time and the best compression.

5 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 [8], a property that distinguishes them from random graphs [18] with exponential tail distribution. The graph shattering has been researched in the viewpoint of attack tolerance [20] and characterizing real world graphs [23]. Chen et al. [24] studied the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

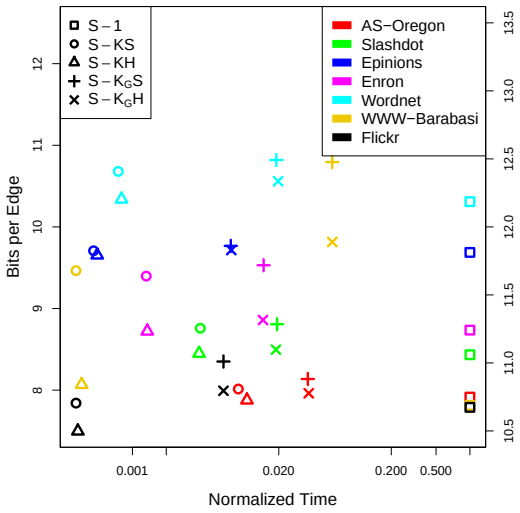


Fig. 14: Bits per edge vs. running time of five versions of SLASHBURN with $k = 0.02n$. AS-Oregon, Epinions, Enron, and WWW-Barabasi correspond to the left axis; the others correspond to the right axis. The colors distinguish the graphs, and the markers distinguish the methods. For each graph, time for S-1 is 1, and the others are normalized accordingly. Note that for every graph, S-KH shows the best compression (smallest bits per edge) with the near-smallest running time.

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 [25], [26], edge betweenness [27], co-clustering [5], [28], cross-associations [6], spectral clustering [4], [29], and shingle-ordering [7]. All of them aimed to find homogeneous regions in the graph so that cross edges between different regions are minimized. A recent result [10] studied real world networks using conductance, and showed that real world graphs don't have good cuts. Graph partition in terms of graph summarization [30] has been also done where each cluster is not required to be structurally homogeneous but corresponds to a certain functional.

Graph compression has also been an active research topic. Boldi [17] studied the compression of web graphs using the lexicographic localities; Chierichetti et al. [7] extended it to the social networks; Apostolico et al. [31] used BFS based method for compression. Maserrat et al. [32] used multi-position linearizations for better serving neighborhood queries, and Fan et al. [33] proposed a query preserving graph compression method by constructing a small graph using reachability equivalence relation of the original graph. Our SLASHBURN, whose preliminary version appeared in [34], 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 [11]. 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 [11], 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 have been several works [28], [1], [35], [36], [37], [38] on large graph mining using MAPREDUCE. Among them, PEGASUS [1] 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.

6 CONCLUSION

In this paper, we propose SLASHBURN, a novel algorithm 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. Moreover, we propose the greedy hub selection and the hub ordering for improving two main operations in SLASHBURN, selecting hubs and ordering spokes, respectively. The former gives the benefit of reducing nonzero blocks in a resulting adjacency matrix, and the latter gives the benefit of reducing bits per edge in resulting compression. Also for two hub selection methods, we analyze how well graphs are shattered by them theoretically.

The main novelty is the focus on real world graphs, that typically have *no good cuts* [10], and thus cannot create good *caveman-like* communities and graph partitions. On the contrary, our SLASHBURN is tailored towards *jellyfish*-type graphs [39], 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-out enjoys

- faster processing times (e.g., for matrix-vector multiplications, that are in the inner loop of most typical graph mining operations, like PageRank, connected

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2014.2320716, IEEE Transactions on Knowledge and Data Engineering

components, etc), and
- lower disk space requirements.

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

ACKNOWLEDGMENTS

This work was supported by the IT R&D program of MOTIE/KEIT [10044970, Development of Core Technology for Human-like Self-taught Learning based on Symbolic Approach].

REFERENCES

[1] U. Kang, C. E. Tsourakakis, and C. Faloutsos, "Pegasus: mining peta-scale graphs," *Knowl. Inf. Syst.*, vol. 27, no. 2, pp. 303–325, 2011.
[2] U. Kang, H. Tong, J. Sun, C.-Y. Lin, and C. Faloutsos, "GBASE: an efficient analysis platform for large graphs." *VLDB J.*, vol. 21, no. 5, pp. 637–650, 2012.
[3] J. Shi and J. Malik, "Normalized cuts and image segmentation," *CVPR*, 1997.
[4] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *NIPS*, 2002.
[5] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *KDD*, 2003, pp. 89–98.
[6] D. Chakrabarti, S. Papadimitriou, D. S. Modha, and C. Faloutsos, "Fully automatic cross-associations," in *KDD*, 2004, pp. 79–88.
[7] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *KDD*, 2009, pp. 219–228.
[8] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the internet topology," in *SIGCOMM*, 1999.
[9] J.-D. J. Han, N. Bertin, T. Hao, D. S. Goldberg, G. F. Berriz, L. V. Zhang, D. Dupuy, A. J. M. Walhout, M. E. Cusick, F. P. Roth, and M. Vidal, "Evidence for dynamically organized modularity in the yeast protein–protein interaction network," *Nature*, vol. 430, no. 6995, pp. 88–93, 2004.
[10] 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.
[11] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *OSDI*, 2004.
[12] J. Rissanen and G. G. L. Jr., "Arithmetic coding," *IBM Journal of Research and Development*, vol. 23, no. 2, pp. 149–162, 1979.
[13] S. P. Borgatti and M. G. Everett, "A graph-theoretic perspective on centrality," *Social Networks*, 2006.
[14] 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.
[15] U. Feige, V. S. Mirrokni, and J. Vondrk, "Maximizing non-monotone submodular functions." in *FOCS*, 2007.
[16] F. Chung and L. Lu, "The average distances in random graphs with given expected degrees," *PNAS*, vol. 99, no. 25, pp. 15 879–15 882, 2002.
[17] P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *WWW*, 2004.
[18] P. Erdős and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.
[19] F. Chung and L. Lu, "Connected components in random graphs with given degree sequences," *Annals of Combinatorics*, vol. 6, pp. 125–145, 2002.
[20] R. Albert, H. Jeong, and A.-L. Barabasi, "Error and attack tolerance of complex networks," *Nature*, 2000.
[21] R. Cohen, K. Erez, D. ben Avraham, and S. Havlin, "Breakdown of the internet under intentional attack," *Phys. Rev. Lett.*, vol. 86, pp. 3682–3685, Apr 2001.
[22] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han, "Attack vulnerability of complex networks," *Phys. Rev. E*, vol. 65, no. 5, p. 056109, May 2002.
[23] A. P. Appel, D. Chakrabarti, C. Faloutsos, R. Kumar, J. Leskovec, and A. Tomkins, "Shatterplots: Fast tools for mining large graphs," in *SDM*, 2009.
[24] Y. Chen, 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.
[25] G. Karypis and V. Kumar, "Multilevel -way hypergraph partitioning," in *DAC*, 1999, pp. 343–348.
[26] V. Satuluri and S. Parthasarathy, "Scalable graph clustering using stochastic flows: applications to community discovery," in *KDD*, 2009.
[27] M. Girvan and M. Newman, "Community structure in social and biological networks," *PNAS*, vol. 99, pp. 7821–7826, 2002.
[28] S. Papadimitriou and J. Sun, "Disco: Distributed co-clustering with map-reduce," *ICDM*, 2008.
[29] U. von Luxburg, "A tutorial on spectral clustering," *Technical Report 149, Max Plank Institute for Biological Cybernetics*, 2006.
[30] B.-S. Seah, S. S. Bhowmick, C. F. D. Jr., and H. Yu, "Fuse: towards multi-level functional summarization of protein interaction networks." in *BCB*, 2011.
[31] A. Apostolico and G. Drovandi, "Graph compression by bfs," *Algorithms*, vol. 2, no. 3, pp. 1031–1044, 2009.
[32] H. Maserrat and J. Pei, "Neighbor query friendly compression of social networks," in *KDD*, 2010.
[33] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *SIGMOD*, 2012.
[34] U. Kang and C. Faloutsos, "Beyond 'caveman communities': Hubs and spokes for graph compression and mining," in *ICDM*, 2011.
[35] U. Kang, E. E. Papalexakis, A. Harpale, and C. Faloutsos, "Gigatensor: scaling tensor analysis up by 100 times - algorithms and discoveries," in *KDD*, 2012, pp. 316–324.
[36] U. Kang, B. Meeder, E. E. Papalexakis, and C. Faloutsos, "Heigen: Spectral analysis for billion-scale graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 2, pp. 350–362, 2014.
[37] 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.
[38] 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.
[39] G. Siganos, S. L. Tauro, and M. Faloutsos, "Jellyfish: A conceptual model for the as internet topology," *Journal of Communications and Networks*, 2006.

Yongsub Lim is a Ph.D. candidate in the Computer Science Department of KAIST. His research interest includes large scale graph mining.

U Kang is an assistant professor in the Computer Science Department of KAIST. He received Ph.D. in Computer Science at Carnegie Mellon University, after receiving B.S. in Computer Science and Engineering at Seoul National University. He won 2013 SIGKDD Doctoral Dissertation Award, 2013 New Faculty Award from Microsoft Research Asia, and two best paper awards. He has published over 20 refereed articles in major data mining and database venues. He holds four U.S. patents. His research interests include data mining in big graphs.

Christos Faloutsos is a Professor at Carnegie Mellon University. He has received the Presidential Young Investigator Award by the National Science Foundation (1989), the Research Contributions Award in ICDM 2006, the SIGKDD Innovations Award (2010), nineteen "best paper" awards (including two "test of time" awards), and four teaching awards. He is an ACM Fellow, he has served as a member of the executive committee of SIGKDD; he has published over 200 refereed articles, 11 book chapters and one monograph. He holds six patents and he has given over 30 tutorials and over 10 invited distinguished lectures. His research interests include data mining for graphs and streams, fractals, database performance, and indexing for multimedia and bio-informatics data.