DeGPar: Large Scale Topic Detection using Node-Cut Partitioning on Dense Weighted Graphs

Kambiz Ghoorchian, Sarunas Girdzijauskas
Software and Computer Systems (SCS)
Royal Institute of Technology (KTH)
Stockholm, Sweden
Email: ghoorian@kth.se, sarunasg@kth.se

Fatemeh Rahimian
Swedish Institute of Computer Science (SICS)
Stockholm, Sweden
Email: fatemeh@sics.se

***Abstract*—*Topic Detection (TD)* refers to automatic techniques for locating topically related material in web documents [1]. Nowadays, massive amounts of documents are generated by users of Online Social Networks (OSNs), in form of very short text, tweets and snippets of news. While topic detection, in its traditional form, is applied to a few documents containing a lot of information, the problem has now changed to dealing with massive number of documents with very little information. The traditional solutions, thus, fall short either in scalability (due to huge number of input items) or sparsity (due to insufficient information per input item). In this paper we address the scalability problem by introducing an efficient and scalable graph based algorithm for TD on short texts, leveraging dimensionality reduction and clustering techniques. We first, compress the input set of documents into a dense graph, such that frequent co-occurrence patterns in the documents create multiple dense topological areas in the graph. Then, we partition the graph into multiple dense sub-graphs, each representing a topic. We compare the accuracy and scalability of our solution with two state-of-the-art solutions (including the standard LDA, and BiTerm). The results on two widely used benchmark datasets show that our algorithm not only maintains a similar or better accuracy, but also performs by an order of magnitude faster than the state-of-the-art approaches.**

***Index Terms*—Topic Detection; Node-cut Graph Partitioning; Distributed Algorithms; Random Indexing; Dimensionality Reduction; Dense Weighted Graph Partitioning; Online Social Networks**

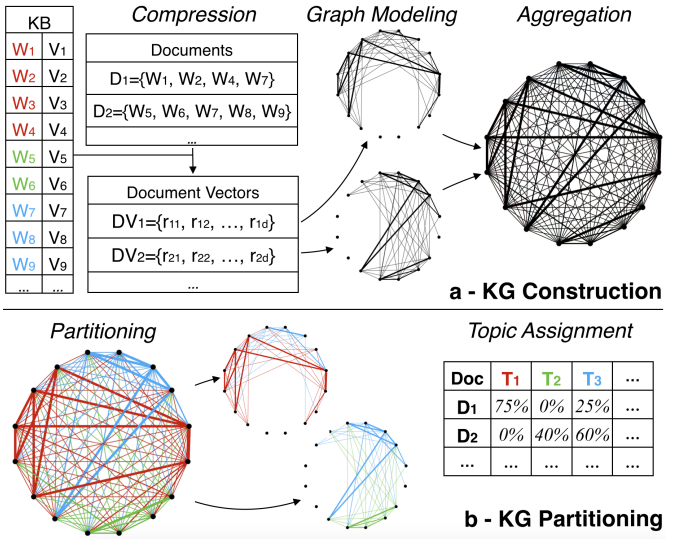

Fig. 1. DeGPar: an efficient graph algorithm for TD leveraging dimensionality reduction. a) Transform frequent correlation patterns to multiple sub-graphs in a dense and weighted graph called *Knowledge-Graph (KG)*, in 3 steps (i) Compression, (ii) Graph Modeling and (iii) Aggregation. b) Partition KG into dense sub-graphs using colors and assign topics to documents by overlapping corresponding document-graph with the partitioned KG.

I. Introduction

Online Social Networks (OSNs) are a dominant source of information dissemination in today's social media. A large number of texts with different topics are created and disseminated in these networks. Some of the topics that are mentioned the most are referred to as trending topics in OSN (like *"#Swineflu"* that became a trending topic in Twitter, after the contagion of the pandemic virus of swine flu in 2009). Thus, it's clear that being the first to know a trending topic is a key to success in today's social media. Topic trends vary over time, and detecting new trending topics is of interest for many companies, including the marketing or web intelligence companies. However, due to the large number of the texts required to be processed, finding these topics is not an easy task (e.g., Twitter users alone, publish around 500M Tweets per day [2]). Such large number of text makes it difficult to apply existing solutions for topic detection on short texts. Therefore, we need solutions that can efficiently apply TD on social media short texts for extracting trending topics.

The most well-known solutions to TD include Latent Semantic Analysis (LSA) [3], [4], and Latent Dirichlet Allocation (LDA) [5]. LSA uses lexical vector representation of documents to generate a word-document co-occurrence matrix. It extracts similarities by decomposing the matrix into its largest possible number of uncorrelated components leveraging eigenvector calculation techniques. The main limitations of LSA lie in its (i) expensive greedy computations caused by the sparseness of the co-occurrence matrix and (ii) inaccurate realization of the real semantic representation of the documents [6]. For example, it does not allow for overlapping topic structures, since the model emphasizes on an orthogonal basis for topic representation.

LDA uses a probabilistic generative approach to alleviate those limitations. The idea is to model word frequencies as a probability distribution of topics over documents. LDA has been initially developed for TD on web documents with long and coherent context structure. However, short messages convey two properties that make application of this approach challenging. The first property is the short average length of documents that causes a problem known as *Sparsity* [7]. In particular, due to the fact that LDA uses co-occurrence patterns in individual documents to infer model parameters, the shorter the length of the document, the lower the quality of the inference model will be [7]. The second property is the large number of documents that significantly hinders the *Scalability* since LDA requires multiple iterations over all documents.

A variety of solutions were proposed to deal with sparsity but scalability is still an issue. For example [7] and [8] proposed to aggregate short texts either with higher similarity or from the same publisher but their experiments do not exceed 2 million documents. Another solution, called BiTerm (BTM) [9], proposed to assign topic probabilities over bi-grams rather than single words and learn topics using aggregated bi-gram patterns over the entire dataset rather than single documents. The solution, although shown successful to overcome sparsity on over 4 million documents, still remained comparable with LDA on it's execution time.

In this paper we focus on the scalability issue. We develop a novel scalable TD algorithm, called *Dense Graph Partitioning (DeGPar)*, based on dimensionality reduction and graph analytics. The algorithm, as its name implies, is comprised of a dense graph modeling to represent documents and a partitioning to extract topics. In particular, given a set of documents, DeGPar first, compresses the documents into a small and dense graph, called *Knowledge Graph (KG)*, such that frequent co-occurrence patterns in the documents create multiple dense areas in that graph. Afterwards, the algorithm interprets the topics that appeared in all the compressed documents by analyzing the topology of the graph. The major advantages of DeGPar lie in (i) its innovative compression-based document representation that allows transformation of an entire dataset into a small and highly dense graph and (ii) its scalable partitioning algorithm that makes large-scale topic extraction possible. Fig. 1 depicts the overall protocol of the algorithm constructed in two phases, called *Knowledge Graph Construction* and *Knowledge Graph Partitioning*.

To construct the KG we use a dimensionality reduction method called *Random Indexing (RI)* [10]. In particular, we use a *Knowledge-Base (KB)* that contains a list of all unique words in the dataset represented by a low-dimensional fixed length vector created by RI. Fig. 1-a shows the overall process of KG construction phase. As we can see, the process is in three steps: (i) compression, (ii) graph modeling and (iii) aggregation. During compression, each document is converted into a low dimensional fixed length vector, called document-vector, using word-vectors in the KB. Then, during graph modeling step, each document-vector is converted into a small weighted graph called document-graph in which nodes represent dimensions of the vector and weights of the edges represent the relation between every two dimensions. Finally, the document-graphs are aggregated to create the KG. Note that KG is very small with the same number of nodes as the document graphs and encodes all frequent co-occurrence patterns in the entire dataset.

To extract co-occurrence patterns encoded in the KG first, we partition the graph into multiple dense sub-graphs and then, we use those sub-graphs to assign topics to each document. Based on that, we should select an appropriate partitioning method that provides scalability. There exist a large number of algorithms [11] for graph partitioning that their application depends on the type of the graph and the way the problem was modeled. In our case, since we modeled co-occurrence patterns into edge weights of the graph, we needed an algorithm to divide the graph into multiple groups of edges by cutting the nodes. Although there are scalable solutions for unweighted sparse graphs [12], but to the best of our knowledge there exists no appropriate solution matching our problem; One that can be directly used for node-cut partitioning of dense and weighed graphs. Therefore, we developed a new algorithm based on ideas from [12] that is known as a scalable solution for node-cut partitioning of un-weighted graphs. The details of the algorithm will be covered in Section 3.

The final step, after partitioning the KG, is to find the distribution of the topics for each document in the dataset. This process is done by first, overlapping the corresponding DG with the partitioned KG (see the colored DGs in the middle of Fig. 1-b) in order to assign topics to the document and then, obtaining the likelihood of each topic as the proportion of the weight of the edges in the DG related to that topic (e.g., D_1 corresponds 75% to the *Red topic*, T_1, and 25% to the *Blue topic*, T_3, as shown in Fig. 1-b).

Contribution: The main contributions of the paper are:

- An innovative document-graph modeling approach to extract and compress co-occurrence patterns in a large number of documents and transform those patterns into a small and highly dense and weighted graph.
- A novel scalable node-cut partitioning algorithm for extracting the topics encoded as a dense weighted graph.

We perform two sets of experiments to compare accuracy and scalability of our algorithm with two approaches namely LDA [13] [1] as the baseline and BTM [9] [2] as the current best known proposal to TD. For accuracy, we use the *SNAP-*

[1] http://gibbslda.sourceforge.net/

[2] https://github.com/xiaohuiyan/BTM

Twitter [3] dataset containing trending topics from 2009 and evaluate the results using B-Cubed [14] score. For scalability, we use the standard *TREC-Twitter2011* [4] dataset containing 16m Tweets and evaluate the results using a standard topic quality measure, called *Coherency* [15]. The results show that our algorithm not only achieves similar or better accuracy but also performs by an order of magnitude faster than the state-of-the-art approaches.

II. Related Work

A large number of solutions were proposed to address the problem of topic detection. Classical approaches relied on statistical methods like TF-IFD [16] or Latent Semantic Analysis (LSA) [3], [4]. The main limitations of these solutions were their scalability and accuracy.

The probabilistic approaches like PLSI [17] and LDA [5] were proposed to address those limitations by modeling the co-occurrence patterns as a probability distribution over documents and inferring the topics using statistical techniques like variational inference and Gibbs Sampling[13]. However, these approaches faced the same problems caused by the domination of short texts during recent years. They again, became un-scalable due to the large cardinality of short texts. Also, they failed creating good results due to the short length of the texts, a problem known as *Sparsity*.

To overcome sparsity a set of solutions, recently grouped under the title *Relational Topic Models (RTM)* by Chang et al. [18], were proposed. The main objective of these solutions was to enrich the problem space by correlating the documents incorporating external knowledge such as *content links*, in traditional web-documents or *network information*, in recent social media. For example Kohn and Hoffman [19] proposed to model content and inter-connectivity of documents using a join probabilistic approach. Weng et al. [8] also, suggested to aggregate Tweets with the same words. Other approaches like Hong et al. [7] and Chang et al. [18] proposed to use network information like publisher profile information (in OSNs) to model the relational semantics between the documents. The main limitation of these approaches is that they highly rely on external knowledge to train the parameters in their inference model. Such information is often hard to obtain specially in today's OSNs like Twitter.

Yan et al. proposed a different solution, called BiTerm (BTM) [9], where they tried to tackle the sparsity using an innovative bi-term document modeling approach. Authors modeled documents as a mixture of multiple bi-terms rather than single words. Their approach was shown successful to outperform relational topic models on large scale datasets of short texts. BTM is the most similar to DeGPar in terms of document modeling. However, DeGPar uses a graph analytical approach to extract topics. We compare the two approaches and show that DeGPar significantly outperforms BTM on scalability.

Despite a large group of approaches developed to solve sparsity there are not many solutions to account for scalability. We only, found a group of approaches that try to achieve scalability leveraging large-scale commodity servers. For example [20], [21] and [22] presented different variations of LDA [5], where they tried to overcome the scalability via parallelization. Also, [23] tried to achieve scalability by proposing a parallel Gibbs Sampling mechanism. These solutions are fundamentally based on the same ideas as LDA and thus, still computationally intensive. Our approach, in contrast, achieves scalability by reducing the size of the problem via dimensionality reduction. Therefore, given the same amount of resources, our solution can solve the problem faster than these approaches.

III. Solution

In this section we describe details of our solution in three steps. First, we explain how random indexing is used for compressing large number of documents into a dense and weighted graph, called KG. Then, we explain our node-cut partitioning algorithm for extracting topics from the KG. Finally, we explain the topic assignment, where we map each document into topics by matching them against the partitioned KG.

A. Knowledge-Graph Construction

A common practice in document-graph representation is to assign a node to each unique word in the dataset and express co-occurrences between words using edges. Such representation results in a large and sparse graph that endures inefficient graph calculations. To prevent this problem we create the graph, not based on the co-occurrences, but based on a compact representation of documents that we acquire using *Random Indexing (RI)*.

RI is a dimensionality reduction technique that allows us to uniquely express a large number of words in a dense representation without significant loss of information [24]. We use this technique to create a *Knowledge-Base (KB)*, which will later be used for construction of the Knowledge-Graph. The KB, shown in Fig. 1-a, is a list of unique words together with a unique vector for each word, called *Word-Vector (WV)*. It contains a large number of words including newly generated words (like hash-tags and acronyms) which are often ignored in existing solutions. The words are collected by constant monitoring of Tweeter streams and the WVs are created using RI. More specifically, after receiving a new Tweet from the stream, RI iterates over all words in the Tweet and for each non-function word W, creates a word-vector WV, as follows:

$$WV = LV + WV' + RV,$$

[3] https://snap.stanford.edu/data/twitter7.html
[4] http://trec.nist.gov/data/tweets/

Where LV is the vector corresponding to the left context word, RV is the vector corresponding to the right context word and WV' is the current vector corresponding to W itself. For each vector, LV, WV' and RV, if its related word has not been visited before, a random vector is created, otherwise, the current word vector will be used. For further details on RI please, refer to the original paper [10]. Here, we focus on two properties of the word vectors that are fundamental to our graph modeling:

1) Each WV has a fixed number of elements, d, of decimal values between 0 and 1. The values of the elements have an skewed distribution with a majority close or equal to 0.
2) Each WV encodes the entire observed history of the context surrounding its corresponding word.

The first property provides the means for dense representation of the documents, and the second property suggests that the WVs corresponding to similar words tend to be similar. Fig. 2 shows these two properties on a sample dataset of Tweets around the topic "Swine flu". We tokenized the documents, removed function-words and extracted 50 most frequent words in the dataset. The upper chart shows skewness in the distribution of values of elements of the WVs corresponding to the frequent words. The lower chart shows cosine similarity between WVs of the frequent words and the word "Swine flu" as the topic representative. As we can see, similar words (like "flu", "influenza", "virus" and "pandemic") have higher values in contrast to non-similar words (like "new-york", "states", "times" and "post"). Next, we describe how these properties are used in the process of KG construction in three steps.

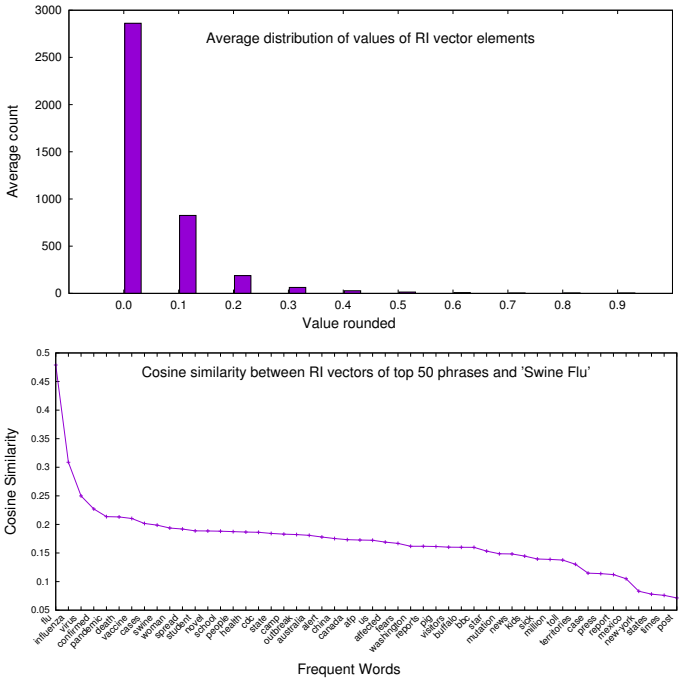

Fig. 2. Two properties of WVs in the KB depicted on a sample dataset of Tweets with topic "Swine Flu". Upper chart: Average distribution of the elements of WVs, corresponding to 51 most frequent words. Lower chart: Cosine similarity between 50 most frequent words and the topic word *"Swine Flu"*.

1) Compression: The first step is to compress each document into a d-dimensional vector, called *Document-Vector (DV)*, using WVs in the KB. The main goal is to combine the WVs such that the constructed DV preserves only the part of each WV that corresponds to its co-occurrence patterns related to the document. Fig. 3 shows this on a sample Tweet. The upper chart shows cumulative densities of WVs corresponding to three words, "kids", "swineflu" and "pigs" from the sample Tweet, *"kids get swineflu from pigs"*. Density is the aggregation of the elements of a vector sorted in descending order. For example, the red circle in the upper chart in Fig. 3 shows the aggregation of the first 500 elements of the WV corresponding to the word "swine flu". The curves show that more specific words like "swine flu" has lower density compared to more general words like "kids" or "pigs". This means, different combination of WVs may result in extraction of different semantic co-occurrence histories of their corresponding words. Therefore, we need to find a combination that extracts only the co-occurrence patterns of the words that is related to this context.

A trivial method is to iterate over words in each document, aggregate their corresponding WVs and normalize the values by the length of the document. This method, although creates a d-dimensional DV, but results in an excessively dense vector with a large number of unnecessary non-zero elements. Intuitively, since each WV encodes the entire observed contextual history and since, each word may appear in multiple semantically different contexts, the aggregation will result in excessive accumulation of all those semantic contexts in the corresponding DV, which does not reflect the specific co-occurrence pattern in the document. Another method is to multiply WVs. By multiplication, we mean the multiplication of mutually corresponding elements of the vectors. For example given two WVs with d elements, $V_1 = \{l_1, l_2, ..., l_d\}$ and $V_2 = \{m_1, m_2, ..., m_d\}$, their multiplication results in another vector, R, that is defined as $R = \{l_1 \times m_1, l_2 \times m_2, ..., l_d \times m_d\}$. This approach prevents high density and preserves the dimensionality. However, in contrast to aggregation, multiplication results in highly sparse DV representation. This can be inferred from the first property above, where most of the elements in WVs are close to zero, thus their multiplication results in high number of non-significant elements in the DV. In particular, multiplication will result in a DV with a highly specific pattern that is related to the co-occurrence of all the words in the document always together.

To prevent those problems, we apply a combination of both multiplication and aggregation, using bi-grams. In particular, for each document we construct the DV in three steps. First,

we extract all bi-grams in the document. Then, for every bi-gram we create a *Bi-gram-Vector (BV)* by multiplying the WVs corresponding to the two words in the bi-gram. Then, we aggregate all extracted BVs to create the desired DV. The lower chart in Fig. 3 shows why combined approach is better than mere multiplication or aggregation. It shows cumulative densities of three DVs created with different approaches, multiplication, aggregation and the combined approach. As we can see, the combined approach perfectly prevents the excessively high and low densities of the other two approaches. Therefore, we can show that the combined approach benefits us in two ways. First, since WVs contain co-occurrence history of the surrounding context, bi-gram multiplication extracts only the overlapping section of the two words, thus captures their co-occurrence pattern in the same semantic context. Second, since ambiguous words tend to share different parts of their vectors with other words from different senses and since it's too rare that two different ambiguous words appear in exactly two same semantic contexts this approach will reduce ambiguity by removing unrelated sections in the WVs of the ambiguous words through bi-gram multiplication. However, we can't totally prevent ambiguity since it's not possible to distinguish ambiguous words like "Apple" and "Blackberry" that appear in two semantic contexts (e.g., computers and fruits).

2) Graph Modeling:: In this step, the goal is to transform the co-occurrence patterns encoded in DVs to a graph representation. This process is performed in two steps. First, we assign a node to each element in DV, which will result in a graph, called *Document-Graph (DG)*, with d nodes and no edges. Then, for every two nodes, i and j in DG we multiply the corresponding values, v_i and v_j in DV and if the resulting value is larger than zero we create an edge, e_{ij} and assign that value as its weight. The result is a set of weighted and un-directed graphs of size d, corresponding to all document in the dataset, which form the building blocks for constructing the KG.

3) Aggregation: The last step is to aggregate the extracted DGs to create the KG. Since, all DGs are of the same size d, the KG will also be of size d. First, we create a graph with d nodes. Then, we iterate over all DGs and for every two nodes i and j, add the weight of the edge between them, W_{ij}, to its corresponding edge in the KG. This process creates the KG as an aggregation of the individual DGs, which contains all co-occurrence patterns in the dataset.

B. Knowledge-Graph Partitioning

In this section we first provide a formal definition of the problem of graph partitioning on a weighted graph and then present our solution.

1) Preliminaries: following definitions are required for understanding the problem description.

Sub-Graph: a sub-graph of a graph $G = (V, E)$ is a graph $G_i = (V_i, E_i)$ such that $V_i \subseteq V$ and $E_i \subseteq E$.

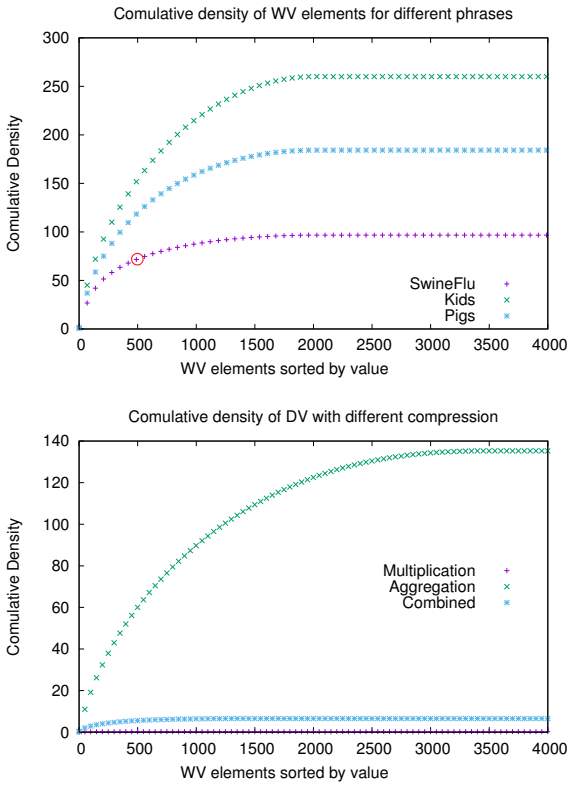

Fig. 3. Comparing different methods for DV creation on a sample Tweet, *"children get swine flu from pigs."* with three non function-words "children", "swine flu" and "pig". Upper chart: Cumulative density of WVs, extracted from KB. Lower chart: Cumulative density of DV created with different methods.

Weighted Density [25]: in a weighted and undirected graph $G = (V, E)$, is defined as the sum of the weights of the actual edges divided by the number of possible edges of weight 1:

$$WD = \frac{\sum_{e \in E} w_e}{n(n-1)/2}. \quad (1)$$

Node-Cut: is a division of a graph into multiple disconnected sub-graphs by dividing one or more nodes into multiple new nodes. The new nodes are called replicas. Each replica takes only a subset of the edges connected to the original node.

Weight of a replica: is defined as the total weight of the edges connected to that replica.

Cut-value (CV): is the value of cut on a node l, denoted by cv_l and is determined as,

$$CV_l = 1 - \frac{R_l^m}{|W|_l} \quad (2)$$

Where R_l^m is the weight of l's replica with maximum weight and $|W|_l$ is the total wight of the edges connected to l. The cut-value varies between 0 (when the node is not cut, $R_l^m = |W|_l$), and $1 - \frac{e_l^m}{|W|_l}$ (when there are same number of replicas as edges and the weight of maximum replica is equal to the maximum edge weight, $R_l^m = e_l^m$).

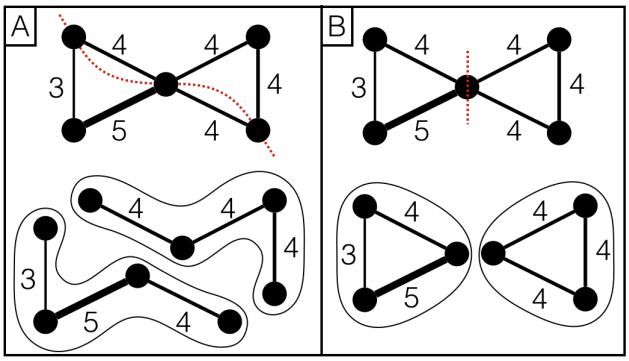

Fig. 4. Two sample node-cuts showing the relation between maximizing the total average weighted density and minimizing the cut-size. According to the *max-flow min-cut* theorem [26] these two approaches are equivalent and result in the same partitioning.

Cut-size (C): is a measure for determining the quality of a cut over a graph, G, calculated as the average cut-value over all nodes, n, in G,

$$C = \frac{1}{n} \times \sum_{i=1}^{n} cv_i. \tag{3}$$

2) Problem Definition: Balanced node-cut partitioning of a weighted graph is the problem of dividing the graph into a predefined number of sub-graphs, k, with the same total weights, $\frac{|w|}{k}$, and maximum average weighted density among all sub-graphs. This problem, according to *Max-Flow Min-Cut* [26] theorem, is equivalent to finding a node-cut with the minimum cut-size over the graph. Fig. 4 shows two balanced node-cut partitioning, A and B, with $k = 2$ on a synthetic weighted graph. The graph has total weigh 24 and the partitions are balanced with total weight 12 each. B shows the best partitioning with maximum average density 4 and minimum cut-size 0.1. In contrast, A depicts a partitioning with a lower average density 2 and a higher cut-size 0.3. As we can see, the average density and the cut-size are inversely proportional. Hence, we define our partitioning problem as the following optimization problem:

Definition-1: Given a weighted graph $G(V, E)$ and a predefined number of partitions k, the k-way balanced node-cut partitioning of G is defined as the problem of finding a cut that divides G into k sub-graphs, $SG = \{G_1, G_2, ..., G_k\}$ of equal size $\frac{|W|}{k}$, such that the cut-size, C, is minimized.

3) Solution: We developed an innovative algorithm based on ideas from [12] to solve the above optimization problem. We use colors on edges to represent partitions. A color is simply, a tag that is assigned to each edge to indicate the current partition that the edge belongs to. The number of colors is equal to the total number of partitions, k. The basic idea is to extract partitions through local exchanges of colors between edges. Initially, edges are colored randomly. We design an optimization process to swap edge colors in the graph, such that over time, each different color is populated in a different dense region of the graph. To enforce balance partitioning, the colors are assigned uniformly at random during the initialization round. This is done by considering the same amount of each color, $\frac{|W|}{k}$, for each partition, where $|W|$ is the total weight of the graph and k is the number of partitions. After the initialization the algorithm continues to perform the optimization.

The optimization is an iterative process over the nodes, where in each iteration every node performs a local optimization process to reduce its cut-value. To ensure the monotonous behavior of the global optimization process, thus, the convergence of the algorithm, we use a utility function that guaranties overall cut-value minimization upon each local optimization. The local optimization process in each node, shown in Algorithm 1, is performed as follows:

Every node, upon its turn for local optimization, first checks if it's not an internal node. An internal node is the one with the same color on all its incident edges. It's obvious that there is no reason to perform an optimization on an internal node as the cut-value is already zero and it can't be further improved. If the node is not internal, then it continues the optimization process by (i) computing its *Dominant Color (DC)*, the color with maximum volume among its incident edges and (ii) trying to reduce the cut-value by increasing the volume of the DC. To achieve that the node goes through a process called *Color Swapping* that changes the color of edges with non-DC to DC.

Color Swapping is performed in three steps. First, the node selects one of its edges with non-DC, called *Candidate* to swap its color. Then, it finds another edge in the network called *Partner*, which its color is equal to DC and is interested in the color of the Candidate. Finally, the node swaps the colors of the two edges. Such simple color-swapping mechanism is easy for un-weighted graphs. However, when the graph is weighted, the two edges cannot simply swap their colors because, it will result in the imbalance of the partitions if their weights are different. In particular, if two edges, e and e', with different weights, $w_e \neq w_{e'}$, swap their colors, c and c', they will change the total amount of the corresponding colors, thus the total size of the corresponding partitions in the graph by a quantity equal to the difference of their weights, $\gamma = |w_1 - w_2|$. A simple solution is to allow exchanges only between edges of the same weights. Our experiments showed the this approach is not effective, since the edges have a skewed weight distribution with a very few heavy-weighted edges, which will never get a chance to exchange their colors.

To solve this problem, we designed a new color-swapping mechanism, in which a node exchanges the color of a candidate with multiple partners. In this mechanism, in order to maintain the balance, the partners should have the same total weight as that of the candidate. However, since weights

Algorithm 1 Optimization

```
1: Pr(θ&δ)
2: g ← Graph
3: k ← Num_Partitions
4: GB ← Global_Bucket
5: procedure OPTIMIZATION(g)
6:     for all n ∈ g.nodes do
7:         if !n.isInternal() then
8:             DC = n.getDominantColor()
9:             C ← n.chooseCandidateEdge(DC)                          ▷ Algorithm 2
10:            if (|GB.getColor(DC) − ⌊|W|/k⌋| ≤ θ)&&(|GB.getColor(C.color) − ⌊|W|/k⌋| ≤ θ) then
11:                P[m] ← g.choosePartnerEdges(DC, C.weight, δ)       ▷ Algorithm 3
12:                SwapColor(C, P)
13:            end if
14:        end if
15:    end for
16: end procedure
```

are continuous value, finding such an exact partner set is highly expensive or even impossible in our weighted graph. Therefore, we present a new parameter, δ, to relax the balance constraint and instead, try to find a set of edges, which their total weight approximately matches that of the candidate edge. In particular, a node with a candidate edge e_c, with wight w_c, tries to find a set of m partners $E_p = \{e_1, e_2, ..., e_m | \forall i : e_i \neq e_c\}$ such that:

$$\sum_{e \in E_p} w_e = w_c \pm \delta$$

The approximate matching mechanism, explained above, trades the partition balance for convergence speed, which results in imbalance of the partition sizes in long term.

To control the imbalance ratio we use another mechanism, called *Global_Bucket (GB)*. The GB is a shared memory data-structure that keeps track of the sizes of the partitions and their variations during the execution of the algorithm. We use a parameter, θ, to control the deviation of partition sizes from the balance. This is shown in line 14 of Algorithm 1. Every node, after specifying the DC and selecting the candidate with color CC, checks the divergence of corresponding partitions with colors DC and CC from balance. If both partitions are less than θ away from the balance then, the node proceeds to the next steps to select partner edges and swap colors between partner edges and the candidate edge. This section explained the color swapping mechanism that involved candidate selection and partner selection. Next, we will explain different approaches for candidate and partner selection functions.

Candidate Selection: Algorithm 2 shows the overall process of candidate selection. A good color swapping, for a node, is the one that reduces the node's cut-value the most. This can be done by selecting the candidate edge with largest weight among all edges with non-CD and swapping its color to DC. To efficiently find such a candidate edge, the algorithm sorts all edges around each node in descending order and traverses through them until, it finds the first non-DC edge that its utility for swapping its color to DC is positive. If no candidate edge is found the algorithm will return null and the process will be canceled until the next iteration.

Algorithm 2 Candidate Edge Selection

```
1: procedure CHOOSECANDIDATEEDGE(DC, NC)
2:     if SE == null then
3:         SE ← edges.sortDes()        ▷ List of edges sorted in descending order
4:     end if
5:     for all e ∈ SE do
6:         if e.color ≠ DC && e.utility(NC) > 0 then
7:             return e
8:         end if
9:     end for
10: end procedure
```

Partner Selection: The goal of partner selection is to find as many edges with color DC from the KG as (i) their total weight approximately matches that of the candidate, and (ii) their utility for changing their color to that of the candidate is positive. The overall process of partner selection, is shown in Algorithm 3.

Algorithm 3 Partner Edge Selection

```
1: procedure CHOOSEPARTNEREDGES(DC, W)
2:     P[] ← ∅                                                  ▷ Partners list
3:     for all e ∈ g.edges do
4:         if e.color == DC && e.utility(DC) > 0 then
5:             P.add(e)
6:         end if
7:         if |W − Σ_{p∈P}(p.weight)| ≥ δ then
8:             break
9:         end if
10:    end for
11:    return P
12: end procedure
```

Utility: Before selecting any edge (candidate or partner) for color swapping, each node must calculate a utility value and make sure this value is positive. The utility for an edge that is going to swap its color c with a new color c', provides a comparison of the strength of the two partitions, indicated with the two colors, c and c', around the two end nodes connected to the edge. Intuitively, if the current partition, c, is weaker than the new partition, c', then, the utility will be positive and the edge will be interested in the swap. In particular, the utility

of an edge e, with two end nodes a and b that is going to swap its current color c, with a new color c' is calculated as follows:

$$U(c,c')_e = gain(c,c')_a^e + gain(c,c')_b^e$$

Where the gain of a node z with a set of edges E_z, for changing the color of its specific edge e, from c to c' is calculated as:

$$gain(c,c')_z^e = \frac{[\sum_{i\in E_z}(w_i^{c'})] - [\sum_{i\in E_z}(w_i^c) - w_e]}{\sum_{i\in E_z}(w_i)}$$

C. Topic Assignment

After partitioning the KG, the next step is to assign topics to documents. In previous step, we extracted topics as single-colored dense sub-graphs in the KG. To assign topics to each document we compare the corresponding document graph, DG, with extracted partitions. Thus, the likelihood of a topic z, given a document D, is determined as the ratio of the total weight of the edges in DG that overlap with edges belonging to topic z in KG over the total number of edges in DG,

$$P(z|D) = \frac{\sum_{\{\forall e: e\in DG, e\in z\}} w_e}{\sum_{\forall e\in DG} w_e},$$

This process will extract a probability distribution of the topics over each document and assign each document to multiple topics unless we clearly define a constraint that allows one topic to be representative topic of the document. This shows the ability of our method to function as an overlapping topic detection approach. In our method, we consider the topic with the highest likelihood among the topic distribution over a document as the representative topic of that document.

IV. Results

A. Experimental Setting

We developed two sets of experiments to compare the efficiency and scalability of DeGPar against two state-of-the-art approaches, LDA [5] and BTM [9]. We run each experiment 100 times on each algorithm and report the average result. A machine with 48 cores of 2GHz CPUs and 100GB RAM is used to perform all experiments. We empirically set the parameters δ and θ to 0.01 for the DeGPar. In both LDA and BTM we set the parameters to the best reported values, $\alpha = \frac{1}{50} \times k$ and $\beta = 0.005$, according to their original report.

B. Evaluation Metrics

1) B-Cubed [14]: Is an statistical measure for evaluating the accuracy of the results of topic detection in text. It calculates the accuracy of a classification for each document compared to a ground-truth. The results can be reported either per topic as average over all documents in that topic or the whole dataset as average over all documents in the dataset. In particular, given a dataset with n documents, tagged with k hand-labels, $L = \{l_1, l_2, ..., l_k\}$ and a classification of the documents into k class-labels, $C = \{c_1, c_2, ..., c_k\}$, the B-Cubed of a document, d, with hand-label l and class-label c is calculated as:

$$B(d) = 2 \times \frac{P(d) \times R(d)}{P(d) + R(d)},$$

where P and R stand for *Precision* and *Recall*, respectively and are calculated as follows:

$$P(d) = \frac{|d_i|_{\forall i: i\in c, i\in l}}{|d_i|_{\forall i: i\in c}}$$

$$R(d) = \frac{|d_i|_{\forall i: i\in c, i\in l}}{|d_i|_{\forall i: i\in l}}.$$

Precision shows the likelihood of documents correctly classified as c, with respect to the total number of documents in c and recall shows that likelihood with respect to the total number of documents in l.

2) Coherency: Is a metric proposed by [15] for measuring the quality of extracted topics in a topic classification solution. Given a set of documents classified into multiple similarity groups, coherency assumes that high-frequent words in each group, tend to co-occur more among the documents inside the group rather than documents across multiple groups. Thus, the higher the coherency, the better the quality of the partitioning. Based on that, the coherency is first, calculated for each similarity group in the classification result and then, the average over all individual coherency values is reported as the coherency of the classification result. In particular, given a set of documents classified into k topics, $T = \{t_1, t_2, ..., t_k\}$, first, we calculate the coherency of each topic, z, with top m probable words, $W^z = \{w_1, w_2, ..., w_m\}$, as,

$$C(z, W^z) = \sum_{i=2}^{m}\sum_{j=1}^{i-1} \log \frac{D({w_i}^z, {w_j}^z)}{D({w_j}^z)},$$

where $D({w_i}^z, {w_j}^z)$ is the co-occurrence frequency of the words v_i and v_j among documents in z and $D({w_j}^z)$ is the total frequency of w_j in z. Then, the total coherency of partitioning is calculated as follows:

$$C(T) = \frac{1}{k} \times \sum_{\forall z\in T} C(z, W^z).$$

C. Datasets

For experiments on accuracy we needed a test dataset that contains Tweets with hand-tagged topics. We used the SNAP-Twitter [27] dataset that contains 476M trending topics published in 2009. To create our test dataset, we used a taxonomy provided by Twitter that classifies different trending topics into a group of 7 top categories including "News Events", "People", "Movies", "TV-Shows", "Technologies", "Sports" and "Hashtags". Then, we chose 6 different topics, one from each category except "Hashtags" and extended it with 2 more topics, from "TV Shows" and "People" categories. The "Hashtags" was excluded since

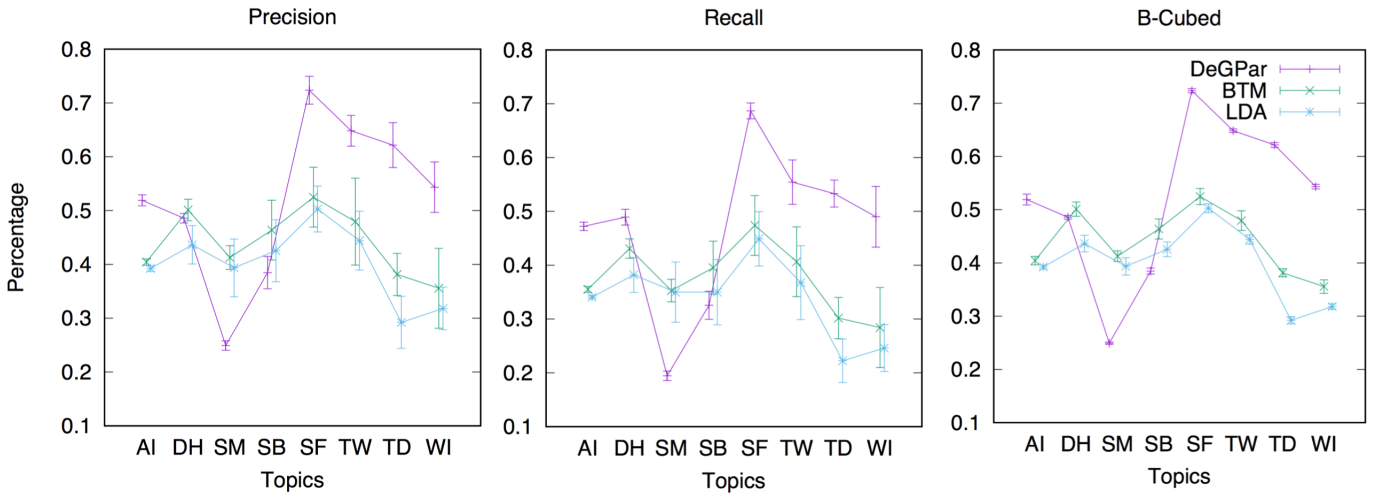

Fig. 5. Precision, Recall and B-Cubed values for DeGPar, BTM and LDA on SNAP dataset with 8 topic from 6 categories. DeGPar significantly outperforms BTM and LDA on most of the topics ($p < 0.001, \alpha = 1 - 0.95$). BTM, as expected, performs better than LDA due to the innovative bigram model.

TABLE I
SUMMARY OF THE TWEETER TOPICS SELECTED FROM SNAP DATASET.

Topic	Category	Acronym	Count
American Idol	TV Show	AI	4511
Dollhouse	TV Show	DH	1388
Slumdog Millionaire	Movies	SM	360
Susan Boyle	People	SB	1177
Swine Flu	News Event	SF	2195
Tiger Woods	People	TW	6841
TweetDeck	Technology	TD	6765
Wimbledon	Sports	WI	2440

the category covers a large range of topics that significantly overlap with topics in other categories. In contrast, we chose the last 2 topics from redundant categories to investigate the performance of the algorithm against overlapping topics. We extracted the Tweets related to each topic by searching their keywords in the dataset, assuming that a Tweet containing a specific keyword represents related topics. The collection was cleaned by removing HTML tags, URLs, function-words, non-English characters and all the words with less than 3 characters. Then, the Tweets with less than 3 words were excluded. It's also common to remove top 100 frequent words. As a result we ended up creating a dataset with 25677 Tweets. Table I shows a summary of the topics, their corresponding categories and the number of Tweets in each topic. Column three of the table shows the acronym of each topic-name that will later be used to refer to topics in the result Section.

For scalability experiments, we used a NIST standard dataset, called TREC-2011 [28], that contains around 16M Tweets collected in a 15 days period in 2011. After collecting the dataset, we first cleaned the Tweets following the same approach used for cleaning the SNAP dataset. Then, we created 5 different test datasets containing different number of Tweets, 10k, 100k, 1M, 3M and 5M, to compare the scalability of DeGPar with BTM against variations in the number of documents.

V. DISCUSSION

A. Accuracy

The comparison between the accuracy of the three approaches, DeGPar, BTM and LDA are shown in Fig. 5 in terms of precision, recall and B-Cubed score. We run all three algorithms on the SNAP dataset and extracted the results for each experiment in terms of precision, recall and B-cubed score for each topic. We performed a T statistical test to measure the significance of the performance of our approach. The results show that DeGPar is significantly better than both BTM and LDA with p-values < 0.01 on 95% confidence interval.

A noticeable point is DeGPar's performance on the two topics "Slumdog Millionaire (SM)" and "Sussan Boyle (SB)", which are the topics with smallest sizes according to Table I. This outcome shows the sensitivity of DeGPar against the topic sizes. In particular, since DeGPar uses a balanced partitioning algorithm, it tends to force the topics not to expand their boundaries beyond the average partition size, $\frac{|w|}{k}$. Therefore, it causes the smaller topics to expand further than their optimal boundaries on the graph and results in diminishing of the performance.

This problem can be solved by increasing the number of partitions (e.g., $k' > k$), which can cause extraction of highly specific topics by splitting true partitions. This is a common problem in any partitioning-based method. However, it is not a problem in our approach. The reason is, we are looking for trending topics in short texts, which are often very specific. For example, we are more interested in specific topics like "spread of deadly pandemic swine flu virus", rather than the

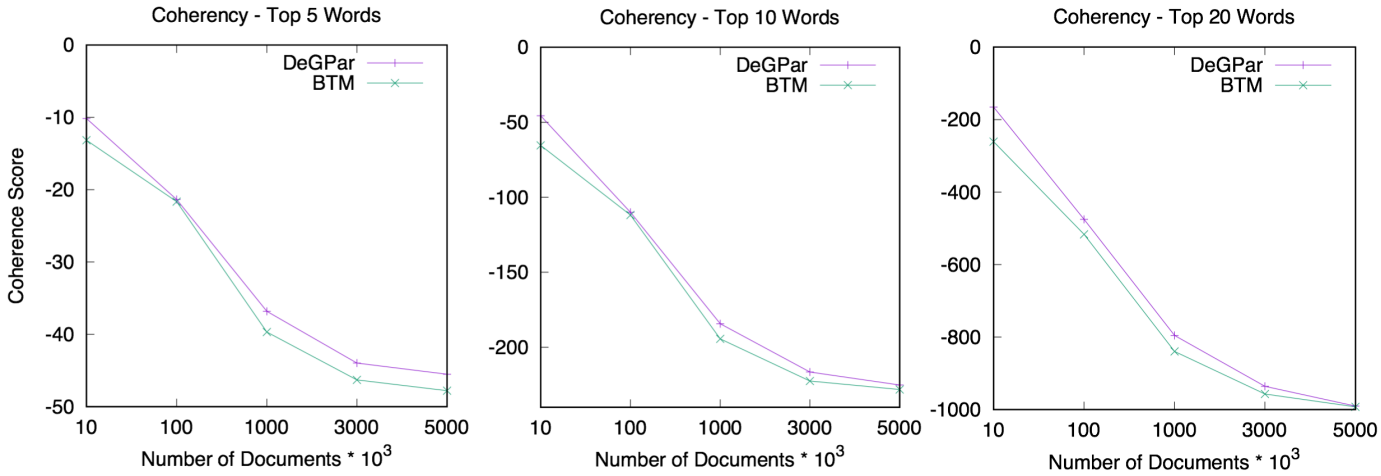

Fig. 6. The coherence-score of the top m words in each topic for BTM and DeGPar against different number of documents. The higher the value is the better. The results show that DeGPar always performs better or the same as BTM.

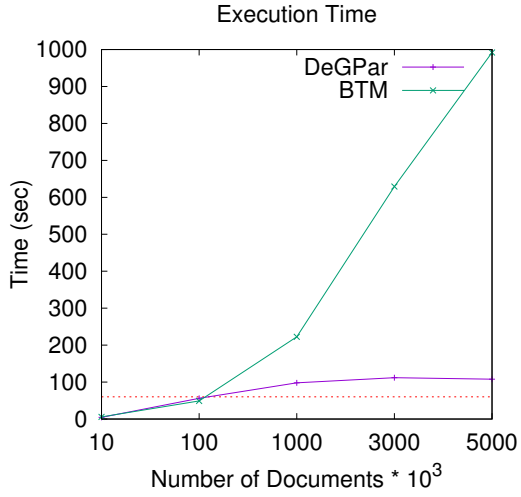

Fig. 7. Average execution time on different number of documents. DeGPer significantly outperforms BTM on scalability. BTM grows exponentially after around 100k documents, whereas DeGPar remains constant.

more general topics like ”Health” or ”Pandemic viruses”. Therefore, we don't consider this as a challenge in our approach. Instead, we allow our algorithm to increase the number of topics and extract more specific topics in the dataset.

B. Scalability

The results of the scalability experiments are summarized in two Figs. 6 and 7. The figures present performance and scalability of DeGPar in comparison to BTM. LDA has already shown less efficient compared with BTM, thus excluded from these experiments. The performance is shown for top 5, 10 and 20 words in terms of coherency in each experiment. The scalability compares the execution time between the two algorithms DeGPar and BTM in the order of seconds. Both results are shown against the variations of the size of the dataset. As we can see, DeGPar significantly outperforms BTM in terms of scalability. In particular, BTM shows an exponential execution time on all datasets with larger than 100k documents, whereas, DeGPar exhibits almost constant execution time over the entire results. In addition, the results of coherency, show that DeGPar always performs better than or similar to BTM in all experiments.

VI. Conclusion

We developed DeGPar, an innovative and scalable algorithm for large scale topic detection on short texts leveraging dimensionality reduction and graph analytics. The algorithm compresses a large number of documents into a small, highly dense and weighted graph representation using a dimensionality reduction technique called *Random Indexing*. The principal assumption is to apply the compression in a way that the frequent co-occurrence patterns in the documents create dense topological areas in the graph. Afterwards, the algorithm proceeds by extracting those patterns by partitioning the graph using a novel node-cut partitioning algorithm. Through a set of experiments, we examined the accuracy and scalability of our algorithm against two state-of-the-art approaches LDA [5] and BTM [9]. The results show that our algorithm outperforms both approaches by an order of magnitude while exhibiting similar accuracy.

We believe that our algorithm, due to the incremental nature of the underlying compression method has the potential to be extended to an streaming-based approach, where the number of topics can be dynamically specified and the algorithm will be able to process an stream of documents (like a Twitter stream). In future, we are going to work on a community detection algorithm that would work with dense weighted graphs and be able to efficiently find the appropriate number of partitions and account for variations in the distribution of topic sizes.

REFERENCES

[1] J. Allan, J. Carbonell, G. Doddington, J. Yamron, Y. Yang, J. A. Umass, B. A. Cmu, D. B. Cmu, A. B. Cmu, R. B. Cmu, I. C. Dragon, G. D. Darpa, A. H. Cmu, J. L. Cmu, V. L. Umass, X. L. Cmu, S. L. Dragon, P. V. M. Dragon, R. P. Umass, T. P. Cmu, J. P. Umass, and M. S. Umass, "Topic detection and tracking pilot study final report," in *In Proceedings of the DARPA Broadcast News Transcription and Understanding Workshop*, 1998, pp. 194–218.

[2] T. S. Portal. (2016) Number of monthly active twitter users worldwide from 1st quarter 2010 to 1st quarter 2016 (in millions). [Online]. Available: http://www.statista.com/statistics/282087/number-of-monthly-active-twitter-users/

[3] T. K. Landauer and S. T. Dumais, "A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge," *Psychological review*, vol. 104, no. 2, pp. 211–240, 1997.

[4] T. K. Landauer, P. W. Foltz, and D. Laham, "Introduction to latent semantic analysis," *Discourse Processes*, vol. 25, pp. 259–284, 1998.

[5] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, mar 2003. [Online]. Available: http://dl.acm.org/citation.cfm?id=944919.944937

[6] M. Steyvers and T. Griffiths, *Latent Semantic Analysis: A Road to Meaning*. Laurence Erlbaum, 2007, ch. Probabilistic topic models.

[7] L. Hong and B. D. Davison, "Empirical study of topic modeling in twitter," in *Proceedings of the First Workshop on Social Media Analytics*, ser. SOMA '10. New York, NY, USA: ACM, 2010, pp. 80–88. [Online]. Available: http://doi.acm.org/10.1145/1964858.1964870

[8] J. Weng, E.-P. Lim, J. Jiang, and Q. He, "Twitterrank: Finding topic-sensitive influential twitterers," in *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, ser. WSDM '10. New York, NY, USA: ACM, 2010, pp. 261–270. [Online]. Available: http://doi.acm.org/10.1145/1718487.1718520

[9] X. Yan, J. Guo, Y. Lan, and X. Cheng, "A biterm topic model for short texts," in *Proceedings of the 22Nd International Conference on World Wide Web*, ser. WWW '13. New York, NY, USA: ACM, 2013, pp. 1445–1456. [Online]. Available: http://doi.acm.org/10.1145/2488388.2488514

[10] M. Sahlgren, "An introduction to random indexing," in *In Methods and Applications of Semantic Indexing Workshop at the 7th International Conference on Terminology and Knowledge Engineering, TKE 2005*, 2005.

[11] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," *CoRR*, vol. abs/1311.3144, 2013. [Online]. Available: http://arxiv.org/abs/1311.3144

[12] F. Rahimian, A. H. Payberah, S. Girdzijaskas, and S. Haridi, "Distributed vertex-cut partitioning," *Distributed Applications and Interoperable Systems*, 2014.

[13] X.-H. Phan and C.-T. Nguyen, "Gibbslda++: A c/c++ implementation of latent dirichlet allocation (lda)," 2007.

[14] A. Bagga and B. Baldwin, "Entity-based cross-document coreferencing using the vector space model," in *Proceedings of the 17th International Conference on Computational Linguistics - Volume 1*, ser. COLING '98. Stroudsburg, PA, USA: Association for Computational Linguistics, 1998, pp. 79–85.

[15] D. Mimno, H. M. Wallach, E. Talley, M. Leenders, and A. McCallum, "Optimizing semantic coherence in topic models," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, ser. EMNLP '11. Stroudsburg, PA, USA: Association for Computational Linguistics, 2011, pp. 262–272. [Online]. Available: http://dl.acm.org/citation.cfm?id=2145432.2145462

[16] T. Brants, F. Chen, and A. Farahat, "A system for new event detection," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, ser. SIGIR '03. New York, NY, USA: ACM, 2003, pp. 330–337. [Online]. Available: http://doi.acm.org/10.1145/860435.860495

[17] T. Hofmann, "Probabilistic latent semantic indexing," in *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '99. New York, NY, USA: ACM, 1999, pp. 50–57. [Online]. Available: http://doi.acm.org/10.1145/312624.312649

[18] J. Chang and D. M. Blei, "Relational topic models for document networks," in *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*, D. V. Dyk and M. Welling, Eds., vol. 5. Journal of Machine Learning Research - Proceedings Track, 2009, pp. 81–88. [Online]. Available: http://jmlr.csail.mit.edu/proceedings/papers/v5/chang09a/chang09a.pdf

[19] D. A. Cohn and T. Hofmann, "The missing link - a probabilistic model of document content and hypertext connectivity," in *Advances in Neural Information Processing Systems 13*, T. K. Leen, T. G. Dietterich, and V. Tresp, Eds. MIT Press, 2001, pp. 430–436. [Online]. Available: http://papers.nips.cc/paper/1846-the-missing-link-a-probabilistic-model-of-document-content-and-hypertext-connectivity.pdf

[20] Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang, "Plda: Parallel latent dirichlet allocation for large-scale applications," in *Proceedings of the 5th International Conference on Algorithmic Aspects in Information and Management*, ser. AAIM '09. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 301–314. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02158-9_26

[21] Z. Liu, Y. Zhang, E. Y. Chang, and M. Sun, "Plda+: Parallel latent dirichlet allocation with data placement and pipeline processing," *ACM TIST*, vol. 2, p. 26, 2011.

[22] J. Chen, K. Li, J. Zhu, and W. Chen, "Warplda: A cache efficient o(1) algorithm for latent dirichlet allocation," *Proc. VLDB Endow.*, vol. 9, no. 10, pp. 744–755, Jun. 2016. [Online]. Available: http://dx.doi.org/10.14778/2977797.2977801

[23] Y. Gao, J. Chen, and J. Zhu, "Streaming gibbs sampling for lda model," *CoRR*, vol. abs/1601.01142, 2016.

[24] W. B. Johnson and J. Lindenstrauss, "Extensions of lipschitz mappings into a hilbert space," *Contemporary mathematics*, vol. 26, no. 189-206, pp. 1–1, 1984.

[25] F. Cicalese and J. A. Amgarten Quitzau, "2-Stage Fault Tolerant Interval Group Testing," in *Algorithms and Computation. 18th International Symposium, ISAAC 2007, Sendai, Japan, December 17-19, 2007. Proceedings*, T. Tokuyama, Ed., 2007, pp. 858–868.

[26] L. R. Ford and D. R. Fulkerson, *Maximal Flow Through a Network*. Boston, MA: Birkhäuser Boston, 1987, pp. 243–248. [Online]. Available: http://dx.doi.org/10.1007/978-0-8176-4842-8_15

[27] J. Yang and J. Leskovec, "Patterns of temporal variation in online media," in *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, ser. WSDM '11. New York, NY, USA: ACM, 2011, pp. 177–186. [Online]. Available: http://doi.acm.org/10.1145/1935826.1935863

[28] N. Baten. (2011) Tweets2011. [Online]. Available: http://trec.nist.gov/data/tweets/