NIFTY: A System for Large Scale Information Flow Tracking and Clustering

Caroline Suen*, Sandy Huang*, Chantat Eksombatchai*, Rok Sosič, Jure Leskovec
Stanford University
{csuen, shhuang, chantat, rok, jure}@cs.stanford.edu

ABSTRACT

The real-time information on news sites, blogs and social networking sites changes dynamically and spreads rapidly through the Web. Developing methods for handling such information at a massive scale requires that we think about how information content varies over time, how it is transmitted, and how it mutates as it spreads.

We describe the *News Information Flow Tracking, Yay!* (NIFTY) system for large scale real-time tracking of "memes" — short textual phrases that travel and mutate through the Web. NIFTY is based on a novel highly-scalable incremental meme-clustering algorithm that efficiently extracts and identifies mutational variants of a single meme. NIFTY runs orders of magnitude faster than our previous MEMETRACKER system, while also maintaining better consistency and quality of extracted memes.

We demonstrate the effectiveness of our approach by processing a 20 *terabyte* dataset of 6.1 *billion* blog posts and news articles that we have been continuously collecting for the last four years. NIFTY extracted 2.9 billion unique textual phrases and identified more than 9 million memes. Our meme-tracking algorithm was able to process the entire dataset in less than five days using a single machine. Furthermore, we also provide a live deployment of the NIFTY system that allows users to explore the dynamics of online news in near real-time.

Categories and Subject Descriptors: H.2.8 **[Database Management]**: Database applications—*Data mining*
General Terms: Algorithms; Experimentation.
Keywords: Networks of diffusion, Information cascades, Blogs, News media, Meme-tracking, Social networks.

1. INTRODUCTION

In less than a decade, the World Wide Web has transformed from a large, static library that people only browse into a vast and dynamic information resource. Today, for example, large media houses and TV stations, small local newspapers, professional online bloggers, as well as casual bloggers and citizen journalists are continuously capturing the pulse of humanity: what we are thinking, what we are doing, and what we know.

Since the early days of the Web, online information content has taken on increasingly dynamic forms, to the extent that the real-time aspect of information has become one of the most pressing concerns in the processing and tracking of Web content. Information on the Web changes rapidly over time due to the rate of production, as well as the ways in which it is transmitted through the network, from website to website. Developing methods for handling such dynamic information at a massive scale poses many challenges. For example, it requires us to think about how content varies and mutates as it is being transmitted over underlying networks. A better understanding of how information spreads on the Web has many practical applications in a wide range of fields, such as social sciences, marketing, and politics.

The goal of this paper is to develop a system that is able to *track information* as it spreads across *billions of documents* on the Web and over time periods spanning *many years.* Studying the spread of online information has been an active research area [1, 3, 15, 21, 39], but reliably tracking the content flow has been extremely challenging [3, 21]. While there has been work on tracking topics [5, 7, 36, 40], tags [13, 24], and keywords [8, 37], scaling difficulties have made it harder to track information across whole websites or across the entire Web.

Additional challenges stem from the fact that for such whole-Web tracking, it is difficult to find the right granularity at which to study the movement of information. For example, entire articles or blog posts (except in rare cases) do not spread and propagate on the scale of the whole-Web [3, 15, 21]. Similarly, terms [16, 18] or topic clusters [7, 36, 40] (e.g. "the presidential election," "the war in Iraq") are generally too broad to truly capture the fine-grained elements of information mutation and diffusion.

In order to study the emergence and the dynamics of Web information, we need to identify the basic units of information that propagate through the Web. We require a level of granularity that is balanced between the coarse-grained structure of whole pages, articles, or posts, and the fine-grained structure of terms, keywords, or topic labels. And even if the basic units of information are successfully identified, the challenge resulting from information that is constantly evolving and mutating as it spreads on the Web remains. Thus it is important to have a robust method to discover and track different mutational variants of the same piece of information.

The sheer volume and time span of Web data requires a system that can process tens of millions of documents per day and billions of documents over several years. Traditional methods for tracking information flow have quadratic time complexity in the number of documents, which makes their running time prohibitively large for practical use over longer time periods. Also, we cannot hope to load all the data in memory, so efficient online incremental algorithms for tracking information flow are needed. And as we would like to run our system over periods spanning several years, a challenge is that our method must not degrade in performance over time.

*Authors contributed equally to this work.

WWW 2013, May 13–17, 2013, Rio de Janeiro, Brazil.
ACM 978-1-4503-2035-1/13/05.

Figure 1: Visualization of the most popular memes in the month of October 2012, with notable memes labeled. Each band represents a meme and the height of the band corresponds to the number of mentions per hour. Real-time visualization is available at http://snap.stanford.edu/nifty.

Overview of results. In this paper we present the *News Information Flow Tracking, Yay!* (NIFTY) system for large scale real-time tracking of new topics, ideas and "memes" across the Web. NIFTY efficiently extracts and traces textual *memes* [20] — short phrases that change, yet remain relatively intact as they propagate from website to website and from blog to blog. NIFTY applies a novel highly-scalable incremental meme-clustering approach for extracting and identifying mutational variants of textual memes. Example output of our system for October 2012 is displayed in Figure 1. NIFTY clearly identified a number of popular memes from that month, such as memes associated with the U.S. presidential election ("binders full of women", "horses and bayonets") as well as memes related to Hurricane Sandy ("this storm is dangerous").

NIFTY builds on our previous MEMETRACKER [20] approach in the sense that it operationalizes the notion of a meme through quotations. Quotations appear in documents as quoted phrases or sentences, operating at a level of granularity between that of individual articles and broad topics. We define memes by combining similar quoted phrases in clusters, each cluster representing one meme.

NIFTY operates as follows. First, we extract quoted phrases from input documents. Once the quoted phrases are extracted, we cluster all the different mutational variants of the same quote to obtain meme clusters. It is challenging to cluster phrases, given that we are working with billions of them. As phrases are short textual fragments containing very little information, traditional bag-of-words representations do not work well. Moreover, we are not interested in simply clustering together all the phrases on the same topic, but rather we want to find all the phrases that evolved through one or more mutational steps from the same original phrase.

Our approach to the problem of identifying meme evolution [20] applies ideas from biological sequence analysis, where we aim to "align" two given phrases using a form of string edit distance and then determine whether one could have evolved from the other. We build a giant *phrase graph* where phrase a has a directed edge to phrase b if there is evidence that a could have evolved from b. We then partition this graph into clusters, such that each cluster is a directed acyclic graph (DAG) with a single root node. The clusters then represent memes, each cluster root node is an original meme phrase and the rest of the cluster nodes are mutational variants of the original phrase. For example, Figure 2 shows a graph of the mutational variants of a meme related to the discovery of the Higgs

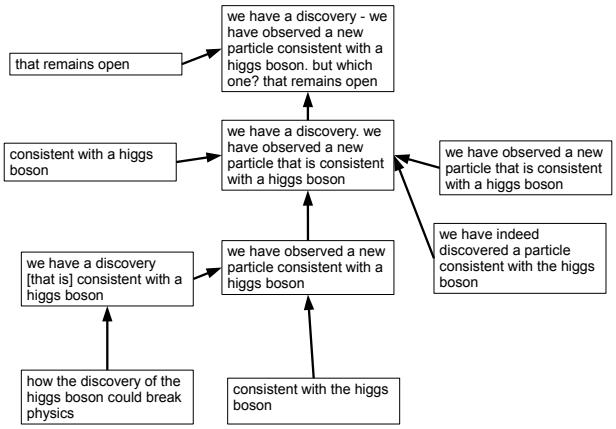


Figure 2: A small subset of mutations of a meme about the discovery of the Higgs boson particle.

boson particle. The advantage of using a graph of phrase variants is that it automatically produces the lineage or ancestry information (as illustrated in Figure 2).

An additional challenge when one works with phrases appearing over longer time periods (e.g., several months or more) is the formation of a "giant component" containing a significant fraction of all phrases. A low distance between two phrases from different, frequently occurring memes can cause their corresponding meme clusters to fuse together, leading to one large cluster over time. As this obscures the structure we are trying to identify, we developed a novel incremental clustering approach to finding mutational variants of a single meme. We keep a dynamic version of the phrase graph and we constantly add new phrases to it. By maintaining the dynamic structure of the phrase graph, we achieve both run time efficiency as well as improved cluster quality. A second essential innovation is that we also continuously remove finalized meme clusters and their corresponding phrases. This means that memes (phrase clusters) that stopped evolving get removed from the graph before a "giant" meme could start swallowing all the phrases.

We demonstrate the effectiveness of our approach by processing a 20 *terabyte* dataset of 6.1 *billion* blog posts and news articles

that we have been collecting for the last four years. This dataset essentially represents complete online news coverage during those four years. NIFTY extracted 2.9 billion unique textual phrases and identified more than 9 million unique memes. Our efficient online incremental algorithm was able to process the massive 20TB dataset in less than five days using a single machine and only 60GB of main memory. Incremental meme-clustering has linear runtime in the number of phrases while maintaining cluster consistency and quality provided by MEMETRACKER. Since the input data is processed incrementally, clustering time depends only on the volume of new data, which is practically constant per time unit, and not on the increasing size of the entire dataset as is the case with the offline batch methods. NIFTY thus allows us to quickly process datasets that are beyond the reach of the offline batch methods.

Last, we also deployed our system so that it processes Web documents in near real-time. The system extracts and clusters memes and then employs data visualization and data exploration techniques that allow users to explore the ongoing dynamics of online news. For example, Figure 1 is a screenshot of our interactive data visualization across a one month period, October 2012.

To the best of our knowledge, the present study analyzes one of the largest collections of news media documents and blog posts. In fact, the largest existing study [20] analyzed 90 million documents. Here we increase the scale of the analysis by 60 fold to 6 billion documents, while requiring approximately the same time and hardware resources. More broadly, we believe that our investigations have the potential to transform our understanding of how to manage real-time Web information, as well as our understanding of the evolving landscape of online news and commentary. The contributions of our present work are the following:

- A novel highly-scalable incremental meme-tracking approach for extracting and identifying mutational variants of short textual phrases that spread through the Web.
- A system level implementation of the incremental meme-tracking approach.
- An application of meme-tracking on 6 billion news articles and blog posts that we collected over the past 4 years.
- A live deployment of the NIFTY system to allow users to explore the dynamics of online news in real-time, available at `http://snap.stanford.edu/nifty`.

Further related work. Taken more broadly, our work here contributes to the growing literature on tracking and studying information dynamics on the Web. Two dominant themes in this work have been the use of algorithmic tools for organizing and filtering news, and the dynamics of online media content. Some of the key research issues concern the filtering and aggregation of online news [6, 12, 14]. While entity based [19, 22] as well as whole-document based [4, 25, 29, 30] approaches have been considered in the past, our work provides a new dimension in which we consider using short textual phrases to track related pieces of information. Another important distinction is that our methods scale to a truly massive dataset, while requiring only modest hardware resources for processing.

Meme-tracking has been successfully applied to other domains and use cases as well. For example, meme-tracking has been used to identify temporal variation of online media news content [10, 28, 39] as well as to reason about the dissemination and mutation of online information [2, 27, 31, 38]. NIFTY complements this line of work as we show that meme-tracking can be efficiently applied to billion-document datasets in order to gain insights about the dynamics of online information.

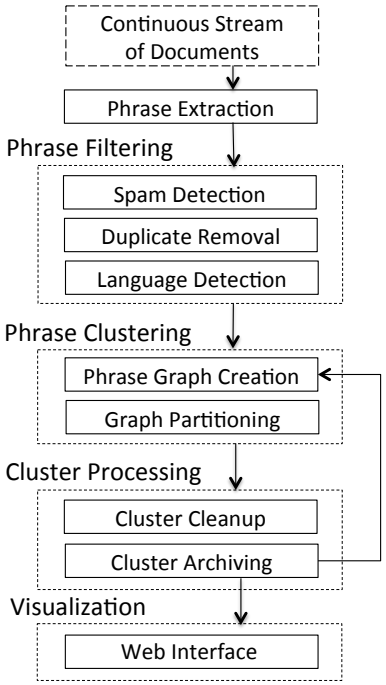


Figure 3: Overview of the NIFTY pipeline.

2. PROPOSED METHOD

NIFTY tracks memes by partitioning quoted phrases from input documents into clusters, such that each cluster represents one meme and the phrases in the cluster are mutational variants of a single original phrase. We now discuss the methods that NIFTY uses to track memes, including our approach to incremental clustering.

NIFTY processes documents through the pipeline of steps illustrated in Figure 3. We start with phrase extraction and proceed with *filtering*, which removes spam and duplicate content. The filtering step is often underappreciated, yet it is crucial for achieving high quality output. Next in the processing pipeline is *phrase clustering*, which groups phrases (i.e., mutational variants) that belong to the same meme. Here, the key challenge is that memes experience significant mutations, so usual text-based distance measures and standard clustering approaches do not work well [20]. The last two steps of the processing pipeline are *cluster processing* and *visualization*. The cluster processing step performs final quality checks of the output, removes old clusters, archives finalized clusters, and provides data updates for incremental clustering. Then, the visualization step uses the finalized cluster archive to present meme clusters via a Web interface.

Next we describe each of these steps in more detail. For ease of exposition we describe the pipeline as if the system operates in a batch setting. Later we discuss how NIFTY extends this batch setting to an incremental, stream based model.

2.1 Phrase Extraction

The input of NIFTY is a set of documents D, where each document in D represents an item from the Web, such as a news report or a blog post. A document contains the document URL, an estimated document publish time, and a list of phrases found in the document.

NIFTY operationalizes the notion of a meme through the extraction of short quoted phrases [20]. Building memes from quotes is natural since quotes are an integral aspect of journalistic practice;

even if a news story is not specifically about a particular quote, quotes are deployed in essentially all stories, and they tend to travel relatively intact through iterations of the story as it evolves [34]. They are also fairly unambiguous and readily identifiable, so that we are studying elements recognizable to consumers of the media, rather than the output of a complex clustering procedure.

2.2 Document and Phrase Filtering

Input documents for NIFTY are crawled off the Web and collected using RSS feeds. Such a data gathering procedure is focused on high coverage and volume. However, this means our data is inundated with spam, duplicates, and irrelevant information. Thus, the task of the filtering step is to clean up the input and select only English phrases, since we want to study only memes in English.

The filtering step passes over data twice. The first pass applies a number of simple heuristics to quickly eliminate as many bad documents and phrases as possible. With fewer documents and phrases, the second pass can use more computationally demanding methods to apply additional filtering criteria.

First pass: Filtering documents and phrases. Documents are eliminated during the first pass, if they are duplicates or their URLs are found on a manually maintained, short blacklist. Phrases are eliminated based on their length and on the portion of ASCII characters.

Phrase length. Intuitively, short phrases do not provide any useful information, while long phrases tend to not be memes. We experimented with different thresholds and found that the best balance between eliminating redundant phrases and preserving useful ones is to discard phrases with less than 3 or more than 30 words.

ASCII characters. As a quick test for English phrases, we apply a simple heuristic that requires at least 50% of the characters in the phrase to be ASCII characters.

Second pass: Advanced phrase filtering. During the second pass, the phrases and their corresponding documents are eliminated based on the phrase frequency, language filtering and URL to domain name ratio.

Infrequent phrases. To further remove noise and spam, we discard all phrases that appear in fewer than 5 distinct documents.

Language filtering. As mentioned, we are interested only in English phrases. To further filter only English documents, we compute for every phrase p the percentage of its words that appear in a list of the 1000 most common English words [35]. After experimenting with a range of values, we found that the best results are obtained by discarding all phrases where the percentage is less than 75%.

URL to domain name ratio. Spam is frequent in our dataset. One property of spam phrases is that they occur frequently but come from a small number of domains. We found the following rule to be our best strategy to remove spam phrases, as most spammers use only a few select domains to publish. If a phrase p appears at more than 20 URLs, we compute a ratio of unique URLs to the number of their unique host names (e.g. "www.cnn.com"). We remove p if this ratio is greater than 6.

Output. At the end of our filtering step we have a sanitized set of documents D, referred to as the **document base**, and a post-filtered set of phrases P, referred to as the **phrase base**. These two sets are used in the subsequent phrase clustering step.

2.3 Phrase Clustering

As memes spread through the Web, they change and mutate. For example, Figure 2 displays a sample of mutational variants of a meme concerning the discovery of the Higgs boson particle. Thus, our next goal is to partition the phrase base P into meme clusters, so that the phrases from the same meme are combined into a single cluster. The clustering of phrases requires a non-traditional clustering approach, since two phrases might have only few words in common, yet they could belong to the same meme.

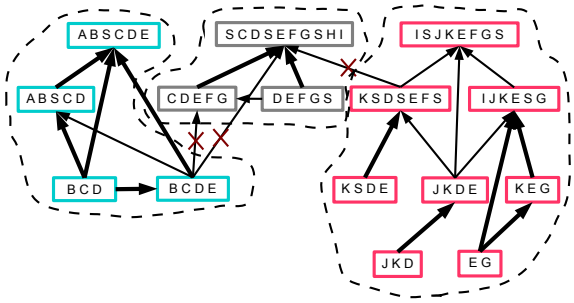


Figure 4: Example phrase graph, in which letters represent words. The letter 'S' indicates a stop word. Directed edges indicate the source phrase is approximately included in the destination phrase. For this example, the edge weight is inversely proportional to the substring edit distance between the two phrases, and is indicated by the thickness of the edge. Deleting the crossed-out edges gives the optimal clustering, shown by the dotted boundaries.

NIFTY extends and heavily improves the phrase clustering algorithm first proposed by MEMETRACKER [20]. The central concept of the algorithm is a phrase graph, which is a directed weighted acyclic graph. Each phrase in the phrase base P is a node in the graph and weighted edges are formed between the nodes based on the textual similarity of the corresponding phrases. After the graph is created, edges are pruned from the graph in order to split the graph into disconnected components with a single root phrase. Each disconnected component then corresponds to a meme where all the nodes (phrases) in the component are meme's mutational variants. In the following, we provide details about the phrase graph creation and partitioning steps.

Phrase graph creation. The purpose of the phrase graph creation step is to construct a directed weighted acyclic graph where phrases are nodes and pairs of similar phrases are connected by directed edges, pointing from shorter phrases to longer ones (Figure 4). The edges capture the intuition that quoted phrases are generally being shortened as they spread over the Web. At the end of this step, every phrase will be connected to a set of its potential "parents", i.e., phrases from which it could have mutated.

In order to create edges between phrases that could have evolved from one another, we apply ideas from biological sequence analysis. We aim to "align" two given phrases using a variant of string edit distance and then determine whether one could have evolved from the other.

Phrase distance. We expand the string edit distance to substrings and determine phrase distances in NIFTY by using what we call *substring edit distance*, an extension of the Levenshtein distance algorithm. Given two strings, we define the substring edit distance to be the minimum number of word insertions, deletions or substitutions needed to transform one string into a *substring* of the other string. We compute the substring edit distance by using a variant of the dynamic programming algorithm for the Levenshtein distance, removing stop words and using stemmed words during this computation.

Algorithm 1 Decision tree to determine if an edge should be created between phrases in the phrase graph

Require: Phrases p_1 and p_2
(1) Set l_p to the minimum number of words in p_1 or p_2.
(2) Compute s_1 and s_2 by stripping the stop words from p_1 and p_2, respectively.
(3) Set l_s to the minimum number of words in s_1 or s_2.
(4) Set d to the substring edit distance between s_1 and s_2.
if $l_s \geq 2$ and $d = 0$ **then**
  return $True$
else if $l_p = 4$ and $l_s = 4$ and $d \leq 1$ **then**
  return $True$
else if $l_p = 5$ and $l_s > 4$ and $d \leq 1$ **then**
  return $True$
else if $l_p = 6$ and $l_s \geq 5$ and $d \leq 1$ **then**
  return $True$
else if $l_p > 6$ and $l_s > 3$ and $d \leq 2$ **then**
  return $True$
else
  return $False$

Edge creation. Next, given a pair of phrases and their substring edit distance, we must determine whether one of them is derived from the other one and thus should be connected by an edge. We hand labeled 1,000 pairs of phrases that are mutational variants of each other as well as 1,000 phrases that are textually similar but are not mutational variants. We then trained a decision tree to distinguish between these two classes of phrases (Algorithm 1). An edge is created between two phrases in the graph, if the decision tree returns $True$.

Speeding up phrase graph creation. A straightforward approach to edge creation performs a pair-wise distance calculation between all the phrases in the phrase base, requiring quadratic number of calculations. This approach is prohibitively slow given the scale of our problem and the size of our dataset.

NIFTY drastically reduces the required number of distance calculations by using locality-sensitive hashing [17] (LSH), an efficient algorithm for identifying candidate pairs of phrases that could potentially be textually similar. LSH uses hash functions to place items into a number of buckets, so that similar items are placed in the same bucket with high probability. Since good locality sensitive hashing functions for substring edit distance are not available, we use shingling [23] with min-hashing [9] (see [26] for an overview) to find candidate pairs of phrases. Although min-hashing is commonly used to identify pairs of phrases with low Jaccard distance, our approach of applying it for substring edit distance makes intuitive sense. We use Jaccard distance as a lower bound for substring edit distance, as high Jaccard distance implies also high edit distance. Our experimental results in Section 3.1 confirm this intuition that LSH with min-hashing works well for substring edit distance and meme clustering.

We create four-character *shingles* from phrases, perform min-hashing by randomly shuffling unique shingles into bands of shingle permutations, and compute phrase signatures. The signature value for each permutation is the index of the first shingle in the permutation that exists in that phrase. Finally, each pair of phrases that have a signature band in common is tested for an edge existence, using the decision tree in Algorithm 1.

Assigning edge weights. Next, edges in the phrase graph are assigned weights based on findings by Yang [39] that most news follow a predictable popularity cycle with two main peaks, one from traditional news reporting and one from blog posts. For an edge from node p_s to p_d, NIFTY uses time and substring edit distance between the nodes and calculates weight for the edge as follows:

$$w(p_s, p_d) = c \cdot \frac{|p_d|}{(D_{edit}(p_s, p_d) + 1) \cdot (T_{peak}(p_s, p_d) + 1)}.$$

$|p_d|$ is the number of documents containing p_d, $D_{edit}(p_s, p_d)$ is the substring edit distance between the phrases, and $T_{peak}(p_s, p_d)$ is the time difference between the first volume peaks for each of the two phrases. We chose this formula so that the edge weight $w(p_s, p_d)$ is proportional to the popularity of p_d and the likelihood that p_s and p_d are mutational variants of the same original meme phrase. Intuitively, if p_s and p_d have a small edit distance and their frequencies first peaked at about the same time, they are more likely to be mutational variants of the same original meme phrase.

At the end, we have a directed acyclic graph $G = (P, E)$ where each phrase $p \in P$ is a node and pairs of similar phrases have weighted edges between them, connecting a phrase to all its potential parents. To obtain clusters, we next partition the phrase graph.

Phrase graph partitioning. Ideally, the creation step produces a phrase graph, such that the connected components of the graph correspond to memes and thus all mutational variants of a single meme are connected together. However, we find that similar phrases may not belong to the same meme and thus the phrase graph needs to be further partitioned.

Traditional graph partitioning criteria (such as the normalized cut or the min cut) are not appropriate here as our aim is to discover all mutational variants of a single meme. With this in mind, our goal can be rephrased as finding clusters in the phrase graph such that each cluster has a single root node. The root node acts as the original phrase from which all other phrases in the cluster evolved through a series of mutations.

Thus more formally, given a weighted directed acyclic graph, our goal is to delete a set of edges with a minimum total weight, so that each of the resulting components is single-rooted. Given that the problem is NP-hard [20], we use the following method.

We partition the phrase graph by repeatedly removing edges from the graph until all outgoing edges for each node belong to the same cluster. This constraint on the outgoing edges follows intuitively from the concept that many phrases are derived from one original phrase, referred to as the *root* phrase. These derived phrases are normally shorter phrase segments that various news sources use. Once all outgoing edges for each node belong to the same cluster, the resulting phrase graph is naturally partitioned into individual meme clusters, where each connected component is considered a meme cluster.

We implement the algorithm by recursively building clusters, starting with a working set that includes all the root phrases, which are nodes in the graph with zero outdegree. At this step, each cluster contains only a single root phrase. Repeatedly, when a node is not in the working set, but all its outgoing neighbors are, we assign the node to a cluster as follows.

For each of the node's neighbors, we find the cluster that the neighbor has been assigned to, then for each cluster found we sum up the edge weights for all the neighbors in this cluster. The node is attached to the cluster with the largest sum and added to the working set; edges to other clusters are removed from the graph. When all the nodes are in the working set, the algorithm terminates (Figure 4).

The final output of our phrase graph partitioning algorithm is a set of clusters, referred to as the **cluster base** C. Clusters have an

easily followable acyclic structure that demonstrates how the root phrases branch into different child phrases.

2.4 Cluster Processing

The phrase graph partitioning creates a number of non-meme clusters such as movies, TV shows or song titles currently being promoted in the online media. To address non-memes and any remaining spam clusters, we next describe how the cluster processing step improves the clusters by filtering them by phrase mutations and the number of peaks. Peaks are defined as spikes in the number of phrase mutations within a time period.

Filtering by phrase mutations. This filtering strategy removes all clusters that include a single phrase mutation. Because many movie, TV show and song titles are frequently short and exhibit little to no variation, this strategy removes many non-memes as desired.

Filtering by peaks. As mentioned earlier, most news follow a predictable popularity cycle with at most two main peaks [39]. A cluster that has many peaks is most likely not a meme, so this step removes clusters with more than five peaks.

To identify peaks, our approach is to find points that are 1 standard deviation higher than the average frequency. The point with the highest frequency in each consecutive sequence of such points is marked as a peak. We use a sliding window of 9 days for these calculations.

Filtering by peaks was effective in both removing spam, since it was less likely to follow the news popularity cycle, as well as non-memes (e.g. "The Dark Knight Rises") that otherwise consistently produced large clusters due to active media promotion efforts.

2.5 Incremental Phrase Clustering

So far, we described a batch approach to clustering. Now, we describe how NIFTY extends this batch approach with incremental clustering.

Motivation. We want NIFTY to be able to update the meme clusters with new stories each day and also to process our entire dataset, spanning more than four years. We need NIFTY to be *fast* and *scalable* while maintaining *consistent* meme clusters over time.

To achieve these objectives, we developed an *incremental clustering* approach that quickly and consistently creates clusters over arbitrarily long periods of time. Our incremental clustering approach is based on the phrase graph of the batch approach, however, we extended the phrase graph creation and partitioning steps for incremental operation. We also introduced two new steps, cluster completion and removal.

Algorithm overview. While the batch clustering processes all the input documents together, the incremental clustering processes the documents in small batches, which we call mini-batches. The processing of each mini-batch of documents takes the phrase graph from the previous mini-batch as input and adds the phrases from the current mini-batch to the graph. The resulting phrase graph is saved as input for the next mini-batch of documents. We extended the phrase graph creation and partitioning steps, so that consistent clusters are maintained across mini-batches.

We also expanded the algorithm with cluster completion and removal steps, which make certain that old clusters do not impact cluster quality and are removed from the phrase graph when they stop evolving. These steps are necessary to maintain cluster quality over a large number of mini-batches and to keep the processing resources from growing over time.

A natural choice for the mini-batch size in NIFTY is one day and this is what we use here. The algorithm can be easily used for other mini-batch sizes, if needed. We provide the details of our algorithm next.

Phrase graph creation. Our incremental clustering approach creates daily phrase graphs by building upon the previous day's phrase graph. We attach each newly created phrase to the graph. An important detail here is that we only consider edges between phrases where at least one phrase is new. This constraint guarantees that new edges will not be added between previously existing phrases, which could disrupt the existing cluster structure. Edges for new phrases can be freely added to the graph, since they did not exist the previous day. Besides improving the cluster consistency, this constraint also drastically reduces the number of comparisons needed in comparison to the batch approach and thus speeds up the processing.

Phrase graph partitioning. During the phrase graph partitioning, we want to preserve existing clusters by preserving all edges that existed in the graph the day before. Our partitioning strategy remains the same - edges are removed until all outgoing edges for each node belong to the same cluster. For incremental clustering, we require that an edge is selected over other edges and kept in the graph if both its phrases already existed the day before. Only edges of newly added nodes can thus be removed from the graph. Otherwise, we proceed as before, computing edge weights for all outgoing edges and keeping only the edges from the cluster with the highest edge weight.

This approach guarantees that all edges from the previous day's phrase graph are preserved, which maintains cluster consistency. Furthermore, bypassing the edge weight computation for existing edges allows incremental clustering to further reduce the system run time.

Cluster completion and removal. If incremental clustering is run without ever removing old clusters, the cluster base grows over time, which reduces cluster quality and increases running time. Common but not overly popular phrase clusters (e.g. "I love you") accumulate similar phrases over time, artificially inflating the cluster popularity. To solve this problem, we introduce the concepts of cluster completion and removal, which allow the cluster base to retain only clusters that are alive and active.

NIFTY treats a cluster as *completed* if its average document frequency within the last three days is less than 20% of its frequency at its highest peak. After a cluster is completed, no new phrases are added to the cluster. However, the document frequencies of existing phrases in the cluster are still updated, so that the full lifecycle of the cluster is recorded. As an example, cluster completion prevents phrases such as "a step in the right direction," a commonly used political phrase, from getting mixed in with older and unrelated phrases such as "a move in the right direction," a phrase describing a Christian novel. By the time "a step in the right direction" appears, the cluster with "a move in the right direction" would have fallen past its peak popularity and therefore would be completed, preventing "a step in the right direction" from being incorrectly added to it.

A cluster is *removed* from the phrase graph, if its highest peak is more than seven days old. We consider such clusters final and remove them and their associated phrases and documents from the current cluster base and the phrase graph. Cluster removal helps to maintain a relatively constant and therefore scalable cluster base, which enables NIFTY to be run over datasets with a large time span. It also prevents a previously mentioned problem of inflated

cluster popularity from occurring. Clusters that remain after the removal step contain memes that are active and relevant to the present time. If newer phrases are similar to some phrases from a removed cluster, they are either being mentioned due to a new and different event, or are spoken under different contexts. These new phrases should therefore be considered a different meme cluster, which is what NIFTY achieves with cluster removal.

Our final addition of cluster completion and removal to the system allows us to achieve the goal of making NIFTY clustering fast, scalable, and consistent. Incremental clustering can be run over any period of time, which was practically impossible before with batch approaches. Detailed speed and quality experiments and comparisons between MEMETRACKER and NIFTY are given in Section 3.

2.6 Visualization

The last step in NIFTY is to rank the clusters by popularity so that the most popular clusters can be visualized (Figure 1). Each cluster c is assigned a time-based popularity score $S(c)$ based on the number of document mentions and cluster recentness using the following exponential decay formula:

$$S(c) = \sum_{p \in c} \sum_{t=t_p}^{t_c} \exp\left[-\left\lfloor \frac{t_c - t}{48} \right\rfloor\right] \cdot M_p(t)$$

p is a phrase in c, t_p is the time of the earliest mention of phrase p, t_c is the current time, and $M_p(t)$ is the number of document mentions of p at time period t. All time entities have one hour resolution with units being hours, so 48 in the formula corresponds to two days. This score ensures that more recent mentions are weighted more heavily than older mentions.

With a popularity score assigned to each cluster, the visualization step simply sorts the clusters by their popularity score to obtain the top clusters for each day.

3. NIFTY SYSTEM EVALUATION

In the following, we evaluate the NIFTY system performance. We discuss experimental results that establish the performance of NIFTY and evaluate our most important design decisions when constructing the NIFTY system. Then we present a series of experiments where we compare the system performance and resource consumption of NIFTY and MEMETRACKER. We establish that NIFTY runs faster than MEMETRACKER, while simultaneously producing meme clusters of higher quality.

The experimental results were obtained using a dataset that we describe in more detail in Section 4.

3.1 Evaluation of NIFTY

In the following we evaluate particular design decisions we made when building NIFTY. First we demonstrate that locality sensitive hashing speeds up the clustering step by reducing the number of comparisons, while still preserving the structure of the graph as created by using brute force with pairwise comparisons. We then evaluate different methods for phrase graph partitioning and find the best performing one.

Candidate pair creation. The Locality Sensitive Hashing (LSH), used in the phrase clustering step (Sec. 2.3), significantly speeds up the algorithm, but is an approximate method. The balance between the quality of approximation and speed is determined by the band size parameter k and the number of bands n. The band size k determines for one band the number of equal hash results that are required for two phrases to be placed in the same bucket. The number of bands n determines how often the process is repeated, once for each band. A smaller k therefore results in more comparisons (increasing running time) but also increases the likelihood of finding all similar edges. A larger n gives two phrases more opportunities to be placed in the same bucket, increasing the quality of approximation and increasing running time.

	baseline	size 1	**size 2**	size 3
#Compares	1.8×10^8	8.7×10^7	**2.6×10^7**	6.4×10^6
Run Time	34m38s	31m28s	**6m56s**	4m01s
Pre Precision	1.00	1.00	**1.00**	1.00
Pre Recall	1.00	0.99	**0.80**	0.57
Pre F1 score	1.00	0.99	**0.91**	0.72
Post Precision	1.00	0.99	**0.96**	0.89
Post Recall	1.00	0.99	**0.95**	0.83
Post F1 score	1.00	0.99	**0.95**	0.86

Table 1: Comparison of LSH band sizes.

While fixing the number of bands to 20, we experimented with different band sizes to find the optimal value. These experiments were done over one week of data with 38,000 phrases. Without LSH, these phrases require over 721 million pairwise comparisons. Using LSH, the number of comparisons is reduced to only 26 million.

Table 1 gives detailed results. The baseline performance is based on an index of shingles. We compare each pair of phrases that shares a shingle, which guarantees that all similar phrases are compared. The precision of a phrase graph is calculated as the fraction of edges in the graph that exist also in the baseline graph. The recall is the fraction of edges from the baseline graph found in the phrase graph. The F1 score is computed as the harmonic mean of precision and recall. "Pre" values are calculated before the partitioning step. "Post" values are calculated after the partitioning step has been performed.

The results in Table 1 show that setting the band size k to 2 achieves the optimal balance between speed and quality. Setting k to 1 achieves barely any time gain over the baseline method. On the other hand, k equal to 3 produces clusters of significantly lower quality (low edge recall) than k equal to 2, while time gain is limited.

It is interesting to observe that the final post scores improved significantly after the partitioning step ("Post" scores are generally higher than "Pre" scores). This improvement demonstrates the robustness of our phrase graph partitioning method — even though there is some noise in the graph creation step, the final clusters practically remain unchanged.

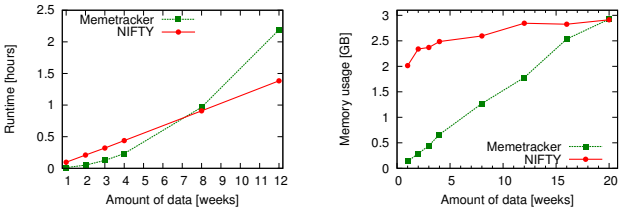
Phrase graph partitioning. Next, we investigate different edge selection methods for our phrase graph partitioning algorithm. Starting from the roots of the phrase graph, children repeatedly select a single outgoing edge or a set of edges to determine their parents, and thus the cluster they belong to. We evaluate the following methods for selecting these outgoing edges:

- **Baseline:** Randomly pick an outgoing edge for each node.
- **Method 1:** Pick the outgoing edge with the highest edge weight (break ties arbitrarily).
- **Method 2:** Pick the outgoing edges from the cluster with the most neighbors to the node.
- **Method 3:** Pick the outgoing edges from the cluster whose neighbors of the node have the highest total edge weight.

We compare the methods using two metrics: the fraction of edges in the pre-partitioned graph that connect nodes assigned to the same cluster, and the ratio of the total edge weight of these edges compared to the total edge weight in the pre-partitioned graph. The re-

	% edges kept	% edge weight kept
Baseline	13.41	70.96
Method 1	17.02	95.38
Method 2	23.62	80.60
Method 3	**21.03**	**95.48**

Table 2: Comparison of edge selection methods.



(a) Running time (b) Memory usage

Figure 5: MEMETRACKER and NIFTY resource usage.

sults are in Table 2. We observe that Method 3 performs best overall. While Method 2 is more successful in retaining edges within clusters, and Method 1 optimizes for the total edge weight, we find Method 3 to give the most balanced performance.

3.2 NIFTY vs. Memetracker

NIFTY builds on MEMETRACKER and we next compare the performance of the two systems. In particular we are interested in comparing resource usage and speed as well as clustering quality.

Resource usage. First in Figure 5 we compare the run time as well as memory usage of the incremental NIFTY algorithm with the batch MEMETRACKER algorithm. Incremental clustering in NIFTY takes on average 1 minute daily to cluster new phrases and documents. As expected, this daily time does not increase over longer time periods, so the total running time increases linearly with the amount of data. Also, because NIFTY archives completed clusters, the amount of memory it requires stabilizes at about 3GB.

On the other hand, MEMETRACKER time complexity and running time is quadratic with respect to the number of days of data that it is processing. Since MEMETRACKER's implementation is less complex than NIFTY's, MEMETRACKER is faster for small datasets. However, we can see that NIFTY is faster once we have at least 8 weeks' worth of data. MEMETRACKER's memory usage is linear with respect to the dataset size because MEMETRACKER must load the whole dataset into memory for clustering.

These results demonstrate that it would be impossible to run MEMETRACKER over our 6 billion document dataset. NIFTY's constant memory usage and linear scaling are the key advantages that allow us to run NIFTY on this large dataset, covering a period of 4 years.

Meme cluster quality. Next we compare the quality of MEMETRACKER and NIFTY clusters. Our first and most important observation is that NIFTY does not suffer from MEMETRACKER's "giant cluster" problem. When MEMETRACKER is run over datasets of non-trivial size, it tends to create a giant cluster that contains a large number of similar phrases that occur over long time periods ("a move in the right direction" vs. "a step in the right direction"). In the giant cluster, multiple phrases are chained together via long and intricate strings of spurious mutations.

For our comparison, we ran NIFTY and MEMETRACKER over the same 1 week input dataset of 102,000 phrases spanning the time

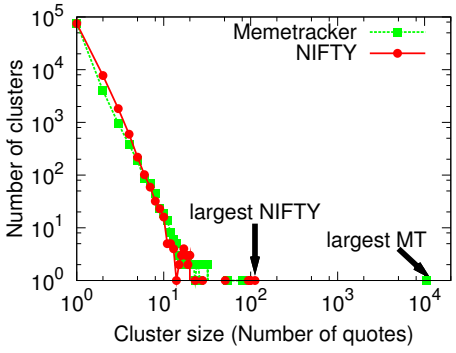


Figure 6: NIFTY vs. MEMETRACKER cluster size distribution. MEMETRACKER produces a giant cluster of 10,000 phrases.

period from Jan 1, 2012 to Jan 7, 2012. As shown in Figure 6, MEMETRACKER identified 5,000 nontrivial clusters, but the largest cluster contained more than 10,000 phrases. On the other hand, NIFTY identified 10,000 clusters (twice as many) and the largest NIFTY cluster contained only 112 phrases. MEMETRACKER fused together 10,000 different phrases into a single giant meme cluster, while NIFTY was able to assign those phrases to separate small meme clusters, which is what we want.

We also found that NIFTY and MEMETRACKER produce very similar small clusters except for those that MEMETRACKER folds into the giant cluster. Larger clusters differ more, and a manual inspection shows that MEMETRACKER clusters, when different, combine phrases that do not belong to the same meme. For example, "there is nothing we can do from here" is combined with "we don't care about data or figures, there's nothing we can do about pollution even it exceeds the limit". In conclusion, there are several clustering concerns with MEMETRACKER that we successfully address in NIFTY.

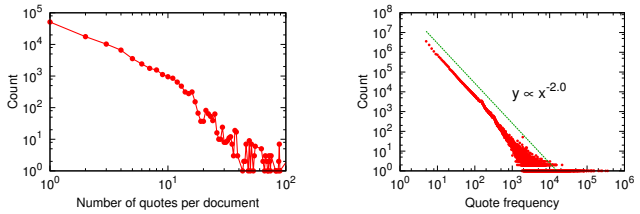
4. PROCESSING 6B DOCUMENTS

Having established the performance of NIFTY we are now ready to run the system over the full 4 year dataset of over 6 billion documents. Here we briefly describe our massive dataset characteristics and give some details of the NIFTY implementation and execution on the dataset.

Data description. Our dataset covers the online media activity since August 1, 2008. It includes posts from the mainstream publishers, blogs, forums and other media sites. At the time of this writing in mid November 2012, the dataset contains over 6.1 billion documents and 2.8 billion unique quoted phrases. Around 3.2 million new documents and 1.5 million unique phrases are added to the dataset daily. The total size of our dataset is 20TB.

We use Spinn3r [33] to obtain new documents. Spinn3r is a service that monitors over 20 million Internet sources, retrieves any new posts and makes them available via an API. The breadth of Spinn3r sources provides essentially complete coverage of online media.

Implementation. The NIFTY pipeline starts with a client that downloads new documents from Spinn3r. The client extracts quoted phrases, which are defined as any string in the document that is enclosed by quotation marks, the URL and an estimated publish time for all new documents. The rest of the pipeline reads the output from the Spinn3r client and implements the methods described in Section 2. The pipeline is initialized with two weeks of data,



(a) Quotes per document (b) Quote frequency

Figure 7: Basic properties of input data. (a) Number of quotes per document vs. number of such documents. (b) Quote frequency vs. number of such quotes.

processed by our batch clustering algorithm. Afterwards, our incremental clustering algorithm is used to add daily updates.

The entire pipeline is implemented in C++ and uses the open source SNAP software for text and graph manipulation [32]. As part of NIFTY, several novel algorithms were developed. We are in the process of integrating these algorithms in SNAP and making them publicly available as open source.

Execution. Although the main purpose of NIFTY is to process new documents as they become available, its ability to work incrementally allows us to run the pipeline over the entire dataset in our news archive. The processing of 20TB of data with 6.1 billion of documents collected over 4 years took less than five days on a single machine.

Figure 7 shows characteristics of our input data. We note that the distribution over the number of quotes per document (Fig. 7(a)) as well as the quote frequency both exhibit a heavy tailed distribution. While documents contain less than hundred quotes, quotes themselves are heavily popular on the Web and some quotes get mentioned hundreds of thousands of times.

The filtering step (Sec. 2.2) starts with 2.9 billion quoted phrases in 6.1 billion documents. The first filtering pass keeps 0.9 billion unique phrases in 0.5 billion documents. The second filtering pass ultimately selects for further processing 378 million phrases in 133 million documents, which means an average of 2.9 phrases per document. 33 million of those phrases are unique.

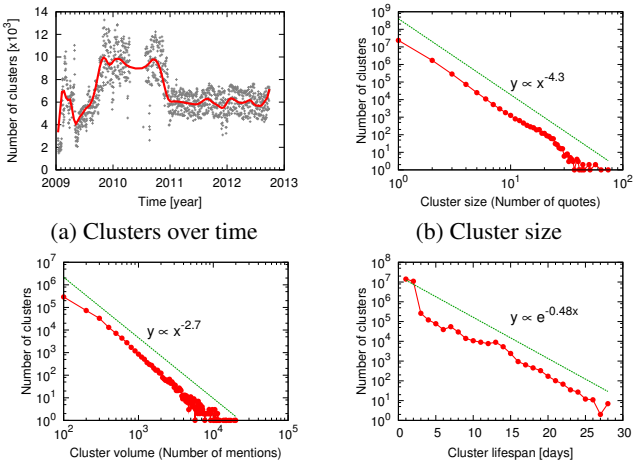
The filtering step takes, on average, 7 minutes and 6 GB of memory to filter a day's worth of data. To reduce execution time, we run 10 instances of filtering concurrently on a single multicore machine, which brings the total filtering time for the entire 4 year dataset to 17 hours.

The clustering step (Sec. 2.3) creates 9 million meme clusters from 33 million unique phrases. An average number of phrase mentions for a meme is 42. The clustering step takes four days at a rate of 8.5 million phrases clustered per day, running as a single thread.

5. ANALYSIS OF MEMES

By extracting memes from our massive 6 billion document 20TB dataset spanning 4 years, we also learn interesting facts about the characteristics and dynamics of online memes. We examine some of our findings here.

Properties of meme clusters. We observed several interesting trends illustrated in Figure 8. First, NIFTY outputs between 6 and 10 thousand clusters per day (Fig. 8(a)). While the number of unique extracted quoted phrases varies between 20 and 50 thousand per day, the number of identified meme clusters is about 5 times smaller, which means an average meme has between 3 to 5 mutational variants found on the Web.



(a) Clusters over time (b) Cluster size

(c) Cluster volume (d) Cluster lifespan

Figure 8: NIFTY meme clusters over a 4 year period.

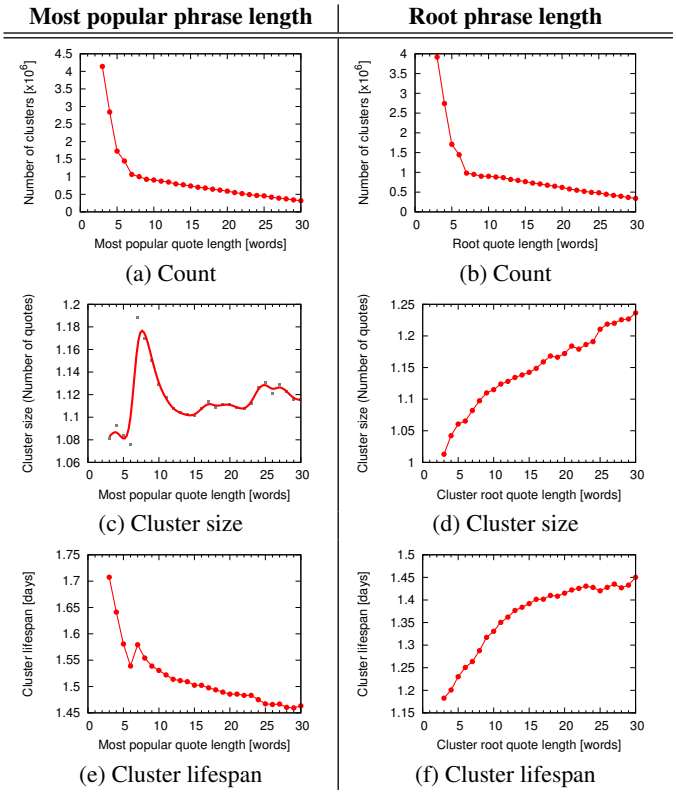
In terms of the meme cluster size measured in the number of phrases contained in the cluster, we observe a nice heavy tailed distribution with the largest clusters containing slightly fewer than 100 variants (Fig. 8(b)). Similarly, the cluster volume measured as the total number of mentions of all the phrases in the cluster follows a power-law like distribution (Fig. 8(c)). We also observe that the most popular memes on the Web are mentioned tens of thousands of times.

Last, we examine the distribution of meme lifetimes. Here we quantify the lifetime of a cluster simply as the time difference between the 5th and 95th percentile of all mentions of phrases in the cluster. Such a definition is more robust than taking the difference between the time of the first and the time of a last mention of any phrase in the cluster. In Figure 8(d) we note an interesting exponential decay in meme lifetime. While most memes live only for a day or two, long-lasting memes remain for about 1 month. This nicely agrees with previous works on human attention and patterns of temporal variation in online media [39].

Properties of phrases inside clusters. Next we examine how properties of meme clusters vary as a function of the phrases that are part of the same meme cluster. In particular, we characterize every meme cluster in two different ways: by its most mentioned phrase, referred to as the popular phrase, and also by its root phrase (i.e., the root node of the cluster). Figure 9 plots various characteristics of clusters based on the word length of the most popular phrase (left column) as well as the word length of the root phrase (right column).

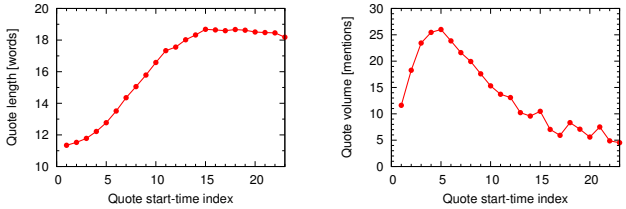
We observe that most popular phrases as well as root phrases are short in most meme clusters (Figures 9(a),(b)). There is surprisingly little difference between the two distributions. However, we observe an interesting distinction between the length of the root vs. most popular phrase when we compare the cluster sizes (i.e., the number of mutational variants in the cluster). Here we notice memes mutate the most when the most popular phrase is relatively short (Fig. 9(c)) and the root phrase is long (Fig. 9(d)). This is interesting as it suggests that memes that mutate a lot contain short catch phrases that appear in the context of a larger phrase.

We observe similar behavior when investigating meme lifespan. We observe that memes with shorter most popular phrases survive longer (Fig. 9(e)), while memes with short roots diminish sooner (Fig. 9(f)). Such behavior is consistent with our explanation above.



(a) Count (b) Count (c) Cluster size (d) Cluster size (e) Cluster lifespan (f) Cluster lifespan

Figure 9: Global characteristics of meme clusters and phrases.



(a) World length (b) Volume

Figure 10: Phrase word length and volume as a function of order of the appearance.

Memes with short catch phrases that are parts of longer narratives survive longer and are also more diverse [11].

Properties of phrases. We also compare how properties of phrases change as the memes evolve over time. Here we simply order the phrases belonging to the same meme cluster in order of their first appearance. Figure 10 shows the results, from which we make several interesting observations. We plot the average phrase word length as a function of the order in which the phrases inside the same cluster appear (Figure 10(a)). We observe that phrases that get mentioned first are generally much shorter than phrases that appear later in the meme lifetime. This phenomenon is consistent with the fact that blogs and social media sites react to trends and news quickly, and mention memes before the mainstream media does [20, 39]. Social media sites tend to mention a shorter and more impactful version of the meme. For example, during the 2012 U.S. presidential election campaign, a popular meme *"binders full of women"* emerged. This catchphrase version of the meme was mentioned much before the original long version of the meme which was: *"I went to a number of women's groups and said, can you help us find folks? And they brought us whole binders full of women."*

Meme property	Popular	Unpopular
Most popular phrase length	7.49	11.64
Root phrase length	11.46	11.94
#Phrase variants	2.84	1.10
Meme lifespan [days]	8.66	1.33
#Peaks	2.62	1.85

Table 3: Comparison of popular and unpopular meme clusters.

When focusing on the number of mentions a phrase receives as a function of the order of appearance (Figure 10(b)), we notice that a phrase variant that appears 5th tends to be the most popular. The popularity quickly drops off with phrases that appear later in the cluster lifetime receiving less and less attention (i.e., volume).

Comparison of popular and unpopular memes. Last, we also examined the differences between memes that get at least some popularity vs. memes that receive little or no attention. For the purpose of this experiment we call a meme popular if it was mentioned at least 350 times, and we call all other memes unpopular.

Table 3 shows that the average length of the most frequently mentioned phrase in a cluster is significantly shorter in popular clusters in comparison to unpopular ones. On the other hand, the root phrase length does not change excessively. This suggests that popular phrases are significantly more likely to be shortened into more memorable sound bites [11]. The fact that popular clusters contain significantly more phrase variants on average supports this hypothesis. We also noticed that popular memes live significantly longer than unpopular ones on average. This also explains why popular memes exhibit more peaks in their volume, as the temporal dynamics of online media tends to follow a strong daily cycle [39].

6. CONCLUSION

In this paper we have developed NIFTY, a system for tracking short, distinctive textual phrases that travel relatively intact through online text. We presented a highly scalable algorithm for meme-tracking which identifies and clusters mutational variants of textual phrases. Our system scales to a collection of 6 billion articles, which makes the present study one of the largest analyses of online news in terms of data scale. Moreover, we provided a live deployment of the NIFTY system which allows users to explore the dynamics of information dissemination and mutation in mainstream and social media.

Our approach to meme-tracking opens a range of opportunities for future work. For example, how can we understand the dynamics of the mutation of memes over both time and space? Given that our data essentially covers the entire online media landscape over the last four years, it may be possible to more generally identify and model the way in which the essential "core" of a widespread meme emerges and enters popular discourse. Similarly, the long time period that our dataset encompasses may ease studies on the evolution of online media commentary and practices, as well as on the kind of collective behavior that leads directly to the ways in which all of us experience news and its consequences.

Acknowledgements. We thank Hyun Goo Kang, Wonhong Lee, and Hai Wei for their help with the phrase clustering decision tree, Andrej Krevl for his help with the website and Spinn3r for providing the document dataset. This research has been supported in part by NSF IIS-1016909, CNS-1010921, CAREER IIS-1149837, IIS-1159679, ARO MURI, DARPA SMISC, DARPA XDATA, ARL AHPCRC, Brown Institute for Media Innovation, Okawa Foundation, Docomo, Boeing, Allyes, Volkswagen, Intel, Alfred P. Sloan Fellowship and the Microsoft Faculty Fellowship.

7. REFERENCES

[1] L. Adamic and N. Glance. The political blogosphere and the 2004 U.S. election: Divided they blog. In Proc. *LinkKDD '05*, pages 36–43, 2005.

[2] L. Adamic, T. Lento, and A. Fiore. How you met me. In Proc. *ICWSM '12*, pages 371–374, 2012.

[3] E. Adar, L. Zhang, L. A. Adamic, and R. M. Lukose. Implicit structure and the dynamics of blogspace. In *Workshop on the Weblogging Ecosystem*, 2004.

[4] A. Ahmed, Q. Ho, J. Eisenstein, E. Xing, A. Smola, and C. Teo. Unified analysis of streaming news. In Proc. *WWW '11*, pages 267–276, 2011.

[5] J. Allan (editor). *Topic Detection and Tracking: Event-based Information Organization*. Kluwer, 2002.

[6] M. Atkinson and E. Van der Goot. Near real time information mining in multilingual news. In Proc. *WWW '09*, pages 1153–1154, 2009.

[7] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. In Proc. *NIPS*, pages 601–608, 2001.

[8] K. D. Bollacker, S. Lawrence, and C. L. Giles. A system for automatic personalized tracking of scientific literature on the web. In Proc. *DL '99*, pages 105–113, 1999.

[9] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In Proc. *STOC '98*, pages 327–336, 1998.

[10] J. Cook, A. Das Sarma, A. Fabrikant, and A. Tomkins. Your two weeks of fame and your grandmother's. In Proc. *WWW '12*, pages 919–928, 2012.

[11] C. Danescu-Niculescu-Mizil, J. Cheng, J. Kleinberg, and L. Lee. You had me at hello: How phrasing affects memorability. In Proc. *ACL '12*, pages 892–901, 2012.

[12] A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In Proc. *WWW '07*, pages 271–280, 2007.

[13] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. *ACM Trans. Web*, 1(2):7, 2007.

[14] E. Gabrilovich, S. Dumais, and E. Horvitz. Newsjunkie: Providing personalized newsfeeds via analysis of information novelty. In Proc. *WWW '04*, pages 482–490, 2004.

[15] D. Gruhl, D. Liben-Nowell, R. V. Guha, and A. Tomkins. Information diffusion through blogspace. In Proc. *WWW '04*, 2004.

[16] S. Havre, B. Hetzler, and L. Nowell. ThemeRiver: Visualizing theme changes over time. In Proc. *INFOVIS '00*, 2000.

[17] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proc. *STOC '98*, pages 604–613, 1998.

[18] J. Kleinberg. Bursty and hierarchical structure in streams. In Proc. *KDD '02*, pages 91–101, 2002.

[19] M. Krstajic, F. Mansmann, A. Stoffel, M. Atkinson, and D. Keim. Processing online news streams for large-scale semantic analysis. In Proc. *Data Engineering Workshop (ICDEW)*, pages 215–220, 2010.

[20] J. Leskovec, L. Backstrom, and J. Kleinberg. Meme-tracking and the dynamics of the news cycle. In Proc. *KDD '09*, pages 497–506, 2009.

[21] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst. Cascading behavior in large blog graphs. In Proc. *SDM '07*, 2007.

[22] L. Lloyd, D. Kechagias, and S. Skiena. Lydia: A system for large-scale news analysis. In *String Processing and Information Retrieval*, pages 161–166, Springer, 2005.

[23] U. Manber et al. Finding similar files in a large file system. In Proc. *USENIX '94*, pages 1–10, 1994.

[24] C. Marlow, M. Naaman, D. Boyd, and M. Davis. Ht06, tagging paper, taxonomy, flickr, academic article, to read. In Proc. *HYPERTEXT '06*, pages 31–40, 2006.

[25] X. Phan, L. Nguyen, and S. Horiguchi. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In Proc. *WWW '08*, pages 91–100, 2008.

[26] A. Rajaraman and J. Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[27] J. Ratkiewicz, M. Conover, M. Meiss, B. Gonçalves, A. Flammini, and F. Menczer. Detecting and tracking political abuse in social media. In Proc. *ICWSM '11*, 2011.

[28] T. Sakaki, M. Okazaki, and Y. Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In Proc. *WWW '10*, pages 851–860, 2010.

[29] D. Shahaf and C. Guestrin. Connecting the dots between news articles. In Proc. *KDD '10*, pages 623–632, 2010.

[30] D. Shahaf, C. Guestrin, and E. Horvitz. Trains of thought: Generating information maps. In Proc. *WWW '12*, pages 899–908, 2012.

[31] M. Simmons, L. Adamic, and E. Adar. Memes online: Extracted, subtracted, injected, and recollected. In Proc. *ICWSM '11*, 2011.

[32] SNAP Software Package. `http://snap.stanford.edu`. 2012.

[33] Spinn3r API. `http://www.spinn3r.com`. 2008.

[34] M. L. Stein, S. Paterno, and R. C. Burnett. *Newswriter's Handbook: An Introduction to Journalism*. Blackwell, second edition, 2006.

[35] 1000 Most Common English Words. `http://www.perlmonks.org/bare/?node_id=310060`. 2003.

[36] C. Wang, D. M. Blei, and D. Heckerman. Continuous time dynamic topic models. In Proc. *UAI '08*, pages 579–586, 2008.

[37] X. Wang, A. McCallum, and X. Wei. Topical n-grams: Phrase and topic discovery, with an application to information retrieval. In Proc. *ICDM '07*, pages 697–702, 2007.

[38] L. Xie, A. Natsev, J. Kender, M. Hill, and J. Smith. Visual memes in social media: tracking real-world news in youtube videos. In Proc. *MULTIMEDIA '11*, pages 53–62, 2011.

[39] J. Yang and J. Leskovec. Patterns of temporal variation in online media. In Proc. *WSDM '11*, pages 177–186, 2011.

[40] L. Yao, D. M. Mimno, and A. McCallum. Efficient methods for topic model inference on streaming document collections. In Proc. *KDD '09*, pages 937–946, 2009.