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

Caroline Suen[*], Sandy Huang[*], Chantat Eksombatchai[*], Rok Sosic, 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 employs a novel highly-scalable incremental meme-clustering approach to efficiently extract and identify 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 constantly capturing the pulse of humanity: what we are thinking, what we are doing, and what we know.

Since the early days of the Web, its information content has been taking on increasingly dynamic forms, to the extent that the real-time aspects of information have become one of the most pressing concerns in the processing and tracking of Web content. Information on the Web changes rapidly over time both because of the rate of production and also because of 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 over time, as well as how the information mutates as it is being transmitted over underlying networks. A better understanding of how the information spreads on the Web has many practical applications in a wide range of fields, such as social sciences, news media, marketing, and politics.

The goal of this paper is to develop a system that will be able to *track information* as it spreads across *billions of documents* on the Web and over the time periods spanning *many years.* Studying the spread of online information has been an active research area [1, 3, 15, 21, 38], but reliably tracking the flow of content has been extremely challenging [3, 21]. While there has been work on tracking topics [5, 7, 35, 39], tags [13, 24], and keywords [8, 36] over time and in restricted settings, difficulties of scale 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 textual units. 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, (again, except in rare cases) terms [16, 18] or topic clusters [7, 35, 39] (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 persists in that information on the Web constantly evolves, changes and essentially mutates as it spreads. Thus it is important to have a robust method to discover and track different mutational variants of the same piece of information.

Additional challenges are presented by the sheer volume and time span of Web data. We require a system that can process tens of millions of documents per day and billions of documents over

[*]Authors contributed equally to this work.

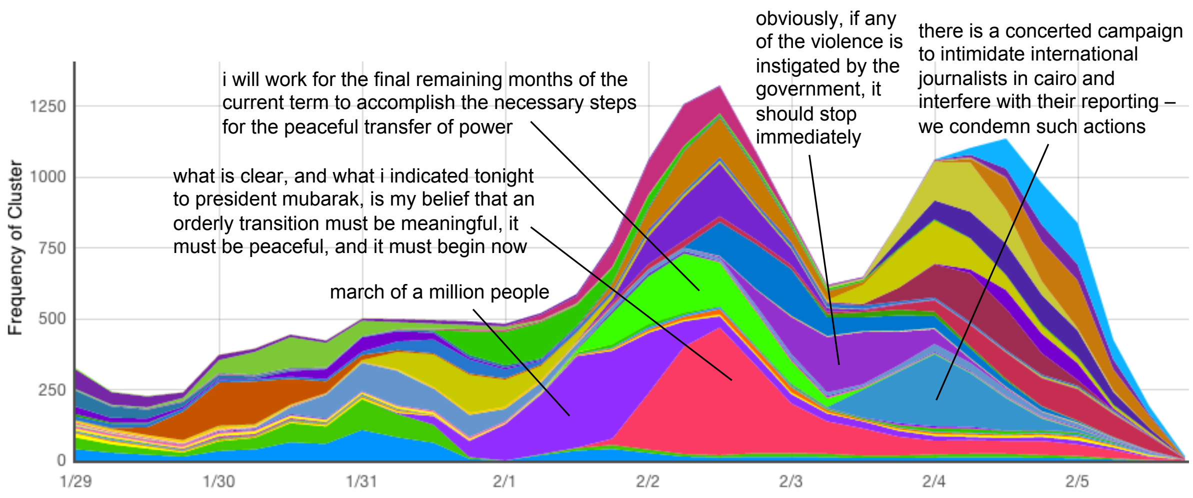


Figure 1: Visualization of most popular memes in the week of January 29, 2011. The top memes are labeled, and are all about the Egyptian revolution, which lasted throughout this week. Each band represents a meme and the height of the band corresponds to the number of mentions per hour.

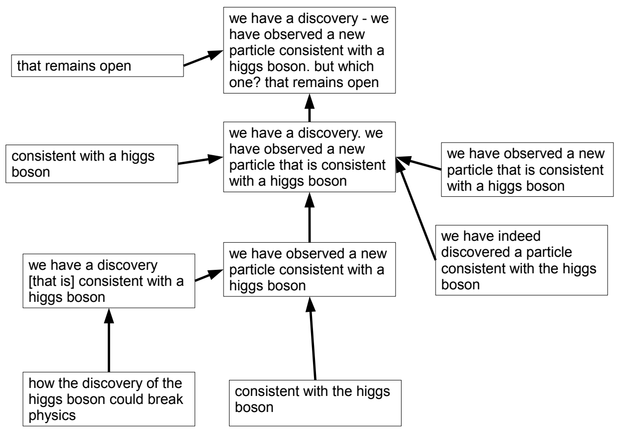

we have a discovery - we have observed a new particle consistent with a higgs boson. but which one? that remains open
that remains open
we have a discovery. we have observed a new particle that is consistent with a higgs boson
consistent with a higgs boson
we have observed a new particle that is consistent with a higgs boson
we have indeed discovered a particle consistent with the higgs boson
we have a discovery [that is] consistent with a higgs boson
we have observed a new particle consistent with a higgs boson
how the discovery of the higgs boson could break physics
consistent with the higgs boson

Figure 2: Mutations of a meme about the discovery of the Higgs boson particle.

several years. Traditional methods for tracking information flow have quadratic time complexity, 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 time periods spanning several years, a challenge is that our method must not degrade in performance over time.

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. It uses a novel highly-scalable incremental meme-clustering approach for extracting and identifying mutational variants of short textual phrases. Example output of our system is displayed in Figure 1.

NIFTY builds on our previous MEMETRACKER [20] approach in the sense that it operationalizes the notion of a meme through extraction of short quoted phrases. Quoted phrases and sentences — that is, quotations attributed to individuals — operate at a level of granularity between that of individual articles and broad topics.

Once the quote phrases are extracted, we need to cluster all the different variants of the same meme, especially given that we will be working with billions of short textual fragments. Clustering textual fragments is interesting because such fragments contain very little information and thus traditional bag-of-words representations do not seem appropriate. Moreover, we are not interested in clustering together all the phrases on the same topic, but rather we would like to find all the phrases that evolved from the same original phrase.

Our approach to this problem [20] applies ideas from biological sequence analysis, where 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. This procedure gives us 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 attempt to partition this graph into clusters, such that each cluster consists of a directed acyclic graph (DAG) with a single root node. The idea is to find a set of phrases that have all evolved from the same ancestor phrase. Such a set of phrases represents one meme with the root node being the original phrase. For example, Figure 2 shows a graph of the mutational variants of a meme related to the discovery of the Higgs boson particle.

The advantage of this approach is that it automatically produces the lineage or ancestry information (as illustrated in Figure 2). However, the drawback is that it is not very robust in its straightforward form, especially when one works with phrases appearing over longer time periods (e.g., more than a year). In such cases, even when working with long phrases that have occurred very frequently, the edit-distance overlaps among the phrases will cause them to chain together and form a "giant component" containing a significant fraction of all phrases. As this obscures the structure we are trying to identify, we propose an alternate approach to tackling the phrase clustering problem as follows.

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 old meme clusters and their corresponding phrases. This means that old phrases get deleted from the phrase 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 a complete online news coverage during that time. NIFTY extracted 2.9 billion unique textual phrases and identified more than 9 million 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. The incremental meme-clustering has linear runtime while maintaining cluster consistency and quality provided by MEMETRACKER. Since the input data is processed incrementally, its processing time depends only on the constant size of new data, 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 novel data visualization and data exploration techniques that allow users to explore the ongoing dynamics of online news. For example, Figure 1 shows a screen shot of our interactive data visualization for a one week period in January 2011. Through such a visual interface users of NIFTY can visualize and explore the dynamics of online news in near real-time.

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 50 fold to 6 billion documents, while requiring about 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://bit.ly/nifty2012`.

Further related work. Taken more broadly our work here contributes to the growing literature on tracking and studying the 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 have been in 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 needing very modest hardware resources for processing it.

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 [38, 10, 28] as well as to reason about the dissemination and mutation of online information [2, 27, 31, 37]. 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.

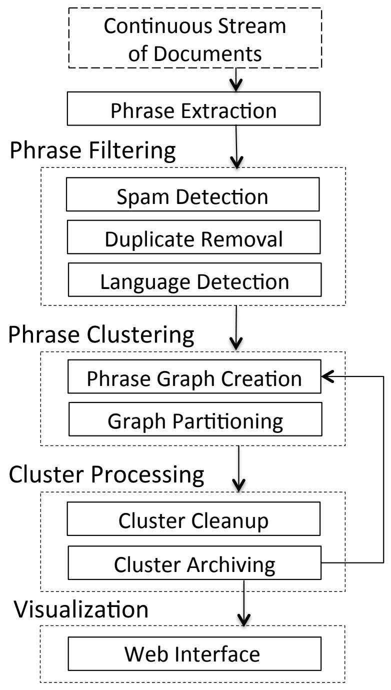


Figure 3: Overview of the NIFTY pipeline.

2. PROPOSED METHOD

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. The second step of the processing pipeline is *phrase clustering*, in which the main task is to group phrases (i.e., find mutational variants) that belong to the same meme. The key challenge here is that memes experience significant mutations, and so usual text based distance measures and standard clustering approaches do not work well [20]. The last two steps of the processing pipeline are *post-processing* and *visualization*. The post-processing step performs final quality checks of the output, removes old clusters, archives stabilized clusters, and provides data updates for incremental clustering. The visualization step uses the 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 will describe the pipeline as if the system operates in a batch setting. Later we discuss how to extend NIFTY to an incremental, stream based model.

2.1 Phrase extraction

The input 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, the document's publish date, and a list of phrases found in the document.

NIFTY operationalizes the notion of a meme through 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 subsequent 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 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. The first pass iterates through documents in D, removes irrelevant documents, and builds a set of phrases, found in the remaining documents. The filter first removes duplicates and documents whose URLs are on a manually maintained, short blacklist. Then it applies the following criteria:

- **Phrase length**: A phrase must contain between 3 and 30 words inclusive to be considered relevant. Short phrases do not provide any useful information, while long phrases tend to not be real phrases. We experimented with different thresholds and found that our current numbers properly balance between eliminating redundant data and preserving valid phrases.
- **English 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. In the second pass, we iterate through all the phrases found in the first pass, and remove phrases and the corresponding documents based on the following criteria:

- **Infrequent phrases**: We discard all phrases that appear in fewer than 5 distinct documents.
- **Advanced language filtering**: With fewer phrases, we can apply a more precise language filter. For every phrase p we compute the fraction of its words that appear in the top thousand English words. After experimenting with different thresholds, we found that a threshold of 75% was most effective. Phrases that do not satisfy this threshold are discarded.
- **URL to domain name ratio**: 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. This rule proved to be our best strategy to remove spam, as most spammers publish spam only to a select few domains.

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.

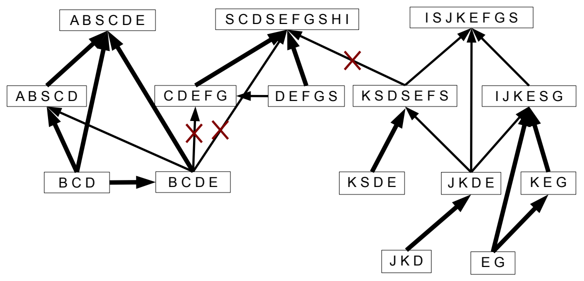


Figure 4: Example phrase graph, in which letters represent words. The letter 'S' indicates a stop word. Directed edges indicate the source phrase is approximately included in the destination phrase. For this example, the edge weight is inversely proportional to the edit distance between the two phrases,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

2.3 Phrase Clustering

As memes spread through the Web, they change and mutate. For example, Figure 2 displays the mutational variants of a meme concerning the discovery of the Higgs boson particle. Thus, our next goal is to partition the phrase base 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.

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. Each disconnected component then corresponds to a meme where all the nodes (phrases) in the component are its mutational variants.

Phrase Graph Creation. The purpose of the phrase graph creation step is to construct a directed weighted acyclic graph where phrases are nodes and similar phrases are connected by directed edges which point from shorter phrases to longer ones (Figure 4). The phrase graph captures the intuition that phrases mutate as they spread over the Web and the goal of this step is to connect every phrase to a set of its potential "parents", i.e., phrases from which it could have mutated.

Phrase distance. Our goal is to create edges between phrases that could have evolved from one another. We apply ideas from biological sequence analysis, where 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.

We determine the presence of an edge between a pair of phrases based on their textual distance; a shorter distance suggests more similar phrases. We apply what we call *substring edit distance*, which expands commonly used Levenshtein distance to substrings. Given two strings, we define the substring edit distance as 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 to determine the Levenshtein distance. For distance computations, we remove stop words from the phrases and use stemmed words.

```
$len \leftarrow min(|p_1|, |p_2|)$
$s_1 \leftarrow StripStopWords(p_1)$
$s_2 \leftarrow StripStopWords(p_2)$
$stoplen \leftarrow min(|s_1|, |s_2|)$
$dist \leftarrow Distance(s_1, s_2)$
$edge \leftarrow False$
if $dist = 0$ and $stoplen \geq 2$ then
    $edge \leftarrow True$
else if $len = 4$ and $stoplen = 4$ and $dist \leq 1$ then
    $edge \leftarrow True$
else if $len = 5$ and $stoplen > 4$ and $dist \leq 1$ then
    $edge \leftarrow True$
else if $len = 6$ and $stoplen \geq 5$ and $dist \leq 1$ then
    $edge \leftarrow True$
else if $len > 6$ and $stoplen > 3$ and $dist \leq 2$ then
    $edge \leftarrow True$
```

Figure 5: Decision tree pseudo code for connecting phrases p_1 and p_2 with an edge in a phrase graph.

Edge Creation. Now given a pair of phrases and their Levenshtein distance we need to determine the presence of an edge. We hand labeled 1,000 pairs of phrases that were mutational variants of each other and also 1,000 phrases that were textually similar but were not mutational variants. We then trained a decision tree to distinguish between these two classes and the resulting decision tree is shown in Figure 5. An edge is created between two phrases in the graph, if the decision tree returns $True$.

Speeding up phrase graph creation. A naive approach to building edges between phrases is to do a pair-wise distance comparison of all phrases in the phrase base. Doing so involves $O(n^2)$ comparisons, which becomes prohibitively slow for the scale of our problem.

In order to drastically reduce the required number of comparisons necessary and make the algorithm practically applicable for our dataset, NIFTY uses locality-sensitive hashing [17] (LSH) based on shingling [23] and min-hashing [9] (see [26] for an overview). Since there are no known good locality sensitive hash functions for substring edit distance, we decided to use a Jaccard-based similarity for our locality-sensitive hash function. Our experimental results in Section 3.2 confirm that LSH works well for meme clustering.

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

Phrase graph partitioning. After the phrase graph creation process one hopes that all mutational variants of a single meme are connected together and thus the connected components of the graph correspond to memes. 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 sink node. The sink node acts as the "mother 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 of minimum total weight, so that each of the resulting components is single-rooted. Given that the problem is NP-hard [20], we proceed as follows.

First, we assign weight to each edge. We base our *edge weight* calculation on findings by Yang [38] that most news follow a predictable popularity cycle with two main peaks, one from traditional news reporting and one from blog posts. NIFTY calculates weight for an edge from node p_s to p_d 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 phrase 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.

Second, we then 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.

Clusters are built recursively, starting by adding all the root phrases to the working set. 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 from 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 a root phrase branches into different child phrases (Figure 2).

2.4 Cluster Processing

After the cluster base is created during the phrase graph partitioning, it contains 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 in the post-clustering step, we next describe how NIFTY improves the clusters by filtering them by phrase mutations and peaks.

Filtering by phrase mutations. This filtering strategy removes all clusters that include only one phrase mutation, if the phrase contains four or fewer words. 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. We chose conservatively low thresholds both in terms of phrase length and the number of mutations to prevent valid phrases from being accidentally thrown away.

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

A good peak detection algorithm is important for this filtering step. We investigated a number of peak detection strategies, such as marking as peaks all points in time where the number of documents citing a phrase in the cluster was more than 1.5 standard deviations higher than the average frequency. The most successful approach has been 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, our clustering method has used a batch approach to clustering. We describe how we extended this batch clustering approach with incremental clustering next.

Motivation. MEMETRACKER methods were developed for datasets with a fixed time period. The original study was done over a static dataset covering three months of news. In contrast, 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 across each day. Our initial approach employed a sliding window, using the data from the previous two weeks to calculate meme clusters for each day. This approach did not yield consistent meme clusters, which were liable to change radically day by day. Furthermore, even with optimizations, this potential solution would be prohibitively slow for large datasets.

Ultimately, we decided to develop an *incremental clustering* approach that quickly and consistently creates clusters over arbitrarily long periods of time. In developing incremental clustering, we extended the phrase graph creation and partitioning steps. We also introduced two new steps, cluster archival and removal.

Phrase graph creation. Our incremental clustering approach creates daily phrase graphs by building upon the previous day's phrase graph. It uses an improved phrase similarity test. Instead of comparing all the pairs of phrases with a signature band in common, we compare only the pairs where at least one of the phrases is a newly introduced phrase. This additional comparison 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 and thus the system run time.

Phrase graph partitioning. During the phrase graph partitioning, we want to preserve existing clusters by preserving any edges that existed in the graph the day before. Our overarching partitioning strategy remains the same - edges are removed until all outgoing edges for each node belong to the same cluster. For incremental clustering, we impose an additional constraint that an edge is automatically selected if both its phrases already existed the day before. Since this constraint is applied before the edge weight calculation, it bypasses the edge weight computation altogether when an edge is selected. 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 any edge that existed in the phrase graph the previous day is selected and preserved in the next day's phrase graph, which maintains cluster consistency. Furthermore, bypassing the edge weight computation allows incremental clustering to reduce the system run time.

Cluster archival and removal. To maintain cluster quality and to improve system scalability of incremental clustering, we introduced the concepts of cluster archival and removal. When running incremental clustering without ever removing 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, NIFTY archives and eventually removes clusters from the cluster base.

A cluster is *archived* if its average document frequency within the last three days is less than 20% of its frequency at its highest peak. When a cluster is archived, 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. This archival strategy prevents phrases such as "a step in the right direction," a political phrase, from getting mixed in with slightly 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 been already archived and the two phrases would not be incorrectly clustered together.

A cluster is *removed*, if its highest peak is more than seven days old. We consider such clusters complete 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 removal contain memes that are recent and relevant to the present time. After a period of time, newer phrases that might have been assigned, without cluster removal, to an older cluster 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 archival and removal to the system allowed 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 for large time frames. 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, Figure 6). 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.

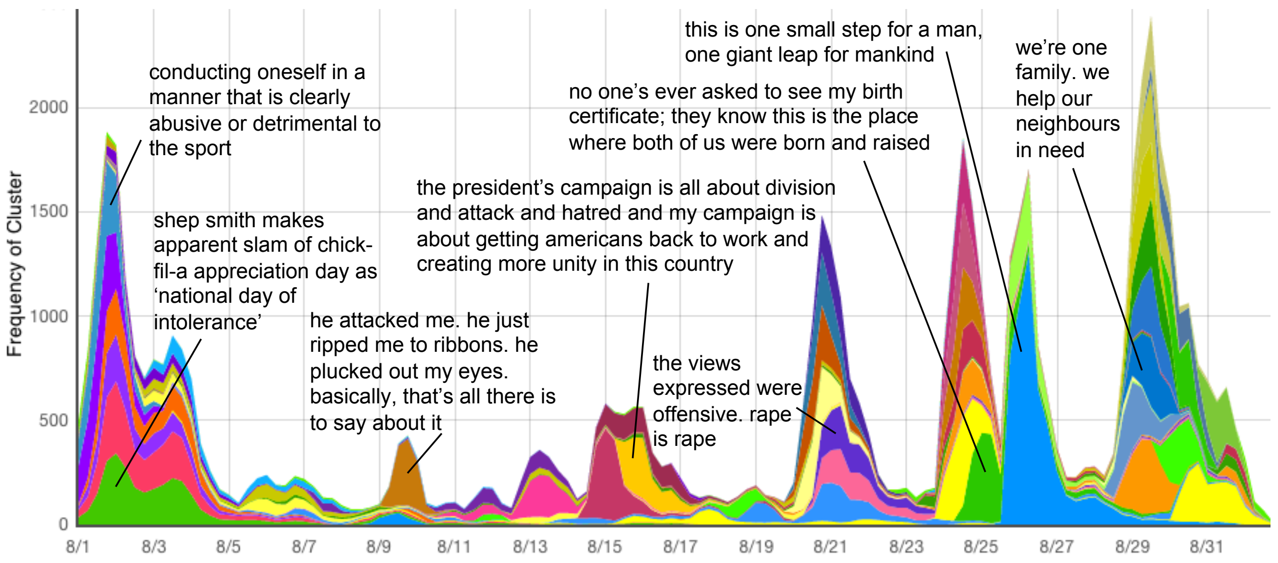


Figure 6: Example of monthly visualization, for August 2012. Notable memes are labeled. Several top memes in the visualization refer to the same event, for example the presidential campaign.

3. NIFTY SYSTEM EVALUATION

This section evaluates the NIFTY system performance and gives further details on the implementation. 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 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. Last, we give a description of the NIFTY system implementation.

3.1 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 quote 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.

3.2 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, which is created by using brute force with pairwise comparisons. We then evaluate different phrase graph partitioning algorithms and find the best performing one.

Locality Sensitive Hashing. 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. Smaller band sizes result in more comparisons (increasing running time) but also in an increased likelihood of finding all similar edges.

	baseline	size 1	**size 2**	size 3
#Compares	1.8×10^8	8.7×10^7	$\mathbf{2.6 \times 10^7}$	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 across 1 week (38,000 phrases, 721 million naive comparisons, 20 bands).

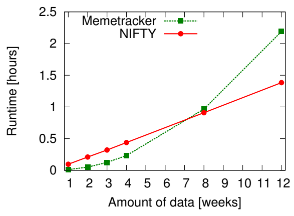
We experimented with different band sizes to find optimal parameter values. Table 1 gives the results. The baseline performance is provided by an index of shingles, where we compare each pair of phrases that shares a shingle. This guarantees that all similar phrases are compared. The precision of a phrase 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 20 bands of size 2 achieve the optimal balance between speed and quality. Band size 1 produces nearly perfect phrase graphs (high values of "Pre"), but with barely any time gain over the baseline method. On the other hand, clusters that result from band size 3 are of significantly lower quality (low edge recall) than clusters of band size 2, while providing only a relatively small time improvement.

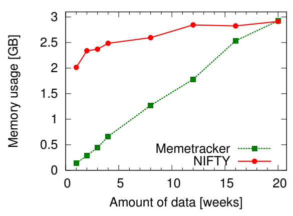
It is interesting to observe that the final post scores improved significantly after the partitioning step ("Post" scores are generally higher than "Pre" scores). This is encouraging as it demonstrates the robustness of our phrase graph partitioning method — even though there is some noise in the graph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

	% 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 7: MEMETRACKER and NIFTY resource usage.

Phrase graph partitioning. Next, we investigate different methods for our phrase graph partitioning problem. Starting from the roots of the phrase graph, children repeatedly select a single outgoing edge or set of edges to determine their parent, 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 and the fraction of the total edge weight kept in the resulting clusters. The results 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.

3.3 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 7 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.

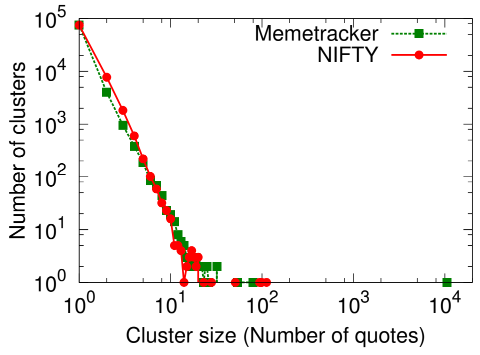


Figure 8: NIFTY vs. MEMETRACKER cluster size distribution. MEMETRACKER produces a giant cluster of 10,000 phrases.

Phrase 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 phrases. In the giant cluster, multiple phrases are chained together via long and intricate strings of spurious mutations.

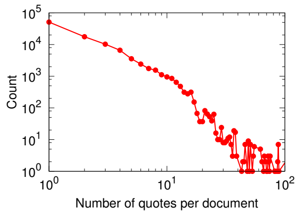
For our comparison, we ran NIFTY and MEMETRACKER over the same 1 week input dataset of 102,000 phrases spanning the time period from Jan 1, 2012 to Jan 7, 2012. As shown in Figure 8, MEMETRACKER identified 5,000 nontrivial clusters, but the largest cluster contained more than 10,000 phrases. On the other hand, NIFTY identified 10,000 clusters (twice as many) and the largest NIFTY cluster contained only 112 phrases. MEMETRACKER clumped together 10,000 different phrases into a single meme, while NIFTY was able to obtain nice, clean and small clusters.

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.

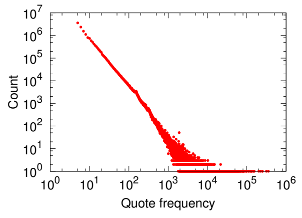
3.4 System Description

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. In the following, we briefly describe dataset characteristics and give some details of the NIFTY implementation and execution on our massive dataset.

Implementation. The NIFTY pipeline starts with a client that downloads new documents hourly from Spinn3r and translates them into a format that is suitable for NIFTY. During this translation, the document URL, its publish date, and the text content are extracted and stored as one item in a text file. The item also contains a list of quote phrases, which are defined as any string in the document that is enclosed by quotation marks. After all the documents from one



(a) Quotes per document



(b) Quote frequency

Figure 9: Basic properties of input data. (a) Number of quotes per document vs. number of such documents. (b) Quote frequency vs. number of such quotes.

download are translated, the output file is compressed and archived. 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, 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 package 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 do the 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. As a side note, the data in the archive is compressed and occupies 2.8TB of disk space.

Figure 9 shows characteristics of our input. data. We note that the distribution over the number of quotes per document (Fig. 9(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 time.

The filtering step (Sec. 2.2) starts with 2.9 billion quote 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.

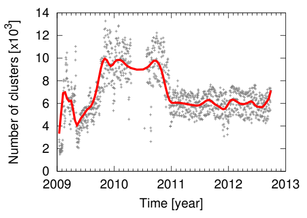
The filtering step takes, on average, 7 minutes and 6 GB of memory to filter a day's worth of data. To reduce execution time, we run 10 instances of filtering concurrently, which brings the total filtering time for the entire dataset to 17 hours.

The clustering step (Sec. 2.3) creates 9 million meme clusters from 33 million unique phrases. Average meme size is 42 non-unique phrases. The clustering step takes four days at a rate of 8.5 million phrases clustered per day, running as a single thread.

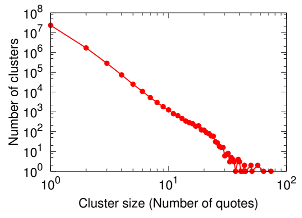
4. ANALYSIS OF MEMES

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

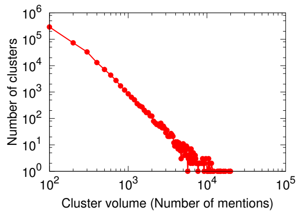
Properties of meme clusters. We observed several interesting trends illustrated in Figure 10. First, NIFTY outputs between 6 and 10 thousand clusters per day (Fig. 10(a)). While the number



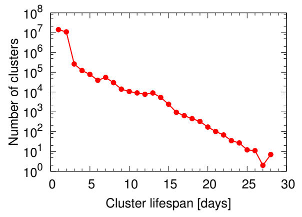
(a) Clusters over time



(b) Cluster size



(c) Cluster volume



(d) Cluster lifespan

Figure 10: NIFTY meme clusters over a 4 year period.

of unique extracted quote 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.

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 just a bit fewer than 100 meme variants (Fig. 10(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. 10(c)). We also observe the most popular memes on the Web get mentioned tens of thousands of times.

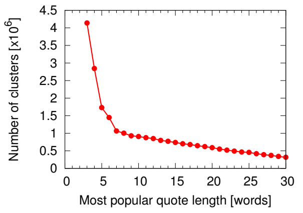
Last, we also examine the distribution of meme lifetimes. Here we quantify the lifetime of a cluster simply as the time difference between 5th and 95th percentile of all mentions of phrases in the cluster. Such a definition is more robust than taking the difference between the time of the first and the time of a last mention of any phrase in the cluster. In Figure 10(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 [38].

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 sink node of the cluster). Figure 11 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).

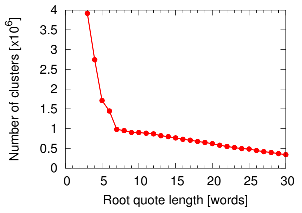
We observe that most popular phrases as well as root phrases are short in most meme clusters (Figures 11(a),(b)). There is surprisingly little difference between the two distributions. However, we observe an interesting distinction between the length of the root vs. most popular phrase when we compare 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. 11(c)) and the root phrase is long (Fig. 11(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.

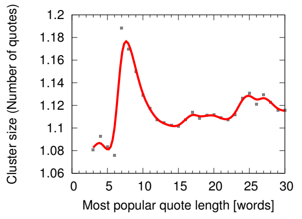
Most popular phrase length | **Root phrase length**



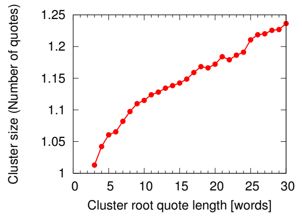
(a) Count



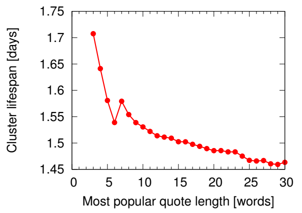
(b) Count



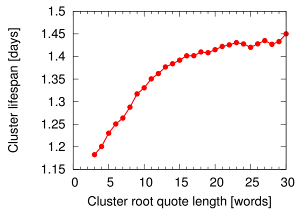
(c) Cluster size



(d) Cluster size



(e) Cluster lifespan



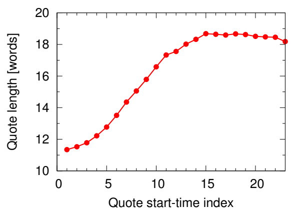
(f) Cluster lifespan

Figure 11: Global characteristics of meme clusters and phrases.

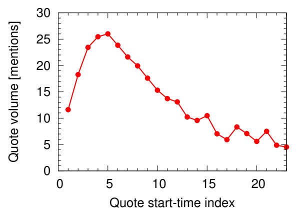
We observe that memes with shorter most popular phrases survive longer (Fig. 11(e)), while memes with short roots diminish sooner (Fig. 11(f)). Such behavior is consistent with our explanation above. Memes with short catch phrases that are parts of longer narratives survive longer and are also more diverse [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 12 shows the results and 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 12(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 dies [20, 38]. 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."* When focusing on the number of mentions a phrase receives as a function of the order of appearance (Figure 12(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



(a) World length



(b) Volume

Figure 12: Phrase word length and volume as a function of order of the appearance.

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.

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 [38].

5. 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 time and also 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 of our dataset may allow for studying the evolution of online media commentary and practices, as well as what kind of collective behavior leads directly to the ways in which all of us experience news and its consequences.

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

[36] 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.

[37] 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.

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

[39] 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.