On Burstiness-Aware Search for Document Sequences

Theodoros Lappas[+] Benjamin Arai[+] Manolis Platakis[*]

Dimitrios Kotsakos[*] Dimitrios Gunopulos[*+]

[+]Dept. of Computer Science and Engineering, University of California, Riverside

{tlappas|barai|dg}@cs.ucr.edu

[*]Dept. of Informatics and Telecommunications, University of Athens

{platakis|d.kotsakos}@di.uoa.gr

ABSTRACT

As the number and size of large timestamped collections (e.g. sequences of digitized newspapers, periodicals, blogs) increase, the problem of efficiently indexing and searching such data becomes more important. Term burstiness has been extensively researched as a mechanism to address event detection in the context of such collections. In this paper, we explore how burstiness information can be further utilized to enhance the search process. We present a novel approach to model the burstiness of a term, using discrepancy theory concepts. This allows us to build a parameter-free, linear-time approach to identify the time intervals of maximum burstiness for a given term. Finally, we describe the first burstiness-driven search framework and thoroughly evaluate our approach in the context of different scenarios.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing—*Indexing methods* ; H.2.8 [**Database Management**]: Database applications—*Data Mining*; G.3 [**Probability and Statistics**]: *Time series analysis*

General Terms

Algorithms, Theory

Keywords

Document Sequences, Search, Burstiness

1. INTRODUCTION

Suppose we are presented with a long *document sequence*, formed by newspaper articles spanning several local titles (e.g., New York Times, The Wall Street Journal, etc.), over a large period of time. Such corpora is becoming increasingly available, due to initiatives such as The National Digital Newspaper Program (NDNP) [15] by the Library of Congress (LC), and other similar ventures for the digitization of periodicals by large corporations such as Microsoft (www.microsoft.com) and Google (www.google.com). The articles in such collections cover newsworthy events that took place at various times. Each event is characterized by a set of descriptive keywords, revealing basic information such as the place where the event occurred, or the names of the persons involved. For the duration of the event's lifespan and consequent coverage in the news, these characteristic terms appear repeatedly in relevant articles, leading to uncommonly high frequencies (bursts). In the typical search paradigm, the user encodes a topic of interest using a query (i.e. a set of keywords), which is then submitted to a search engine. Typical search engines rely on static, frequency-based measures (e.g. *tf-idf*) for the purposes of indexing and querying the underlying collection. These measures record the frequency of a term in each document, typically normalized by a global frequency measure, in order to capture the impact of the term in the entire collection. The underlying assumption is that an occurrence of a term has the same significance, regardless of the moment in time it occurs. Our claim is that, for a contiguous document sequence observed through time, this assumption is invalid: the importance of terms varies through time, as they are used to describe current influential events that are discussed in the corpus. Therefore, it is essential to consider the temporal dimension of the data in the indexing and ranking process. The ultimate purpose of our work is the creation of an efficient, end-to-end framework that, given a document sequence, identifies "bursty" intervals for each term and utilizes this information toward an efficient, burstiness-aware search mechanism.

Even though some work has been devoted to measuring burstiness in different contexts, the concept has yet to be formalized. A major contribution of our work is a formal definition of burstiness that is based on the concept of discrepancy. Discrepancy theory has applications in several fields including machine learning, computer graphics and computational geometry [7, 8, 4, 1]. The concept is generally used to describe the deviation of a situation from the

KDD'09, June 28–July 1, 2009, Paris, France.

"expected" behavioral baseline. Based on our definition, we present a parameter-free, linear-time algorithm to identify the time-intervals that maximize the burstiness score of any given term. We present the theoretical foundations of our work and proceed to evaluate it thoroughly on a new dataset.
Our Contributions: In this paper we make the following contributions:

i. A formal definition of term burstiness in the terms of numerical discrepancy.

ii. A parameter-free, linear-time method to identify the maximum burstiness intervals for a given term.

iii. An efficient search framework for documents, that considers term burstiness in the indexing and ranking process. The framework uses an extension of the well-known TA algorithm [9] for finding the top intervals.

The rest of this paper is organized as follows: Section 2 discusses related work. Section 3 describes the basic notation used in the paper. Section 4 introduces our definition of burstiness and discusses efficient techniques for the identification of bursty intervals for a given term. In Section 5, we present two different versions of a complete, burstiness-aware search framework. Finally, we conclude with a thorough experimental evaluation in Section 6.

2. RELATED WORK

The concept of burstiness has been studied in several domains. A significant portion of this work has been inspired by Kleinberg's seminal paper on the bursty and hierarchical structure of streams [13]. We discuss Kleinberg's approach in more detail in Section (6.2). A considerable amount of work has been devoted to developing efficient burst-detection methods [10, 11, 18, 19]. Even though we propose a method of our own, we do so in the process of creating a complete search framework, which is the main contribution of our work. The main benefits of our method are that it runs in linear-time and is also completely parameter-free. This makes it ideal for very large sequences of documents, spanning significant periods of time. That being said, our search framework is compatible with *any* burst detection method that can report non-overlapping bursty intervals and their respective scores, for any given term.

Another burst-detection method is presented by Fung et al.[10]. In this work, bursty terms are clustered to represent events discussed in the data. In [11], the authors classify terms in four burstiness categories, based on their frequency trajectory. Their use of spectral analysis is similar to the one used by Vlachos et al. in [18], where the authors focus on periodic and bursty artifacts in query logs. In [19], the authors use a wavelet-based structure for aggregate monitoring of data streams.

Burstiness has also been evaluated in the context of other applications, such as stream clustering [12], and even in the context of graphs [14]. Further, He et al. [16] apply Kleinberg's model to topic clustering.

Bansal and Koudas [2, 3] have presented a system for the analysis of streaming blogs. Even though no details on the employed methods are given, their work is relevant to ours, in that they ultimately map bursty terms to specific blogposts. To the best of our knowledge, our work is the first that directly incorporates burstiness information in the indexing and ranking of documents, leading to a complete burstiness-aware search framework.

3. PRELIMINARIES

In this paper, we explore corpora that are formed as a sequence of documents, spanning a pre-defined timeline. For a timeline of m consecutive timestamps, we define a *document sequence* $\mathcal{S}$ as:

$$\mathcal{S} = S_1, S_2, ..., S_m \tag{3.1}$$

where S_i represents the set of documents appearing on the i_{th} timestamp. Further, we define the *frequency sequence* Y_t for a given term t as:

$$Y_t = y_{t1}, y_{t2}, ..., y_{tm} \tag{3.2}$$

where y_{ti} expresses the frequency of term t on the i_{th} timestamp of the timeline. We assume that y_{ti} is equal to the number of documents in S_i that include term t. Finally, by $Y_t[l:r]$, we represent an interval of Y_t that includes all timestamps from y_{tl} to y_{tr} (inclusive). Note that the words "interval" and "segment" are used interchangeably throughout the paper.

4. DEFINING TERM BURSTINESS

In this section we present a formal definition of term burstiness in the terms of *numerical discrepancy*. We then show how the problem of finding the maximum burstiness intervals can be solved in linear time.

4.1 A Discrepancy Model of Burstiness

We first present the general definition of numerical discrepancy [7, 8, 4]. Let $\mathcal{P}$ be a set of points distributed over random locations in $[0,1]^d$, where d is the number of dimensions on the plane. For any region $\mathcal{R}$ in $[0,1]^d$, let $\mu(R)$ be the Euclidean measure of $R \cap [0,1]^d$ (i.e. the area of R), and $\mu_{\mathcal{P}}(R)$ be the discrete measure $|R \cap \mathcal{P}|/|\mathcal{P}|$ (i.e. the fraction of points of $\mathcal{P}$ inside R). Then, the *numerical discrepancy* of R with respect to P is defined as:

$$\mathcal{D}_{\mathcal{P}}(R) = |\mu(R) - \mu_{\mathcal{P}}(R)| \tag{4.1}$$

Even though the concept is meaningful for any $d > 1$, we will focus on the one-dimensional case, suitable for our sequence representation. For $d = 1$, a region R is reduced to a one-dimensional interval I, defined within the unit interval $[0,1]$. Following Equation (4.1), the discrepancy of a given interval I is defined as the absolute value of the difference between its length and the ratio of points from $\mathcal{P}$ that fall within I:

$$\mathcal{D}_{\mathcal{P}}(I) = |\mu(I) - \mu_{\mathcal{P}}(I)| = \left| len(I) - \frac{|\mathcal{P} \cap I|}{|\mathcal{P}|} \right|, \tag{4.2}$$

where $len(I)$ is the length (i.e. the euclidean measure) of interval I. Conceptually, $\mu(I)$ expresses the baseline, i.e. the fraction of points from $\mathcal{P}$ that is *expected* to fall within I, while $\mu_{\mathcal{P}}$(I) represents the observed fraction. This constitutes an appropriate definition for term burstiness, which is similarly expressed by increased frequency values that diverge from a term's individual baseline. In the mapping of term burstiness to numerical discrepancy, the set of points $\mathcal{P}$ is represented by the total frequency of a term, observed throughout the entire document sequence. Next, we formally define the baseline $\mu(I)$ and the observed fraction $\mu_{\mathcal{P}}(I)$ in the context of term burstiness.

Depending on the nature of the data, $\mu(I)$ can be either pre-defined or based on the underlying distribution. In [1], the authors explore discrepancy in the context of different

distributions (Poisson, Binomial). Even though such an approach may work well in some scenarios, the assumption that the entire dataset can be accurately described by a single distribution is not always valid. In addition, the use of a probability distribution introduces parameters that are not always intuitive to tune. Given an interval $Y_t[l:r]$ of the frequency sequence for a term t, we define the baseline as

$$len(Y_t[l:r]) \times Avg(Y_t),$$

i.e. the average frequency observed over all timestamps, multiplied by the length of the interval. To conform with the definition of numerical discrepancy, we then project $Y_t[l:r]$ on the unit interval $[0,1]$ by dividing the baseline by $\sum_{i=1}^{n} y_{ti}$.

Formally, let I be the projection of $Y_t[l:r]$ on $[0,1]$. Then, we define the baseline $\mu(I)$ for I as:

$$\mu(I) = \frac{len(Y_t[l:r]) \times Avg(Y_t)}{\sum_{i=1}^{m} y_{ti}} \tag{4.3}$$

By replacing the average and solving further, we get:

$$= \frac{len(Y_t[l:r]) \times (\sum_{i=1}^{m} y_{ti})/m}{\sum_{i=1}^{m} y_{ti}}$$

$$= \frac{len(Y_t[l:r])}{m} = len(I) \tag{4.4}$$

Indeed, the baseline is equal to the Euclidean measure (length) of I, as mandated by Eq. (4.2).

Next, we define $\mu_{\mathcal{P}}(I)$, the fraction of the term's frequency observed within the interval, as the frequency of t observed within interval $Y[l:r]$, divided by the total frequency of t throughout the sequence:

$$\mu_{\mathcal{P}}(I) = \frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} \tag{4.5}$$

By replacing Equations (4.4) and (4.5) in Eq. (4.2), we get:

$$\mathcal{D}_{\mathcal{P}}(I) = \left| \frac{len(Y_t[l:r])}{m} - \frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} \right| \tag{4.6}$$

Note that Eq. (4.6) takes positives values if the observation is either *greater* or *less* than the baseline. Conceptually, the latter occurs if the frequency of a term within an interval is *less* than expected. Even though this typically constitutes a case of discrepancy, it is of little value for the purposes of measuring term burstiness. Instead, we would like burstiness to be positive only for **uncommonly high** frequency observations. Thus, given a term t and an interval $[l:r]$ on the timeline, we define the **Burstiness of t in $[l:r]$** as:

$$\mathcal{B}(t,[l:r]) = \left(\frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} - \frac{len(Y_t[l:r])}{m} \right) \tag{4.7}$$

4.2 Maximizing Burstiness

Using Eq. (4.7), we can measure the burstiness of a given term for any interval on the timeline. The next step is to identify high-burstiness intervals for each term. On a higher level, the problem definition is the following:

Problem 1. **Bursty Intervals Problem:** Given the frequency sequence Y_t of a given term t, identify the set of intervals that maximize the Burstiness function $\mathcal{B}(t,\cdot)$.

Next, we argue that Problem 1 is equivalent to the well-known *maximum sum segments problem*, defined as such:

Problem 2. **Maximum Sum Segments Problem:** Given an input sequence $X = x_1, x_2, ..., x_n$ of real numbers, identify the K segments with the highest total scores, where the score $f(X[i:j])$ of a segment $X[i:j] = x_i, x_{i+1}, ..., x_j$ is equal to the sum of its elements:

$$f(X[i:j]) = \sum_{k=i}^{j} x_k \tag{4.8}$$

The **Maximum Sum Segments** Problem comes up in different domains and has been extensively researched in the past [5]. To show that Problems (1) and (2) are equivalent, it is sufficient to show that, given a term t, the Burstiness score of any given interval is equal to the sum of the Burstiness values observed in the individual timestamps of the interval. Formally:

$$\mathcal{B}(t,[l:r]) = \sum_{k=l}^{r} \mathcal{B}(t,[k:k]) \tag{4.9}$$

PROOF.

$$\sum_{k=l}^{r} \mathcal{B}(t,[k:k]) = \sum_{k=l}^{r} \left(\frac{y_{tk}}{\sum_{j=1}^{m} y_{tj}} - \frac{1}{m} \right)$$

$$= \left(\frac{\sum_{i=l}^{r} y_{ti}}{\sum_{j=1}^{m} y_{tj}} - \frac{len(Y_t[l:r])}{m} \right) = \mathcal{B}(t,[l:r])$$

□

Problem (1) is now reduced to solving the **Maximum Sum Segments** Problem on the *burstiness sequence* $\mathcal{B}_t$, defined as such:

$$\mathcal{B}_t(i) = \mathcal{B}(t,[i:i]) = \left(\frac{y_{ti}}{\sum_{j=1}^{m} y_{tj}} - \frac{1}{m} \right), 1 \le i \le m \tag{4.10}$$

Eq. (4.10) assumes the global baseline given by Eq. (4.3). Alternatively, one could use the local average of each interval as a baseline. In that case, consecutive segments of $\mathcal{B}_t$ could be computed separately, and then concatenated to form the entire sequence. This could allow for a more flexible calculation of burstiness, and avoid reporting anticipated periodical bursts (e.g. a burst of the term "Christmas" during December).

The standard formulation of the **Maximum Sum Segments** problem has the following disadvantage: given a high-scoring segment, one can easily generate several others by simply appending or removing a small number of elements. These segments convey little extra information regarding the burstiness of a term. To address this, we adopt a slightly different formulation, based on the concept of the *maximal segment*:

Definition 1. **Maximal Segment:** Let X be a non-empty score sequence. A segment $X[i:j]$ is **maximal in** X if

i. All proper sub-segments of $X[i:j]$ have a lower score

ii. No proper super-segments of $X[i:j]$ in X satisfies (i).

Figure (1) illustrates the Burstiness Sequence $\mathcal{B}_t$ for the term "Earthquake", as it manifested in a daily newspaper

over a fixed period of time. The values on the x-axis represent consecutive timestamps. Following Eq. (4.10), negative values correspond to points when the observed frequency was less than the baseline. Here, "WZ" is identified as a maximal segment; conceptually, extending the interval from either side can only reduce its score, since more negative than positive values will be included.

No two maximal sequences can overlap. A formal proof appears in [17], but essentially, given two maximal overlapping sequences, either the union or the intersection of the two would have higher discrepancy than one of the two, creating a contradiction. Thus, every element of the input sequence belongs to exactly one maximal segment. Therefore, for any given sequence of real numbers, there exists a finite set that contains *all* the maximal scoring segments. We can now formalize the problem as such:

Problem 3. **All Maximal Segments Problem:** Given an input sequence $X = x_1, x_2, ..., x_n$ of real numbers, identify the set of *all* segments of X that satisfy Definition (1).

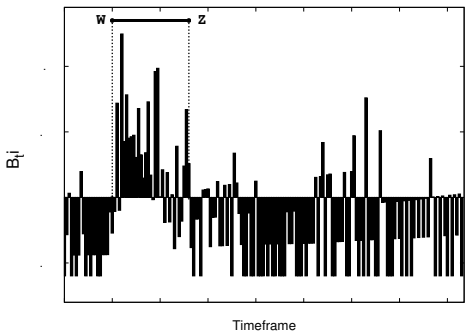

Figure 1: Burstiness sequence $\mathcal{B}_t$ for t="earthquake"

4.3 Algorithms for the All Maximal Segments Problem

In [17] the authors present a linear-time algorithm for solving the **All Maximal Segments Problem**. The algorithm accepts as input a sequence of real numbers and reports the set of all maximal segments. For the rest of this paper, we refer to this algorithm as `MAX-1`. The details and pseudocode of the algorithm can be found in [17]. `MAX-1` filters out maximal segments with a negative score. This is ideal for the purposes of burstiness evaluation, since negative-scoring intervals represent regions where the observed frequency of a term was less than the expected. Finally, in addition to being linear, the approach is completely parameter-free. Next, we present an extension of `MAX-1` and discuss its advantages.

In [13], Kleinberg discusses *anisochronies*, the non-uniform relationships between the time spanned by a story's events and the amount of time devoted to these events in the actual telling of the story. Considering the coverage of events in news streams (e.g. newspapers, blogs), we identify two primary levels of bursty behavior for the terms describing an event: the first level represents the extended time period when the event was generally discussed in the news. Depending on the nature and significance of the event, this period can be extended to include weeks or even months. The second burstiness level pertains to smaller intervals within this extended period, when the event was particularly popular and extensively covered in the news. In the context of a newspaper, such intervals may represent the first time an event made the headlines, or a new development in an older event that brings it back to the front page.

Conceptually, the intervals reported by `MAX-1` capture the first level of burstiness activity for a given term. By reapplying the algorithm on each of the reported maximal intervals independently, we can easily identify the second-level burstiness intervals. For the rest of this paper, we refer to this algorithm as `MAX-2`. The pseudocode is shown in Algorithm (1). Multiple iterations of `MAX-2` could be used to obtain a hierarchical structure of the bursty intervals. As we demonstrate in the Experiments section, a single iteration is enough to capture the burstiness patterns of events.

Algorithm 1 : `MAX-2`

Input: $\mathcal{I}$: Set of first-level maximal intervals for Y_t
Output: $\mathcal{I}'$: Set of second-level maximal intervals for Y_t
1: $\mathcal{I}' \leftarrow \emptyset$
2: **for** every interval $I \in \mathcal{I}$ **do**
3: $\quad \mathcal{I}' \leftarrow \mathcal{I}' \cup$ `MAX-1`(I) // `MAX-1` returns 1st level intervals
4: Return $\mathcal{I}'$

5. QUERY EVALUATION

In this section, we describe two different ways to utilize burstiness information to create a complete, burstiness-aware search framework. The described search frameworks constitute the main contribution of our work. Our first approach focuses on indexing and ranking documents directly, while the second approach is more advanced and performs a more informative, interval-based evaluation of a given query. For each approach, we start by discussing the underlying indexing mechanism, and then proceed to discuss the respective query evaluation algorithms.

It is important to note that both approaches are compatible with *any* method than can evaluate the frequency sequence of a term over a specified timeline, and report **non-overlapping** bursty intervals and their respective scores.

5.1 Evaluating Documents

Next, we describe a burstiness-aware query evaluation framework that retrieves and ranks documents based on a given query. First, we discuss the employed indexing mechanism.

Indexing: In the standard inverted index structure, each term is mapped to the list of documents that contain the term. In a more advanced scenario, the document lists are sorted on a pre-computed score that expresses the strength of the connection between the term and the document. In order to use such a structure in our framework, we need to define a formula that evaluates a document with respect to a given term, in the context of burstiness:

Definition 2. Given a term t and a document d, let $I_{t,d}$ be the bursty interval of t that includes (the timestamp of) d. Then, the **Burstiness of d with respect to t** is defined as:

$$d\text{-}score(t,d) = \begin{cases} \mathcal{B}(t, I_{t,d}) \times freq(t,d) & , \text{ if } I_{t,d} \neq \emptyset \\ 0, \quad \text{otherwise} & \end{cases} \tag{5.1}$$

where *d-score* stands for *document score*. Conceptually,

$\mathcal{B}(t, I_{t,d})$ returns the burstiness score of $I_{t,d}$, as defined by Eq. (4.7). Also, $freq(t,d)$ returns the frequency of term t with respect to document d. In our experiments, we assume $freq(t,d) = \log(TF(t,d)+1)$, where $TF(t,d)$ returns the number of occurrences of t in d. The logarithm is used to moderate the effect of the frequency and ensure that burstiness is the dominant factor. Finally, if $d\text{-}score(t,d) = 0$, d is not included in the sorted list. Note that

We can now build an inverted index structure, where each term is mapped to a list of documents, sorted on their *d-score*. Next, we discuss how we can use this index to evaluate multi-term queries.

Evaluation: First, we formally define the *Document Evaluation Problem* as such:

Problem 4. **Document Evaluation Problem**: Given a query of terms $q = \{t_0, t_1, ...\}$, retrieve the k documents with the k highest values for $\sum_{t \in q} d\text{-}score(t,d)$.

With an appropriate index structure at our disposal, the next step is to find a query evaluation algorithm to address Problem 4. For this purpose, we use the Threshold Algorithm (`TA`)[9], an efficient top-k evaluation algorithm, which is able to deal with multi-predicate queries. The algorithm goes through the sorted lists mapped to the terms of a query, evaluating documents in descending order. For every document seen under sorted access in some list, a random access probe retrieves the respective scores of the document from the other lists. The cumulative score is then calculated, and the document is considered as a top-k candidate. The algorithm maintains a threshold value T, based on the score of the last document seen from every List. As soon as k documents with a cumulative score of at least T have been found, the algorithm terminates. The authors prove that, while this mechanism allows for early termination, it does not affect the optimality of the result. Algorithm (2) contains the pseudocode of the `TA` algorithm. The algorithm is designed so that candidates from different sorted lists can be also evaluated in parallel. In that case, lines 5-11 of the algorithm can be handled by independent threads.

The proposed Inverted Index structure and the `TA` algorithm compose a complete search framework that efficiently solves the **Document Evaluation Problem**. The framework is thoroughly evaluated in the Experiments Section.

5.2 Evaluating Intervals

The framework described in the previous section focuses on the indexing and ranking of documents. In this section, we describe an alternative approach that places the focus on intervals. Given a query of terms, we would like to find periods of time when all terms simultaneously displayed bursty behavior, indicating the occurrence of an underlying event. Next, we describe a search framework for this problem.

Indexing: First, we define a formula that evaluates the burstiness of interval with respect to a given term:

Definition 3. Let $\mathcal{I}_t$ be the set of bursty intervals for a term t. Then, given a query of terms $q = \{t_0, t_1, ..\}$, an interval I is identified as **bursty with respect to** q, if $\forall t \in q,\ \exists\ I' \in \mathcal{I}_t,\ s.t.\ I \subseteq I'$. Then, the burstiness score of I with respect to q is defined as:

$$i\text{-}score(I,q) = \sum_{t \in q} B(t, super(\mathcal{I}_t, I)) \quad , \qquad (5.2)$$

Algorithm 2 `TA Algorithm`

Input: `query q = {t0, t1, ...}, int k`
Output: `Set of top-k documents`

```
1: TopK ← ∅  // sorted, holds at most k elements
2: Threshold T ← 0
3: L = {L0, L1, ..} // set of Doc Lists for each term in q
4: while (Not All lists in L Have Been Exhausted) do
5:     for (every List L ∈ L) do
6:         cand ← getNext(L)
7:         total ← cand.score  // Holds cumulative score
8:         T ← (T − lastSeen(L) + cand.score)
9:         for (every List L' ∈ L, L' ≠ L) do
10:            total+ = getDScore(L', cand)
11:        TopK.insert(cand, total)
12:        if ((TopK.size() == k) && (TopK.last() >= T))
13:            return TopK // Early Termination
14: return TopK
```

- `getNext(L)` returns the next candidate to be evaluated from list L, under sorted access.
- `getDScore(L', cand)` is a random access probe that retrieves the *d-score* of the candidate document from list L'.
- `lastSeen(L)` returns the score of the last candidate seen under sorted access in List L
- `TopK.last()` returns the score of the lowest-scoring element in the Result.

where *i-score* stands for *interval-score*. Also, $super(\mathcal{I}_t, I)$ returns the interval $I' \in \mathcal{I}_t,\ s.t.\ I \subseteq I'$ (i.e. I' is a **super-segment** of I).

Conceptually, I is bursty with respect to a query if it has been included in a bursty interval for *all* the terms in the query. The *i-score* of I is then the sum of the scores of all the bursty intervals that include it. Note that this definition requires the bursty-interval set $\mathcal{I}_t$ for a given term t to consist of **non-overlapping intervals**. This guarantees that **at most one** interval $I' \in \mathcal{I}_t$ is a super-segment of I. Using Eq. (5.2), we can now build an inverted index structure, where each term is mapped to a list of intervals, sorted on their *i-score*. Next, we discuss how we can use this index to evaluate multi-term queries.

Evaluation: First, we formally define the *Interval Evaluation Problem* as such:

Problem 5. **Interval Evaluation Problem:** Given a query of terms $q = \{t_0, t_1, ...\}$, retrieve the k intervals with the highest values for $\sum_{t \in q} i\text{-}score(t,d)$.

For the top-k evaluation phase, we introduce a modified version of the `TA` Algorithm, which we refer to as `TA`* (Algorithm (3)). `TA`* is similar to `TA`, differing only in the use of the random access probe. In the standard version, a random access probe looks for the candidate document in the various document lists and retrieves its *d-score* (line 10 of Algorithm 2). In the case of intervals, this step is more complicated, since the candidate may overlap with multiple intervals in a list. Procedure (1) provides an implementation of the Random Access probe. Given an Interval I and a list of intervals L the probe returns **the set** of (sub)intervals of I that overlap with some interval in L. The procedure can be easily implemented with the use of interval-trees [6].

Procedure 1 `RandomAccess(Interval I, Interval List L)`

Return a set of intervals $\mathcal{I}$, s.t.
$\forall$ I$'$ $\in$ L, where I$'$ $\cap$ I $\neq \emptyset$,
$\exists$I* $\in \mathcal{I}$, where I* = I$'$ $\cap$ I AND
I*.score = I.score + I$'$.score

Algorithm 3 `TA*` Algorithm

Input: query q = {t0, t1, ...}, int k
Output: Set of top-k Intervals

```
1: TopK ← ∅  // sorted, holds at most k distinct elements
2: Threshold T ← 0
3: L ← {L0, L1, ..} // set of Doc Lists for each term in q
4: while (Not All lists in L Have Been Exhausted) do
5:     for (every List L ∈ L) do
6:         cand ← getNext(L)
7:         T ← (T − lastseen(L) + cand.score)
8:         X[i] ← {cand}
9:         for (every List L' ∈ L, L' ≠ L) do
10:            X[j] ← RandomAccess(cand, L')
11:        F ← merge(X[0], X[1], ...)
12:        for every Interval I in F do
13:            TopK.insert(I, I.score)
14:        if ((TopK.size() == k) && (TopK.last() >= T))
15:            return TopK // Early Termination
16: return TopK
```

- `getNext(L)`, `lastSeen(L)` and `TopK.last()` are as in the `TA` Algorithm.
- The $\mathcal{X}[\,]$ variables represent sets of intervals.
- The `RandomAccess()` function is described in Procedure 1.
- The use of the `merge()` function is shown in Figure (2).

After the sets of overlapping (sub)intervals from each List have been retrieved, they are merged to produce the final set $\mathcal{F}$, consisting of segments included in bursty intervals for *all* the terms of the query (Line 11 of Algorithm 3). Figure (2) shows an example of the merging process for a query $q = \{t_0, t_1, t_2, t_3\}$. The interval-set $\mathcal{X}[0]$ contains only one interval: the candidate, selected under sorted access from the bursty-interval list of term t_0. Also, $\mathcal{X}[1], \mathcal{X}[2]$ and $\mathcal{X}[3]$ contain intervals that overlap with the candidate, retrieved by applying the `RandomAccess` Procedure on the bursty-interval lists of terms t_1, t_2 and t_3, respectively. According to Definition 3, only the interval $I = [5:7]$ qualifies as **bursty with respect to q**. Following Eq. (5.2), the burstiness score of candidate I is equal to $\sum_{t \in q} B(t, super(\mathcal{I}_t, I)) = 6 + 4 + 5 + 4 = 19$.

The top-k set produced by `TA*` optimally solves the **Interval Evaluation Problem**. The reported intervals reveal bursty periods for any multi-term query. This allows us to not only locate events correlated with particular terms, but also estimate their lifespan. Further, the framework can be easily extended to report the documents that appear within each interval, and also contain all the query terms. Thus, we can obtain ranked groups of documents, where each group is relevant to a specific bursty period. Clearly, this is more informative than a mechanism that simply reports k documents from completely arbitrary timestamps.

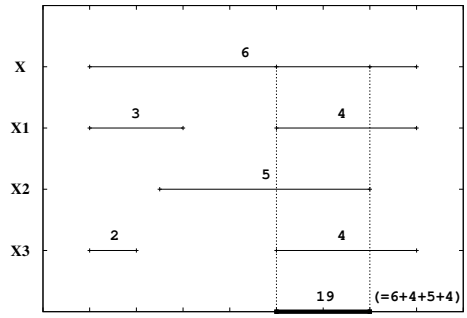

Figure 2: Interval Merging Process

6. EXPERIMENTS

In this section, we illustrate the efficacy of our search framework through a rigorous experimental evaluation. Section 6.1 describes the datasets we used. Section 6.2 discusses the different burst-detection methods used in our experiments. Finally, Sections 6.3-6.5 evaluate our search framework in different scenarios.

6.1 Datasets

Newspaper Datasets: We have conducted a series of experiments using real-world datasets from the Center for Bibliographical Studies and Research (CBSR) at the University of California, Riverside (UCR). CBSR has received two grants from the National Endowment for the Humanities to participate in the National Digital Newspaper Program (NDNP). The NDNP is a joint venture of the National Endowment for the Humanities and the Library of Congress to create a national digital newspaper resource, representing papers from all states, published between 1836-1922.

For the experimental evaluation we have gathered over 390,000 articles from the *San Francisco Call*, a daily newspaper with publication dates between 1900-1909. After the removal of stopwords, approximately 120,000 distinct terms were identified. We have several attributes for each article, including the title, the date of publication, and the raw (punctuation and capitalization included) content. Due to the age and size of the corpus, some issues were not located for digitization, leaving small gaps in the data set. To address this, we extracted 3 independent document sequences from the data, for which all the articles were available:

- `SF-Call-1`: A sequence of 122,114 articles spanning from Jan 01, 1900 to Dec 31, 1901.
- `SF-Call-2`: A sequence of 144,289 articles spanning from Jan 01, 1903 to Dec 31, 1904.
- `SF-Call-3`: A sequence of 153,412 articles spanning from Jan 01, 1908 to Dec 31, 1909.

These large sequences of chronologically ordered articles will serve as datasets for the experiments described in this section. **The data is available on request.**

Major Events List: In order to perform a qualitative evaluation of our approaches, we manually composed a list of major events that took place at a time covered by one of the three **Newspaper Datasets**. The events were taken from Wikipedia (www.wikipedia.com), which maintains annual lists of major events. For every event, a query was

composed, consisting of keywords chosen for their particular significance with respect to the event. Table (3) contains the list of events and their respective queries.

6.2 Burst Detection

Throughout the experiments section, we evaluate the performance of the proposed search frameworks, using the `MAX-1` and `MAX-2` algorithms, described in Section 4, to obtain the required bursty intervals. As an alternative, we try the popular burst-detection method proposed by Kleinberg in [13]. This algorithm is based on a Hidden Markov Model, with states that correspond to frequency levels for individual terms. State transitions (bursts) correspond to points in time, around which the frequency of a term changes significantly. Given the frequency sequence Y_t of a term t, dynamic programming is used to fit the most possible state sequence that is likely to have generated Y_t. The state assigned to each interval will serve as its burstiness score, which is required by our framework. For the rest of this paper, we refer to this algorithm as `KLEIN`.

The states reported by `KLEIN` form a hierarchical structure, with a long burst of low intensity including several bursts of higher intensity. Clearly, this violates our requirement for non-overlapping bursty intervals. To address this, we give priority to higher-state intervals, by assigning to every timestamp i the highest state observed over all the reported intervals that include i. To be fair, if the length of the highest-state interval is too small($<$3), we take the interval with the second-highest state. We believe this to be a reasonable and intuitive aggregation method.

Further, by assigning a high cost to state transitions, one can restrain the number of states in the hierarchy reported by `KLEIN`, thus eliminating short bursts and leading to longer intervals. Reasonably long intervals that reflect the true lifespan of an event are desirable, since they are likely to contain more relevant documents. On the other hand, the assignment of very high costs will limit the score-space to a small set of (low-intensity) states. Consider having to rank 10 documents based on their state, where each document has 1 of 2 distinct states; inevitably, multiple ties will lead to a meaningless ranking. For our experiments, `KLEIN` was tuned to find a balance between reasonably long intervals and an adequate number of distinct states. Note that our parameter-free algorithms resolve such issues by using the concept of the maximal segment to automatically extend a segment as long as it can benefit its score.

6.3 Document Ranking

The purpose of this experiment is to evaluate the Document Evaluation framework described in Section 5.1. The evaluation is done as follows: First, an inverted index is built on top of each on the three **Newspaper Datasets**, as described in Section 5.1. Then, the queries from the **Major Events List** are evaluated using the `TA` Algorithm. Queries mapped to events from 1900 and 1901 are evaluated using the index built on top of `SF-Call-1` and so forth. The entire process is repeated 3 times, each time using one of the three burst-detection algorithms (`MAX-1`, `MAX-2` and `KLEIN`) to build the search framework. We also compare against **Lucene** (lucene.apache.org), a popular text-search engine. Lucene uses frequency-based measures such as the frequency of the term within each document and the term's global frequency to rank documents in the context of a given query.

Table 1: Achieved Precision on Major Events List

ID	Lucene		MAX-1		MAX-2		KLEIN	
1	1/5	2/10	5/5	10/10	5/5	9/10	5/5	8/10
2	3/5	5/10	5/5	7/10	5/5	8/10	5/5	8/10
3	3/5	6/10	5/5	10/10	5/5	10/10	-	-
4	3/5	7/10	5/5	10/10	5/5	10/10	5/5	10/10
5	1/5	2/10	3/5	6/10	4/5	8/10	-	-
6	4/5	6/10	5/5	9/10	5/5	9/10	5/5	10/10
7	3/5	6/10	5/5	10/10	5/5	10/10	4/5	9/10
8	3/5	4/10	5/5	10/10	5/5	10/10	4/5	9/10
9	5/5	9/10	5/5	10/10	5/5	10/10	5/5	9/10
10	0/5	1/10	5/5	10/10	5/5	10/10	5/5	10/10
11	5/5	10/10	5/5	10/10	5/5	10/10	5/5	10/10
12	4/5	8/10	5/5	10/10	5/5	10/10	-	-
13	3/5	6/10	5/5	10/10	5/5	10/10	5/5	8/10
14	4/5	7/10	5/5	8/10	5/5	7/10	-	-
15	3/5	5/10	4/5	8/10	5/5	9/10	5/5	7/10
16	2/5	2/10	5/5	9/10	5/5	9/10	-	-

A human annotator studied each of the top-10 documents reported for each event, marking them as "relevant" or "non-relevant". This allows us to evaluate the achieved precision, defined as the ratio of the number of relevant documents over the total number of retrieved documents. The results are shown in Table (1). The table contains a separate column for the achieved recall in the top-5 documents, to provide more insight on the quality of the produced ranking. Our framework performs consistently well, clearly outperforming Lucene in almost every case. Regarding the different burst detection algorithms, `MAX-1` and `MAX-2` achieved near-perfect precision values for all submitted queries. `KLEIN`'s precision was just as good, although it failed to retrieve any documents for 5 of the 16 queries. This can be due to the fact that `KLEIN` did not identify any intervals as bursty for **all** the terms in the query. Alternatively, even if such a region was identified, it did not include any documents containing all the query-terms. This could be addressed by separately tuning the parameters of the algorithm for each term. In practice, however, this is not desirable for obvious reasons.

6.4 Interval Ranking

The purpose of this experiment is to evaluate the Interval Evaluation framework described in Section 5.2. The experiment is similar to the one described in the previous Section. In this case, the index built on top of each of the **Newspaper Datasets** was the one described in 5.2, which considers term burstiness to index **intervals** rather than documents. Also, the `TA`* Algorithm was used to evaluate the queries from the **Major Events List**. For each event, we identify the interval among the reported 10 that is closest to the actual date of the event. We then report the start and end dates of that interval. The process is again repeated 3 times, each time using one of the three burst-detection algorithms. The results are shown in Table (4).

Both `MAX-1` and `MAX-2` produce reasonable intervals for the evaluated queries. As anticipated, `MAX-2` gives tighter intervals, which commonly span a few days or weeks around the actual date of the event. The intervals produced by `KLEIN` are of similar or smaller length. Also, no bursty intervals were identified for queries 3, 5, 12, 14 and 16. As discussed in Section 6.2, even though the algorithm could be tuned to report larger segments, this would also reduce the number of states and thus have an adverse effect top-k evaluation.

In general, KLEIN produced accurate results, indicating that our search framework is compatible with any efficient burst-detection method. Finally, it is also important to note that, for all three algorithms, the intervals closest to the actual event date were always ranked **first** in the top-10 list.

In order to illustrate the utility of the proposed Interval Evaluation Framework, we do an additional experiment: let $\mathcal{A}$ be the set of all articles within the interval reported by a query. Also, let $\mathcal{V}$ be the set of distinct terms appearing in the titles of the articles in $\mathcal{A}$. We then report the top-10 terms from $\mathcal{V}$, ranked in descending order on the number of titles they appeared in. For lack of space, we only report the results reported by MAX-2, since it produced the most reasonable intervals for all the queries in the **Major Events List**. The results, shown in Table (5), prove that the documents of a top-k interval can be used to identify terms that describe the underlying event. In the context of a search engine, these terms can compose an informative cloud that suggests insightful queries to the user.

6.5 Index Statistics

In this experiment, we show that, by focusing only on bursty intervals, we can greatly reduce the number of documents mapped to each term. This fact, combined with the high-quality results shown in the previous experiments, proves that our index structure is compact, while preserving all the useful information for each term.

First, we build the Document Evaluation framework, described in Section 5.1, for each of the three **Newspaper Datasets**. For each dataset, we compute the average number of documents mapped to a term. The process is repeated 3 times, once for each of the three burst-detection algorithms. We compare against Lucene, which essentially maps each term to all the documents that include it. A similar evaluation is done for the Interval Evaluation Framework, described in Section 5.2: for each term, we compute the percentage of the timeline (spanned by each collection) that is covered by bursty intervals. We then report the average over all terms. The results are shown in in Table (2). As can be seen from the Table, our framework achieves a

Table 2: Statistics Table

	SF-Call-1	SF-Call-2	SF-Call-3
Avg. number of documents per term			
Lucene	124.45	119.3	112.4
MAX-1	85.6	83.5	74.9
MAX-2	73.5	75.2	63.2
KLEIN	72.35	74.9	72.7
Avg. covered timeline % per term			
MAX-1	0.27	0.24	0.27
MAX-2	0.09	0.08	0.08
KLEIN	0.1	0.12	0.14

significant reduction in the number of documents. As anticipated, MAX-2 and KLEIN result in higher reductions, since they generally produce smaller intervals. Further, only a small percentage (as low as 8%) of the timeline is covered by bursty intervals. Nonetheless, as illustrated by our previous experiments, these intervals provide all the information that our search framework needs to effectively evaluate queries.

7. CONCLUSIONS

In this paper we explored how term burstiness can be used to enhance the search process for large document sequences. We provided a formal definition of burstiness and proposed efficient, parameter-free algorithms for the identification of bursty intervals for any given term. The main contribution of our work is an efficient search framework that considers term burstiness in the indexing and ranking process. We describe two alternative versions of our framework, and discuss how they can be useful to a user querying a document sequence. Finally, we thoroughly evaluated our approaches on a new dataset, in the context of different scenarios.

8. ACKNOWLEDGMENTS

This work was supported by NSF IIS-0534781, NSF 0803410, the ONR N00014-07-C-0311 Aware, the Health-e-Child, and the SemsorGrid4Env projects.

9. REFERENCES

[1] D. Agarwal, J. M. Phillips, and S. Venkatasubramanian. The hunting of the bump: on maximizing statistical discrepancy. In *SODA '06*, pages 1137–1146, New York.

[2] N. Bansal and N. Koudas. Blogscope: a system for online analysis of high volume text streams. In *VLDB '07*.

[3] N. Bansal and N. Koudas. BlogScope: spatio-temporal analysis of the blogosphere. In *WWW '07*.

[4] B. Chazelle. *The discrepancy method: randomness and complexity*. Cambridge University Press, NY, 2000.

[5] C.-H. Cheng, K.-Y. Chen, W.-C. Tien, and K.-M. Chao. Improved algorithmms for the k maximum-sums problems. *Theor. Comput. Sci.*, 362(1):162–170, 2006.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[7] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Comput. Syst. Sci.*, 52(3):453–470, 1996.

[8] D. Dobkin and D. Eppstein. Computing the discrepancy. In *SCG '93*, pages 47–52, New York, NY, USA, 1993. ACM.

[9] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *PODS '01*.

[10] G. P. C. Fung, J. X. Yu, P. S. Yu, and H. Lu. Parameter free bursty events detection in text streams. In *VLDB '05*.

[11] Q. He, K. Chang, and E.-P. Lim. Analyzing feature trajectories for event detection. In *SIGIR '07*.

[12] Q. He, K. Chang, E.-P. Lim, and J. Zhang. Bursty feature representation for clustering text streams. In *SIAM '07*.

[13] J. Kleinberg. Bursty and hierarchical structure in streams. In *KDD '02*, pages 91–101, New York, USA.

[14] R. Kumar, J. Novak, P. Raghavan, and A. Tomkins. On the bursty evolution of blogspace. In *WWW '03*.

[15] National Digital Newspaper Program (NDNP), http://www.loc.gov/ndnp.

[16] Qi He and Kuiyu Chang and Ee-Peng Lim. Using burstiness to improve clustering of topics in news streams. In *ICDM '07*, Washington, DC, USA, 2007.

[17] W. L. Ruzzo and M. Tompa. A linear time algorithm for finding all maximal scoring subsequences. In *ISMB 1999*.

[18] M. Vlachos, C. Meek, Z. Vagena, and D. Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *SIGMOD '04*, pages 131–142, New York.

[19] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *KDD '03*, pages 336–345, New York.

Table 3: Major Events List

ID	Description	Date	Query
1	Mormon Leader B. H. Roberts is refused a seat in the US Congress due to his polygamy.	Jan 17 1900	polygamy
2	The German passenger ship Saale, owned by the North German Lloyd, catches fire at the docks in Hoboken, killing 326 people.	Jun 30 1900	saale
3	King Umberto I of Italy is assassinated by Italian-born anarchist Gaetano Bresci.	Jul 29 1900	king assassination
4	A powerful hurricane hits Galveston, Texas killing about 8,000.	Sep 8 1900	texas disaster
5	Queen Victoria dies at the age of 81.	22 Jan 1901	victoria death
6	The Great Fire of 1901 begins in Jacksonville, FL .	May 3 1901	jacksonville
7	Serbian King Alexander Obrenovic and Queen Draga are assassinated.	Jun 11 1903	serbian kings
8	Pope Leo XIII dies. He is later succeeded by Pope Pius X.	July 20 1903	pope death
9	A fire at the Iroquois Theater in Chicago kills 600.	Dec 30 1903	theater disaster
10	The Great Baltimore Fire in Maryland destroys over 1,500 buildings in 30 hours.	Feb 7 1904	baltimore
11	Battle of Guru: British troops under Colonel Francis Younghusband battle with Tibetan Troops, marking the beginning of the British Expedition to Tibet	Mar 31 1904	guru
12	A fire aboard the steamboat General Slocum in New York City's East River kills 1,021.	Jun 15 1904	steamboat disaster
13	Eugen Schauman assassinates Nikolai Bobrikov, Governor-General of Finland	Jun 16 1904	finland governor
14	King Carlos I of Portugal and Prince Luiz are shot dead in Lisbon.	Feb 1 1908	carlos luiz
15	Louis Bleriot is the first man to fly across the English Channel in an aircraft.	Jul 25 1909	english channel
16	A 7.0 Richter scale earthquake destroys Messina, Sicily and rocks Calabria, killing over 75,000 people and living thousands homeless.	Dec 28 1909	italy homeless

Table 4: Predicted Intervals for Major Events

Event ID	Actual Date	MAX-1	MAX-2	KLEIN
1	Jan 17 1900	5 Jan - 3 Apr (1900)	5 Jan - 26 Jan (1900)	5 Jan - 23 Jan (1900)
2	June 30 1900	25 Jan - 12 Jul (1900)	1 Jul - 12 Jul (1900)	1 Jul - 12 Jul (1900)
3	Jul 29 1900	15 Jul - 19 Aug (1900)	30 Jul - 5 Aug (1900)	-
4	Sep 8 1900	3 Sep - 10 Mar (1900/01)	9 Sep - 6 Oct (1900)	10 Sep - 14 Sep (1900)
5	Jan 22 1901	5 Oct - 17 Mar (1900/01)	28 Dec - 8 Feb (1900/01)	-
6	May 3 1901	24 Apr - 29 Jul (1901)	27 Apr - 20 May (1901)	4 May - 23 May (1901)
7	Jun 11 1903	11 Jun - 25 Oct (1903)	12 Jun - 25 Jun (1903)	12 Jun - 19 Jun (1903)
8	July 20 1903	5 Jul - 4 Jan (1903/04)	7 Jul - 22 Jul (1903)	20 Jul - 22 Jul (1903)
9	Dec 30 1903	22 Dec - 20 Aug (1903/04)	31 Dec - 26 Jan (1903/04)	31 Dec - 17 Jan (1903/04)
10	February 7 1904	19 Jul - 20 Mar (1903/04)	5 Feb - 20 Feb (1904)	8 Feb - 20 Feb (1904)
11	Mar 31 1904	1 Apr - 6 Apr (1904)	3 Apr - 5 Apr (1904)	1 Apr - 6 Apr (1904)
12	Jun 15 1904	14 May - 4 Sep 1904 (1904)	16 Jun - 20 Jun (1904)	-
13	Jun 16 1904	20 Mar - 30 Oct (1904)	17 Jun - 31 Jul (1904)	20 Jun - 23 Jun (1904)
14	Feb 1 1908	2 Feb - 20 Feb (1908)	2 Feb - 11 Feb (1908)	-
15	Jul 25 1909	5 Mar - 10 Nov (1909)	19 Jun - 8 Aug (1909)	18 Jul - 27 Jul (1909)
16	Dec 28 1909	28 Nov - 28 Oct (1908/09)	26 Dec - 18 Jan (1908/09)	-

Table 5: Frequent keywords extracted from the top-k documents for each query

ID	Cloud
1	January state washington roberts present practice utah member house law
2	burn german york lloyd bodies fires recovered north river hoboken
3	humbert july state anarchist italiy unit rome bressi general police
4	city people galveston state sufferers received great money reported relief
5	state present great king queen people passed service palace city
6	city people fire state florida unite part sufferers generous report
7	belgrade queen peter officers alexander minister murder governor assassin palace
8	july leo rome cardinal holy pontiff church vatican present great
9	chicago fire place city building iroquois work time manager people
10	February state city general york fire company aid busy american
11	tibet british fight chinese mission general hostile colonel petersburg influence
12	bodies general york slocum fire boat hoboken police dead
13	general bobrikoff russia petersburg assassin government author people condition land
14	queen king crown assassination portugal prince oporto royal brother lisbon
15	flight july miles aviator attempt cross return bleriot condition machine
16	italian earthquake people city aid messina sufferers ruins relief stricken