Indexes Analysis for Matching Subscriptions in RSS feeds

Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cédric Du Mouza, Michel Scholl, Nicolas Travers

▶ **To cite this version:**

Zeinab Hmedeh, Harris Kourdounakis, Vassilis Christophides, Cédric Du Mouza, Michel Scholl, et al.. Indexes Analysis for Matching Subscriptions in RSS feeds. Bases de Données Avancées, BDA'2012, Oct 2012, Clermont-Ferrand, France. pp.1-20. hal-00737232

HAL Id: hal-00737232
https://hal.science/hal-00737232v1

Submitted on 1 Oct 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Indexes Analysis for Matching Subscriptions in RSS feeds

Zeinab Hmedeh[1] Harris Kourdounakis[2] Vassilis Christophides[2]
Cedric du Mouza[1] Michel Scholl[1] * Nicolas Travers[1]

[1] CEDRIC Laboratory - CNAM - Paris, France
firstname.lastname@cnam.fr
[2] FORTH/ICS and Univ. of Crete - Heraklion, Greece
(christop,kourdoun)@ics.forth.gr

Abstract

The explosion of published information on the Web leads to the emergence of a Web syndication paradigm, which transforms the passive reader into an active information collector. Information consumers subscribe to RSS/Atom feeds and are notified whenever a piece of news (item) is published. The success of this Web syndication now offered on Web sites, blogs, and social media, however raises scalability issues. There is a vital need for efficient real-time filtering methods across feeds, to allow users to follow effectively personally interesting information. We investigate in this paper three indexing techniques for users' subscriptions based on inverted lists or on an ordered trie. We present analytical models for memory requirements and matching time and we conduct a thorough experimental evaluation to exhibit the impact of critical workload parameters on these structures.

Keywords: Pub/sub, Subscription Indexing, Web Syndication

1 Introduction

Web 2.0 technologies have transformed the Web from a publishing-only environment into a vibrant information place where yesterday's passive readers have become active information collectors and content generators themselves. In this context, Web syndication formats such as RSS or Atom emerge as a popular mean for timely delivery of frequently updated Web content. According to these formats, information publishers provide brief summaries (textual snippets) of the content they deliver on the Web [8], called *information items*, while information consumers subscribe to a number of RSS/Atom *feeds* (i.e., channels) and get informed about newly published items. Today, almost every personal weblog, news portal, discussion forum, user group or social media (e.g., Facebook, Twitter, Flickr) on the Web employs RSS/Atom feeds. Given that the amount and diversity of the information generated on a daily basis in Web 2.0 is unprecedented, there is a vital need for efficient *real-time filtering methods across feeds* which allow users to effectively follow personally interesting information. For these reasons, we advocate a *content-based Publish/Subscribe* paradigm for Web 2.0 syndication in which information consumers are decoupled (in both *space* and *time*) from information providers and they can express their interest to specific information items using content-based subscriptions. *Keyword-based subscriptions* will be matched on the fly against the content of incoming items originating from different feeds.

*Michel Scholl passed away on Nov 15th, 2011, too early after a short battle with cancer. We would like to express our gratitude for everything Michel offered us on a personal and professional level during our long-lasting collaboration.

To efficiently check whether all keywords of a subscription also appear in an incoming item (*i.e.*, *broad match* semantics) we need to index the subscriptions. *Count-based* (CI) and *Tree-based* (TI) are two main indexing schemes proposed in the literature for counting explicitly vs implicitly the number of contained keywords. The majority of related data structures [15, 7, 1] cannot be employed for conjunctions of keywords (rather than attribute-value pairs) due to the space high-dimensionality. In this paper, we are interested in efficient implementations of both indexing schemes using *Inverted Lists* (IL) [23] for CI and a variant for distinct terms of *Ordered Tries* (OT) [11] for TI and study their behavior for critical parameters of realistic Web syndication workloads. Although these data structures have been employed to evaluate broad match queries in the context of selective information dissemination [21] and sponsored search [12] or for mining frequent Item sets [3, 14], their memory and matching time requirements appear to be quite different in our setting. This is due to the peculiarities of Web syndication systems which are characterized [8] (a) by information items of average length (25-36 distinct terms) which are greater than advertisement bids (4-5 terms [12]) and smaller than documents of Web collections (12K terms [21]) (b) by very large vocabularies of terms (up to 1.5M terms) Note also, that due to broad match semantics Information Retrieval techniques for optimizing ILs (e.g., early pruning [23]) are not suited in our setting.

A detailed analysis of Trie structures has not been conducted in the past while the Ordered Trie usage has been discouraged in Pub/Sub systems due to the prohibiting performance exhibited in other application areas studied in related work (e.g. document filtering in [22]). In our work we are going one step forward in identifying real setting parameters under which Trie structures become competitive. In a nutshell, the main contributions of this work are:

(1) In Section 2 we consider three index structures implementing different counting techniques for pruning as early as possible non matching subscriptions to an incoming item. The first two implement the CI scheme and rely on an *Inverted List* (IL) of terms to the subscriptions that contain them. The *Count-based Inverted List* (CIL) variant stores each subscription into the ILs of all the terms it contains while the *Ranked-key Inverted List* (RIL) only once in the IL of its least frequent term. We finally consider an *Ordered Trie* (OT) of distinct terms implementing TI with factorization of common subscription prefixes while and a variation that compacts paths of unary nodes called *POT*.

(2) Section 3 provides a detailed probabilistic analysis of the size and number of nodes visited during matching of the three indices which takes into account the term occurrence distribution and the distribution of subscription lengths. To the best of our knowledge an analysis of *OT* matching time has not been previously reported in the literature.

(3) In Section 4 we conduct a thorough experimental evaluation of *CIL*, *RIL* and *POT* [1] using generated subscriptions of terms appearing in a real RSS/Atom testbed[2] of items [8]. To the best of our knowledge, this is the first study investigating (a) how critical workload parameters, such as terms distribution, size of the vocabulary and lengths of subscriptions affect the morphology of OT, *i.e.*, the level of achieved factorization and (b) their scalability and performance in realistic settings (*e.g.* for 100M of subscriptions with 100K of distinct terms and real distribution of terms in items).

Related work is presented in Section 5 while a brief summary along with plans for future work are given in Section 6.

[1] The regular OT scheme called ROT turned out to be outperformed in terms of memory and matching time requirements

[2] Available on line at deptmedia.cnam.fr/∼traversn/roses

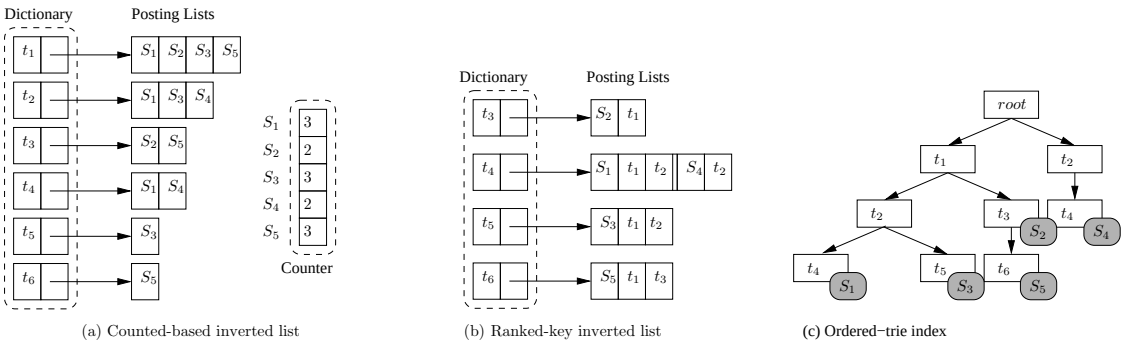

Figure 1: Subscription Indexes

2 Subscription Indexes

In the pub/sub paradigm for web syndication, users submit *long lasting* (continuous) queries under the form of keyword-based subscriptions. Whenever a news item is published, it gets evaluated against the set of subscriptions submitted to the system and for every matching subscription the corresponding subscriber is notified. The set of stored subscriptions is denoted by $\mathcal{S}$ and their total number by $|\mathcal{S}|$. Each subscription $s \in \mathcal{S}$ includes a set of (distinct) terms from a vocabulary $\mathcal{V}_S = \{t_1, t_2, ..., t_n\}$. The length of s, denoted by $|s|$, is the total number of (distinct) terms it contains. $\mathcal{I} = [I_1, I_2, ...I_m]$ denotes the stream of incoming news items. News item $I \in \mathcal{I}$ is also formed by a set of terms[3] ($I \subseteq \mathcal{V}_I$, with $\mathcal{V}_I$ the vocabulary of items). Like [21] we make the common assumption that $\mathcal{V}_S \subseteq \mathcal{V}_I$. However it is worth noting that in reality, $\mathcal{V}_S$ may diverge from $\mathcal{V}_I$ significantly. In this context, a match occurs if and only if all of the terms (keywords) of a subscription s are also present in a news item I (*i.e.*, *broad match semantics*).

Subscription (Terms)	$S_1(t_1 \wedge t_2 \wedge t_4)$	$S_2(t_1 \wedge t_3)$	$S_3(t_1 \wedge t_2 \wedge t_5)$
	$S_4(t_2 \wedge t_4)$	$S_5(t_1 \wedge t_3 \wedge t_6)$	

Table 1: Example of keyword based subscriptions

Consider the set of subscriptions $\mathcal{S}$ illustrated in Table 1. Matching item $I = \{t_2, t_4\}$ against $\mathcal{S}$ will result in the set of matched subscriptions $\mathcal{S}_M = \{S_4\}$ since t_2 and t_4 of S_4 are contained in I. A naive matching approach consists in testing whether the terms of every subscription are contained in the incoming news item. Clearly, this naive solution does not scale to millions of subscriptions. For this reason, index structures have to be found which allow to prune as early as possible non matching subscriptions. A widely used structure is the inverted list (IL) which maintains an inverse mapping from terms t_j to the subscriptions s that contain them. It essentially confines the original search space only to subscriptions containing *at least a term* present within the item being matched. In particular, two variants of IL namely the *Count-based Inverted List* and the *Ranked-key Inverted List* are studied in this paper. Additionally an *Ordered Trie* structure is considered which exploits the *term subset relations between subscriptions*.

Count-based Inverted List

Count-based Inverted List (*CIL*) is essentially a *mapping dictionary* whose key is a term $t_j \in \mathcal{V}_S$ and value the corresponding *posting list* $Posting(t_j)$, *i.e.* the set of subscriptions that contain

[3]Duplicate terms in items are eliminated after sorting during items preprocessing.

Algorithm 1: $CIL\_MATCH(I)$

Require: An item I.

1: $Counter\_copy \leftarrow$ copy of $Counter$
2: **for all** terms $t_j \in I$ **do**
3: $PostSet \leftarrow Posting(t_j)$
4: **for all** s in $PostSet$ **do**
5: $Counter\_copy[s] \leftarrow Counter\_copy[s] - 1$
6: **if** $Counter\_copy[s] = 0$ **then**
7: $S_M \leftarrow S_M \cup \{s\}$
8: **end if**
9: **end for**
10: **end for**

Algorithm 2: $RIL\_MATCH(I)$

Require: An item I.

1: $sorted\_terms \leftarrow sort(I)$
2: **while** $sorted\_terms \neq \{\}$ **do**
3: $t_j \leftarrow less\_frequent(sorted\_terms)$
4: $sorted\_terms \leftarrow sorted\_terms - t_j$
5: $PostSet \leftarrow Posting(t_j)$
6: **for all** s in $PostSet$ **do**
7: **if** $s.remaining \subseteq sorted\_terms$ **then**
8: $S_M \leftarrow S_M \cup \{s\}$
9: **end if**
10: **end for**
11: **end while**

t_j. Furthermore, to implement broad match semantics, an additional structure has to be maintained: a counter per subscription keeps track of the number of remaining terms to be matched for a given subscription. The structure that maps every subscription $s \in \mathcal{S}$ to the number of remaining terms to be tested before reporting a matching is denoted by *Counter*.

Figure 1(a) depicts the *CIL* index for the example of Table 1. The posting list associated with t_2 is $Posting(t_2) = \{S_1, S_3, S_4\}$. Initially, $Counter = \{S_1 : 3, S_2 : 2, S_3 : 3, S_4 : 2, S_5 : 3\}$. Consider an incoming news item $I = \{t_2, t_4\}$. The subscription elements of $Posting(t_2)$ are first accessed and their corresponding counters are decremented, and thus $Counter = \{\mathbf{S_1 : 2}, S_2 : 2, \mathbf{S_3 : 2}, \mathbf{S_4 : 1}, S_5 : 3\}$. Finally, after processing t_4, $Counter = \{\mathbf{S_1 : 1}, S_2 : 2, S_3 : 2, \mathbf{S_4 : 0}, S_5 : 3\}$ and a matching is reported for subscription S_4 since its counter becomes 0.

CIL - Construction When a new subscription s is posted to the system, a new element labeled with its identifier s is added to the posting lists of *all* its terms. Additionally, a new entry is inserted into *Counter* with the total number of distinct terms of s ($|s|$).

CIL - Matching The matching process for an item I is given in Algorithm 1. Initialization consists of an exact copy of *Counter* into *Counter_copy* (1.1). For every term $t_j \in I$, $Posting(t_j)$ is accessed (1.3). For each subscription s in $Posting(t_j)$, the corresponding subscription value in *Counter_copy* is decremented (1.5). Whenever a counter reaches zero, a matching is reported.

Ranked-key Inverted List

In contrast to *CIL*, in the *Ranked-key Inverted List* (*RIL*) a subscription is added only to the posting list of the least frequent among its terms. This term is called *key* of the subscription. Besides the subscription identifier, the elements of a posting list include the set of the remaining subscription terms. This variant obviates the need for an explicit *Counter* structure while accessing only the posting list of the most discriminating term of a subscription. More precisely, for every term of an incoming item the corresponding posting list is accessed and for each of its subscriptions it is checked whether it contains the remaining item terms. Clearly the posting lists of the frequent terms is reduced in comparison with *CIL* and subscriptions are now distributed over a large number of posting lists of medium-frequent terms.

Figure 1(b) depicts the *RIL* index for the example of Table 1 where the term rank in frequency distribution is given by the term subscript (t_1 has the higher rank in $\mathcal{V}_S$). Not all the terms have entries in the dictionary. For instance, t_1 and t_2 do not appear as least frequent

terms in any of the subscriptions in $\mathcal{S}$. $Posting(t_4)$ has two elements (S_1 and S_4) for which the remaining (more frequent) terms to be checked are stored. With item $I = \{t_2, t_4\}$ we start by checking for subscriptions in $Posting(t_4)$; for S_1: I does not contain t_1 thus S_1 is not satisfied by I. Then in the next subscription S_4, t_2 appears in $I' = I - t_4 = \{t_2\}$ and there are no more terms in S_4 so it is reported as matching I.

RIL - Construction When a new subscription s is posted to the system, the posting list where it should be added is located. This implies to sort the terms of s by their rank. Then an entry labeled with the subscription identifier s is added to the posting list of the least frequent term followed by the remaining terms.

RIL - Matching Matching an item I is given in Algorithm 2. When item I arrives its terms are sorted by their rank (line 1). In the rest of the paper, unless otherwise specified, term subscripts are used to denote ordering. Then, the posting lists $Posting(t_j)$ of the least frequent terms are iteratively accessed (lines 2-4). For every subscription element of $Posting(t_j)$, it is checked whether its remaining terms also appear in I (lines 6-7).

Regular Ordered Trie

As an alternative to *IL* indexes, an *OT* index is capable of exploiting the *term subset relations between subscriptions* in order to build a hierarchical (as opposed to a flat) search space for sets of terms taking advantage of common prefixes of terms in subscriptions (i.e. *factorization*). A Trie node represents a term and a subscription is stored at node n of the Trie iff its terms are found in the path from the root to node n. Then, two subscription paths sharing a subset of k nodes can be merged in a single subpath of length k (i.e. common prefix), followed by two distinct paths representing the remaining subscription terms.

Clearly, in this structure, subscriptions are stored only once and there is no need anymore for an explicit *Counter* structure. Compared to the commonly used Trie structures for storing sentences on a given vocabulary [11] two features characterize the Ordered Trie: (*i*) there is *no repetition of terms* in any sentence (*i.e.* a subscription is a *set* of terms; (*ii*) terms in the subscriptions and therefore in the Trie *are totally ordered*. This total order could be random, follow the ranking of the terms occurrence distribution in subscription/new items whenever available, etc. This structure referred to as *Regular Ordered Trie* (ROT) has been investigated in a different setting [21], as discussed in Section 5. It was also used more recently in data mining [4, 14].

Figure 1(c) depicts the ROT index for the example subscriptions of Table 1 where the term rank is given by the term subscript (t_1 has the highest rank in $\mathcal{V}_S$). Factorization leads to a single node t_1 for all subscriptions that share this term. Consider now an incoming news item $I = \{t_2, t_4\}$ (whose terms are already sorted). Initially, term t_2 is searched as child of the root. Since such a node exists, navigation continues by looking for a t_4 child node. Since such a node also exists, the stored subscription S_4 is reported as matching. Finally, the last term t_4 of I is processed and since no such path from the root exists, the matching concludes. Collapsing single paths in the trie to single nodes should not only reduce the number of nodes therefore the memory occupied by the index but also should accelerate matching. A variant of a *Patricia Ordered Trie* (POT) obeys this principle. The single paths corresponding to multiple nodes are compacted into one single compact node. Each compact node is labeled with the set of terms corresponding to the nodes in the path. For instance, nodes t_2 and t_4 in Figure 1(c) are merged in a single node labeled with $t_2.t_4$ in POT.

ROT - Construction Initially, the terms of a new subscription s are sorted according to their ranking order. Then the path corresponding to the first term of the ordered set of terms is followed from the root. This procedure is repeated for every term $t_j \in s$. If a particular path does not exist, then a new node labeled with the term under consideration is created and inserted into the Trie structure. The node at which the top down traversal concludes, after consuming the whole set of terms, is where s is finally stored.

ROT - Matching The matching process for an item I is given in Algorithm 3. When a news item I arrives its terms are also sorted. The paths corresponding to all the terms of I, whose ranks are superior to that of the term assigned to the currently considered node are followed (lines 5-10). For every node visited the subscriptions stored at that specific node are reported as matching (lines 2-4).

Algorithm 3: $TRIE\_MATCH(TNode, I)$

Require: $TNode$: the current trie node, I an item

```
1: sorted_terms ← sort(I)
2: if TNode contains subscriptions then
3:    S_M ← S_M ∪ {S_i | S_i ∈ TNode}
4: end if
5: sorted_terms ← sorted_terms − Term(TNode)
6: for all term t_j ∈ sorted_terms do
7:    childNode ← get child for term t_j
8:    if childNode ≠ NULL then
9:       TRIE_MATCH(childNode, sorted_terms − {t_j})
10:   end if
11: end for
```

An index dependent definition of nodes

The three indexes presented previously essentially implement a relation R of vocabulary terms within subscriptions. The specific technique employed for counting the matching terms in subscriptions to satisfy broad match semantics identifies each index. As a consequence, the node definition, inherent to each data structure, is also different. In *CIL*, an index node is a tuple (t_j, s) expressing that subscription s contains term t_j. If s has k terms, k nodes (so k R-tuples) are needed to represent s. In *RIL*, an index node is a nested tuple $(t_k, \{(s, \{t_j\})\})$ where t_k is the key term, s the corresponding subscription and $\{t_j\}$ the remaining terms to be checked. In *ROT*, an index node is a nested tuple $(\{t_j\}, s)$ where $\{t_j\}$ is the set of terms encountered in the path from root to this node which form the actual content of subscriptions in s. When subscriptions share the same prefix, it is expected to create less index nodes in Trie-based indexes than in Inverted List-based indexes. The rationale in studying the index behavior in terms of abstract nodes is to understand the impact in the morphology of the three indexes (and thus of the search space) of critical parameters of web syndication systems (*i.e.*, the distribution of term occurrences as well as the distribution of subscription lengths).

3 Analytical models

This section is devoted to analytical modeling for predicting the number of nodes of the CIL, RIL and ROT structures presented in Section 2 as well as to predict the number of visited nodes upon matching. The set of parameters and notations that affect construction time, memory requirements and matching time are summarized in Table 2.

$\|\mathcal{S}\|$	total number of subscriptions
$\|\mathcal{V}_I\|$, $\|\mathcal{V}_S\|$	vocabulary size of items and subscriptions
$\|s\|_{avg}$, $\|s\|_{max}$	resp. average and max. subscription length
$\|I\|$	average news item length
$P(t_j)$	frequencies distribution of terms in $\mathcal{V}_I$
$\theta(k)$	proba. for a subscription to have a length k
σ_i	proba. to have a term with a rank $\leq i$
$w(c), w(v)$	size of Counter/Dictionary entry
$w(p), w(n)$	size of Subscription Posting entry/Trie Node

Table 2: Parameters that characterize the workload

3.1 Building time

The average length of a subscriptions is: $|s|_{avg} = \Sigma_{k=1}^{|s|_{max}} \theta(k) \times k$, with $k \in [1, |s|_{max}]$. It is easy to check that the time required to insert a subscription into *CIL* is $O(|s|_{avg})$ (we insert $|s|_{avg}$ new postings and one new counter). *RIL* requires to sort the terms before insertion to determine the less frequent one. Consequently an insertion is performed in $O(|s|_{avg} \times \log |s|_{avg})$. The time required to insert a subscription into *ROT* is $O(|s|_{avg} \times \log |s|_{avg})$. This is due to the fact that using a hash-based implementation of Trie nodes, the time required to sort the terms of subscriptions dominates. Observe that indexing time for all structures is independent of the total number of stored subscriptions $|\mathcal{S}|$.

3.2 Memory requirement

Let $P(t_j)$, denote the frequency of occurrences of term $t_j \in \mathcal{V}_I$. It is assumed that the choice of $t_j \in s$ is independent of the choice of any other term $t_k \in s$. The probabilities that t_j is one of the terms of a subscription s, denoted by $Pr(t_j \in s)$ and that t_j is one of the terms of at least one subscription in $\mathcal{S}$ denoted by $Pr(t_j \in \mathcal{S})$ are:

$$Pr(t_j \in s) = 1 - Pr(t_j \notin s) = 1 - \sum_{k=1}^{|s|_{max}} \theta(k) \times (1 - P(t_j))^k \tag{1}$$

$$\begin{aligned} Pr(t_j \in \mathcal{S}) &= 1 - Pr(t_j \notin \mathcal{S}) = 1 - \prod_{i=1}^{|\mathcal{S}|} (1 - Pr(t_j \in s)) \\ &\overset{(1)}{=} 1 - (\sum_{k=1}^{|s|_{max}} \theta(k) \times (1 - P(t_j))^k)^{|\mathcal{S}|} \end{aligned}$$

Then the number of terms in the vocabulary of subscriptions $\mathcal{V}_S$ is equal to:

$$|\mathcal{V}_S| = \Sigma_{j=1}^{|\mathcal{V}_I|} Pr(t_j \in \mathcal{S})$$

CIL Memory Requirement Recall that the count based index is composed of two structures, the *Counter* and the *inverted lists* that are themselves further decomposed into the *dictionary* and the *subscription postings*. Thus the overall memory required by the index is:

$$Size(CIL) = Size(Dictionary) + Size(Postings) + Size(Counter)$$

The memory required by the *Counter* is equal to:

$$Size(Counter) = |\mathcal{S}| \times w(c) \tag{2}$$

We assume a low collision rate. Then the memory occupied by the *Dictionary* is equal to:

$$Size(Dictionary) \quad = \quad |\mathcal{V}_S| \times w(v) \tag{3}$$

Finally, since each term of any subscription leads to an entry in the corresponding posting list the expected total number of entries in the posting lists is equal to $|\mathcal{S}| \times |s|_{avg}$ and the size of the posting lists is equal to:

$$Size(Postings) \quad = \quad |\mathcal{S}| \times |s|_{avg} \times w(p) \tag{4}$$

In addition, for computing the matching time we need to know the size $Size(Postings(t_j))$ of the posting list of a term t_j:

$$Size(Posting(t_j)) = \frac{Pr(t_j \in S)}{\sum_{i=1}^{|\mathcal{V}_S|} Pr(t_i \in S)} \times |\mathcal{S}| \times |s|_{avg} \tag{5}$$

where $\Pr(t_j \in S) / \sum_{k=1}^{|\mathcal{V}_S|} Pr(t_k \in S)$ is the normalized frequency of t_j in the postings. Thus the space consumed by the index is:

$$Size(CIL) \overset{(2,3,4)}{=} |\mathcal{S}| \times w(c) + |\mathcal{V}_S| \times w(v) + |\mathcal{S}| \times |s|_{avg} \times w(p)$$

RIL Memory Requirements *RIL* is decomposed into the *Dictionary* which stores the set of terms that are the less frequent in at least one $s \in \mathcal{S}$, and the *postings lists* which store the set of subscriptions. Since virtual nodes in *CIL* and *RIL* consist in subscription or term ids, we assume that both structures share the same size $w(p)$ for posting entry. Thus $Size(Postings)$ is computed as for *CIL* (equation 4). $Size(Dictionary)$ is however less than the *CIL*'s one.

A subscription s belongs to a posting list $Postings(t_j)$ iff $t_j \in s$ and there is no term $t_i \in s$ with $i < j$. Thus $Post(s, t_j)$, the probability that a subscription s belongs to $Posting(t_j)$ is:

$$Post(s, t_j) = \sum_{k=1}^{|s|_{max}} \theta(k) \times k \times P(t_j) \times (\sigma_{j-1})^{k-1}$$

where $\sigma_j = \Sigma_{i=1}^{j} P(t_i)$ is the probability to have a term with a rank higher than i. Indeed, if the subscription length is k, then there are k ways of choosing t_j, the remaining terms being chosen among the terms with higher rank. The probability to have a term t_j in the *Dictionary* is the probability to have at least one subscription in his $Posting(t_j)$:

$$Pr(t_j \in Dictionary) = 1 - (1 - Post(s, t_j))^{|\mathcal{S}|}$$

So the size of the *Dictionary* is:

$$Size(Dictionary) = \sum_{j=1}^{|\mathcal{V}_I|} (1 - (1 - Post(s, t_j))^{|\mathcal{S}|}) \times w(v) \tag{6}$$

The overall memory required by the index is:

$$Size(RIL) \quad \overset{(4,6)}{=} \sum_{j=1}^{|\mathcal{V}_I|} 1 - (1 - Post(s, t_j))^{|\mathcal{S}|} \times w(v) + |\mathcal{S}| \times |s|_{avg} \times w(p)$$

Finally, when considering the length of each subscription to be stored in the posting list, we deduce:

$$Posting(t_j) = |\mathcal{S}| \sum_{k=1}^{|s|_{max}} k \times \theta(k) \times P(t_j) \times (\sigma_{j-1})^{k-1} \quad (7)$$

ROT Memory Requirements Although analyses of the regular trie can be found in particular in [6], to our knowledge, the following is the first attempt to predict the expected number of nodes of a ROT as defined in the previous section. The analysis takes into account any term distribution and any distribution of the subscriptions length. However it does not provide a closed form. Therefore its applicability is limited to vocabularies with size $|\mathcal{V}_I| < 100$ and short subscriptions with length ($|s| < 12$). It turns out that this is the case when the vocabulary is restricted to the terms of an item. We show that the analysis is useful for computing the expected time to match an item against a set of subscriptions. Let $\mathcal{P}$ be a path from the root to a node in the trie representing term (labeled with) t_i. Its label Λ is defined as follows:

i) $\Lambda = \lambda$ is the empty path label

ii) if $\mathcal{P}$ with label Λ is a path from root to a node labeled with t_i then $\Lambda \bullet j$ is the label of a path ending at node labeled with t_j whose parent is tail($\mathcal{P}$) labeled with t_i

In the following, for short, we shall say that node j has for a prefix Λ or that j (tail($\mathcal{P}$)) has for an address $\Lambda \bullet j$ (Λ). Let s be a subscription. There is a path in the trie with label the (ordered) sequence of ranks of terms in s. It is noteworthy than there are possibly $\binom{k}{k-|P|}$ subscriptions sharing prefix Λ. Given s, $Q(\Lambda, j)$ denotes the probability that the node with address $\Lambda \bullet j$ belongs to s.

Lemma 1 $Q(\Lambda, j) = \sum_{k=|\mathcal{P}|+1}^{|s|_{max}} \theta(k) \times Q(\Lambda, j, k)$ *where* $Q(\Lambda, j, k)$ *denotes the probability that the node with address* $\Lambda \bullet j$ *belongs to a subscription with length* k *and is equal to:*

$$Q(\Lambda, j, k) = \binom{k}{k-|\mathcal{P}|} \prod_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times (1-\sigma_j)^{(k-|\mathcal{P}|-1)} \quad (8)$$

Proof The probability that term t_j belongs to s at address $\Lambda \bullet j$ is $\Pi_{m \in \mathcal{P}} P(t_m) \times P(t_j) \times (1-\sigma_j)^{(k-|\mathcal{P}|-1)}$, since the remaining $k-|\mathcal{P}|-1$ nodes of s must be drawn in $\mathcal{V}_I - \{t_1, \ldots, t_j\}$ (recall σ_j denotes the sum of probabilities of the first j terms). Since there are $\binom{k}{k-|\mathcal{P}|}$ such possible subscriptions, we obtain equation 8. □

We denote $P(\Lambda, j)$ the probability that node n with address $\Lambda \bullet j$ exists in at least one subscription. So node n is said to be occupied with probability $P(\Lambda, j)$. Then if one observes that a node with address $\Lambda \bullet j$ is not occupied if it is not occupied in all $|\mathcal{S}|$ subscriptions, we have:

$$P(\Lambda, j) = 1 - (1 - Q(\Lambda, j))^{|\mathcal{S}|} \quad (9)$$

Finally let $E(\Lambda, j)$ denote the expected number of nodes of the trie with root $\Lambda \bullet j$ where j has for a prefix Λ.

Theorem 1 $E(\Lambda, j) = \sum_{m=i+1}^{|\mathcal{V}_I|-|s|_{max}+|\mathcal{P}|} P(\Lambda \bullet j, m) \times [1 + E(\Lambda \bullet j, m)]$

with $E(\Lambda, j) = 0$ *if* $|\mathcal{P}| > s$ *or* $j \geq |\mathcal{V}_I|$

Indeed, if node with address $\Lambda \bullet i \bullet j$ is occupied the expected size of the trie with root $\Lambda \bullet i \bullet j$ is equal to $E(\Lambda \bullet i, j)$ Last the expected size of ROT is expressed as:

$$Size(ROT) = E(\lambda, 0) \times w(n)$$

3.3 Matching time

CIL Matching Time Requirements The time complexity of matching an item I against the set of indexed subscriptions $\mathcal{S}$ with Algorithm 1 is equal to the time needed to copy the counter, $Time(Copy\_counter)$, and the time for dealing with posting lists entries. The latter depends on the number of times the critical inner loop (line 4-9) is executed, *i.e.*, the sum of the sizes of all of the posting lists corresponding to terms t_j in item I. The constant time required to perform a counter decrement or test (resp. to copy an entry of the counter) is denoted by τ_{decr} (resp. τ_{copy}). Then the time needed to perform matching is:

$$\begin{aligned} TimeMatch(CIL) &= Time(Copy\_counter) + Time(Postings) \\ &= |\mathcal{S}| \times \tau_{copy} + \sum_{j=1}^{|I|} Size(Posting(t_j)) \times \tau_{decr} \\ &\overset{(5)}{=} |\mathcal{S}| \times \tau_{copy} + \left[\sum_{j=1}^{|I|} \frac{Pr(t_j \in \mathcal{S})}{\sum_{i=1}^{|\mathcal{V}_S|} Pr(t_i \in \mathcal{S})} \times |\mathcal{S}| \times |s|_{avg} \right] \times \tau_{decr} \end{aligned}$$

RIL Matching Time Requirements The time needed to match an item I depends on the number of its terms and the size of the corresponding postings lists. First we must sort its terms (set up phase) and then run through the postings lists to check the inclusion of the subscriptions. The matching cost is estimated as:

$$\begin{aligned} TimeMatch(RIL) &= Time(Sort) + Time(Postings) \\ &= |I| \times \log |I| + \sum_{j=1}^{|I|} Size(Posting(t_j)) \times \tau_{chk} \\ &\overset{(7)}{=} |I| \times \log |I| + \sum_{j=1}^{|I|} |\mathcal{S}| \sum_{k=1}^{|s|_{max}} k \times \theta(k) \times (k \times P(t_j) \times \sigma_{j-1})^{k-1} \times \tau_{chk} \end{aligned}$$

where τ_{chk} is the time needed to check term inclusion in I.

ROT Matching Time Requirements Given a set of subscriptions S whose length obeys distribution $Dist_k$ using vocabulary $\mathcal{V}$ the resulting ROT is denoted by $T(S, Dist_k, \mathcal{V})$.

Definition 1 (Restriction of a trie) *The restriction $T'(S, Dist_k, \mathcal{V}') = \Delta_{V'}(T(S, Dist_k, \mathcal{V}))$ is the subtrie of T on vocabulary $\mathcal{V}' \subset \mathcal{V}$ with same maximal depth the maximal size of a subscription $|s|_{max}$.*

By definition T' has been pruned from terms not in $\mathcal{V}'$. It contains subscriptions s whose terms are all in $\mathcal{V}'$ as well as subscriptions whose prefix is defined in $\mathcal{V}'$ but tail is out of $\mathcal{V}'$. All nodes of the prefix of the latter subscriptions are occupied as well. Theorem 2 allows to predict the number of nodes visited upon matching an item against the set of subscriptions. The set of terms of an item ($\mathcal{V}(I)$) is very small: it is on the average equal to 25 in our datasets for experiments. Then the expected number of visited nodes of the restriction of the ROT to the vocabulary of item $\mathcal{V}(I)$, $T'(S, Dist_k, \mathcal{V}(I))$ is the expected number of nodes visited for matching I. T is used as a short cut for $T(S, Dist_k, \mathcal{V})$.

Theorem 2 *The expected number of visited nodes for matching item I against the subscriptions of trie T (Algorithm 3) is equal to the expected number of nodes of the restriction $\Delta_{\mathcal{V}(I)}(T)$ where $\mathcal{V}(I)$ is vocabulary of I.*

Then the expected number of visited nodes for matching item $I = \{t_1, \ldots, t_k\}$ which has been sorted on the term ranks, is expressed, using the $P(\Lambda, j)$ formula 9, as:

$$\begin{cases} VisitedNodes = E(\lambda, 0) \qquad where \\ E(\lambda, j) = \sum_{m=j+1}^{|I|-|s|_{max}+1} P(\lambda \bullet j, m) \times (1 + E(\Lambda \bullet j, m)) \end{cases}$$

Proof Theorem 2 evaluates the average number of nodes occupied. A node n to be visited by Algorithm 3 is a node occupied. Either n contains subscriptions or the subtrie of root n possibly contains subscriptions to be checked for matching. A node has no child (line 8 of Algorithm 3) if it is of maximal depth or the child is defined on terms not in $\mathcal{V}_I$. There are no other nodes to be visited. □

Table 3 validates this model against actual measures on real items obeying the Zipf distribution and 1M subscriptions with fixed lengths. An average was taken over 5K real items chosen randomly from the English items crawled in [8]. We notice that the deviation slightly increases with subscriptions size. With large subscriptions, the depth of the Trie is higher, so the approximations of the computations at each level are propagated. For a large number of leaf nodes, the sum of approximations done becomes more significant.

$\|s\|_{max}$	real	calc. theorem 1	deviation (%)
2	426.7	426.89	+0.043
3	538.99	538.86	−0.024
4	576.52	575.77	−0.13
5	594.66	590.03	−0.77
6	600.15	595.64	−0.75
7	603.36	596.06	−1.2
8	599.8	592.17	−1.27

Table 3: Real vs estimated # visited nodes

The total matching time is equal to the time needed to sort the item's terms and the sum of the time spent on the visited nodes. The average time spent on a single nodes is τ_n.

$$TimeMatch(ROT) = Time(Sort) + Time(Nodes) = |I| \times \log|I| + E(\lambda, 0) \times \tau_n$$

4 Performance evaluation

The core implementation choices and the characteristics of the dataset of items and subscriptions are first presented, then experiments illustrate the impact of the different workload parameters on the morphology, the space requirement and the matching time.

4.1 Implementation

All indexes were implemented using the standard Java Collection Framework v1.6.0_20. All experiments were run on a 3.60 GHz quad-core processor with 16 GB in JVM memory. The *Dictionary* representing the inverted list in both *CIL* and *RIL* indexes is implemented using a static hash table inherited from Java *Hash Map*. Subscription Ids are encoded in all structures as 4-byte integers. In *CIL*, due to collisions, a dictionary entry may correspond to more than

one term. For this reason, with each entry is associated a *linked list of term nodes*, whose nodes are $(term_{id}, \uparrow subs_{list}, \uparrow next)$, where $\uparrow subs_{list}$ is a pointer to a *subscription list* implemented as an array list and $\uparrow next$ a pointer to the next node. Finally, *Counter* is a *byte* array of size $|\mathcal{S}|$ (a subscription cannot exceed 2^8 terms). *RIL* is implemented in a similar way with two noteworthy differences: (i) the lack of *Counter*, and (ii) beside subscriptions' ids the elements of subscription lists also keep track sequentially of the terms that remain to be checked per subscription.

Many different data structures (e.g. linked lists, arrays or trees) have been suggested for implementing efficiently a Trie structure. We choose a hash tree based [16] implementation of the (ROT). Every internal node includes: (i) a 4-byte integer that stores the term's rank, (ii) an array list for storing the corresponding subscriptions, and (iii) a Java *Hash Map* for storing the children nodes. A leaf node consists only of the term's id and the subscription list. We have paid particular attention in optimizing the memory requirements of internal and leaf nodes w.r.t. the number of their children. So, we distinguish nodes with or without associated subscriptions, and with zero, one or several children. Each node type is equipped with a different implementation. An internal node has a hash map to index its children, but has only when required subscription list associated with. Oppositely leaf nodes do not contain any hash map since they have no child. Since nodes may store only one subscription, we further reduce the node size by using a single subscription's id instead of an array list. Finally, in the *POT* variant, paths of unary nodes are compacted into a *compact* node labeled with the set of terms of the compacted unary ones. A 4-byte array is used to store the terms sorted by their ranks. Despite its factorization gain in terms of abstract nodes (see Section 4.3), the memory requirements of a concrete *POT* node are clearly more important than *CIL* and *RIL*: while for *CIL* and *RIL* a node occupies only $4 - byte$, it occupies on average $128 - byte$ for the *POT* (a complex Java object with a hash map, pointers, array lists, etc).

Experiments focus on the evaluation of the memory space required by each structure and the matching time. We assume that updates will be maintained in a small tail data structure, which is periodically, offline, merged and then swapped with the main read-optimized data structure.

4.2 Description of synthetic and real datasets

The experimental evaluation relies on a large-scale testbed acquired over a 8-months campaign from March to October 2010. A total number of 10,7M items was collected originating from 8K productive feeds (spanning over 2K different hosting sites) [8]. From the textual content of items a vocabulary of 1.5M distinct terms was extracted which has been used for the synthetic generation of subscriptions. More precisely, we rely on the Alias sampling method [18] to generate subscriptions whose distinct terms follow a given occurrence distribution *Dist*. Three distributions are chosen for the subscriptions in the experiments: *real* (subscriptions obey the same term distribution as the news items term distribution), *uniform*, and *inverse* (subscriptions follow the inverse term occurrence distribution of items). Generated subscriptions are characterized by three features: (a) the vocabulary size and the occurrence distribution of terms in subscriptions $\mathcal{V}_S$, (b) the total number of generated subscriptions $|\mathcal{S}|$, and (c) the subscription length k that can be constant for all subscriptions, or follow a particular distribution. When not specified, we use the length distribution of web queries reported in [2]. It is characterized among others by a maximal length equal to 12 and an average equal to 2.2.

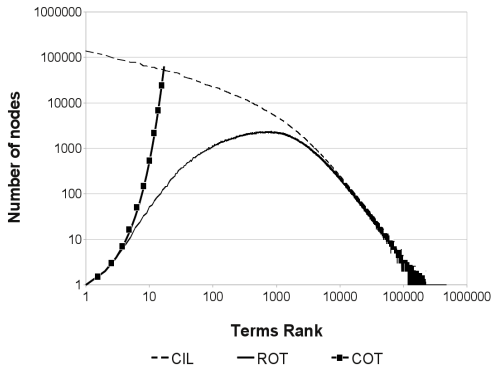

Figure 2: # nodes vs term's rank

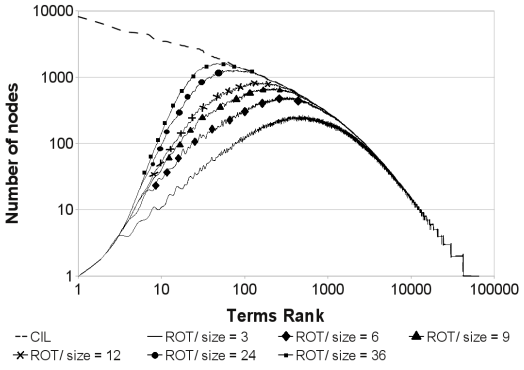

Figure 3: # nodes vs term rank for $\neq |s|$

4.3 Size and morphology of indexes

This subsection is devoted to the impact of the subscriptions length and the terms' occurrence distribution on the index size and morphology. These parameters determine the degree of factorization achieved by ROT (or POT) compared to CIL (or RIL) on common subscription prefixes as well as the rank of terms for which factorization is actually taking place. Both are essentially affecting the pruning opportunities of the indexes during matching. In order to provide a common basis for comparing the morphology of the indexes, the number of index nodes is measured (for the index dependent definition of abstract nodes see Section 2). Figure 2 depicts the number of ROT nodes created per term rank compared to CIL when indexing the same set of 10M subscriptions with a vocabulary $\mathcal{V}_S = 470K$ following the *real distribution* of terms in items [8]. Clearly, the distribution of the size of the CIL posting lists is identical to the distribution of terms' occurrences in subscriptions. We observe that the number of ROT nodes is significantly reduced not only w.r.t CIL (due to factorization of common prefixes) but also w.r.t. the *Complete Ordered Trie* (COT) of depth the maximal subscription length 12.

Table 4 highlights the gain achieved in number of nodes per term rank tr. The number of occurrences for a given term rank in the generated subscriptions as well as the number of ROT nodes that hold this term are also given in this table. As expected the gain decreases from almost 1 for rank 1 (most frequent term) to 0 for rank $470,000$ (no factorization). The closer the number of nodes in ROT to CIL, the smaller the factorization. In this experiment for all terms having a rank greater than $18,789$ the gain is equal to 0.

Rank	# occurrences	# ROT nodes	Gain (%)
1	138090	1	99.99
10	60469	52	99.91
1000	4967	2201	55.69
10000	251	218	13.15
470000	1	1	0

Table 4: Gain per term's rank

4.3.1 Impact of subscription length

We now focus on how the length of subscriptions affects nodes factorization in ROT. Sets of subscriptions are generated using the same vocabulary as previously, but with a fixed length $k \in \{3, 6, 9, 12, 24, 36\}$. To provide a common comparison ground, the total number of term occurrences T in each set of subscriptions is fixed to $T = |\mathcal{S}| \times s = 1.5M$. We observe in Figure 3

that the number of ROT nodes increases with k, *i.e.* factorization decreases with k. Indeed, the larger the subscriptions, the deeper the ROT. In this context, the probability that two subscriptions share the same terms decreases for lengthy subscriptions and the ROT behavior is close to the COT one. In other words, lengthy subscriptions imply more distinct paths. For instance, for subscription length $k > 24$ even for frequent terms (*i.e.* rank around 100) there is no gain. Oppositely, for subscription lengths $k \leq 24$ more occurrences of frequent terms are encountered in subscriptions and thus the structure fully benefits from the factorization gain. Of course for $s \leq 2$, the number of subscriptions associated with leaves becomes large and the structure degenerates to an inverted list over term combinations (rather than individual terms).

4.3.2 Impact of the term order

To estimate the impact of the total order of terms when building ROT or POT four orders are considered (a) a *frequency order* (descending order in the number of term occurrences in subscriptions), (b) a *reverse order* (ascending order in the number of occurrences), (c) *arrival order* (order of arrival of terms in the subscriptions) and (*d*) *random order* (the term rank is randomly drawn).

The term order has a limited impact on the size of ROT (see Figure 4). Compared to the frequency order, reverse, random and arrival order require respectively 23.9%, 12.1% and 1.6% more nodes. The difference between the frequency order and the arrival order is expected since more frequent terms appear statistically earlier than infrequent ones and there is an important factorization on the terms with the lowest ranks. For the reverse order, there is now a (low) factorization on the highly ordered terms but since these terms are infrequent, more subscriptions will be indexed in the sub-tries rooted at medium-frequency terms where factorization is more important. This explains why this order leads to more Trie nodes, but the increase is limited. Figure 5 shows that the term order impacts more seriously the number of nodes visited during matching: *e.g.* with the frequency order, for an incoming item 36.1% more nodes are visited than with the reverse one. The reverse order allows a better pruning than the frequency order since subscriptions featuring less frequent terms can be quickly filtered out. A similar result, in a different context, is reported in [21]. Surprisingly enough the order has an almost negligible effect both on the POT size and as on the number of nodes visited during matching. The number of Trie nodes differs by less than 1% w.r.t. the considered order, and the difference between the numbers of visited nodes does not exceed 8%. Indeed, whereas the reverse order requires more nodes for ROT, subscriptions in sub-tries rooted at the low-ordered terms are poorly factorized which leads to many unary paths that benefit from the path compaction of the POT.

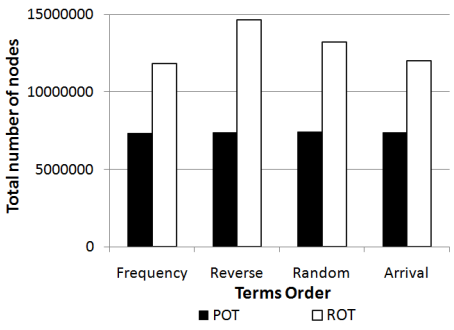

Figure 4: Impact of terms order on # created nodes

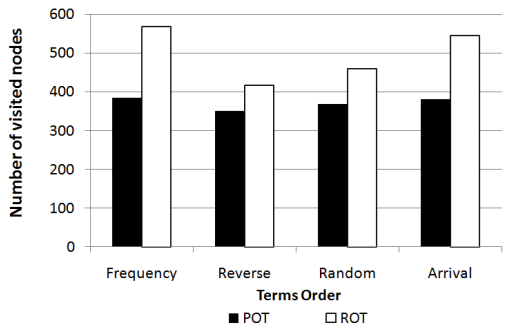

Figure 5: Impact of terms order on # visited nodes

4.3.3 Impact of the terms distribution

To investigate the impact of terms' occurrence distribution on the indexes, in addition to the *real* distribution employed previously, two more distributions are considered when generating subscriptions: (a) *uniform* and (b) *inverse* where the most frequent terms in subscriptions correspond to the least frequent terms in the items. Note that terms are ranked according to their *frequency* order in items. Figure 6 shows that the terms' occurrence distribution in subscriptions has no impact on the size of the four indexes. For *CIL* and *RIL*, the number of nodes corresponds to the number of terms in subscriptions so it is independent from their occurrence distribution. Regarding *OT*, *real* or *inverse* distributions have no impact on the index size. But in all cases, a similar factorization gain is observed as we used the same order to rank terms. As expected, an *uniform* distribution results in a significant larger *ROT* index. More combinations of terms are drawn, resulting in a more balanced Trie with more paths rooted at internal nodes, and thus less factorization opportunities. Paths compaction attenuates this effect for *POT*.

In contrast, distributions seriously impact the numbers of visited nodes during matching (Figure 7). For *CIL*, this number reaches $300,000$ nodes for *real*, but only $1,400$ for *uniform* and it drops to 8 for *inverse*. *ROT* and *POT* exhibit a similar behavior with respectively 400, 26 and 1 nodes visited on the average for the different distributions. For *CIL*, *inverse* leads to the scanning of shorter subscription lists leading to a lower matching cost. For a *uniform* distribution , subscription lists have almost the same size, while in *real*, subscription lists of frequent terms are particularly large with a high probability to be scanned for an incoming item. Regarding *ROT* and *POT*, *uniform* and *inverse* distributions yield a fast pruning, because of low factorization. For *inverse*, frequent item terms correspond to quite few subscriptions, and it is quite rare to match more than one term. For *uniform*, less subscriptions are associated with a given prefix compared to *real* and consequently less nodes have to be visited.

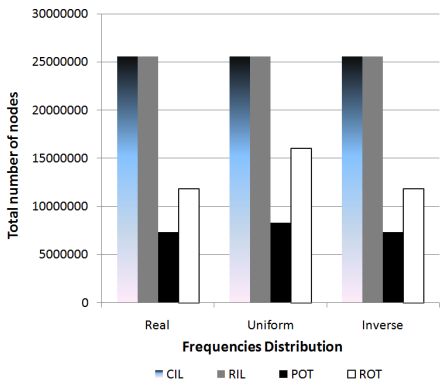

Figure 6: Impact of the distribution on index size

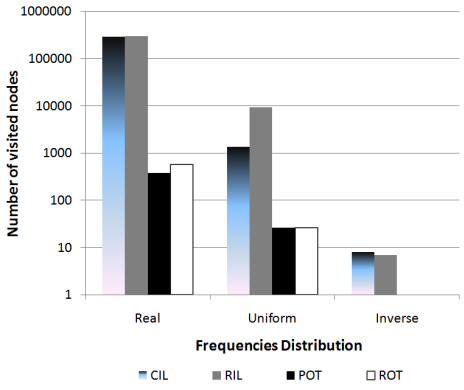

Figure 7: Impact of the distribution on matching

4.4 Scalability and performance

We turn now our attention to benchmarking memory space and matching/indexing time for *CIL*, *RIL* and *POT*. *ROT*, whose space consumption reveals quickly scalability issues, has been discarded.

4.4.1 Memory requirements

Figure 8 illustrates the evolution of the memory space for the three indexes for $10M$ of subscriptions when scaling vocabulary size $\mathcal{V}_S$. Using vocabularies of items $\mathcal{V}_I$ ranging from $100K$ to $1.5M$ terms, we generate subscriptions whose vocabularies $\mathcal{V}_S$ ranges from $87,839$ to $471,324$ terms. In general, IL indexes require a third of the memory required by *POT*. *CIL* and *RIL* space requirements slightly increase with vocabulary size, from 250 to 280MB , so around a 10% increase. This is due to the fact that a larger vocabulary leads to a larger hash table for the *Dictionary* (we fix it as half size of the vocabulary) while subscription lists are constant ($|s|_{avg} \times |\mathcal{S}|$ nodes). On the other hand, *POT* is more sensitive to the vocabulary size, since its space requirement grows from 710 to 925MB, so a 30% increase. This is due to the appearance of more terms combinations in subscriptions so more paths in the Trie and less factorization opportunities.

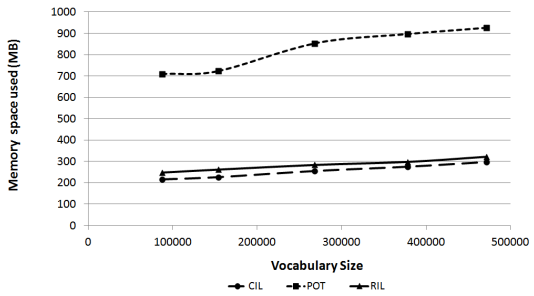

Figure 8: Memory footprint by scaling $|\mathcal{V}_S|$

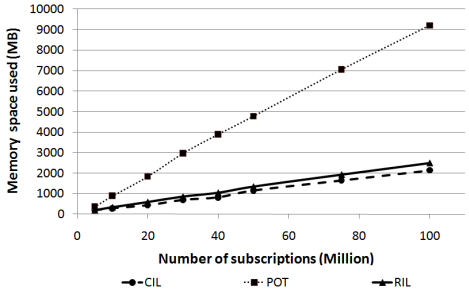

Figure 9: Memory footprint by scaling $|\mathcal{S}|$

Figure 9 illustrates the memory space consumed by the three indexes for a vocabulary $\mathcal{V}_I$ (resp. $\mathcal{V}_S$) of $1.5M$ (resp. $1.2M$) terms when scaling the number of subscriptions from $5M$ to $100M$. As expected, the memory space consumed by *CIL* and *RIL* increases linearly with the number of subscriptions. Since the *Dictionary* size is fixed to half of the $\mathcal{V}_S$ size, only subscription lists consume more space to store the incoming subscriptions (*i.e.*, a $4 - byte\ id$ per new subscription). Surprisingly enough, *POT*'s also exhibits a linear size growth. While a sub-linear growth is expected with factorization, this effect competes with the creation of new nodes. This happens when a subscription does not match with any of the Trie paths or with alteration of existing ones when adding a new subscription list, resizing the array of an existing subscription list, or adding to a new hashmap for its children. The gradient of the memory curve is four times larger for *POT* than for *CIL* and *RIL*. For instance for 100M subscriptions, the first requires $9,200$MB while the other two only $2,320$MB. Remember that despite its factorization gain in terms of abstract nodes (see Section 4.3), the memory requirements of a concrete *POT* node in our implementation is on average $128bytes$ versus $4bytes$ for *CIL* and *RIL*.

4.4.2 Matching time

Figure 10 reveals that for 10M of subscriptions, *RIL* and *POT* outperform *CIL* by a one or two orders of magnitude for all vocabulary sizes. For instance, a matching is performed in $4.33ms$ (resp. $6.21ms$) for *CIL* with $|\mathcal{V}_S|$ equal to $87K$ (resp. $378K$), while it requires only $0.55ms$ (resp. $0.89ms$) for *RIL* and $0.03ms$ (resp. $0.94ms$) for *POT*. *CIL* matching leads to scanning large subscription lists, and consequently to decrement many counters, especially for items with 25 terms on the average that are likely to contain several frequent terms. *RIL* takes advantage of the terms distribution to scan smaller subscription lists than with *CIL*, since by construction only few subscriptions appear in the posting lists of frequent terms. *POT* benefits from a more

drastic pruning of the search space, since despite the theoretically large number of paths, in POT not many term combinations actually exist (see the difference between the complete trie COT and the ROT in Figure 2).

The vocabulary size $\mathcal{V}_S$ also affects indexes. CIL exhibit a convergent behavior: with larger $\mathcal{V}_S$ more subscription lists are scanned but since each one is smaller, on the average the same number of nodes are eventually visited. RIL matching cost grows linearly with $\mathcal{V}_S$: for large vocabularies, subscriptions are less likely to contain the least frequent terms, and consequently the subscription lists of the *medium* frequent terms are larger; since these lists have a higher probability to be scanned when matching an incoming item, more nodes are expected to be visited. In POT this number increases exponentially with $\mathcal{V}_S$: trie has less factorization opportunities (nodes for frequent terms degenerate to inverted lists) and for an incoming item more paths need to be explored. As a consequence, while POT outperforms RIL with one magnitude order for small vocabularies, RIL provides better performances for large ones (*i.e.* $|\mathcal{V}_S| > 378K$).

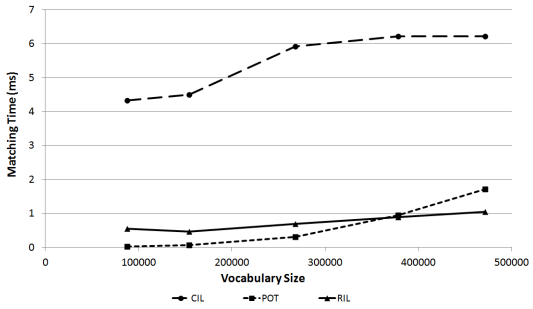

Figure 10: Matching time by scaling $|\mathcal{V}_S|$

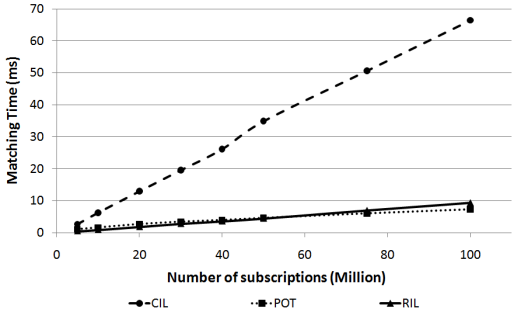

Figure 11: Matching time by scaling $|\mathcal{S}|$

Figure 11 depicts the matching time for a vocabulary $\mathcal{V}_I$ of 1.5M terms when scaling the number of indexed subscriptions from 5 to 100M. In the three indexes, matching time scales linearly with the number of subscriptions. CIL and RIL subscription lists grow linearly with $|\mathcal{S}|$, thus achieving a constant gain which explains this linear behavior (with a gradient that remains 6.5 times higher for CIL). In POT, the larger the number of subscriptions, the larger the size of the index, so the pruning effect decreases since more paths are possibly explored when matching an incoming item. Observe that POT slightly outperforms RIL for a large number of subscriptions: for 100 million subscriptions, matching time is only $7.2ms$ for POT versus $10ms$ for RIL.

4.4.3 Indexing time

Last, we measure the average time required to index $10M$ subscriptions generated from a vocabulary $\mathcal{V}_I$ of $1.5M$ terms. IL building time is in general faster than OT, with only $0.7ns$ (resp. $1.0ns$) required to insert a subscription to RIL (resp. to CIL) while POT needs $17ns$. The additional POT overhead stems from the cost of converting a Trie node from one type to the other (to accommodate added subscriptions, children, etc.). On the contrary, CIL requires only to add the new subscription to the corresponding posting lists of its terms while RIL indexing is even simpler since a subscription is added only to one posting list of its key term.

4.4.4 Summary

The morphology study conducted with our first set of experiments, highlights the factorization gains in terms of abstract nodes of *TI* compared to *IL* indexes. However, as we have seen in the second set of experiments, these gains are somehow eroded, by the concrete representation requirements of nodes in memory of the three indexes: *CIL* (vs. *ROT*) exhibits the lowest (vs. biggest) memory requirements while *ROT* (vs *CIL*) offers better (vs. worst) matching performances. *RIL* appears as a good compromise with memory requirements close to that of *CIL* and matching time close to *ROT* (and can even outperform *ROT* for some settings). More precisely, for critical parameters of web syndication systems, the three indexes exhibit the following behavior:

i) Nodes factorization is particularly important in *ROT* for small *subscription lengths*; short subscriptions are expected to be the reality of web syndication systems as for web queries [17]. In addition, such factorization is almost not sensitive to the order of terms chosen in *OT*;

ii) Although not affecting the number of created nodes in the three structures, the *occurrence distribution of terms* seriously impacts their matching time. In particular, under the realistic assumption that terms distribution in subscriptions follows the terms distribution in items, the search space for frequent terms in *CIL* is extremely large compared to *ROT*. When terms in subscriptions follow the same distribution as in items [8] the latter has to visit on the average three orders of magnitude less nodes than the former. Of course, when they follow the inverse distribution, the number of visited nodes drastically drops for both indexes; *iii*) In the three indexes, the larger the *vocabulary size* $|\mathcal{V}_S|$ is, the more memory is consumed. $\mathcal{V}_S$ additionally impacts matching time: for small vocabularies, *POT* outperforms *RIL* (resp. *CIL*) by an order (resp. two orders) of magnitude; but for large vocabularies employed in web syndication systems, while *CIL* matching time converges, *POT* matching time exponentially grows and is outperformed by *RIL*;

iv) Finally memory and matching time scale linearly in all indexes w.r.t. the *number of indexed subscriptions*.

5 Related work

Several indexes have been proposed for matching efficiently structured (*i.e.* attribute-value pairs) or unstructured (*i.e.* keywords) subscriptions with incoming items, commonly called Publish/Subscribe. The goal of our matching is to find all subscriptions fully contained in an incoming item and not to find the most synthetic or relevant subscriptions (that may be useful for the inverse matching problem of documents to queries). Neighborhood topics such as LCA/SLCA or top-k keyword queries are left for future work since they can be applied only on semi-structured text (*e.g.* XML) or support approximate subscription matching. None of the related works provide an exhaustive experimental evaluation of the most suited indexes supporting broad matching semantics as in our paper. Mostly, they rely on *CI* schemes rather than on a *TI* structure, since memory requirement is larger for TI than CI systems. For example, Le Subscribe [15] system employs a CI index on predicates where subscriptions match by counting contained predicates. But all partially matching subscriptions are examined, and some optimizations have been proposed which group subscriptions according to their size with several variants: by focusing on disjunctive predicates [5], by guiding cost-based algorithms with statistics on subscriptions and incoming events [7], or by sorting matching lists and subscription ids in lists in order to skip unmatching ids quickly [20].

In [1], a TI with a two phase matching of conjunctive subscriptions has been proposed, assuming a fixed total ordering among subscription predicates. The pre-processing phase creates a matching tree over the subscription predicates, in which nodes are predicates. This TI's matching time complexity is sub-linear, with respect to the number of indexed subscriptions.

[10] proposes a two-level TI partitioning of the subscriptions, taking advantage that in most applications events have only a few 'relative' attributes. This allows to efficiently identify a small subset of matching subscriptions. Multi-dimensional TI [19, 13] rely on spatial reduction of the search space to points (for subscriptions) and range queries (for incoming events). [13] extends this index for conjunctive predicates for a Pub/Sub ranked version.

Few Pub/Sub systems have been proposed for keyword-based subscriptions. Most noteworthy is the SIFT [22] selective dissemination of text documents whose alternative indexes have been thoroughly studied [21]. Those studies are only based on disk implementations of (a) the ranked key Counter-Less IL in which subscriptions are sorted on their smallest rank keyword, (b) the Regular Trie and (c) the Regular Ordered Trie (ROT), but not the POT (our Patricia Trie variant). It is expected that this ordering leads to many more common prefixes between subscriptions. In contrast to the disk-based approach, we chose a main memory implementation since it enhances the performance when scaling up to a large number of subscriptions (about 10M). The main differences in their observed behavior are due to (a) the size of the incoming set of words (52 in average for web syndication [8] vs 12K in text documents), (b) the size of the subscription vocabulary (1,5M in web syndication vs. 18,000 in text documents), and (c) the distribution of terms' occurrences. This explains why authors observe that ROT matching takes significantly more time than IL indexes as documents take more I/O to be read. The only work exploiting CI ranks information of terms occurrence distribution in order to optimize subscription matching by clustering techniques is presented in [9]. This proposal does not scale in our context due to our vocabulary size.

On the other hand, ILs have been exploited in [12] for indexing advertisement bids in sponsored search. This optimization relies on a multi-term indexing which combinations are the most frequent in the bids. Since the number of index probes grows exponentially with query size, authors consider a maximum length on indexed term combinations and propose a mapping scheme that reorganizes bids sharing the same subset of terms to the same data nodes based on a memory access cost model. Clearly this optimization does not scale in our setting given that the number of incoming items (on average 25-36 terms) are clearly larger than web queries (on average 2-3 terms [2]).

6 Conclusion and future work

We present and compare three index structures implementing different counting techniques for pruning as early as possible non matching subscriptions to an incoming item. The technical novelty of our work comes from the thorough analysis of the complexity and experimental evaluation of the three chosen indexes. The first two, CIL and RIL, rely on an inverted list while the last one, POT, is based on a Patricia ordered trie. We found that for small vocabularies POT matching time is one order of magnitude faster than the best IL (RIL), while for large vocabularies, both exhibit a matching time of the same order. The actual distribution of term occurrences has almost no impact on the size of the three indexing structures while it significantly affects the number of nodes that need to be visited upon matching something that justifies OT performance gains. Finally, the smaller the subscription length, the larger the OT factorization gain w.r.t. IL and the larger the rank of the term from which the OT substructure degenerates to an IL. Moreover, not only we experimentally evaluate the redundancy saving provided by a Trie w.r.t. IL structures, but also we propose a first analysis of the ROT structure, especially based on variation of the vocabulary, the subscription size distribution and several term occurrence distributions.

We intend as future work to extend the matching process to take into account a similarity score between the subscription and the item. This could imply to weight the subscription terms or to consider additional taxonomic relationships like broader and narrow terms We will also

study the behavior of the three indexes in a parallel and distributed setting.

References

[1] M. K. Aguilera, R. E. Strom, D. C. Sturman, M. Astley, and T. D. Chandra. Matching Events in a Content-Based Subscription System. In *PODC*, pages 53--61, 1999.

[2] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. A. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR*, pages 321--328, 2004.

[3] F. Bodon. Surprising Results of Trie-based FIM Algorithms. In *FIMI - ICDM Workshop*, 2004.

[4] F. Bodon. A Trie-based APRIORI Implementation for Mining Frequent Item Sequences. In *OSDM*, pages 56--65, 2005.

[5] A. Carzaniga and A. Wolf. Forwarding in content- based network. In *SIGCOMM*, pages 163--174, 2003.

[6] J. Clément, P. Flajolet, and B. Vallée. Dynamical Sources in Information Theory: A General Analysis of Trie Structures. *Algorithmica*, 29(1):307--369, 2001.

[7] F. Fabret, H.-A. Jacobsen, F. Llirbat, J. Pereira, K. A. Ross, and D. Shasha. Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In *SIGMOD*, pages 115--126, 2001.

[8] Z. Hmedeh, N. Vouzoukidou, N. Travers, V. Christophides, C. du Mouza, and M. Scholl. Characterizing web syndication behavior and content. In *WISE*, pages 29--42. Springer Heidelberg, 2011.

[9] U. Irmak, S. Mihaylov, T. Suel, S. Ganguly, and R. Izmailov. Efficient Query Subscription Processing for Prospective Search Engines. In *USENIX*, pages 375--380, 2006.

[10] S. Kale, E. Hazan, F. Cao, and J. P. Singh. Analysis and algorithms for content-based event matching. In *ICDCS*, pages 363--369, 2005.

[11] D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

[12] A. C. König, K. W. Church, and M. Markov. A Data Structure for Sponsored Search. In *ICDE*, pages 90--101, 2009.

[13] A. Machanavajjhala, E. Vee, M. N. Garofalakis, and J. Shanmugasundaram. Scalable ranked publish/subscribe. *PVLDB*, pages 451--462, 2008.

[14] H. H. Malik and J. R. Kender. Optimizing Frequency Queries for Data Mining Applications. In *ICDM*, pages 595--600, 2007.

[15] J. Pereira, F. Fabret, F. Llirbat, R. Preotiuc-Pietro, K. A. Ross, and D. Shasha. Publish/Subscribe on the Web at Extreme Speed. In *PVLDB*, pages 627--630, 2000.

[16] B. Philip. Ideal hash trees. Technical report, EPFL Swtzerland, 2000.

[17] N. Vouzoukidou. On the statistical properties of web search queries. Technical report, ISL, ICS-FORTH, Greece, 2010.

[18] A. Walker. An efficient method for generating discrete random variables with general distributions. *TOMS*, 3:253--256, 1977.

[19] B. Wang, W. Zhang, and M. Kitsuregawa. UB-Tree Based Efficient Predicate Index with Dimension Transform for Pub/Sub System. In *DASFAA*, pages 63--74, 2004.

[20] S. Whang, J. Shanmugasundaram, S. Vassilvitskii, E. Vee, R. Yerneni, and H. Garcia-molina. Indexing boolean expressions. *PVLDB*, 2:37--48, 2009.

[21] T. W. Yan and H. Garcia-Molina. Index structures for selective dissemination of information under the boolean model. *TODS*, 19(2):332--364, 1994.

[22] T. W. Yan and H. Garcia-Molina. The SIFT Information Dissemination System. *TODS*, 24(4):529--565, 1999.

[23] J. Zobel and A. Moffat. Inverted files for text search engines. *CSUR*, 38(2), 2006.