Improving Space-Efficiency in Temporal Text-Indexing

Kjetil Nørvåg* and Albert Overskeid Nybø

Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway

Abstract. Support for temporal text-containment queries is of interest in a number of contexts. In previous papers we have presented two approaches to temporal text-indexing, the V2X and ITTX indexes. In this paper, we first present improvements to the previous techniques. We then perform a study of the space usage of the indexing approaches based on both analytical models and results from indexing temporal text collections. These results show for what kind of document collections the different techniques should be employed. The results also show that regarding space usage, the new ITTX/VIDPI technique proposed in this paper is in most cases superior to V2X, except in the case of patterns of high number of new documents relative to number of updated documents.

1 Introduction

Temporal text indexes are used to reduce the cost of performing temporal text-containment queries, i.e., query for all versions of documents that contained one or more particular words at a particular time. The importance of such indexes will increase as the ability to manage timestamped or temporal documents becomes common. For example, an increasing amount of documents in companies and other organizations is now only available electronically, and exist in several versions updated at different times. These documents can be in a number of formats like plain text, HTML, XML, Microsoft Word, Adobe PDF, etc. Many organizations already have searchable repositories or intranet search engines that can be used to retrieve documents based on keywords search, and possibly also other searchable parameters like create or update. Another example is web warehouses which collect web pages from a number of sites at regular intervals, and whose information contents can be queried and analyzed.

We have previously proposed two text-indexing techniques for transaction-time temporal document database systems: the *V2 temporal text index* (V2X) [1] used in the V2 temporal document database system, and the *interval-based temporal text index* (ITTX) [2]. V2X is a combination of full-text indexes and time indexes for performing efficient text-containment queries, and is most suitable for documents with few versions or with a high degree of change between versions. In the ITTX, word occurrences and stored in a way that is particular space-efficient when most documents have several versions and the change between versions is relatively small.

This paper is the first comparative study of temporal text-indexing techniques, and the contributions of this paper are 1) a more detailed study of the space usage of the

* Email of contact author: Kjetil.Norvag@idi.ntnu.no

indexing approaches, 2) improvements to the ITTX, and 3) a study for what kind of document collections the different techniques should be employed.

The organization of the rest of this paper is as follows. In Sect. 2 we give an overview of related work. In Sect. 3 we give an overview our two basic techniques for temporal text indexing, the V2X and ITTX indexes. In Sect. 4 we present several improvements to the ITTX approach. In Sect. 5 we study the space usage of the different indexing alternatives, and for what document collection types the different alternatives should be used. Finally, in Sect. 6, we conclude the paper.

2 Related Work

There has been a large amount of research on indexing temporal data in context of traditional data types, see [3] for an extensive survey. However, as explained in detail in [1], the traditional temporal indexing methods are not directly applicable to temporal text indexing.

The only research work we are aware of that directly focuses on access methods for general temporal document querying, is the proposal from Anick and Flynn [4] on how to support versioning in a full-text information retrieval system. In their proposal, the current version of documents are stored as complete versions, and backward deltas are used for historical versions. This gives efficient access to the current (and recent) versions, but costly access to older versions. They also use the timestamp as version identifier. This is not applicable for transaction-based document processing where all versions created by one transaction should have same timestamp. In order to support temporal text-containment queries, they based the full-text index on bitmaps for words in current versions, and delta change records to track incremental changes to the index backwards over time. This approach has the same advantage and problem as the delta-based version storage: efficient access to current version, but costly recreation of previous states is needed. It is also difficult to make temporal zig-zag joins (needed for multi-word temporal text-containment queries) efficient.

Related to the task of temporal full-text indexing, is indexing temporal XML documents [5]. In this case the focus is on improving path queries. It should be noted that temporal full-text indexes like the ones presented in our paper can also be used to improve performance of temporal XML queries, and this is described in more detail in [6].

The inverted file indexes used as basis in our work is based on traditional text-indexing techniques, see for example [7].

3 Basic Temporal Text-Indexing Techniques

The basic lookup operation in non-temporal text indexing is to retrieve the document identifiers of all documents that contain a particular word w. The most common access method for text indexing is the inverted file, which is also the basis of our approaches.

An inverted file index is a mapping from a term (text word) w to the documents $d_1, d_2, \ldots, d_j$ where the term appears. Inverted files are also the basis of our approaches. In the inverted file index, a *posting list* $PL = (w, d_1, d_2, \ldots, d_m)$ is created for each

index term, where w is the text word, and d_i are the document identifiers of the documents the term appears in. The tuple $P = (w, d_i)$, i.e., an index term and a document identifier, is called a *posting*.

In order to make this paper self-containing, and provide the context for the rest of this paper, we will in this section give a short overview of the V2 index (V2X) and the interval-based temporal text index (ITTX). Both the V2X and ITTX have been implemented in temporal document database prototypes built on top of Berkeley DB [8].

The V2X Temporal Text-Index. A document version stored in V2 is uniquely identified by a *version identifier* (VID). In order to support partial retrieval of documents, the document versions are chunked and stored in a B-tree-based document-version index. The VID is essentially a counter, and given the fact that each new version to be inserted is given a higher VID than the previous versions, the document-version index is append-only and always compact. A document is identified by a *document name*. Conceptually, the document name index has for each document name some metadata related to all versions of the document, followed by specific information for each particular version, including both timestamp and VID for each version. Thus, the document name index can be used to retrieve particular versions of a particular document by providing the VIDs to be used in the lookup in the document-version index.

The words in the document versions are indexed by variants of inverted lists, which essentially provides a mapping from a word to the VIDs of all document versions containing the word. In order to support efficient temporal text-containment queries, a separate index called *VIDPI* is employed. The VIDPI provides the mapping from VID to validity period (start- and end-timestamp), which is the timestamp of the document version identified by VID and the timestamp of the next version (or time of deletion) of the particular document.

Temporal text-containment queries using the VIDPI-index-based approach can be performed by the following two-step algorithm:

1. A text-index query using the text index that indexes all versions in the database. The result is a set of VIDs of all document versions containing the particular word.
2. A time-select operation selects the actual versions (from stage 1) that were valid at the particular time or time period. For this purpose the VIDPI is used. One lookup is needed for each of the VIDs returned in stage 1.

The ITTX Temporal Text-Index. One problem with the V2X is that each unique word in a document version requires a separate posting in the text index. This makes the size of the text index proportional to the size of the document version database. In a document database with several versions of each document, the size of the text index can be reduced by noting the fact that the difference between consecutive versions of a document is usually small: frequently, a word in one document version will in also occur in the next (as well as the previous) version. Thus, we can reduce the size of the text index by storing word/version-range mappings, instead of storing information about individual versions.

In order to benefit from the use of intervals, we use *document version identifiers* (DVIDs) instead of the version identifiers used in the V2X. Given a version of a document with DVID=v, then the next version of the same document has DVID=v+1. In contrast to a VID that uniquely identifies a document version stored in the system, different versions of different documents can have the same DVID, i.e., the DVIDs are not unique between different versions of different documents. In order to uniquely identify (and to retrieve) a particular document version, a *document identifier* (DID) is needed together with the DVID, i.e., a particular document version in the system is identified by (DID||DVID). In this way, consecutive versions of the same document that contain the same word can form a range with no holes.

Conceptually, the text index that use ranges can be viewed as a collection of (w,DID,DVID$_i$,DVID$_j$)-tuples, i.e., a word, a document identifier, and a DVID range. Note that for each document, there can be several tuples for each word w, because words can appear in one version, disappear in a later version, and then again reappear later. A good example is a page containing news headlines, where some topics are reoccurring.

When a new document version with DVID=DVID$_i$ is inserted, and it contains a word that did not occur in the previous version, a (w,DID,DVID$_i$,DVID$_j$) tuple is inserted into the index. DVID$_i$ is the DVID of the inserted version, but DVID$_j$ is set to a special value UC (until changed). In this way, if this word is also included in the next version of this document, the tuple does not have to be modified. This is an important feature (a similar technique for avoiding text index updates is also described in [4]). Only when a new version of the document that does not contain the word is inserted, the tuple has to be updated. It is important to note that using this organization, it is impossible to determine the DVIDs of the most recent versions from the index. For the [DVID$_i$,UC] intervals only the start DVID is available, and we do not know the end DVID. As will be described later, this makes query processing more complicated.

In order to save some space and increase performance of queries for current documents, a separate index is used for the entries that are still valid, i.e., where the end of the interval is UC. In this index the end value UC is implicit, so that only the start DVID needs to be stored. We denote the index for historical entries *HTxtIdx* and the index for valid entries *CTxtIdx*.

One of the main reasons why the VIDPI is very attractive in the context of V2, is that storing the time information in the VIDPI is much more space efficient than storing the timestamps replicated many places in the text index (once for each word). However, when intervals are used, one timestamp for each start- and end-point of the intervals is sufficient, and the increase in total space usage, compared with using a VIDPI index, should be less than what is the case in V2 (although, as we shall see later in this paper, this is unfortunately not always the case). It could also be more scalable, because the V2 approach is most efficient when the VIDPI index can always be resident in main memory. To summarize, our final solution for the ITTX as presented in [2] was to store (w,DID,DVID$_i$,DVID$_j$,T_S,T_E) in the HTxtIdx (where T_S and T_E are the start- and end-timestamps of the interval [DVID$_i$,DVID$_j$>), and to store (w,DID,DVID,T_S) in the CTxtIdx. In [2] we outline the algorithms to be applied when inserting, updating and deleting documents, as well as algorithms for temporal text-containment queries using the ITTX.

4 Improving the Interval-Based Temporal Text Index

After some experimenting with the ITTX we have discovered that for many types of temporal document collections the assumption that ITTX is usually more space-efficient than the V2X does not hold. As a result, we have developed several variants of ITTX where the space efficiency is improved. The ITTX improvements will now be presented together with a discussion about space usage of the variants. We start with the original ITTX, which we from now on will denote *ITTX-24/14* in order to avoid any confusion between the variants.

ITTX-24/14. In the original implementation of ITTX we used 4 bytes to represent DIDs and DVIDs, and 6 byte for each timestamp. This means a total of 24 bytes to represent a posting interval in the historical index, and 14 bytes in the current index where the end of interval is not known and therefore does not have to be stored. In V2X, on average just a little over 2 bytes where needed to represent a posting. This means that in order to be competitive, the ITTX posting intervals need to cover on average at least 12 versions in order to be competitive compared to the V2X. For many application areas this was not the case, and the space usage when using the ITTX was much higher than if V2X was used.

ITTX-16/11. One way to reduce the space usage is to use a compressed representation of the identifiers. In traditional non-temporal posting lists, the difference is often small between two consecutive document identifiers in large posting lists. This makes it possible to encode the identifiers very efficiently, for example using Elias encoding [9] or variable length encoding [10]. In the case of the intervals in ITTX this is more difficult, but one possible approach is to reduce the size of the representation of the DVIDs. Instead of using 4 bytes for each DVID, we can use 3 bytes for the start DVID, and 1 byte to represent the difference between the end DVID and start DVID. The consequences of using this representation is that we can only have $2^{24} \approx 16$ million versions of each document, and that no interval can have more than 256 versions. It is not very likely that a document in a document database should have more than 16 million versions, and intervals over 256 versions can simply be represented by two or more intervals instead. The result of this representation is 16,7% reduced space usage.

Using the same difference-based technique to reduce space usage of timestamps is not possible. The problem is that by using 1 byte to represent a difference, the technique is only useful if most differences is less than what 1 byte represents, which is only about 7 minutes. This will definitely not be the case in general. Even 2 bytes is not sufficient, as it only increases the possible difference to 18 hours. However, in the case of document databases it should be enough with a coarser granularity than the one provided by the 6 bytes for each timestamp that were used in the original ITTX. Similar to the V2X, 4-byte-timestamps with resolution of 1 second should suffice. The result is then that the space needed for storing a posting interval, i.e., DID/DVID/DVID/T/T, is $4 + 3 + 1 + 4 + 4 = 16$ bytes, a total reduction of 33% from the original size.

ITTX/VIDPI. The improvements proposed so far are fairly simple and give only a moderate reduction of space. In order to reduce the size more drastically, some change to the indexing architecture itself is necessary. One possibility is to make the text index itself "non-temporal" by not including the timestamps in the main index, but instead using a strategy similar to the VIDPI [1] approach used in V2. The result is that it is sufficient to store the time interval once for each version, instead of once for each interval in the text index. The text index itself still contains intervals of version identifiers, thus still has the property of not increasing proportional to the version database size which was the case of the index used in V2.

Using the ITTX/VIDPI approach, each posting interval in the index is a DID/DVID/DVID record, using $4 + 3 + 1 = 8$ bytes. The records in the new VIDPI index are DID/DVID/T/T structures, using 16 bytes each assuming 4 byte large timestamps. In order to support efficient search in this index, the records should be sorted on DID/DVID. Similar to what is done in the VIDPI index in V2, DVIDs are sequential so they do not really have to be stored. In addition, the end timestamp of one version is the start timestamp of the next, so only one timestamp for each version needs to be stored, i.e., 8 bytes is sufficient for each version.

There is one important difference between the VIDPI index used in V2 and the one proposed here: in V2 the VIDPI index was sorted on VIDs and was append-only, thus having very low update cost. In the ITTX/VIDPI approach this is not the case, so the update will have a higher cost, approximately one block to be update per version, instead of one per transaction as was the case for V2. However, compared to the cost of indexing words in documents, the VIDPI update cost is only marginal.

ITTX/ND. During a temporal text-containment query using the implemented version of ITTX (the ITTX-24/14 variant), a lookup in the text index returns for each document where the word appears, an interval of versions (DVID,DVID) and a time period (T,T). In order to determine the actual versions, a separate lookup in the document name index is necessary. The DVDIs can be used to reduce the amount of work during the lookup in the document name index, but are not strictly necessary. The reason for still including the DVIDs in the ITTX, is that they are needed to support efficient removal of individual versions from the database. If removal of intermediate versions will not occur, it is possible to omit explicit storage of the DVID interval in the index, and instead having a DID together with the time interval. In this case, only 8 byte is needed for an entry in the CTxtIdx, and 12 bytes in the HTxtIdx.

5 Evaluation of Space Usage

In this study the main focus will be on reducing space usage, instead of studying the access cost directly. When using the indexes discussed in this paper, the space usage also indirectly determines the access cost because access cost is a function of posting list/interval sizes and buffer-hit probability. We will now first describe the test data that is used in the study, then the evaluation approach will be described, and an evaluation of space usage will be performed.

Index type	HTxtIdx	CTxtIdx	Space
ITTX-24/14	DID(4) DVID(4) DVID(4) T(6) T(6)	DID(4) DVID(4) T(6)	$N_{\text{IH}} * 24 + N_{\text{IC}} * 14$
ITTX-16/11	DID(4) DVID(3) DVID(1) T(4) T(4)	DID(4) DVID(3) T(4)	$N_{\text{IH}} * 16 + N_{\text{IC}} * 11$
ITTX/VIDPI	DID(4) DVID(3) DVID(1)	DID(4) DVID(3)	$N_{\text{IH}} * 8 + N_{\text{IC}} * 8$
ITTX/ND	DID(4) T(4) T(4)	DID(4) T(4)	$N_{\text{IH}} * 12 + N_{\text{IC}} * 8$
V2X	VID(2)	VID(2)	$N_{\text{P}} * 2$

Table 1. Space usage of different indexing alternatives. The numbers in paranthesis are the number of bytes used to represent the fields in the index. Number of historical intervals is denoted N_{IH}, intervals in CTxtIdx is denoted N_{IC}, and number of postings in total is denoted N_{P}.

5.1 Test Data

Acquiring real-world temporal document collections is difficult. In some of our previous studies in temporal document databases, we have used a document collection that is based on the evolution of pages from a set of web sites. Even though that collection was sufficient for the use in our previous work, it has a number of shortcomings that makes it less satisfactory for the purpose of this paper: it has a very high number of documents that are never updated, and it only presents one application area (temporal web warehouses). When comparing different indexing approaches, it is necessary with a number of test collections with different characteristics/statistical properties, and is also an advantage if we know and can control these characteristics, in order to make it easier to explain the results. For this purpose we have developed *TDocGen*, temporal document generator.

TDocGen creates a temporal document collection whose characteristics are decided by a number of configurable parameters. For example, the probability of update, average number of new documents in each generation, etc., can be configured. One of the important properties of TDocGen is that the documents it creates have vocabulary, vocabulary size, and words distribution according to what is expected in the real world. The created documents contain real words taken from histograms based on real (but non-temporal) documents and follows empirical laws like Heaps' law and Zipf's law. In order to capture the aspect of dynamic and static documents, every new document created by TDocGen is characterized as being dynamic or relatively static. The percentage of documents in each partition and the probability of updates to each partition is configurable. In a typical configuration 20% of the documents are defined to be dynamic, and 80% of the updates are performed on dynamic documents. TDocGen is described in more detail in [11].

5.2 Evaluation Method and Validation

Our approach to comparison is to use the simple disk usage models as summarized in Tab. 1 as basis for calculating the space usage (note that the extra space needed for the VIDPI indexes in the case of the ITTX/VIDPI and V2X alternatives is very small compared to the rest of the index structure and is therefore omitted from the

space usage models, the same is the case for the start-VID of each chunk in the V2X index). In order to calculate the space usage for the different indexing alternatives, the models will be instrumented with words, validity intervals, etc., based on the document collections created by TDocGen. The statistics is acquired by inserting the collections into an IDDB database and using the statistics from the ITTX index.

In order to have confidence in the result using our evaluation methods, a validation of the approach is necessary. The modeling approach is the same for all the ITTX-variants, so validation of one of them suffices. The ITTX-24/14 variant is implemented in the IDDB prototype, and we insert one of the test collections into an IDDB database and compare the actual disk space usage of the ITTX index in IDDB with the predicted values resulting from instrumentation of the model as described above.

In order to predict the actual space usage from the values described in the previous section, page utilization and page overhead has to be taken into account. Figure 1 illustrates space usage of storing a test collection in IDDB for different amounts of data. The uppermost curve shows the actual disk space used for data stored in the IDDB system (having page fill factor of 67%), the second curve shows space usage adjusted for page utilization (i.e., space usage if fill factor was 100%), and the lower curve shows the space usage as predicted by the model which assumes 100% fill factor. The discrepancy between the model and real values is the space occupied by keys (words) and overhead on each page.

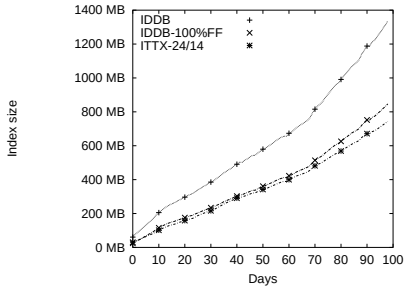


Fig. 1. Space usage of storing a test collection in IDDB for different amounts of data.

The effects of page utilization and space for keys and overhead is approximately the same for the indexing alternatives, so in the rest of this study we will use the values predicted from the models. These values reflects space usage after a reorganization of the database, which would bring page utilization close to 100%. Figure 1 shows that the accuracy of the model is good, and the small difference also gives high confidence in the models for the indexing techniques that are not actually implemented (ITTX-16/11, ITTX/VIDPI, and ITTX/ND).

A similar approach is followed to validate the V2X model. With the test data we have studied, the difference between predicted and real values is approximately 3%, this is the result of omitting the start VID of each chunk in the V2X index from the model.

5.3 Space Usage

Different document database applications have different access pattern and document characteristics. A big advantage of having document collections created by a synthetic document generator is that we are able to produce collections reflecting the different applications. We will now study space usage for a number of different temporal document collections. Our application case that is behind the parameters, is a company or

Parameter	Default value	Normal distributed	
		Average	Std. dev.
Number of documents first that exists the first day	20		
Percentage of documents being dynamic	20		
Percent of updates applied to dynamic documents	80		
Number of words in each line in document	10		
Number of new documents created/day		20	10
Number of deleted documents/day		3	1
Number of updated documents/day		200	75
Number of lines in new document		150	50
Number of new lines when updating		50	10
Number of deleted lines when updating		20	5

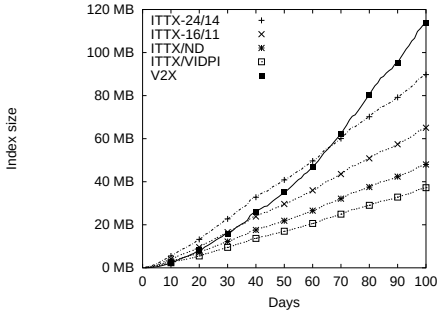
Table 2. Default document generator parameters.

department involving a number of persons that each day create and update a certain number of documents. The starting point for the study is a document collection created by TDocGen using the parameters in Tab. 1. These parameters can for example reflect a group of 10 people where each of them every day on average creates 2 new documents and updates 10 documents, and once in a while delete documents. Assuming 50 lines on a page, a typical new document has 3 pages, and an update is typically addition of text equivalent to one new page, and removal of text equivalent to half a page. In addition to performing a study using the default parameters in Tab. 1, we have also changed some of the parameters to understand how these changes will affect space usage.

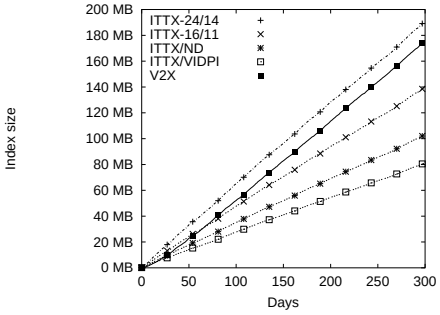
The results are presented in Fig. 2. In all graphs, the space usage for the index is presented as a function of document collection size, which is given as number of days there have been actions performed on the database. In order to make it easier to see details in the graphs, we have only included the interesting part of the range (days), i.e., up to the interesting points of crossing. The actual document collection size at the end of each experiment is typically in the order of 15 times the space usage of the V2X, i.e., between 1 GB and 4 GB (depending on number of days and update/create patterns).

Figure 2(a) illustrates space usage using the default parameters as presented in Tab. 1. As can be seen in the figure, already early in the experiment it becomes obvious that using the default parameters, the interval-bases indexing techniques based on the ITTX excel.

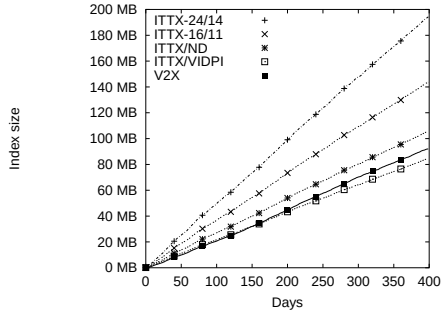
The default parameters represent a quite aggressive update pattern in terms of amount of updated documents, although the percentage of documents that are updated decreases as the size of the database grows larger. In order to study the effect of the update rate, we have run the experiments based on the default parameters, but with number of updated documents each day reduced to 100. The results are illustrated in Fig. 2(b), and shows that the space usage of the V2X does not increase at the same very high rate as in Fig. 2(a). It also shows that it takes a longer time before the difference between V2X and the other techniques becomes significant. However, the difference increases with time, so it is obvious there are great benefits gained from using intervals-based indexes also in this case.



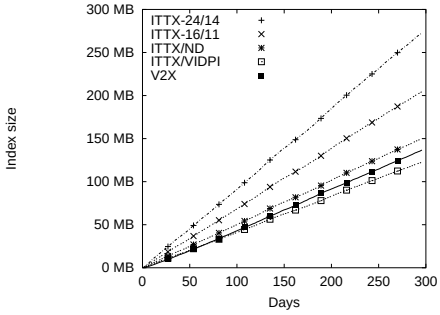
(a) Default parameters.



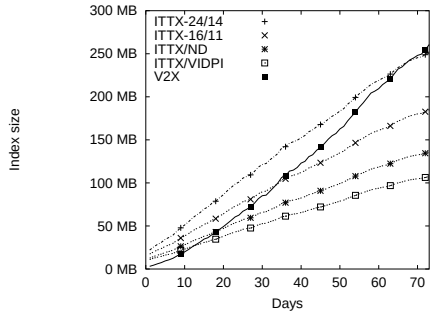
(b) Number of updated documents/day reduced to 100.



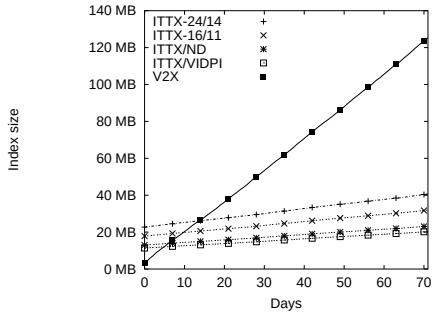
(c) Number of updated documents/day reduced to 50.



(d) Number of new documents/day increased to 40.



(e) Increasing intial number of document and the update rate.



(f) Pattern consisting of mostly small updates.

Fig. 2. Space usage for different temporal document collections.

Figure 2(c) shows the space usage when the update rate is further reduced, to 50 updates documents per day. It illustrates well the fact that the V2X is best when most documents have few updates, because short intervals make the interval-based indexes inefficient. The same aspect can also be illustrated by increasing the number of new documents created each day, instead of reducing the update rate. This is illustrated in Fig. 2(d), which shows space usage when the number of new documents has been increased to 40. What happens is essentially that new documents are created so fast that it is not possible to update all old documents with the given update rate. For the values as shown in Fig. 2(c) and 2(d), there is not a very significant difference between ITTX/ND, ITTX/VIDPI, and V2X, and access performance can be just as important as space when choosing which index to use. However, when the number of new documents relative to number of updated documents increases even more, using the V2X will become more and more beneficial. The extreme parameters are when there are no updates, only new created documents. In that case, a new interval has to be created for each document, and space usage will increase linearly with increasing document collection size for the interval-based indexes as well. However, an interval occupies much larger space in the index than just a VID as is the case for the V2X, so V2X will in total have a much lower space usage in that case.

In order to study how the index structures scales to a larger number of documents, we increased the number of documents at the start to 1000, and the number of updated documents each day to 400. The space usage is illustrated in Fig. 2(e), and the difference between V2X and the interval-based alternatives is significant.

In addition to the experiments presented so far on basis of the parameters in Tab. 1, we have also studied the impact of varying the other parameters. One interesting case is when most updates are very small, i.e., only one line added by during each update. One possible application area where this could occur is a CV database. This is essentially a best-case for the interval-bases index, and is very obvious when we see the graphs in Fig. 2(f): very little increase in index size of the interval-based approaches, but very high increase in space usage for the V2X approach.

6 Discussion and Conclusions

Support for temporal text-containment queries is of interest in a number of contexts, both temporal document databases and temporal XML databases[6]. In this paper, we have presented improvements to the previous temporal text-indexing techniques and studied in more detail the space usage of the indexing approaches. As has been shown, regarding space usage the ITTX/VIDPI is in most cases superior to V2X, except in the case of:

- High number of new documents relative to number of updated documents. In that case, many intervals will be one-version intervals, which are expensive in terms of space usage.
- Possibility of physically deleting historical versions from the database, for example if granularity reduction [12] or vacuuming is performed (note that in the case of ordinary/logical deletions of documents, previous versions will be retained in the

database). In that case, intervals will be destroyed and the relative space usage of interval-based approaches will be very high.

For both ITTX/VIDPI and V2X, a query has to be performed by a lookup in the text index followed by a lookup in the VIDPI index. As long as the VIDPI index can fit in main memory the cost of the VIDPI lookup is not significant and does not have to be taken into account. In this case, choice of index structure can be based on create/update pattern. However, if the number of document versions is very large, the VIDPI index might not fit in main memory. This can for example be the result of a very large document collection, but can also happen in the case of a small collection that contains many small documents. In this case, the extra lookups in the VIDPI index can contribute much to the overall access time, and using an index variant without the need up the extra lookup can be beneficial. The ITTX/ND normally occupies more space than an ITTX/VIDPI index, but the difference is small enough to consider it a good alternative when the VIDPI index does not fit in main memory.

The proposed indexing techniques work well even for large document collections. However, we believe there still are possible ways of improving indexing performance in the case of very large document collections, and our current work focuses on designing index structures that are truly scalable. These indexes will be needed when the document collection and indexes are of such sizes that only small parts of the index structures can be assumed to be resident in main memory.

References

1. Nørvåg, K.: Supporting temporal text-containment queries in temporal document databases. Journal of Data & Knowledge Engineering **49** (2004) 105–125
2. Nørvåg, K.: Space-efficient support for temporal text indexing in a document archive context. In: Proceedings of the 7th European Conference on Digital Libraries (ECDL'2003). (2003)
3. Salzberg, B., Tsotras, V.J.: Comparison of access methods for time-evolving data. ACM Computing Surveys **31** (1999) 158–221
4. Anick, P.G., Flynn, R.A.: Versioning a full-text information retrieval system. In: Proceedings of SIGIR'1992. (1992)
5. Mendelzon, A.O., Rizzolo, F., Vaisman, A.A.: Indexing temporal XML documents. In: Proceedings of VLDB'2004. (2004)
6. Nørvåg, K.: Algorithms for temporal query operators in XML databases. In: Workshop on XML-Based Data Management and Multimedia Engineering. (2002)
7. Witten, I.H., Moffat, A., Bell, T.C.: Managing Gigabytes: Compressing and Indexing Documents and Images. Morgan Kaufmann (1999)
8. Olson, M.A., Bostic, K., Seltzer, M.: Berkeley DB. In: Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference. (1999)
9. Elias, P.: Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory **IT-21** (1975) 194–203
10. Fraenkel, A., Klein, S.: Novel compression of sparse bit-strings — preliminary report. In: Combinatorial Algorithms on Words, NATO ASI Series Volume 12. Springer Verlag (1985)
11. Nørvåg, K., Nybø, A.O.: Creating synthetic temporal document collections. Technical Report IDI 6/2004, Norwegian University of Science and Technology. Available from `http://www.idi.ntnu.no/grupper/DB-grp/` (2004)
12. Nørvåg, K.: Algorithms for granularity reduction in temporal document databases. (Accepted for publication in Information Systems)