

# Sussex Research

## A concept/use of temporal Search Engine

Monika Arora

### Publication date

21-06-2008

### Licence

This work is made available under the **Copyright not evaluated** licence and should only be used in accordance with that licence. For more information on the specific terms, consult the repository record for this item.

### Citation for this work (American Psychological Association 7th edition)

Arora, M. (2008). *A concept/use of temporal Search Engine* (Version 1). University of Sussex.  
<https://hdl.handle.net/10779/uos.23313710.v1>

### Published in

Corporate Strategies & Innovations in the emerging Global Economy

### Copyright and reuse:

This work was downloaded from Sussex Research Open (SRO). This document is made available in line with publisher policy and may differ from the published version. Please cite the published version where possible. Copyright and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners unless otherwise stated. For more information on this work, SRO or to report an issue, you can contact the repository administrators at [sro@sussex.ac.uk](mailto:sro@sussex.ac.uk). Discover more of the University's research at <https://sussex.figshare.com/>

## A Concept /Use of Temporal Search Engine

Prof MONIKA ARORA

Department of Computer Science  
Apeejay School of Management  
Dwarka, New Delhi  
marora.asm@gmail.com

### ABSTRACT

As we come to rely on the World Wide Web as a primary information resource, maintaining versions and histories of documents becomes important for every project. If the document corrupts, then one can take the previous version of document consider for the change. Since content can easily change on a page, having an archive of temporal implementation versions of a web page allows users to maintain the content of their bookmarks instead of simply the URL. Also, with a recorded document history to compare versions of pages over time to see how the user change.

The role of Temporal Search Engine tracks these changes and allows a user to keep record through an archive specifying a date range. The Temporal Search Engine collected and maintained an archive of searchable web pages. This concept is to keep track of the archive documents and a model implementation for viewing the versions of pages, and the differences between them. Also keep the track of missing documents. This paper, attempts to discuss implementation model of temporal index system over the archive. And also the retrieval time calculation change and Temporal Search Engine prototype model.

**Keywords:** Temporal, Temporal search Engine, Temporal index, history of documents

### Introduction

The World Wide Web is emerging a lot. The content is not only important to the creator but also the people who are using it. The website is dynamic to construct and use. A search engine technique gathers and provides you the relevant and important information. It is like a snapshot of the Web. However, conventional search engines only return results for the current state of the Web but they do not have a history archive, to find old versions or history of web pages. The ability to track the changes and history of a website could reveal a great deal of information about how the world is changing. If a page is cited as a source, the content of the page could be retrieved even after the original page is long gone.

Also the content of bookmarks would not be lost if a page is removed or changed considerably.

This concept not only facilitates to take backup but also fast in retrieving the updated documents. It also facilitates the intranet organization where one can always maintain and the supervisor can keep the track to the subordinates. It is also helpful to the backup mechanism where your older version is always saved. As add the archive document, A temporal index is updated every time a new day. The constructing the index however is a server side process that can take place without users noticing. This indexes decreases in retrieval time is important to users wishing to utilize the system.

**Temporal Knowledge** is based on distinction between time points and time intervals. They propose that treating time intervals as an ordered pair with a beginning and endpoint as a primitive. It depends on defining operations over this data type, such as *T occurs before S* or *T occurs during S*. The concept of intervals is treated, as grouped time points are basics temporal systems. **Time Transaction Databases** is the semantics of temporal, or time transaction. In databases, time intervals are called *states* and time points are called *events*. States are then delimited by events. Information in a time transaction database is called a *fact*. *Transaction time* deals with when the fact is current in the database or when a fact was actually added to the database. This information cannot be changed and would be automatically created when facts are entered in the database. Retrieving facts with queries returns a *time-slice* of the database.

For **example** if a document corrupted, so instead of that document that is lost. Instead of restarting you have to manage with the just previous version of the There incremental tree assumes that a user will attempt to retrieve more recent facts. The implications of temporal information can be vast and difficult to handle. As new information is added, it must be related to previous facts about how they work together. There are many possible temporal relations between just two pieces of information. With the continual addition of new facts to a database it becomes impossible to track the temporal relations between all of them. Information retrieval delves deeper into document histories and temporal aspects of documents; we will need to incorporate more of the work that has been developed for databases. The ability to efficiently retrieve records from a database is very closely related to information retrieval over a set of documents.

**Source Code Control Systems** Source code control systems are important when trying to maintain software projects, especially for commercial purposes that need a reliable working version while developing an improved, but unreliable,



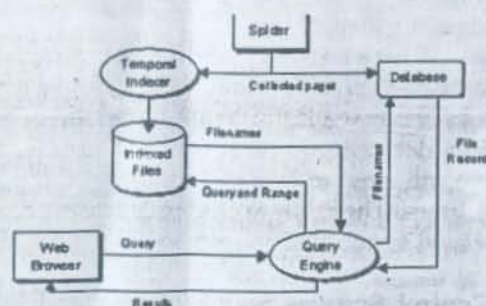
version between the different files. Code control systems have laid an excellent groundwork on how to version files. Some of these techniques, which catch, hold of the versioning of online documents. This technique is a method for saving storage space since you only record versions that have been modified and read.

**Tracking Versions:** There are a variety of techniques that have been used to track the versioning of documents. There are two primary levels at which to add versioning support for documents. The first method stores version information as part of the document, allowing any program, most likely a web browser, to determine how recently the document has been modified. The second method employs an external system to manage document versions. Since they already store the documents and information about documents for general viewing, Web servers ideal for this activity. A proxy based server, or any client side history tracking, works well for an individual that wishes to track their own web surfing.

**Version Management:** There is also an edit-based management that is similar to SCCS and RCS, in that it stores the original and only the changes. The server and proxy systems simply store the entire document when new versions are found, which is the approach taken in the Temporal Search Engine.

#### Implementation Model of Temporal Search Engine

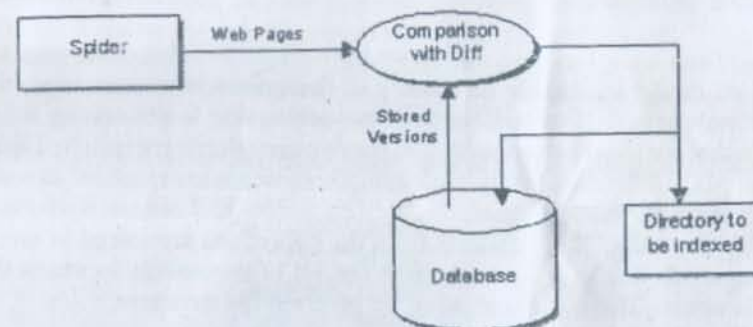
The basic Temporal Search Engine altered the spider, index, and retrieval components to support the notion of time. It also utilized a database to aid in tracking different versions of web pages and to assist in the retrieval process. By maintaining an archive of collected documents and comparing them to newly collected ones, it is possible to create a searchable history. Figure 1 gives an overview of the Temporal Search Engine.



(Figure 1: System Architecture Overview)

A **spider** is a program that gathers data from the World Wide Web. The spider takes a given URL and follows all hyperlinks found, creating a mirror image of the site. However, the basic spider will only retrieve web pages and store them to directories. We have added a wrapper to the spider for extra functionality.

By keeping the most recent versions, we can compare the newly collected page to previously collected pages for the same URL to see whether or not any differences are found. The spider is run once a day and, if it encounters a new page, it adds the page to the database. When the spider finds a previously collected page, it performs a *diff* check against the stored version and the current version. If there has been a change, a new entry is created in the database and the current version is stored, as Figure 2 details.



(Figure 2 - Spider Functions)

Otherwise, the spider will discard the page and continue onwards. Files are named by using a unique identifier, assigned in ascending order, and then they are stored in sub-directories denoted by their date.

**Archive Description** The Temporal Search Engine uses any database as the backend to store information about all versions of the collected web pages. Although this information could be stored in additional files. There are three fields to track information: the **URL**, the **date spidered**, and the **filename**. By tracking the URL, we can easily find all versions of a document from a URL. The date spidered field is stored to determine, during retrieval, whether or not a page is within the range specified by the user. Finally, the filename field is used during retrieval to display a page to a user. The database structure is proposed as follows:



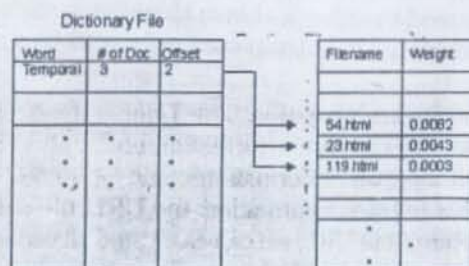
Description	Field	Datatype	Example
Uniform Resource Locator	URL	String	http://www.abc.org
Date when this file was added	date_spidered	String	20050322
File name used in indexing	Filename	String	91.html

(Table 1: Database Overview)

In addition to the database, a version of each page collected is stored. The database is used to keep a record of the collected pages, but the pages themselves are stored in a publicly accessible directory. The pages are then displayed during the retrieval process.

**Indexing:** Once a site has been spidered, we have create an index of the pages. An index is an easily searchable dictionary of the information contained in the archived web pages. The end result of pre-processing is generating a list of keywords that are in each document. The user's query words are matched against the keywords and the documents containing these words are returned as results. The indexing process takes the entire raw data archive and outputs a structured, searchable file. All of the keywords from the collections are stored in one file. These keywords link to a file containing the list of documents in which these keywords occur. This system is called an inverted file structure.

The two files created for the index, are called the Dictionary file and the Postings file. A hash table data structure is used to store the Dictionary, which allows for O(1) retrieval time. A Dictionary record has three parts: a word, the number of documents the word occurs in, and the offset in the Postings file. A Postings record has two parts: the file name and weight of the word in that file. The Postings file stores these fixed length records in sequential order. As seen in Figure 3, this setup allows us to search based on words and easily retrieve any given file in the archive along with the importance of the word in that specific file.

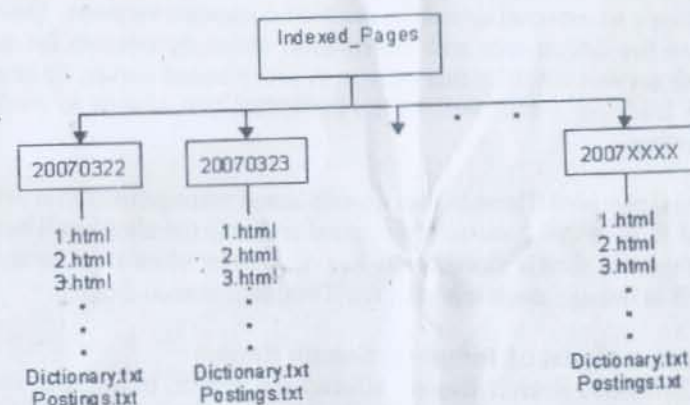


(Figure 3: Dictionary/Postings File Relationship)

**Error!**

The pilot Temporal Search Engine created a separate index for each day that was archived. Creating the index in this fashion causes a problem with retrieval, which we will describe shortly. The final directory structure created by the system is shown in Figure 4.

The indexer must be run daily to create a new dictionary and posting files from materials collected by the spider. Once an index has been created for a day it does not need to be modified or rebuilt in the future. The drawback, however, is that weights of documents are not calculated across the entire archive, but just the individual day's index.

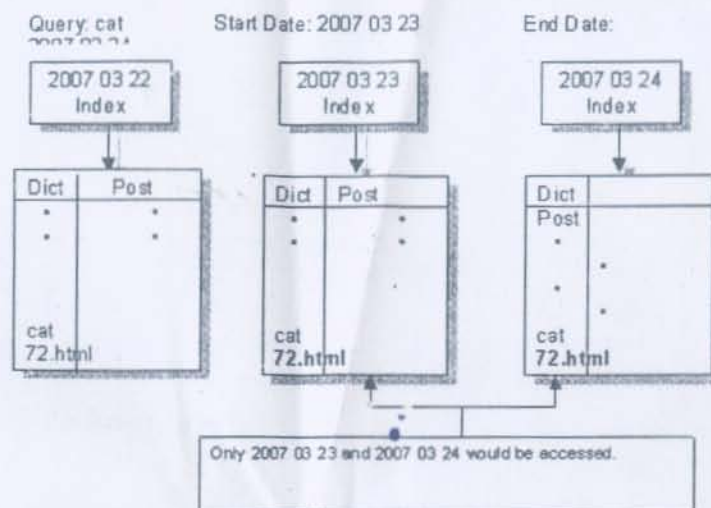


(Figure 4 - Pilot Index Directory Structure)

**Retrieval Functionality** The final component of the Temporal Search Engine is the retrieval mechanism. Once the web pages have been indexed, a quick retrieval scheme must be used to search the indexes and return accurate results. Since all of the pages are divided into directories for indexing, we use them as our archive as well. When we need to return a page to a user, findings in which day it was archived and display the appropriate page.

To search a multiple-index system is by searching every index from the beginning of the archive; otherwise a searching error occurs during retrieval. Since each day's index only includes pages that have been modified, older unchanged pages will not appear in the index for that day. If we take the approach of only searching over the range entered by the user, pages that do not specifically change within the range will not be shown. In Figure 5, we can see that it is possible to miss a valid result, if the query range is specified to be March 23 to March 24, it will not return the results the March 22 as that invalid.





(Figure 5 - Search Error)

The Temporal Search Engine also supports the ability to review all versions of a document, any two versions can be compared and the differences between them will be displayed. When a query is returned, a check is made to see if other copies of a page exist. If they do, a link is added beside the result to allow the user to access these versions. With the URL stored in the database, a document history can be retrieved without difficulty. This allows a user to examine all versions of a document easily, if they so desire.

Allowing users to view the changes between documents is also in the Temporal Search Engine. Another use of the *diff* function allows us to determine the changes between two versions of a document. By pulling both pages from the archive and comparing the two, we can then create a patch of the changes. The user can easily see the differences since the old information is struck through in red and the new additions appear in green.

Lastly, there is a sort option that enables the user to decide how they wish to see the results. Users can sort by relevance, returning the highest rated pages. They can also sort by date, returning the most recent pages. This allows the user to easily search for fresh content if they wish.

**Implementation Model** The indexing and retrieval are the two important aspects of the Temporal Search Engine. Our goal was to decrease retrieval

times for queries for the end user. By creating a temporal index instead of multiple daily indexes we can eliminate the necessity of looping over the entire archive and improve the accuracy of the return results by considering weights of words over the entire documents set. **Temporal Index** Firstly, we have modified how many indexes are created and used in the system. The modified approach creates a single index over the entire archive. A temporal index simplifies the retrieval process, as we will see momentarily. Since every document is included in one archive, we must have a way to distinguish on what day it was archived.

By combining a directory name, which includes year, month, and date of archival, and a filename we can create a unique identifier for each file in our archive. This is an important aspect for speeding up retrieval and vital for the range based searching. Also, with this alteration the same spidering scheme can be used without modification because there cannot be a repeat filename in a single day. Figure 6 shows the new filenames stored in the Postings file.

Filename	Weight
20070322_54.html	0.0042
20070329_54.html	0.0033
20070404_119.html	0.0029
20070327_15.html	0.0012
.	.

(Figure 6 - Temporal Index Postings File)

**Retrieval from a Temporal Index:** By moving to the temporal index, retrieval can avoid the brute-force search required by multiple indexes. When retrieving from a single index with embedded dates, we remove the need to repeat the same query across multiple Dictionary files. A single lookup is required in the Dictionary file, and then from the Postings file all documents with a query word can be retrieved. We parse the records from the Postings file to find the date the file was archived and the filename. Using the date we can filter files that are in the user's specified range. The filenames and dates remaining are then passed as keys to the database to retrieve the URL to display on the results page. The benefits of this implementation become more important as the size of the archive, and the number of days the spider is run, increases.



### Proposed Layout /Screenshots

The following are proposed Layout /screenshots of the four major pages of the Temporal Search Engine. Each page is intended for simple easy design, without overloading the user with unnecessary information.

**By Date:** The opening search page is featured below where the user may enter the query. They can control the start and end date of their query range, which by default are the beginning and end of the archive. The default sort option is by relevance (Figure 7). A Results screen will display a link to the archived file, the relevance of the document, the day the file was archived, and the original URL. To the right side, if a file has more versions a link will appear which allows the user to retrieve those versions

Fig. 7:First Proposed Screen Layout

**By document** The All Versions page will allow the user to see what day each new version was archived, along with a link each version of the file. They also can select two of the files and compare the differences between them. If it shows File 2 superimposed onto File 1. The red material that is struck through only occurs in File 1. Green lines only appear in File 2. Everything else shown appears in both files. Images are not saved by the archive and cannot be seen. This can be tested on any public site. The websites were selected based on an update schedule ranging from low to medium to high. After seeing a minimal variability in the first sites selected, we added five more sites on March 24 to

create a higher versioning rate. On the bases the websites can be seen as that have a low update frequency, which means most of their pages should appear only once or twice in the archive. On the other hand sites like www.pbs.org have a high update rate with almost daily changes to all pages. The documents of the extension can be seen. No of documents updated day wise.

### Conclusion

This paper, described a Temporal Search Engine that supports queries over a user specified range of days. It also concluded that the only accurate search over such a system is by starting at the beginning of the archive. It also utilizes a temporal index to improve retrieval times.

An unforeseen benefit of the temporal index system is the reduced storage requirements due to the single dictionary file. Since the weight of words in each document is across all documents in the archive, a more accurate representation is given with a temporal index.

The Temporal Search Engine only supports storing full copies of new versions, even though the changes could be minor. By pulling from the RCS separate, reverse delta technique, it could reduce the space need to store the archive considerably. This smaller storage may also allow for more efficient indexing of documents, since only the changes would need to be re-indexed.

This application also brings into question on where the Temporal Search Engine would be most effective. With rapidly changing large sites, constant addition and modification of pages would cause the archive to grow to an incredible size. The Temporal Search Engine may be better served over smaller sites that update less frequently. Testing the efficiency and archive size versus site size and update rate is a viable research consideration. This should also consider the image amendments also

### References

1. Ryan Sheahan, Improving Query Retrieval Times in the Temporal Search Engine, Master's Project Report, University of Kansas, 2003
2. Batheja, Suchit, Temporal Search Engine, Master's Project Report, University of Kansas, 2003
3. M. Nascimento, M. Dunham, R. Elmasri, Using incremental trees for space efficient indexing of bitemporal databases, *Proceedings of the Second International Conference on Application of Databases (ADB'95)*, pp. 235 - 248, Santa Clara, California, December 1995.