

TimeLinks: Exploring the link structure of the evolving Web

Reiner Kraft (rekraft@cse.ucsc.edu) and Enes Hastor (enes@cse.ucsc.edu)

April 4, 2003

1 Introduction

The Web is growing and evolving at a rapid pace. The data of the Internet archive (<http://www.archive.org>) represents an unique opportunity to explore and investigate how the Web evolves over time. However, it is difficult to conveniently extract the link structure of the Web at a certain point in time. For instance, one problem is that crawls may overlap or use inconsistent naming conventions for storage, which makes it difficult to access hyperlinks at a particular moment in time using conventional data extraction methodologies.

We introduce the notion of *TimeLinks* and motivate to use these as a convenient infrastructure for experimenting with the evolving link structure of the Web graph. *TimeLinks* are directed edges that incorporate a first and last crawler access time stamp. *TimeLink* collections can then be used to generate the link structure of the Web graph at any given moment in time. In particular, we are interested in how these changes of hyperlinks can be used for various ranking algorithms in the context of Web information retrieval (e.g., indegree and host-indegree of popular web sites over time). Since the Internet archive contains currently over 100 Terabytes of compressed data we were able in our experiments to examine a large collection of billions of hyperlinks.

Our goals are to first develop infrastructure and tools to build large *TimeLink* collections. Second, we then want to use these *TimeLink* collections as a basis to compile statistical information that provides insights on how link structure of the Web is changing. In particular, we are interested in how these changes of hyperlinks can be used for various ranking algorithms in the context of Web information retrieval (e.g., indegree and host-indegree of popular web sites over time).

2 Related Work

The Web graph with billions of nodes, which grows exponentially, is a fascinating object of Study. Kleinberg et al. [3] investigated and analyzed the graph structure of the Web. Also, Broder et al. [1] provide more insights on the overall

Web graph structure. Randall et al. [4] built a connectivity server that provides fast access to a URL’s successors and inlinks. Overall a lot of research has been done that focuses on link analysis of a snapshot of the Web at a particular time. However, the integration of time as a dimension of evolution of the Web graph is mostly neglected.

3 Overview

The paper introduces *TimeLinks* and motivates to use these as a convenient and useful infrastructure to generate the link structure of the Web graph at any given time.

We first introduce the idea and concepts of *TimeLinks* in detail. Furthermore, we present a precise theoretical foundation for *TimeLinks* and devise an algebra to manipulate them and extract data. Extracting information ranges from iterating over *TimeLinks* to more complex operators. For instance, there are operators that merge *TimeLinks*, retrieve a set of hyperlinks within a specified time interval.

We then show how a collection of *TimeLinks* can be derived from existing link feature data obtained from the Internet archive. For this purpose we developed basic *TimeLink* infrastructure and tools (e.g., `parseDAT`, `convertTL`, `mergeTL` to convert existing link feature data to *TimeLink* collections. These collections can then be used and queried using *TimeLink* run-time tools (e.g., `extractTL`, `linksPerMonthTL`, `printTL`) to extract and output graph data at any given moment or interval in time.

We further discuss practical issues on how *TimeLink* collections fit into the current crawling and data extraction infrastructure of the Internet Archive, along with implementation details. Scalability and performance issues that arise with the large amount of link data will be enumerated.

We then describe first the experiments we conducted using a large sample set of data comprising 7 billion hyperlinks from the Web archive. First, we explain the processing steps using the tools and infrastructure we developed to transform the raw link data into data structures that represent *TimeLinks*. We compiled statistical information on this transformation process that shows how many links were originally extracted, and how many *TimeLinks* were derived. We then show how the links are distributed over time. Since this is work in progress we expect more interesting results downstream.

4 TimeLinks: A model to represent hyperlinks over time

4.1 Overview of the model

First, we extend URLs to incorporate the time as a new dimension: A URL is a pair (URL, t) , where URL represents a URL, and t the content of URL at time

t . This is similar to Alexa’s archival URL specification [5], that also associates time to a URL. However, in their model there is a archival host that needs to be specified, that will then retrieve the content of a (URL, t) request (or list of documents when patterns are used), whereas our model does not enforce any particular implementation. We also define a constant NOW that is added as as a default time parameter if t is omitted, which can be resolved to the actual time of the request.

In the proposed model a URL conceptually is a container that comprises a sequence of content versions. Each content version of a URL has a time of creation, and an end time (when the content changed to a new version). In addition, each content version may also have a set of hyperlinks to other URL destinations. Start and end times are obtained from the Web crawler that accessed the document. Therefore these start and end times of a content version may not reflect the actual content version time, and therefore only represent an approximation based on the schedule of the crawler that visited the URL. A more detailed discussion on this topic of approximating page change and implications for an incremental crawler is discussed by Cho and Garcia-Molina [2].

There are different models on how to systematically access the data on the Web using Web crawlers or automatic robots. For instance, a Web crawler can be started for a certain time period with the task to collect a specific set of data (batch mode). Once it collected all the required URLs or after a certain amount of time (e.g., a week), the crawler shuts down and the crawled data is processed (e.g., indexed). Another model is that a crawler is running continuously and uses heuristics to determine good access frequencies for documents (e.g., to optimize bandwidth, to detect page changes more precisely).

A hyperlink (or simply link) is a pair (S, D) , where S represents the source URL, and D the destination URL. Typically the first time a crawler visits a page S , it downloads its contents and extracts its links that can be then used for further crawling destinations. Our model will use the access time stamp of the crawler, which then will be associated with the content version and all hyperlink pairs (S, D) . This is the most generic design, since it is not dependent on the type of crawler that is used. A hyperlink then is a triple (S, D, t) .

Suppose a crawler accesses a page several times over the course of a year. This will result in a set of hyperlink triples. More precisely, let k be the number of times the crawler accessed URL S , and m be the number of generated hyperlink triples (S, D, t_i) , where $1 \leq i \leq m$. We immediately see that $m \leq k$, since it may be the case that the crawler accesses S , but cannot extract its content. Also, the content may have changed and there is no link to D anymore.

Now we can introduce the notion of “time links”. These are directed edges (hyperlinks) to incorporate a time interval $[t_i : t_j]$, where $t_i \leq t_j$. A *TimeLink* tl then is a triple $(s, d, [t_i : t_j])$, where s is the source URL, pointing to the destination URL d during the time interval $[t_i : t_j]$.

We can iteratively construct time links from extracted link triples over time as follows: Let $l_1 = (S, D, t_i)$ be a link triple and $l_2 = (S, D, t_j)$ be a link triple. We define a time link $tl(l_1, l_2)$ to be the triple $(S, D, [t_i, t_j])$ if $t_i < t_j$

or $(S, D, [t_j, t_i])$ if $t_j < t_i$. Essentially, the first component of the time interval represents the time the link was discovered, and the second component when the page was last accessed.

Note that we do not know what happened to a link before its start time nor do we know what happened after its end time. Also, it might be the case that the link disappeared between two consecutive crawler accesses temporarily. For instance, a crawler was not able to successfully access the page, the server was down, the link was temporarily removed or the link was not on the crawler’s fetch list. There are many plausible explanations why a link that was access successfully before was not discovered during a sub-sequent crawl. However, what we know is the time of link creation and the last time we were able to successfully access it. Our approach therefore is to approximate a link’s lifetime by not incorporating technical access difficulties of the crawler infrastructure. We argue that if an author places a hyperlink on a page, which is then first discovered by a crawler, we should use this access time as its start time. A link should always have its end date on the date of the latest successful crawl access. Every time we successfully access a link we can update the correspondent end date of the *TimeLink*. An analogy to justify this approach would be to consider the moon orbiting the earth. An observer would when first discovering the moon write down its discovery time, and then periodically check if its still there. At some point (probably during day light) the moon would not be visible anymore. However, it’s still there and it will be visible at some point later in time. We can see that declaring that the moon is not there anymore if not observed is not justified. If the moon is discovered at a later time again we can assume that it did not really disappear and was still there all the time, otherwise the same object would have been created twice (which doesn’t make sense if we treat links as original objects).

Let TL be the collection of all time links. TL can then be transformed into a regular graph using a transformation function $f(TL, t)$ that produces G_t , which takes two arguments: The time link collection TL , and a point in time t , which results in the Web graph G_t . We will describe in the next section in more detail how this transformation can be done.

Based on this idea we define a new dynamic Web graph $G_t = (V, E)$ at a moment in time t . Although the structure of this graph is subject to change over time, we refer to G_t as the *Web graph*. We will define later algorithms that span over several time intervals of this graph and calculate various measurements.

4.2 Containment model for Time Links

The collection of time links TL can become very large considering several billion of links, and several content versions per URL. It is therefore desirable to exploit a containment model for time links for compression purposes to reduce the size of TL without losing structural information.

We therefore define a sequence t_i of points in time, where $1 \leq i \leq n$, and $1 \leq j \leq n$. If $i < j$ then $t_i < t_j$, that is t_i is before t_j on the time line (happens to be earlier). Consider then two time links $tl_1 = (s, d, [t_i : t_j])$, where $i < j$ and

$tl_2 = (s, d, [t_k : t_l])$, where $k < l$. Both have the same source and destination URL, but a (possibly) different time interval. We have three cases when these time intervals overlap. This containment model is convenient, since it allows us to collapse time links, and still capture the same link semantics in a more efficient representation (that is it leads to a smaller collection TL).

- $k < i$ and $l \geq i$ and $l \leq j$. In this case both intersect, and we can derive a new time link $tl_{12} = (s, d, t_k : t_j)$ that contains the collapsed version of both time links in a more efficient representation.
- $i < k$ and $l \leq j$. In this case tl_2 is contained within tl_1 . We can therefore discard tl_2 to capture the same link structure.
- $k \geq i$ and $k \leq j$ and $l > j$. Here we can derive a new time link $tl_{12} = (s, d, t_i : t_l)$ without losing semantics.

4.3 The structure of the Web at a particular moment in time

As mentioned earlier we can use the time link collection TL to generate the Web graph G_t using a transformation function $f(TL, t)$. Conceptually an algorithm that would implement f would first perform a compression step described in the previous section, and then perform a linear scan over all time links in TL . It would start with an empty Web graph G_t and then gradually add nodes and edges to it. Basically, for each $tl \in TL$ it would make a decision whether to add the source and destination nodes, and the edge between these to G_t . Such a decision function $c(tl, t)$ would simply check whether t is in the time interval of the time link. More precisely, if $tl = (s, d, t_i : t_j)$, and $i \leq t \leq j$, then we add the nodes s and d to G_t , as well as the edge (s, d) .

We can see that such a transformation can be done in $O(n \cdot \log n)$ time: The first compression step requires a sorting on source URL, and then destination URL. Links with the same source and destination URL are then candidates for compression.

5 An Algebra for TimeLinks

We define an algebra for *TimeLinks* to manipulate them and to be able to extract link information from collection of *TimeLinks*.

The *TimeLink* algebra consists of several operators and functions that operate on the three concepts of time stamps, *TimeLinks*, and *TimeLink* sets.

A time stamp represents a specific moment or point in time. *TimeLinks* are triples $(s, d, [t_i : t_j])$, where s is the source URL, pointing to the destination URL d during the time interval $[t_i : t_j]$, and *TimeLink* sets are a means to group *TimeLinks*.

The semantics should be clear and as expected. In future work we will need to go in more depth and details on this.

5.1 Comparing TimeLinks

For comparison we introduce and define the following operators:

- equal
- contains
- inside
- after
- before
- overlap

5.2 Basic Operators

We define the basic set operators for *TimeLink* sets as follows:

- union
- intersection
- exclusion

5.3 Positional operators

These are operators that express relationships between *TimeLinks* according to their order on the time line.

- Indexing $TL[i]$. The index operator $[]$ extracts the i th element of a *TimeLink* set. We apply standard numbering from $0 \dots Size(TL) - 1$.
- S before/not before T
- S directly before/not directly before T
- S after/not after T
- S directly after/not directly after T
- S overlap/not overlap T

5.4 Containment operators

These operators express containment relationships between *TimeLink* sets.

- S inside/not inside T
- S contain/not contain T
- S without T

Average number of links per HTML doc	20
Average number of docs per DAT file	12,000
Average number of DAT files per machine	7,000
Average number of links per machine	1.8 Billion
Total number of links on 4 machines	7.14 Billion
Estimated number of links on 200 machines	360 Billion

Figure 1: Link statistics

6 Experiments with the Internet Archive data

The Internet archive currently comprises over 100 Terabyte of compressed data. This data is collected from regular crawls and spans roughly from 1995 to present. We built basic tools and infrastructure for *TimeLink* processing, and a run-time component that can be used on top of *TimeLink* collections to extract information.

6.1 Statistical information on compiling Timelink collections

We compiled statistics on how many links we extracted from DAT files (these are pre-processed files that contain extracted link information), how many were converted, and eventually merged.

We were processing DAT files from 4 machines (ia00100 to ia00103) to test and optimize our tools. During the past weeks we processed about 7.2 billion links. The conversion step eliminated about 1.2 billion links, resulting in 6 Billion TimeLinks before the merge process. Elimination takes place when we have two links which share the same source and destination, but have different time stamps. Using the containment model described earlier these two can be represented as one *TimeLink*.

The merge process was then carried out first within each directory, then over several directories, and eventually we copied the data to one big machine (ia00400) to process a merge over different machines to further compress the *TimeLink* collection. Overall the merge process reduced the size of the *TimeLink* collection by 1% only. We expected more compression here initially. However, based on the findings we expect now a higher compression rate for the merge once we have a larger number of machines processed. For the merge to have a good compression rate we need to have a large common base of links over time.

Overall we have to process about 3 billion of text lines per machine It takes 3-5 sec. to fully process 100 000 lines (1.5 days). As we pointed out earlier there is much room to improve efficiency.

6.2 Number of links over time

We compiled a large *TimeLink* collection of roughly 2 billion links. On this collection we applied the tool `linksPerMonthTL` that counts the number of

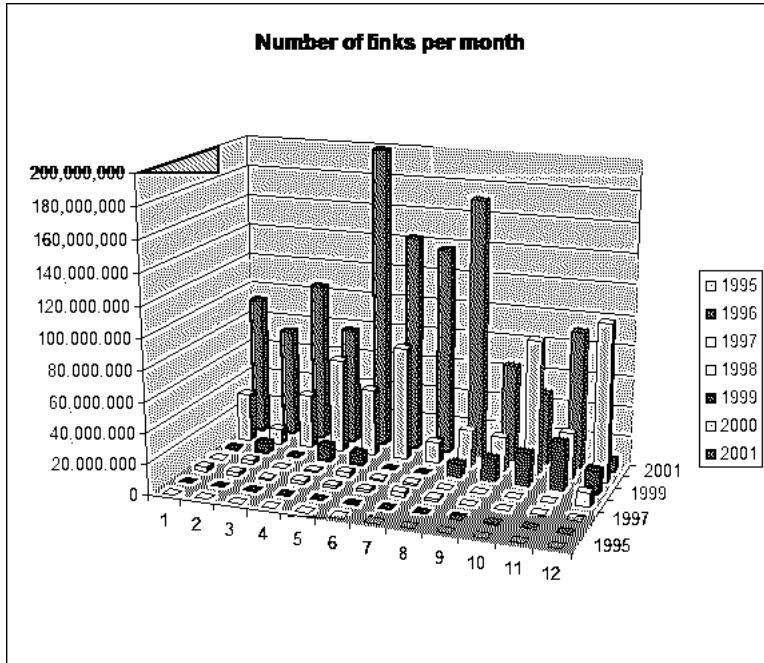


Figure 2: Number of links per month between 1995 and 2002

links per month given a time interval. In this experiment we specified the time interval from 1995 to 2002 (inclusive). The tool then prints out a statistics on number of links per month, i.e. the size of a Web graph in each month during that time. In this scenario a *TimeLink* can result in multiple counts if it spans over several months. We used data that was distributed over four machines taken from one directory on each of these machines to generate the *TimeLink* collection.

It is interesting to look at the distribution of links (see figure 2). We can see that crawls are spread over machines. For instance, if one directory or one machine would store data of one crawl only, then all the link data would go into one bucket.

Furthermore, we can see that the number of links per month is continuously growing with exponential growth over the years (see figure 3). For example, in 2000 the average number of links per month is about 45 million, whereas in 2001 it is about 100 million. Since crawlers are collecting more data nowadays from the fast growing Web we are able to extract more links from the most recent years (e.g., 2001).

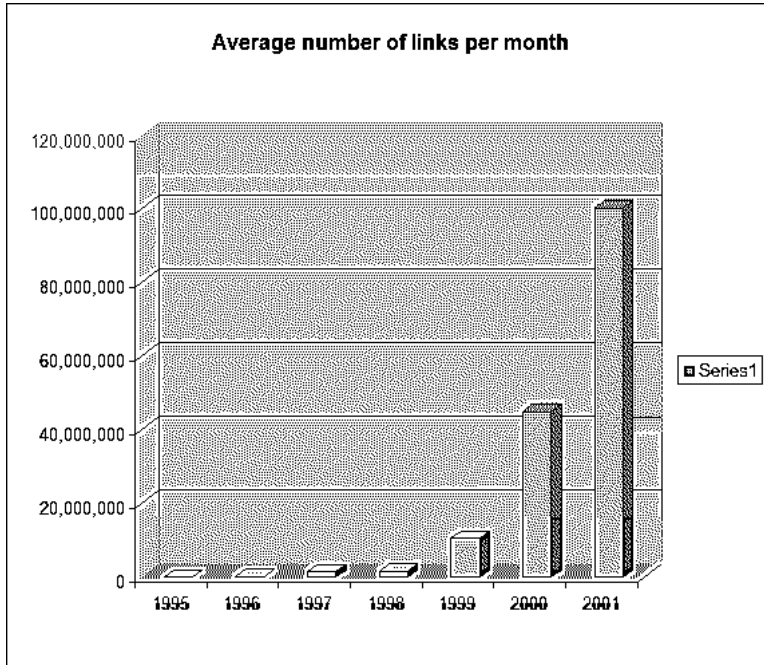


Figure 3: Average number of links per month between 1995 and 2002

7 Future Work

On a very large *TimeLink* collection we then plan to use the discussed tools for data extraction to build five Web graphs at different moments in time. The crawler of the Internet archive typically works in a batch mode, so we need to pick times of different crawls to get data that shows how the Web evolved both from crawl-to-crawl, or over several crawls (multi-crawl).

We then want to compile more interesting statistics (both crawl-to-crawl statistics and multi-crawl statistics) on how the link structure of the Web evolves over time.

First, we want to explore how the number of outlinks for a set of documents changes over time to learn about dynamics of hyperlinks.

Second, we want to investigate how the set of incoming links per document (*indegree*) for a given sample set of documents evolves over time.

Third, similar to the *indegree* of a page the *host-indegree* of a document counts the number of incoming links on a per host basis. As an example consider a document that has three incoming links, so the *indegree* of that document is three. However, all of these links are from the same server. That results in a *host-indegree* of only one. We are interested on how the *host-indegree* changes over time, and how the correlated to the *indegree*.

8 Conclusion

We showed that *TimeLinks* do not rely on possibly faulty Metadata (e.g., file-names) to identify collection of links that were crawled at the same time to compile crawl-to-crawl or multi-crawl statistics.

Moreover, it handles the problem of overlapping crawl collections elegantly, since each link itself has an associated time interval: If the same link was accessed twice in an overlapping crawl we can use the *TimeLink* algebra to combine these.

To sum up *TimeLinks* represent a convenient and robust infrastructure for extracting, collecting, and analyzing the Web graph over time.

A formal Algebra for manipulation and operations on *TimeLinks* and *TimeLink* collections provides a sound theoretical foundation.

We plan to extend the *TimeLink* infrastructure over time at the Internet archive as a basis for future research and projects on Web graph algorithms.

9 Acknowledgements

We are grateful to Raymie Stata for all his comments and helpful discussions. Furthermore, we thank Bruce Baumgart, Brad Tofel and Igor Ranitovic for their help and advice related to the Internet archive infrastructure.

References

- [1] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web (winner best paper award). In *Proceedings of the Ninth International World Wide Web Conference*, 2000.
- [2] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, 2000.
- [3] Jon M. Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan, and Andrew S. Tomkins. The Web as a graph: Measurements, models and methods. *Lecture Notes in Computer Science*, 1627:1–??, 1999.
- [4] K. Randall, R. Stata, R. Wickremesinghe, and J. Wiener. The link database: Fast access to graphs of the web. In *2002 Data Compression Conference (DCC)*, pages 122–131. IEEE, April 2002. Also published as SRC Research Report 175, Compaq Systems Research Center, November 2001.
- [5] Dave Sherfese. Alexa archival url specification. February 2000.