

Project #2 – Big Data

Compressing large collections of web pages

Federico Conte

fconte90@gmail.com

<https://github.com/Draxent>

University of Pisa
Department of Computer Science
Master in Computer Science and Networking

20th March 2015

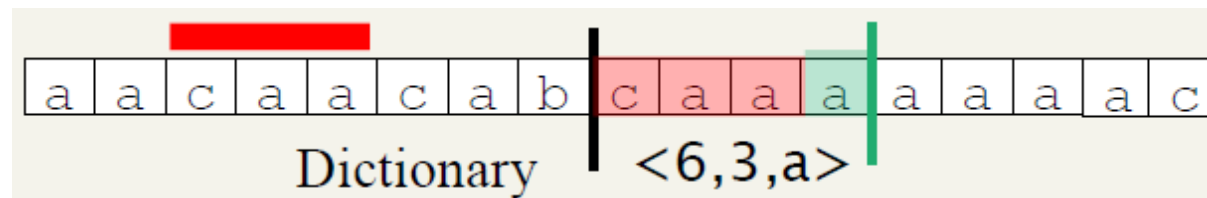
Problem Statement

INPUT: A file containing a collection of **Web pages**

OUTPUT: A file containing the permuted collection, where the permutation is driven by the **similarity** between pages

GOAL: Find the best way of permuting in order to **minimize the compression** of the output file (using Lzma2 compressor)

↓ LZ77



Analysis of the Web pages

```
warc/0.9.10757 response http://www.mattenltd.co.uk/0061006192531 message/http :
BUbING-guessed-charset: iso-8859-1
BUbING-content-digest: eceb3de18182b34aedab72fb9a62846

HTTP/1.1 200 OK
content-type: text/html
accept-ranges: bytes
connection: close
server: Apache/1.3.33 (Unix) mod_auth_passthrough/1.8 mod_log_bytes/1.2 mod_bwli
content-length: 10123
last-modified: Thu, 13 Oct 2005 22:41:57 GMT
etag: "4f1df5-278b-434ee2b5"
date: Wed, 05 Jul 2006 07:44:13 GMT
ubi-http-equiv-charset: iso-8859-1

<html>
<head>
<META name="description" content="Matten Press Makers">
<META name="keywords" content="Press, Hand, ...">
<title>Matten Engineering</title>
<script language="JavaScript" type="text/JavaScript">
  <!--
    function MM_openBrWindow(theURL,winName,features) { //v2.0
      window.open(theURL,winName,features);
    }
  //-->
</script>
```

Start new web Page

Length of the Page

URL

Content-Type

Start HTML Page

TAG

Word

Script

Analysis of the Web pages

Assumption:

If someone **COPY** some TAGs / WORDs / SCRIPTs from an HTML page, he will **NOT MODIFY** them.

Start new web Page

Length of the Page

URL

Content-Type

Start HTML Page

TAG

Word

Script

```
<html>
<head>
<META name="description" content="Matten Press Makers">
<META name="keywords" content="Press, Hand, ...">
<title>Matten Engineering</title>
<script language="JavaScript" type="text/JavaScript">
  <!--
    function MM_openBrWindow(theURL,winName,features) { //v2.0
      window.open(theURL,winName,features);
    }
  //-->
</script>
```

Proposed Solution

Algorithm: *CompressingWebPages(InputFile)*

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b, r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

Min-Hashing: *used to permute the fingerprints*
Initialize 20 MURMUR HASH function

Proposed Solution

Algorithm: *CompressingWebPages*(InputFile)

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

LSH-initialization:

- pre-compute sampling of r elements from *Sketch*
- create b hash tables with chaining

Proposed Solution

Algorithm: *CompressingWebPages(InputFile)*
begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

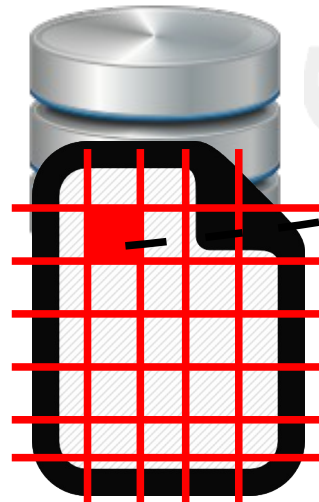
```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

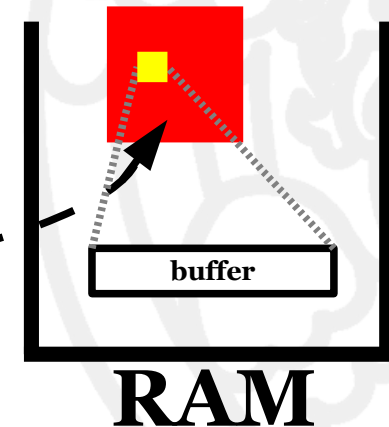
```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

Scanning File:



memory
mapping



Proposed Solution

Algorithm: *CompressingWebPages*(InputFile)
begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()

while(NOT sf.EOF)
{
    wp <- Recognize in sf a Web page
    if (wp.Signature != NULL)
    {
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else notCategWP.Add(wp)
}

foreach wp in categWP
{
    simDoc <- sort lsh.UnionFind(wp) by URL
    calculate permutation order of simDoc
}

simDoc <- sort notCategWP by URL
calculate permutation order of simDoc

foreach wp in (categWP union notCategWP)
    write wp in OutputFile

end
```

Recognize Web page:

- Divide the Web page in **words** (*TAGs* / *WORDS* / *SCRIPTs*) → pair (start, length)
- Karp-Rabin hashing for every **word**
- Shingling of *x*-words, with *x* depending #characters (*Q*=25)
- Karp-Rabin hashing for every **shingle**
- Sketch Vector using **Min-Hashing** (*pi*)
- Return the end of the page → next step will start here

Proposed Solution

Algorithm: *CompressingWebPages(InputFile)*

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
```

```
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
```

```
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

LSH-AddDocument:

- pick **3** elements from the Sketch
- compute the sum
- add result to 1 of **37** buckets
- if there are collision → we list them

Threshold
~30%

Proposed Solution

Algorithm: *CompressingWebPages(InputFile)*

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

← **Calculate Permutation:** *For every page:*

- *find all similar pages (union find LSH)*
- *sort by URL*
- *calculate the position of the permutation*

```
foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

Proposed Solution

```
Algorithm: CompressingWebPages(InputFile)
begin
  pi <- CalculatePermutations(20)
  lsh <- new LSH(b , r)
  sf <- new ScanningFile(InputFile)
  categWP <- new Set(), notCategWP <- new Set()

  while(NOT sf.EOF)
  {
    wp <- Recognize in sf a Web page
    if (wp.Signature != NULL)
    {
      lsh.AddDocument(wp)
      categWP.Add(wp)
    }
    else notCategWP.Add(wp)
  }

  foreach wp in categWP
  {
    simDoc <- sort lsh.UnionFind(wp) by URL
    calculate permutation order of simDoc
  }
  simDoc <- sort notCategWP by URL
  calculate permutation order of simDoc

  foreach wp in (categWP union notCategWP)
    write wp in OutputFile
end
```

Write on OutputFile:

Since we have calculated the new position that the Web pages will occupy:

- scan sequentially the **input file**
- write randomly into **output file**

Results – 1^o phase

Algorithm: *CompressingWebPages*(InputFile)

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
{
    wp <- Recognize in sf a Web page
    if (wp.Signature != NULL)
    {
        lsh.AddDocument(wp)
        categWP.Add(wp)
    }
    else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
{
    simDoc <- sort lsh.UnionFind(wp) by U
    calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc
```

```
foreach wp in (categWP union notCategWP)
    write wp in OutputFile
```

end

Dimension (K)	Time
1	00:00:01
10	00:00:26
100	00:05:26
3000	>> 2 days



CompressingWebPages

2%	94%	100%
CPU	Memory	Disk
1,5%	2.971,6 MB	1,3 MB/s

Results – 2^o phase

Algorithm: *CompressingWebPages(InputFile)*

begin

```
pi <- CalculatePermutations(20)
lsh <- new LSH(b , r)
sf <- new ScanningFile(InputFile)
categWP <- new Set(), notCategWP <- new Set()
```

```
while(NOT sf.EOF)
```

```
{
  wp <- Recognize in sf a Web page
  if (wp.Signature != NULL)
  {
    lsh.AddDocument(wp)
    categWP.Add(wp)
  }
  else notCategWP.Add(wp)
}
```

```
foreach wp in categWP
{
  simDoc <- sort lsh.UnionFind(wp) by URL
  calculate permutation order of simDoc
}
simDoc <- sort notCategWP by URL
calculate permutation order of simDoc

foreach wp in (categWP union notCategWP)
  write wp in OutputFile
```

end

Dimension (K)	Time
1	00:00:02
10	00:01:07
100	00:27:46

Results – compression

Dimension (K)	Original Compression	Achieved Compression
1	89.75%	89.83%
10	91.41%	91.97%
100	93.53%	94.64%

Future Work

■ Parallelize:

- File Scanning: find **#pages** and **start & length** of each page
- Divides **#pages** in group depending **#CPU**
- Computes **signature** of each group in **parallel**
- After found the final permutation, in **parallel**, reads each group of pages and write them into the output file, accordingly the permutation

■ Save data of Web pages on file:

- Avoids stressing the memory
- Collects information when needed

Assumption:

If someone **COPY** some TAGs / WORDs / SCRIPTs from an HTML page, he will **NOT MODIFY** them.

■ Remove initial assumption:

- Collects from the **shingle 3** groups of 7 random characters each
- Compute **Karp-Rabin** hashing of each group → **Sketch 20×3**

Future Work

Thanks !